



Matthias Haun

# Handbuch Robotik

Programmieren und Einsatz intelligenter  
Roboter

*2. Auflage*

VDI



Springer Vieweg

---

# Handbuch Robotik

---

Matthias Haun

# Handbuch Robotik

Programmieren und Einsatz intelligenter  
Roboter

2. Auflage



Springer Vieweg

Matthias Haun  
Altrip  
Deutschland

ISBN 978-3-642-39857-5      ISBN 978-3-642-39858-2 (eBook)  
DOI 10.1007/978-3-642-39858-2

Die Deutsche Nationalbibliothek verzeichnetet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <http://dnb.d-nb.de> abrufbar.

Springer Vieweg  
© Springer-Verlag Berlin Heidelberg 2007, 2013  
Das Werk einschließlich aller seiner Teile ist urheberrechtlich geschützt. Jede Verwertung, die nicht ausdrücklich vom Urheberrechtsgesetz zugelassen ist, bedarf der vorherigen Zustimmung des Verlags. Das gilt insbesondere für Vervielfältigungen, Bearbeitungen, Übersetzungen, Mikroverfilmungen und die Ein-speicherung und Verarbeitung in elektronischen Systemen.

Die Wiedergabe von Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. in diesem Werk berechtigt auch ohne besondere Kennzeichnung nicht zu der Annahme, dass solche Namen im Sinne der Warenzeichen- und Markenschutz-Gesetzgebung als frei zu betrachten wären und daher von jedermann benutzt werden dürften.

Gedruckt auf säurefrei und chlorfrei gebleichtem Papier

Springer Vieweg ist eine Marke von Springer DE. Springer DE ist Teil der Fachverlagsgruppe Springer Science+Business Media  
[www.springer-vieweg.de](http://www.springer-vieweg.de)

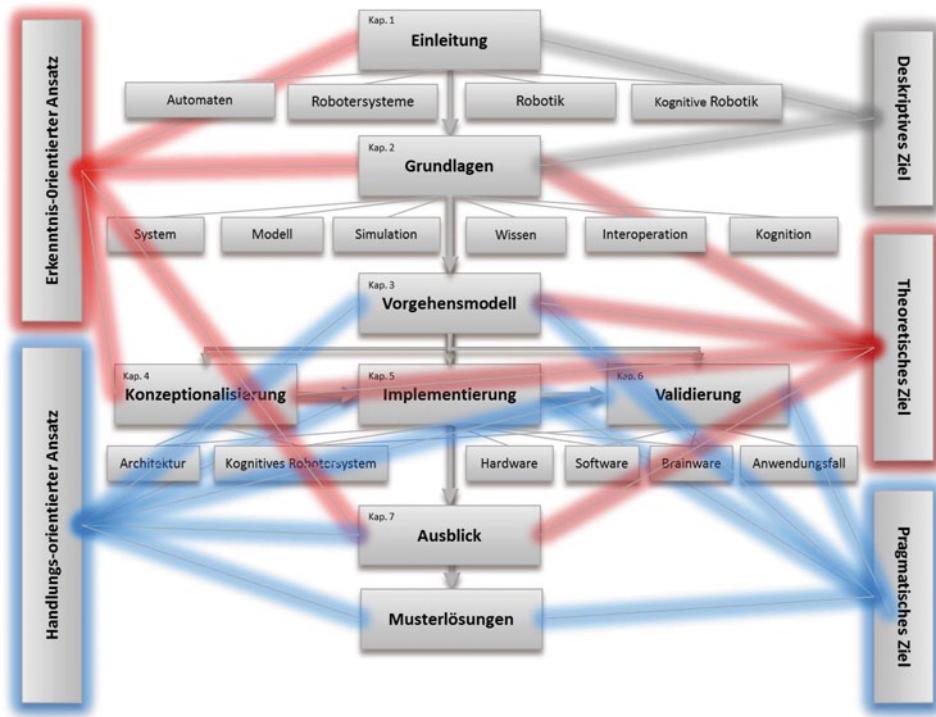
---

## Vorwort zur 2. Auflage

Nach bereits 6 Jahren erscheint jetzt die zweite Auflage des Handbuchs Robotik. Inhaltlich wurden die Sachverhalte dabei sorgfältig überarbeitet und aktualisiert. Eine Reihe von Abbildungen wurde neu konzipiert oder überarbeitet und bei einigen Kapiteln waren Ergänzungen notwendig. Hierbei wurden die zahlreichen Rückmeldungen und Vorschläge weitgehend berücksichtigt. In Anbetracht der rasanten Entwicklung der Forschung, Lehre und Praxis und der dadurch bedingten Einarbeitung der erzielten Ergebnisse, konnte die Konzeption und die Gliederung leider nicht beibehalten werden. Die neue Konzeption, als auch die neue Gliederung spiegelt den aktuellen Ansatz dieses Buches wieder, den Roboter nunmehr als kognitives Modell und damit als wissensbasierten Agenten zu konzeptionalisieren und dieses Agentenmodell durch rechnerbasierte Cognitive Computing Technologien in prozessualer und funktionaler Hinsicht zu einem kognitiven Robotersystem auszuimplementieren. Das Ziel eines solchen Ansatzes liegt in der Steigerung des systemischen Intelligenzquotienten ( $IQ_s$ ) des kognitiven Robotersystems. Das Buch wendet sich gleichermaßen an Studierende, Fachleute aller Fachrichtungen als auch den interessierten Leser. Indem die einzelnen Kapitel einen Brückenschlag zwischen Standardwissen und Wissen aus Nachbargebieten, wie Kognitionswissenschaft oder Informatik darstellen, versucht dieses Handbuch ein tiefgreifendes Verständnis des komplexen Themengebietes „Robotik“ zu ermöglichen (Abb. 1).

Speziell strebt das Buch drei Ziele an. Als *deskriptives* Ziel beschreibt es die Begriffe der technologisch induzierten Robotik. Das *theoretische* Ziel besteht darin, die Konzepte, Theorien und Modelle so zu erfassen, so dass der Leser nicht nur zur Entwicklung von Robotersystemen befähigt wird, sondern darüber hinaus auch über Möglichkeiten der Erkenntnisgewinnung in Bezug auf die der Robotik inhärenten wissenschaftstheoretischen Fundamente und angewendeten Technologien verfügt. Letztlich umfasst das *pragmatische* Ziel, praktische Empfehlungen und Muster dem Leser zur Bewältigung der Herausforderungen an die Hand zu geben.

Als Weg zu diesen Zielen verfolgt das Buch einen erkenntnis- und handlungsorientierten Ansatz. Dadurch werden Theorie (Modell) und Praxis (Technologie) miteinander verknüpft.



**Abb. 1** Ansatz, Ziel und Buchdesign

Trotz dieser Ziele und der angekündigten Erweiterungen bleibt das Buch auch weiterhin voraussetzungsfrei lesbar.

Altrip, im Juni 2013

Matthias Haun

---

## Vorwort zur 1. Auflage

Die Robotik stellt sich bisher als ein weit ausgedehntes Forschungsgebiet dar, dass zahlreiche Disziplinen streift, hier und da sich in diesen vertieft und damit eher an eine zerklüftete Bergkette als an eine harmonische Landschaft erinnert. In diesem Buch wird die Robotik als Wissenschaft formuliert, verstanden als Gesamtheit naturwissenschaftlicher Analysen von Erkennen, Wissen und Handeln in allen Dimensionen und Funktionsweisen von Systemen. Legitimiert wird dies auch dadurch, daß die Robotik in den letzten Jahren theoretisches wie praktisches Interesse weit über wissenschaftliche Fachgrenzen hinaus gefunden hat. Sie gilt als die bedeutendste theoretische und technische Revolution seit der Atomphysik, mit unabsehbaren Folgewirkungen auf die gesellschaftliche Entwicklung dieses Jahrhunderts.

Ihre Wirksamkeit liegt vor allem darin, dass Wissenschaft und Technik hier eine auch von außen unübersehbare enge Wechselbeziehung eingegangen sind. Mit der Entstehung der Robotik als Wissenschaft haben sich gesellschaftliche Problemfelder, wie Daten, Information, Kommunikation, Wissen, deren Verarbeitung oder die Robotik selbst revolutionär verändert bzw. neu konstituiert.

Insofern wird die Robotik bewusst als Wissenschaft betrieben. Eine solche Wissenschaft treiben, wird als eine besondere Form sozialen Handelns nach bestimmten Regeln aufgefasst, mit dem Ziel, Strategien zur Lösung von Problemen zu entwerfen. Solche Problemlösungsstrategien sind explizite Konstruktionen, die ein Wissenschaftler entwickelt, um nach ihren Kriterien Erklärungsdefizite zu beheben, die in dem jeweiligen Problembereich als Bedarf vorliegen. Solche Lösungsstrategien werden in diesem Buch als Theorien vorgestellt. Ein in diesem Sinne wissenschaftliches Handeln liefert Problemlösungstechniken, deren Lösungsverfahren interpretierbar sind und zu einem anwendbaren Wissen führen. Gerade dieser wissensorientierte Problemlösungsansatz skizziert Lesern ein Modell wissenschaftlichen Handelns, dass das Ziel der systematischen Problemlösung nach wissenschaftlichen Kriterien verfolgt.

Trotz ihrer theoretischen Ergiebigkeit und praktischen Effizienz ist die Wissenschaft der Robotik bis heute noch keine etablierte reife Naturwissenschaft mit einer homogenen Forschergemeinschaft, sondern eher ein multidisziplinäres, dynamisches Forschungsfeld, in dem sich disziplinäre Ansätze überschneiden und gegenseitig beeinflussen.

Dieses Buch stellt somit einen engagierten Versuch dar. Ein Spezifikum dieses Versuches liegt darin, daß innerhalb der 600 Seiten die Wissenschaft der Robotik und Robotiktechniken nicht voneinander getrennt werden, sondern postuliert wird, daß Vitalität und Zukunftspotential dieses Forschungsbereiches erst dann in den Blick kommen, wenn man den Grundlagenaspekt mit den technischen Aspekten zusammen sieht.

Damit bezieht der Autor bewusst eine wissenschaftssoziologische und auch wissenschaftspolitische Position, die Heterodoxie, Intradisziplinarität und Vielfalt propagiert und jeweils forschungsdominierende Ansätze aus gründlicher Kenntnis einer gründlichen Kritik unterzieht. Ihr Engagement bezieht diese Position aber nicht nur aus einer allgemeinen skeptischen Einstellung; gerade im Fall der Robotik kann der Autor zu Recht auch darauf verweisen, dass gegenwärtig höchst kreative kognitive Denkmodelle bzw. Gedankenexperimente, wie die der Selbstorganisation und Emergenz schon vor fast dreißig Jahren entwickelt worden sind, aber wieder vergessen wurden, weil sie der kognitivistischen Orthodoxie zufolge nicht in die Konzepte der „klassischen“ Robotik passten.

Die in diesem Buch erstmals verfolgte Strategie ist klar und wirksam: Auf Basis der bereits klassischen Informationsverarbeitung entwickelt der Autor deren basale theoretische Konzepte (Daten, Information, Symbol, Repräsentation) weiter aus (Wissensverabreitung). Mit anderen Worten, die naturwissenschaftliche Erforschung des menschlichen Erkennens und Handelns bietet der Robotik mit ihrer Technik einen so noch nie wahrgenommenen Fundus, der weit über Philosophie<sup>1</sup>, Psychologie und Informatik<sup>2</sup> hinausführt. Ein Spezifikum dieser naturalistischen Erkenntnistheorie liegt im schöpferischen Zusammenspiel von Forschung, Technik und Öffentlichkeit, das die Geschwindigkeit wissenschaftlicher, wie technischer Entwicklungen von Beginn an geprägt hat. Dieses Zusammenspiel drückt sich auch im Interesse der Massenmedien für wissenschaftliche Fragestellungen aus. Themen, wie „künstliche Intelligenz“, „maschinelles Sprach- und Bildverständen“, „Wissens-“, bzw. „Expertensysteme“, oder letztlich „intelligente Roboter“, prägen auch die populärwissenschaftliche Diskussion der letzten Jahre.<sup>3</sup>

Ein Ergebnis dieser Diskussion ist sicher auch darin zu sehen, dass heute fast jeder ohne Probleme von Information- und Informationsverarbeitung spricht und das Gehirn als einen informationsverarbeitenden natürlichen Computer betrachtet. Gerade der Informationsbegriff<sup>4</sup> und die damit verbundenen Vorstellungen sind höchst problematisch. Werden Informationen wirklich, wie Kognitionstheoretiker immer wieder sagen, aufgenommen, gespeichert, kommuniziert und verarbeitet, oder entstehen Informationen und Wissen nicht erst durch kognitive Tätigkeiten?

---

<sup>1</sup> Vgl. auch Stork 1977.

<sup>2</sup> Vgl. Ernst 2003.

<sup>3</sup> Vgl. Lee 1991.

<sup>4</sup> Vgl. Capurro 1978.

Die einzelnen Konzepte dieses Buches gehen mit einer Neuorientierung der kognitivistischen Konzepte von Symbolverrechnung und Repräsentation einher. Diese Konzepte implizieren drei ontologisch wie erkenntnistheoretisch folgenreiche Annahmen:

- Die Welt ist vorgegeben.
- Die Kognition bezieht sich auf diese Welt – wenn auch oft nur auf einen Teil derselben.
- Die Art, auf die diese vorgegebene Welt erkannt wird, besteht darin, ihre Merkmale abzubilden und sodann auf der Grundlage dieser Abbildungen zu agieren.

Diese Annahmen werden durch einen wissenschaftstheoretischen Ansatz flankiert, der einerseits im kritischen Rationalismus basiert und andererseits auf den Erkenntnissen der Phänomenologie und der Hermeneutik aufbaut. Dieser wissenschaftstheoretische Ansatz, unter Einbeziehung der Systemtheorie liefert folgende Annahmen:

- Das Ganze ist nicht gleich der Summe der Teile.
- Komplexe Systeme sind vernetzte, dynamische Ganzheiten.
- Offene Systeme sind mit ihrer Umwelt vernetzt und tauschen mit ihr Materie, Energie und Informationen aus.
- Das Verhalten komplexer Systeme lässt sich nicht im Einzelnen vorhersehen, jedoch beeinflussen.
- Komplexe Systeme weisen erkennbare Ordnungsmuster auf, die gestaltet werden können.
- Lenkung (Steuerung, Regelung) hält ein System unter Kontrolle.
- Rechnerbasierte Robotersysteme können lernen, sich entwickeln und entsprechend intelligent handeln.
- Fehlertoleranz ist oftmals erfolgreicher als Exaktheit.

Das Buch plädiert für eine Modellierung der erkenntnistheoretischen Grundfrage, die Wahrnehmung und Erkenntnis als handlungsbezogene kreative Dimensionierungen von Bedeutungen im Kontext des Lebenszyklus eines Systems begreift. Die kognitiven Module eines solchen Systems bringen ständig Welten im Prozeß gangbarer Handlungsalternativen hervor; diese Module legen Welten fest statt sie zu spiegeln. Der Grundgedanke besteht dabei darin, daß intelligente/kognitive Fähigkeiten untrennbar mit dem Lebenszyklus eines Systems verflochten sind, wie ein Weg, der als solcher nicht existiert, sondern durch den Prozess des Gehens erst entsteht. Daraus folgt, dass eine Problemlösung nicht mit Hilfe von Wissensrepräsentationen erfolgt, sondern dass durch die kognitiven Komponenten in kreativer Weise eine Welt produziert/modelliert wird, für die die einzige geforderte Bedingung die ist, dass sie erfolgreiche Handlungen ermöglicht: sie gewährleistet die Fortsetzung der Existenz des betroffenen Systems mit seiner spezifischen Identität. Insofern erscheinen Probleme und deren Lösung nicht als vorgegeben, sondern werden, bildlich gesprochen, handelnd erzeugt, aus einem wie auch immer gearteten Hinter- bzw. Untergrund hervorgebracht. Dieses Hervorbringen spiegelt die totale Zirkularität zwischen Handeln, Erkennen und Verstehen.

Zur Unterstützung dieser handlungsorientierten Theorie baut der Autor unter anderem auch auf konnektionistische Modelle. Während orthodoxe Modelle Kognition nach dem Muster logischer Rechenprozesse mit symbolischen Repräsentationen konzipieren, arbeiten konnektionistische Modelle mit der Annahme, dass Gehirne neuronale Netzwerke sind, die auf der Grundlage zahlloser, weit verzweigter Verknüpfungen arbeiten. Die Beziehungen zwischen Neuronengruppen verändern sich durch Erfahrungen und Lernprozesse, d. h. sie zeigen Fähigkeiten der Selbstorganisation, die sich in der Logik nirgendwo finden.

Damit erweist sich der aus dem Kognitivismus hervorgegangene Konnektionismus als wichtiger Ausgangspunkt für die handlungsorientierte Interaktionstheorie, die erstmalig in diesem Buch vorgestellt wird. In dieser Konzeption spielen aber auch Konzepte der Symbolverarbeitung und Repräsentation eine entscheidende Rolle. Alle Ansätze zusammen bewirken in ihrer Kombination eine Steigerung des systemischen Intelligenzquotienten eines Systems ( $IQ_s$ ). Dabei wird der übliche Intelligenzbegriff nicht länger verstanden als Fähigkeit des Problemlösens, sondern als Fähigkeit eines Systems, in einer mit anderen geteilte Welt aktiv einzutreten.

So liegt denn auch ein weiterer Schwerpunkt des Buches eben nicht nur auf dem technischen Aspekt der Robotik, wie beispielsweise dem Bau von Robotern (Mechanik), der Steuerung der Gelenke (Elektronik) oder der Mechatronik (als die Verbindung von Mechanik und Elektronik). Vielmehr beschreibt das Buch auch die Möglichkeiten der Programmierung von Robotersystemen. Am Ende dieser Kapitel wird sich dann zeigen, dass in der zukünftigen Brainware das Potenzial zu suchen ist, was letztlich Roboter zu intelligenten Robotersystemen avancieren lässt.

Welche wissenschaftlichen, philosophischen, ethischen und technischen Konsequenzen mit dieser Konzeption verbunden sind, versucht der Autor am Ende des Buches, sozusagen als Ausblick und Motivation, deutlich zu machen.

Neben dieser inhaltlichen Ausrichtung liegt dem Buch auch ein didaktisches Grundprinzip zugrunde, das davon ausgeht, dass die beste Methode der Auseinandersetzung mit diesem Gebiet darin besteht, eigene Robotersysteme zu entwickeln und zu erproben. Es ist inzwischen möglich, relativ günstig Bausätze oder Bauteile zu kaufen, aus denen sich kleine, aber dennoch leistungsfähige Robotersysteme entwickeln lassen. Was hierzu fehlte, war ein allumfassendes Buch, das in die grundlegenden Probleme, Konzepte und Techniken so einführt, dass eine direkte Umsetzung des vermittelten Wissens in die Praxis möglich ist. Auch diese Lücke möchte das vorliegende Buch schließen.

Insofern wendet sich das Buch an Philosophen, Psychologen, Informatiker, Ingenieure der Fachrichtungen Maschinenbau, Elektrotechnik und Mechatronik, Praktiker in der Fertigungswirtschaft, Studenten aller technischen Studiengänge, sowie alle Personen, die Interesse an Robotern zeigen.

Das Konzept dieses Buches impliziert, dass sich die behandelten Themengebiete über viele Fachgebiete erstrecken. Es ist evident, dass der Autor eines solchen Buches nicht Spezialist für die einzelnen Themen sein kann. Insofern sei an dieser Stelle die Bitte erlaubt, dass die Fachleute der einzelnen Fachgebiete, denen die Darstellung ihres Spezialgebietes

zu kurz oder zu oberflächlich erscheint, Nachsicht üben. Wenn dieses Buch auch an vielen Stellen ein Kompromiss zwischen Gesamtschau und Detailblick darstellt, sind eventuelle Fehler allein dem Autor anzulasten. Einen Hinweis auf diese, sowie Anregungen oder konstruktive Kritik nimmt der Autor gerne unter matthias.haun@web.de entgegen. Immerhin lebt nicht nur die Wissenschaft, sondern auch ein Buchvorhaben von solchen Dialogen.

Jeder, der Ergebnisse langwieriger Überlegungen zu formulieren versucht, hat sicherlich die Erfahrung gemacht, daß man vor seinem Schreibtisch oder vor dem Computer sitzt, auf die geschriebene Seite, deren Absätze und Sätze schaut und sich dann zusichert, dass eigentlich alles in Ordnung zu sein scheint. Doch plötzlich macht sich eine Unsicherheit breit, indem Fragen wie „Woher habe ich das?“, „Habe ich es irgendwo gelesen?“ auftauchen. Man zerbricht sich den Kopf, wer das wohl so oder in ähnlicher Form formuliert haben könnte. Bei dem Schreiben eines solchen Buches hat man eine Menge Bücher konsultiert und man verbringt manchmal Stunden und Tage damit, Ideen und Formulierungen nachzuspüren, gewöhnlich ohne Erfolg. Es ist wahrlich nicht leicht, in Hunderten von Seiten einen ganz bestimmten Satz zu finden, vor allem wenn man noch in verschiedenen Sprachen suchen muß. Greift man dabei auf mehrere Bibliotheken zu, ist es oft praktisch unmöglich, vergessene Quellen wieder aufzutreiben. Der Autor hat es inzwischen aufgegeben. Auch ist klargeworden, dass alles, was man schreibt, schon einmal gesagt oder geschrieben worden ist. Ideen sollten ohnehin niemals Privatbesitz sein, denn es kommt ja darauf an, was man aus ihnen macht. Der Autor hat daher versucht wo immer möglich, die Quellen anzugeben und lebt in der Hoffnung, keinen falschen Gebrauch von all dem gemacht zu haben, was man sich bei solch einem Tun unwissentlich angeeignet hat.

Altrip, im Dezember 2006

Matthias Haun

---

## Lesehinweise

An gleich mehreren Stellen in diesem Buch werden sich mathematischer Formeln bedient und damit zunächst die nachvollziehbare Tatsache mißachtet, daß jede solche Formel den Leserkreis von vorne herein halbieren wird. Diesem Leserkreis wird daher empfohlen, auch hier das Prinzip des „Mutes zur Lücke“ zunächst zu verfolgen, diese mathematisch-kontaminierte Zeile völlig zu ignorieren und zur nächsten Textzeile überzugehen. Offener ausgedrückt: Man sollte die Formel eines flüchtigen Blickes würdigen, ohne sie unbedingt verstehen zu wollen, und dann rasch weiterlesen. Nach einer Weile kann man mit neuem Selbstvertrauen zu der vernachlässigten Formel zurückkehren und versuchen, sich diese Schritt für Schritt zu erschließen.

Um den Weg des Lesens zu erleichtern, wurden folgende Formatierungen benutzt:

- *Kursiv* wird für Hervorhebungen und für neue Wörter und Bezeichnungen benutzt, die im Text dann auch näher erklärt werden.
- Die Listingschrift wird für Programmanweisungen benutzt.
- In den durch das Symbol eingeleiteten Abschnitten werden interessante Themen näher beleuchtet oder aber durch konkrete Beispiele aus der Praxis verdeutlicht.
- In der Robotik und dort speziell beim Cognitive Computing werden sehr oft anthropomorphe Redeweisen verwendet, wie beispielsweise „Das Robotersystem weiß X“, „Das Robotersystem will Y“ oder gar „Das Robotersystem ist sich bewusst, dass Z“, ohne darauf hinzuweszen, dass es sich dabei um eine metaphorische und modellhafte Anreicherung von Sprache handelt. Erschwerend kommt hinzu, dass Begriffe wie beispielsweise „erkennen“, „wissen“ usw. verwendet werden, die in ihrer Daseinsberechtigung bzw. Geltung im Zusammenhang mit künstlichen Systemen zumindest hinterfragt werden müssen. Die Notwendigkeit einer solchen Reflexion bezüglich der Redeweisen bzw. Begrifflichkeiten wird einem anderen, dann allerdings wissenschaftsphilosophischen Buch überlassen. Um dieser Notwendigkeit allerdings bereits in diesem Buch Rechnung zu tragen, wird zumindest auf die fragliche Verwendung dieser Begrifflichkeiten durch deren Einbettung in Hochkommata hingewiesen.

---

# Inhaltsverzeichnis

<b>1 Einleitung als Motivation .....</b>	1
1.1 Mechanische Automaten .....	1
1.2 Robotersysteme .....	6
1.3 Klassische Robotik .....	14
1.4 Kognitive Robotik .....	27
Literatur .....	31
<b>2 Grundlagen .....</b>	33
2.1 System .....	33
2.2 Modell .....	54
2.3 Simulation .....	88
2.4 Wissen .....	95
2.5 Interoperation .....	103
2.6 Kognition .....	138
Literatur .....	152
<b>3 Vorgehensmodell .....</b>	155
3.1 Vorgehensmodelle .....	155
3.2 Methodik als Randbedingungen .....	164
3.3 Kognitives Vorgehensmodell .....	171
Literatur .....	183
<b>4 Konzeptionalisierung .....</b>	185
4.1 Architekturen .....	185
4.2 Kognitives System .....	199
4.3 Kognitives Robotersystem .....	204
Literatur .....	220
<b>5 Implementierung .....</b>	221
5.1 Hardware .....	221
5.2 Software .....	235
5.3 Brainware .....	271
Literatur .....	401

<b>6 Validierung .....</b>	405
6.1 Problem .....	405
6.2 Hardware .....	406
6.3 Software .....	424
6.4 Brainware .....	426
Literatur .....	462
<b>7 Ausblick als Motivation .....</b>	465
7.1 Implikationen einer artifiziellen Kognition .....	465
7.2 Implikationen einer kognitiven Robotik .....	473
7.3 Empfehlungen als Plädoyer .....	483
Literatur .....	494
<b>8 Musterlösungen .....</b>	497
8.1 Musterlösung: Vorgehensmodell .....	497
8.2 Musterlösung: Wissensakquisition .....	511
8.3 Musterlösung: Cognitive Robotic Plattform .....	520
8.4 Musterlösung: UML4Robotik .....	525
8.5 Musterlösung: Java4Robotic .....	535
Literatur .....	592
<b>Sachverzeichnis .....</b>	595

---

## 1.1 Mechanische Automaten

Im Laufe der Menschheitsgeschichte betete der Mensch seine Götter nicht nur an, sondern trat hier und da in Konkurrenz zu ihnen, wie beispielsweise dann, wenn er versuchte, sich die Fähigkeiten des Fliegens anzueignen, beim Streben nach ewiger Jugend, bei der Suche nach dem Lebenselexier und besonders bei der Schaffung künstlichen Lebens. Unterstützt durch Anstrengungen auf mythologischem, magisch-biologischem und technisch-mechanischem Gebiet zeigen sich die Ergebnisse in verschiedenen Erscheinungsformen:

- von der belebten Statue bis zur einfühlsamen Puppe,
- von der plastischen Wachsfigur bis zur mechanischen Marionette,
- vom Golem bis zum Homunkulus,
- vom Maschinen- und Uhrwerk bis zum Androiden,
- vom mechanischen Spielzeug bis hin zum intelligenten Automaten,
- um letztlich beim kognitiven Roboter zu landen.

Der erste sagenhafte Mechaniker und Konstrukteur von Maschinen und automatenhaften Gebilden war Daidalos. Am bekanntesten ist seine Flugmaschine, die ihn und seinen Sohn Ikarus vor der Ungnade des Königs Minos retten sollte. Ähnliches gilt für die Produkte des Schmiedes Hephaistos, dessen eisernem Riesen Talos Wächterdienste bei König Minos zugeschrieben werden. Diomedes soll Statuen geschaffen haben, die aus eigener Kraft schwammen. Der König Ptolemaios von Ägypten förderte im dritten Jahrhundert v. Chr. nicht nur die Automatenproduktion als Mäzen, sondern soll eigene Automaten gebaut haben. Man erzählt, dass bei einem Fest zu Ehren Alexanders des Großen eine Statue der Nymphe Nisa auf ihrem Wagen aufgestanden sei und Milch in eine goldene Schale gegossen habe. Die Erwähnung solcher belebter Statuen und mechanischer Kunstwerke zu Repräsentationszwecken findet sich in nahezu allen Kulturen. Gerade den tatsächlich oder scheinbar belebten Statuen kam als göttlichem Orakel eine wichtige Rolle zu. Lukian

berichtet so von einer Orakelstatue des Apollo. Sprechende Statuen erlaubten den Regierenden und Priestern, das gläubige Volk mit deren Prophezeiungen beliebig zu manipulieren, seine Gier nach sinnlichen Gottesbeweisen zu befriedigen. Dass derartige Leistungen durchaus nicht nur im Bereich der Mythologie existiert haben mussten, beweisen die grundlegenden technischen Entwicklungen griechischer Mechaniker, insbesondere derer, die zur Schule von Alexandria gehörten (Ktesibios, Heron von Alexandria und Philon von Byzanz). Als Antriebsprinzipien ihrer mechanischen Kunstwerke standen ihnen primär Vorrichtungen nach dem Sanduhrprinzip zur Verfügung, als Energiequellen Wasser, Wasserdampf, Quecksilber, als Energieübertragungsmittel Winde, Hebel, Flaschenzug oder Schrauben. Es fehlte noch das fortschrittliche Zahnrad. Dennoch konstruierte Ktesibios seine Wasserorgel, eine Wasserruhr und eine Luftpumpe. Heron hinterließ ein zum Teil verschollenes, reichhaltiges und breitgefächertes theoretisches Werk, das von der Geometrie bis zur Vermessungs- und Spiegellehre reicht. Philon, ein Schüler des Ktesibios, (3. Jahrhundert v. Chr.) soll darüberhinaus ebenfalls Wasseruhren, mechanische Geschütze und einzelne Automatenfiguren angefertigt haben.

Nach dem Niedergang der hellenischen Kultur verlagert sich das Zentrum des Automatenbaus nach Byzanz. Arabische Mechaniker schaffen prunkvolle Automaten, besonders zu Repräsentationszwecken, wie den berühmten Thron Salomons, der um 835 nach Christus von dem Astronomen, Architekten und Mathematiker Leo für Kaiser Theophilos Ikonomachos aus Gold und Silber gebaut worden sein soll. Er war verziert mit Greifen und Löwen, die den Kopf bewegen, sich erheben und sogar brüllen konnten. Hervorragendes Produkt der arabischen Automatenbaukunst ist die etwa 500 n. Chr. entstandene Uhr von Gaza, die mit unzähligen automatischen Figuren verziert war. Diese Verwendung der Automaten war die gebräuchlichste, da der Islam im Übrigen Nachahmungen der menschlichen Gestalt verbot. Ausnahmen bilden Trinkautomaten, die zur Belustigung des Publikums zuvor aufgenommene Getränke wieder ausschieden.

Die Weiterentwicklung der Uhrwerke, Maschinen und Automaten wurde entschieden durch die Erfindung einer Vorrichtung gefördert, die es erlaubte, den Ablauf der bisher sanduhrmäßig funktionierenden Bewegung zu verzögern und damit zu verlängern: die Hemmung. Der Uhr kam eine über ihren Gebrauchswert hinausgehende Faszination zu, die sie trotz der Ungenauigkeit der ersten Exemplare zur Metapher und zum Symbol von Ordnung und Harmonie werden ließ in all den Bereichen, in denen diese Werte gefragt waren. Die Uhrmacherkunst erreicht im 14. Jahrhundert eine ungeahnte Blüte. Ihre Produkte sind nicht mehr bloße Zeitmessgeräte, sondern Mittel zur Nachahmung der Planeten. Der Mensch versucht, „imitatio die“, ein maschinalisiertes Universum nachzubauen, noch ohne den Gedanken, mit Gott konkurrieren zu wollen. Die besten Beispiele dafür sind die Uhren des Abtes von St. Albán, Richard von Wallingford, und die des Giovanni de Dondi. Zusätzlich wird die Uhr zunehmend künstlerisch ausgestaltet mit Automatentableaus von überwiegend religiösem Gehalt. Beispielhaft dafür ist die Uhr des Straßburger Münsters von 1352 mit ihrem an die Verleugnung des Petrus erinnernden Hahns. Andere Automatentableaus spielen auf politische Gegebenheiten an.

Die Automaten entwickelten sich aber auch zum beliebten Spielzeug, das wegen der hohen Herstellungskosten allerdings den oberen Schichten vorbehalten blieb. Die kunstvollen und kostbaren Figuren waren kaum als Kinderspielzeug, sondern vielmehr nur als Repräsentationsobjekt geeignet. Die Skala reicht von der automatischen Weihnachtskrippe Hans Schlottheims (1589) bis zu dem berühmten Schiff Karls V., dem sein Hofmeister Juanelo Turriano den Rückzug ins Kloster San Yuste mit mechanischen Spielzeugen versüßt haben soll. Charakteristisch für die ständig zunehmenden Automatentableaus war die Mischung aus mechanisch bewegten und unbeweglichen bzw. marionettenhaften Figuren. Ein anderer Zweig der Automatenbaukunst entwickelte immer kunstvollere Wasserspiele für feudale Parkanlagen und Gärten. Diese Wasserkunstfiguren dürften wesentlich die Cartesiansche Konzeption vom Mechanismus des menschlichen Körpers beeinflusst haben.

Aber nicht nur Ingenieure und Techniker, sondern selbst Philosophen beschäftigen sich zunehmend mit den Werken Herons und der Alexandrinischen Schule. Die einzelnen Werke und ihr Inhalt sind weniger wichtig als die sich generell darin abzeichnende Tendenz zur Verbindung von Technik und Philosophie, die erstmals in den Theorien Descartes kulminierte.

Die praktische technische Entwicklung der Automaten zielt auf immer perfektere Nachahmung der Natur ab, sichtbar etwa in den Automaten Jacques de Vaucansons. 1738 stellt dieser Mechaniker seinen nahezu lebensgroßen, hölzernen Flötenspieler der Akademie vor. Er konnte zwölf Melodien auf seinem Instrument spielen und erzeugte dabei die Töne nicht mehr einfach durch ein Uhrwerk in seinem Innern, sondern natur- und kunstgerecht durch die entsprechenden Zungen- und Fingerbewegungen. Ein von verschiedenen Uhrwerken angetriebenes Blasebalgsystem setzte den dazu nötigen komplizierten Mechanismus in Gang. Dem Flötenspieler folgte bald der Joueur de Galoubet, der eine provenzalische Hirtenflöte blies und sich auf einem Tamburin begleitete. Man berichtet von ihm, dass er sogar besser und schneller gespielt haben soll, als ein Mensch. Vielleicht der erste, wenn auch noch harmlose Fall einer Maschine, die den Menschen übertrifft. Höhepunkt des Vaucansonschen Erfindungsreichstums ist aber sicherlich die mechanische Ente, die zum ersten Mal nicht nur äußerlich die Natur imitiert, sondern auch innerlich korrekt die biologischen Vorgänge nachzuahmen suchte. Es lässt sich nicht genau entscheiden, wie die Verdauungsvorgänge im Einzelnen abliefen, ob tatsächlich im Magen der Ente ein kleines Chemielaboratorium untergebracht war, in dem die Nahrung zerkleinert wurde. Sicher ist allein, dass die Ente ihr Futter auf natürlichem Wege aufnahm und auch wieder ausschied, die verabreichte Nahrung also den Körper durchlaufen hatte und dabei auf irgendeine Weise zerkleinert worden war. Die Idee eines Tierautomaten war nicht völlig neu, einmalig war jedoch die Konzeption der Vaucansonschen Ente als dreidimensionaler, korrekter anatomischer Atlas.

Tatsächliche Automaten wurden nur noch im Miniaturformat gebaut – als verspieltes Beiwerk für Taschenuhren oder als Tafelaufsätze, die auf Grund der hohen Fertigungskosten weiterhin wohlhabenden Bürgern vorbehalten blieben. Weitere Verbreitung erfuhren dagegen in zunehmendem Maße die immer mehr als Massenprodukte angefertigte, mechanische Spielzeuge.

Die vorzüglichsten Techniker wendeten sich deshalb den Nutzmaschinen zu, deren Entwicklung durch die Entdeckung neuer Antriebskräfte, wie Dampf und Elektrizität, noch beschleunigt wurde. Es erscheint fast als Ironie des Schicksals, dass ausgerechnet diese neuen, äußerlich so menschenähnlichen Maschinen dem Menschen bedrohlicher wurden, als die ihm nachgebildeten Androiden.

DESCARTES steht bei seinen Versuchen, das Universum, das Leben und den Menschen rational, mechanistisch-materialistisch zu erklären bzw. auch zu reproduzieren, ganz in der Nachfolge der atomistischen Lehren eines Epikur, Demokrit und Titus Lucretius Carus. Anders als diese begnügen sich Nikolaus Kopernikus (1473–1543), Johannes Kepler (1571–1630) und Galileo Galilei (1564–1642) nicht mehr mit spekulativen Konzeptionen des Universums, sondern suchen nach empirischen Beweisen. Ihre von mechanischen Gesetzen bestimmten Systeme bilden natürlich eine eminente Gefahr für die entsprechenden Lehren der Theologen, stellen diese durch die Reduktion alles Seienden auf Materie, beobachtbare Fakten und Gesetze alle Theorien hinsichtlich Unsterblichkeit und Geisthaftigkeit des Menschen doch in Frage. Gerade die negativen Erfahrungen Galileis machen Descartes vorsichtig mit eigenen radikalen Thesen. Er begegnet allen Vorwürfen dadurch, dass er als Objekt seiner Überlegungen nicht das reale Universum bzw. den konkreten Menschen nimmt, sondern vielmehr deren künstlich geschaffene *Modelle*. Diese entsprechen völlig den empirischen Wissenschaften, den Erkenntnissen der Mathematik, Physik und Mechanik. Das einzig Sichere scheint die eigene Denkfähigkeit zu sein.

Am Anfang des 20. Jahrhunderts begannen die Bemühungen, Automaten zu bauen, die nützliche Aufgaben übernehmen konnten. Dabei konnte man in relativ kurzer Zeit wichtige Ergebnisse erzielen<sup>1</sup>:

- 1946 entwickelte der Amerikaner G. C. Devol ein Steuergerät, das elektrische Signale magnetisch aufzeichnen konnte. Diese konnten wieder abgespielt werden, um eine mechanische Maschine zu steuern.
- 1951 begann die Entwicklung ferngesteuerter Handhabungsgeräte (Teleoperatoren) zur Handhabung radioaktiver Materialien.
- 1952 wurde am MIT der Prototyp einer numerisch gesteuerten Werkzeugmaschine entwickelt. Die zugehörige Programmiersprache APT wurde 1961 veröffentlicht.
- 1954 reicht der Brite C.W. Kenward ein Patent einer Roboterentwicklung ein. Zur gleichen Zeit arbeitet der Amerikaner George C. Devol an dem „programmierten Transport von Gegenständen“. Er erhält 1961 dafür ein US-Patent.
- 1959 wird von der Firma Planet Corp. der erste kommerzielle Roboter vorgestellt. Dieser wurde mechanisch durch Kurvenscheiben und Begrenzungsschalter gesteuert.
- 1960 wurde der erste Industrieroboter („Unimate“) vorgestellt. Er basierte auf den Arbeiten von Devol. Der Roboter war hydraulisch angetrieben. Er wurde durch einen Computer unter Verwendung der Prinzipien numerisch gesteuerter Werkzeugmaschinen kontrolliert.

---

<sup>1</sup> Vgl. Groover et al. 1987, S. 54 ff.

- 1961 wurde bei Ford ein Roboter des Typs Unimation installiert.
- 1968 wurde der mobile Roboter „Shakey“ am Stanford Research Institute (SRI) entwickelt. Er war mit einer Vielzahl von Sensoren, u. a. Kamera, Tastsensor, ausgestattet.<sup>2</sup>
- 1973 wurde die erste Programmiersprache (WAVE) für Roboter am SRI entwickelt. Im Jahr 1974 folgte die Sprache AL. Ideen dieser Sprachen wurden später in der Programmiersprache VAL von Unimation verwendet. Etwa zur gleichen Zeit entstanden auch vollständig elektrisch angetriebene Roboter.
- 1978 wird der Roboter PUMA (programmable universal machine for assembly, Programmierbare Universalmaschine für Montage-Anwendungen) von Unimation vorgestellt. Er ist elektrisch angetrieben und basiert auf Entwürfen von General Motors. Der Robotertyp PUMA wird nachfolgend immer bei konkreten Anwendungsbeispielen zugrunde gelegt.

So findet man beispielsweise recht früh in den Richtlinien für die Montage- und Handhabungstechnik des Vereins Deutscher Ingenieure (VDI), Düsseldorf, nachstehende Begriffsbestimmung für einen Industrieroboter:

Industrieroboter sind universell einsetzbare Bewegungsautomaten, mit mehreren Achsen ausgestattet, deren Bewegung hinsichtlich Bewegungsfolge und Wege bzw. Winkeln frei, d. h. ohne mechanischen Eingriff, programmierbar und gegebenenfalls sensorgeführt sind. Sie sind mit Greifern, Werkzeugen oder anderen Fertigungsmitteln ausrüstbar und können Handhabungs- und Fertigungsaufgaben ausführen.

Die Automaten der ersten Generation hatten mit ihren metallenen Armen und Beinen noch eine sehr menschenähnliche Statur. Allerdings hatten diese Apparaturen wenig mit den Robotern im heutigen Sinne zu tun. Jede einzelne Aktion musste vorher vom Menschen durch einen Befehl abgerufen werden. Daher scheint es legitim, den Begriff „Automat“ anstelle von „Roboter“ zu verwenden. Auf jeden Fall waren die Apparate nicht in der Lage, diverse Tätigkeiten selbstständig durchzuführen. So stellte zum Beispiel der Amerikaner Wensley 1928 der Öffentlichkeit seinen Maschinenmenschen „televox“ vor. Das Gerät konnte auf ein akustisches Kommando hin den Elektroherd einschalten oder einen Lampenschalter betätigen. Keine große Sache, „televox“ leistet damit auch nicht viel mehr als beispielsweise ein Fernsehgerät: Auf Kommando – in der Regel per Infrarotsignal – wird auf ein anderes Programm umgeschaltet, erhöht sich die Lautstärke etc.

Als Nächstes baute man Maschinen, die man aus sicherer Entfernung leicht steuern konnte. Automaten dieser Kategorie sind prädestiniert für lebensgefährliche Arbeiten, z. B. in den „heißen“ Zonen von Kernkraftwerken (z. B. setzte die UdSSR bei dem Reaktorunfall 1986 in Tschernobyl solche Geräte ein) oder bei der Produktion von Kernbrennstoffen, bei der Reparatur von Unterwasserpipelines in großen Meerestiefen, bei Giftunfällen und Umweltkatastrophen, bei Waldbränden, bei der Instandhaltung von Satelliten im Welt Raum, als Besatzung von Raumschiffen oder als Auskundschafter auf fremden Planeten.

---

<sup>2</sup> Vgl. Jones und Flynn 1996.

Gerade in diesem zuletzt angesprochenen Einsatzgebiet waren Automaten ausgesprochen nützlich. Zu Beginn der 60er Jahre setzte man zur Erkundung der Mondoberfläche sogenannte „rangers“ ein, „Kamikaze“-Flugkörper, die vor dem Aufprall auf dem Erdtrabanten Fotos aufnahmen, die per Funk an die Erde übermittelt wurden. Die gewonnenen Aufnahmen zeigten zwar bis dahin unbekannte Details, genaue Untersuchungen der Mondoberfläche ließen sich naturgemäß nicht durchführen. Also ging man an die Entwicklung kleiner Automatenfahrzeuge, die auf dem Mond herumkutschieren sollten, um den Boden zu untersuchen und Ergebnisse auf die Reise zu unserem Planeten zu schicken. Diese Automaten wurden von der Erde aus „on-line“ ferngesteuert, d. h. der Steuermann auf der Erde sah über Fernsehkameras, wie das Gelände vor Ort sich gestaltete und konnte dadurch entsprechende Aktionen einleiten. Von Autonomie war man also noch weit entfernt.

Als die ersten Computer verfügbar waren, lag natürlich der Gedanke nahe, die Automaten mit einem Rechner zu kombinieren. Von nun an waren die Automaten in der Lage, auch komplizierte Arbeitsvorgänge durchzuführen, für die zahlreiche Bewegungen nötig sind. Mit anderen Worten, man konnte von jetzt an dem Automaten Anweisungen, wie beispielsweise „Bohre ein Loch mit einem Durchmesser von 15 mm“ oder „Lege die gefertigten Teile in den Transportbehälter“, geben. Der Automat war somit in der Lage, alle notwendigen Berechnungen durchzuführen bzw. das entsprechende Programm ablaufen zu lassen. Um 1970 wurden dann auch die ersten computerunterstützten Industrieautomaten entwickelt.<sup>3</sup> Der Übergang von den Vorläufern der Automaten hin zu echten Robotern hatte begonnen.<sup>4</sup>

---

## 1.2 Robotersysteme

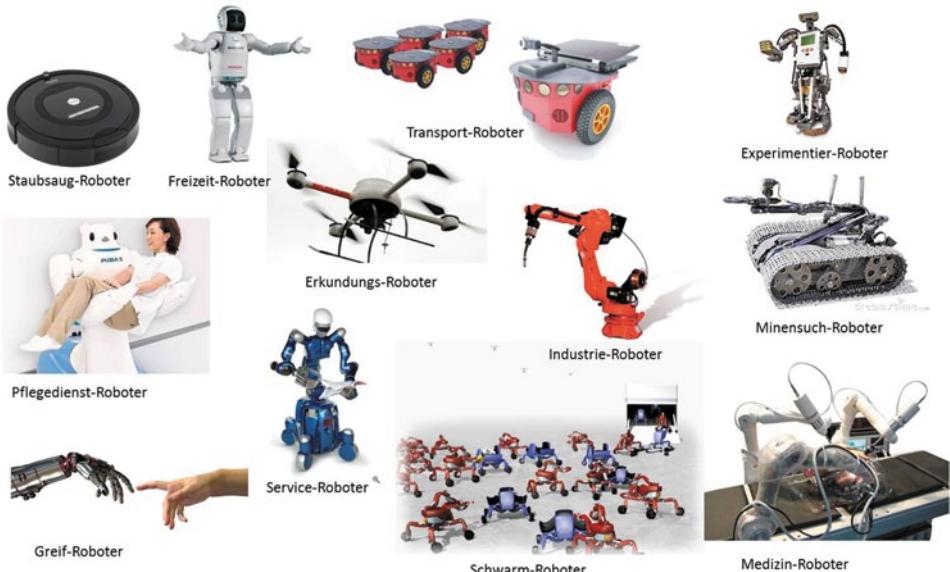
Was immer noch mit gemischten Gefühlen empfunden wird, lässt sich nicht leugnen: Roboter integrieren sich langsam aber stetig in die Lebenswelt des Menschen. In der industriellen Fertigung längst unentbehrlich, erledigen die modernen Robotersysteme zunehmend auch Dienstleistungsaufgaben. So entschärfen sie Bomben, überwachen Fabrikgelände und saugen die Wohnung. Als hochpräzise Assistenten unterstützen sie Chirurgen bei heiklen und difizilen Operationen. Und anstelle von Pflegern sollen autonome Roboter in Zukunft sogar einen Großteil der Altenbetreuung übernehmen. Selbst beim Sport entwickeln die Maschinen Ehrgeiz: Verläuft alles nach Plan ihrer Konstrukteure, sollen Fußballroboter bei der Weltmeisterschaft 2050 nicht nur gegen menschliche Gegner antreten, sondern auch gewinnen (Abb. 1.1)!<sup>5</sup>

---

<sup>3</sup> Siehe auch Albert und Ottmann 1983.

<sup>4</sup> Vgl. Dillmann und Huck 1991.

<sup>5</sup> Auch der Autor dieses Buches arbeitete mit seinem Team in den letzten Jahren an dieser visionären Herausforderung.



**Abb. 1.1** Roboter-Impressionen

### 1.2.1 Serviceroboter

Wie das Eingangskapitel dieses Buches verdeutlicht hat, zogen menschenähnliche Maschinen die Menschen immer wieder in ihren Bann. In den 70er Jahren schrieb George Lucas Kinogeschichte, als er den aufrecht gehenden Roboter C3PO auf Zelluloid bannte und als Weggefährten des Helden Luke Skywalker in den „Krieg der Sterne“ ziehen ließ. Dass diese Geschichte noch nicht am Ende ist, zeigt der riesige Erfolg des Roboter-Epos „I Robot“. Flankierend zu den Filmgenres treibt auch die Unterhaltungsindustrie den lang geträumten Traum voran. Die Computerhunde Aibo oder Qrio, als erste dynamische zweibeinige Laufroboter aus dem Hause Sony, dienen als publicityträchtige Vorboten eines vielleicht neuen Zeitalters. Man schätzt, dass in ca. 30 Jahren mehr persönliche Roboter produziert werden als persönliche Computer. Hintergrund dieser Prophezeiungen ist der ermittelte Bedarf an Servicerobotern, die auf Zuruf die alltägliche Arbeit „vollautomatisiert“ erledigen: Staub saugen, Rasen mähen, Fenster putzen, einkaufen und den Müll wegbringen, Betten machen und den quengelnden Nachwuchs betreuen.<sup>6</sup> Allerdings darf man bei all der berechtigten Euphorie die derzeitige Realität nicht ganz außer Acht lassen. Vieles, was für den Menschen selbstverständlich ist, erweist sich für Roboter auch heute noch als große Herausforderung. Beispielsweise können sie nach wie vor nur bedingt Auto fahren. Auch die Anatomie des Menschen entpuppt sich immer wieder als schwer kopierbares Wunderwerk. Zum Beispiel weist die menschliche Hand bei einem Durchschnittsgewicht

<sup>6</sup> Siehe auch Schraft und Volz 1996.

von 600 Gramm 23 verschiedene Freiheitsgrade auf, die sie in die Lage versetzt, unterschiedlichste Aufgaben, wie Schrauben drehen, Bälle fangen oder Menschen begrüßen mit einer stufenlos verstellbaren Feinfühligkeit zu erledigen. Dazu verfügt sie über rund 17.000 Sensoren, die Druck, Temperatur, Wärmeleitfähigkeit und Berührung in direkter Muskelnähe messen. Solch ein perfektes Regelwerk bringt selbst modernste Technik derzeit „ins Schwitzen“:

Es sind vor allem japanische Forscher, die sich bemühen, die „überlebenswichtige“ Mechanik, Elektronik, Sensorik und Steuerungstechnik eines zweibeinigen Laufroboters zu integrieren und zu miniaturisieren. So wiegt das Modell *Asimo* (Advanced Step in Innovative Mobility), das Honda im Jahre 2001 vorstellt, bei einer Größe von 1,20 m nur noch 43 kg. Ein wahres Federgewicht im Vergleich zu seinen Vorgängern, wie dem Pl, den die Ingenieure des Konzerns Mitte der 90er Jahre entwickelt hatten. Der Pl brachte mit seinem Gardemaß von 1,92 m satte 175 kg auf die Waage. Zu viel, als dass Batterien die Energie hätten liefern können, um den Koloss auf Trab zu bringen. Wie einen Schwanz zog der Pl daher sein Elektrokabel hinter sich her. Der Nachfolger P2 verfügte zwar bereits über eine eingebaute Energiequelle; bei einem Gewicht von 210 kg konnte ihn seine Batterie aber höchstens 15 min lang ausreichend mit Energie versorgen.

Eine der großen Herausforderungen stellt nach wie vor die Energieversorgung der Robotersysteme dar. Ein Serviceroboter, der ständig an der Ladestation pausieren muss, hilft dem Menschen schließlich nicht weiter. Anders als die stationären Automaten, die in der Industrie selbstständig ihre Arbeit rund um die Uhr verrichten, müssen autonome Assistenten auch dann zuverlässig arbeiten, wenn sie nicht an der Steckdose hängen. Beliebig viele Akkus zu montieren, ist dabei keine Alternative, denn Akkus sind schwer und der Stauraum knapp. Und nur in wenigen Fällen lösen Solarpanels alle Energieprobleme.

Bei Drucklegung der ersten Auflage dieses Buches hatten sich ca. 200 Firmen, die sich zum Ziel gesetzt haben, marktfähige Serviceroboter zu entwickeln, bei der europäischen Wirtschaftskommission der Vereinten Nationen (UNECE) registriert. Fast die Hälfte der aufgeführten Unternehmen stammt aus den USA. Japan, Deutschland und Großbritannien beheimaten jeweils rund acht Prozent. Die Bandbreite ihrer Produktideen reicht von Reinigungs-, Inspektions-, Unterwasser- und Abrissrobotern für den professionellen Einsatz bis hin zu Unterhaltungs- und Pflegerobotern für den häuslichen Gebrauch. Mehr als 2,1 Mio. dieser Maschinen sollen laut UNECE bis 2006 verkauft worden sein. Insofern hat die Zukunft auch laut diesen Zahlen bereits begonnen.

Die artifiziellen Robotersysteme erobern also die natürliche Lebenswelt des Menschen. Allerdings verdient nicht alles, was diese Statistik aufzählt, den Titel eines Roboters im Sinne dieses Buches. Denn Roboter, die Staub saugen oder Wache schieben, spulen eigentlich nur stumpf Programme ab, die ihnen von Entwicklern eingegeben werden, und befahren Routen, die sie nicht selbstständig wählen. Allerdings sind sie frei programmierbar und in mindestens drei Achsen frei beweglich. Intelligent im Sinne eines systemischen Intelligenzquotienten ( $IQ_s$ ) aber sind die Geräte nach den Maßstäben dieses Buches noch lange nicht.

Die Interaktion zwischen dem Robotersystem und dem Menschen scheint das Problem zu sein. Dazu muss das Robotersystem Daten nicht nur messen und analysieren, sondern

auch bewerten können. Denn die intelligente, sprich intuitive Kommunikation mit dem Menschen erfolgt über unscharfe Botschaften wie Gestik, Mimik und Sprache. Schon die einfache Aufforderung, seinem Besitzer ein Buch aus dem Regal zu bringen, stellt ein Robotersystem vor riesige Probleme. Zunächst muss er den Menschen identifizieren und lokalisieren, anschließend das Regal finden, auf das der Mensch verweist, sich in der Wohnung orientieren, das Buch erkennen und dieses aus dem Regal ziehen und gegebenenfalls zurücktransportieren. Sollte das Buch nicht aufzufinden sein, darf der Roboter auf keinen Fall stillschweigend den Dienst quittieren, sondern sollte beispielsweise eine verständliche Meldung liefern. Aber auch hier muss der Realität allerdings Rechnung getragen werden. Noch fehlt es an einer leistungsfähigen und robusten Spracheingabe, die einen genuschelten Befehl in Digitalcode übersetzen kann. Stattdessen müssen sich die Anwender eines Robotersystems oftmals mit einem Notebook behelfen, das vorprogrammierte Befehle per Funk an das Robotersystem übermittelt. Schon kleine Lichtreflexe oder ein überlastetes Funknetz führen dazu, dass der Roboter die Orientierung im Raum verliert. Nur das Drücken des Not-Aus-Schalters und ein Neustart bringen ihn zurück in die Wirklichkeit. Solche Robotersysteme sind also noch weit davon entfernt, ihr Experimentierumfeld bzw. Trainingslabor verlassen zu können.

### 1.2.2 Industrieroboter

Während Roboter für den Haushalt sich derzeit gerade etablieren, haben sich die mechanischen Helfer in der Industrie längst durchgesetzt. Wenngleich man bei den sozialen Servicerobotern durch deren menschenähnliche Gestaltung versucht, deren Akzeptanz zu erhöhen, haben die realen Produkte für die Industrie äußerlich wenig gemein mit den ersonnenen humanoiden Sklaven. Statt zwei haben die meisten von ihnen nur einen Arm, auf Beine müssen sie, ebenso wie auf einen Kopf, ganz verzichten.<sup>7</sup> Dafür gibt es Modelle, die selbst tonnenschwere Lasten mühelos heben können. Und sie tun genau das, wofür Capek sie eigentlich vorgesehen hatte: pausenlos unangenehme und gefährliche Arbeiten versehen, bei denen es auf Intellekt nicht ankommt. In der Autoindustrie, immer noch bei weitem der größte Roboteranwender, heißt das vor allem schweißen und schwere Sachen heben bzw. tragen. Überhaupt hat sich am Grundkonzept des Industrieroboters relativ wenig geändert, seit General Motors im Jahr 1961 das erste Exemplar, den von George Devol und Joseph Engelberger entwickelten „Unimate“, in Betrieb nahm. Dennoch steckt in diesen Industrierobotern eine Reihe von Technologien: Antrieb, Steuerung, Sensoren, Mechanik und Informationstechnologie.<sup>8</sup> All diese Komponenten werden stetig weiterentwickelt und verbessert. Dieser stetige Fortschritt zeigt Wirkung: So kann ein heutiger Industrieroboter doppelt so viel wie früher, bei gleichzeitiger Minimierung seiner Bauteile

---

<sup>7</sup> Vgl. Warnecke und Schraft 1990.

<sup>8</sup> Vgl. Nof 1999.

um nahezu die Hälfte.<sup>9</sup> Nach Berechnungen der United Nations Economic Commission for Europe ist der qualitätsbereinigte Preis von 1990 bis 2002 um 81,5 % gefallen. Feinere Steuerung, Kameras und Sensorik erschließen den einarmigen Robotersystemen mittlerweile Einsatzgebiete, für die sie anfangs schlicht zu grob strukturiert und damit zu tollpatschig waren. Hierzu zählen:

- montieren,
- schweißen, kleben, nielen, schrauben, schneiden, schleifen,
- beschichten, spitzen, entgraten, einfärben,
- beschicken, pressen, verketten,
- verpacken und
- befördern.

So können mit einem Erkennungssystem aufgerüstete Robotersysteme heute bei VW das Dach auf voller Länge und höchst präzise mit dem Rahmen so verschweißen, wie es ein menschlicher Schweißer kaum könnte.

Auch wenn man theoretisch heutzutage jedes Fertigungsproblem maschinell lösen kann, klafft auch hier eine Kluft zwischen Theorie und Praxis. So gibt es in der Praxis, wie beispielsweise in der Autoproduktion, weite Bereiche, wo Roboter nach wie vor nicht zum Einsatz kommen. Bei VW-Nutzfahrzeugen etwa dominieren die Robotersysteme die gesamte Halle für die Herstellung der Karosserie; Menschen sind nur vereinzelt zu sehen, wenn sie mit Gabelstaplern neue Teile herankarren oder Kontrollen vornehmen. In der Endmontage zeigt sich allerdings das umgekehrte Bild: Es wimmelt von Menschen, nur an wenigen Stationen unterstützt ein Robotersystem mit seinen Greifern den Produktionsprozess.

Es ist davon auszugehen, dass die inzwischen gereifte Roboterbranche auch in den nächsten Jahren mit ansehnlichem Wachstum aufwarten wird. Zusätzliche Reife werden die Roboter erlangen, indem, statt umständlich und in teils proprietären Sprachen, die zukünftigen Systeme bequem über eine grafische Benutzeroberfläche oder eine Sprachsteuerung ihre Aufgaben erklärt bekommen.

### 1.2.3 Medizinische Roboter

Wenn auch didaktisch sicherlich nicht unbedingt empfehlenswert, soll dieser Abschnitt gleich mit einem Rückschlag beginnen. Weil er gegen das oberste Gesetz der Robotik verstieß, wurde Robodoc am 16. April 2004 abgeschaltet. Ganz gemäß des Postulats des Science Fiction-Autors Isaac Asimov aus seinem Buch „I, Robot“, dass ein Roboter einem Menschen keinen Schaden zufügen darf. Robodoc ist ein Chirurg aus Stahl. Er hat weit über 10.000 Menschen operiert und manche allerdings dabei auch verletzt. Bei Patienten

---

<sup>9</sup> Vgl. Volmer 1992, S. 40 ff.

mit defekten Hüftgelenken fräst beispielsweise das Robotersystem nach einem vor der Operation festgelegten Plan die Knochen so exakt aus, dass sich die Ersatzteile aus Titan und Keramik sogar ohne Zement wackelfrei ins Skelett einfügen. Führen menschliche Operateure den Eingriff durch, müssen sie manchmal die Fugen mit Füllstoff kitten. An die Präzision der Maschine gereicht keine noch so geschickte menschliche Chirurgenhand heran. Erklärtes Ziel dieses vom Roboter präparierten Gelenkersatzes war es, dass dieses länger halten sollte und Patienten nach dem Eingriff wieder schneller laufen können sollten. Über 60 Robodocs operierten einmal in Deutschland. Tausende Patienten haben die Vorzüge des Robot-Operateurs schätzen gelernt. Einige hundert jedoch mussten erfahren, dass auch eine Maschine fehlbar ist. Ihnen durchtrennte der Automat Nerven und zerstörte Muskeln. Schmerzen und ein hinkender Gang waren die Folgen.

Die Ansätze und Ziele des apparativen Fortschritts klingen überzeugend, nicht nur im Fall Robodoc: Roboter arbeiten frei von emotionalen Einflüssen, sie kennen keine Tagesform, sondern schaffen flink und unermüdlich, stets reproduzierbar auf den hundertsten Teil eines Millimeters genau, egal ob sie Hüftknochen fräsen, Schrauben im Schädel verankern oder bei einer Gehirnoperation assistieren. Doch Präzision allein heilt leider nicht immer alle Patienten.

Robotersysteme, gleich welcher Anwendungskategorie, sind teuer und kompliziert. Jeder Operation geht eine millimetergenaue Planung voraus, kein OP-Roboter agiert autonom. Auf sich plötzlich verändernde Gegebenheiten, wenn sich ihnen etwa ein Muskel in den Weg schiebt, können die Automaten nicht reagieren, sie setzen ihr befohlenes Programm stur fort. Viele Prototypen wurden nach kurzer Zeit wieder demontiert, weil der Aufwand, den sie verursachten, unverhältnismäßig hoch war gegenüber ihrem Nutzen. Insofern hat der Einsatz solcher Robotersysteme im Operationssaal zunächst einen herben Rückschlag erlitten.

Insofern reduziert sich der derzeitige Einsatz im Operationssaal eher in Form eines Assistenten. Die Maschine hält dabei Skalpell und Nadel und führt als eine Art verlängerter Arm sklavisch aus, was ihr der menschliche Operateur per Joystick-Steuerung befiehlt. Solche Systeme – sie haben Namen wie Zeus, Aesop oder DaVinci – werden bereits erfolgreich bei Herzoperationen eingesetzt. Sie setzen die Bewegungen wie ein Getriebe um und filtern dabei jedes Zittern des Operateurs heraus, so dass er auch an kleinsten Gefäßen oder Hirnstrukturen arbeiten kann, ohne sie zu zerstören. Mit ihnen kann der Chirurg ein Herz durch drei kleine Löcher hindurch behandeln und muss nicht mehr den ganzen Brustkasten öffnen (invasive Operationstechnik).

Ein Weiteres kommt hinzu, indem – dank der elektromechanischen Verknüpfung – Patient und Arzt nicht mehr unbedingt an einem Ort zusammenkommen müssen. An solcher Technologie ist natürlich auch das Militär interessiert. Im Kriegsfall steht die Manipulator-Hälfte von solchen verteilten Robotersystemen oder einem seiner Kollegen im umkämpften Feldlazarett, der Steuerteil bleibt samt Spezialisten an einem sicheren Ort.

Die Präzision der Maschinen mit der Reaktionsfähigkeit und Intuition des Menschen zu verknüpfen, ist momentan das Ziel vieler Forschungen im Bereich der medizinischen Robotik. Dabei sollen die Vorteile beider Welten vereint werden. Dazu muss man die

Verhältnisse erst mal umkehren – die Maschine überwacht dann die Arbeit des Menschen. Weicht die von menschlicher Hand geführte Knochenfräse zum Beispiel vom geplanten Weg ab, stoppt ein Computer blitzschnell das Werkzeug, bevor es etwas zerstört, was nicht zerstört werden soll. Ist es wieder an der richtigen Position, regelt das System die Drehzahl erneut hoch. Als Navigationshilfe kann der Computer Bilder aus Kernspin- oder Computertomographen samt exakter Operationsplanung auf eine Spezialbrille oder einen Bildschirm projizieren und den Menschen durch Knochen und Gewebe lotsen.

Nicht nur dort, wo es auf Millimeterarbeit ankommt, sind Maschinen in der Medizin gefragt. Roboter taugen vor allem für stupide, immer gleiche Bewegungsabläufe, beispielsweise in der Rehabilitation nach Schlaganfällen. Jeder dritte Schlaganfallpatient behält Lähmungen zurück. Dann müssen die Betroffenen mühsam lernen, wieder ihre Arme oder Beine zu bewegen. In der Regel helfen Physiotherapeuten beim Üben dieser Bewegungsabläufe. Ein Prozess, der sich täglich und oft über Monate hinweg zieht. Je länger, intensiver und also auch teurer die Therapie ist, umso besser sind die Genesungschancen. Den Job der Krankengymnasten können jedoch in vielen Fällen einfühlsame Robotersysteme übernehmen. Teilweise erzielen die Maschinen sogar bessere Ergebnisse als menschliche Physiotherapeuten, denn die Automaten ermüden auch nach 1000 Wiederholungen nicht und führen die Bewegung des Arms oder des Beins mit der immer gleichen Exaktheit durch oder variieren den Bewegungsablauf nach genau definierten Mustervorgaben.

### 1.2.4 Freizeitroboter

Die Roboter sind in der alltäglichen Lebenswelt angekommen, sie dringen in die Häuser ein und begegnen den Menschen dort als Spielzeuge oder als nützliche Helfer. Insofern nehmen die Robotersysteme einen ähnlichen Lebenszyklus, wie es die Computer seit den 80er Jahren vormachen. Auch damals begann der Siegeszug der Computersysteme eher verhalten und vorsichtig, um dann in eine Welle der sukzessiven und unaufhaltbaren Verbreitung umzuschlagen. Insofern kann man prophezeien, dass die Robotersysteme in 10 Jahren überall sein werden, so wie heute E-Mail und das Web. Angetrieben von einem viele Milliarden großen Markt, werden sie sich immer weitere Bereiche erschließen und sich in den Alltag des Menschen nahtlos und harmonisch integrieren. Aber auch bereits heute kann ein Roboter in Form eines Staubsaugers Zeit sparen oder als Experimentierbausatz die Zeit vertreiben helfen, was die folgende exemplarische Auswahl verdeutlicht:<sup>10</sup>

- *Staubsauger*: Obwohl ihre Saugleistung begrenzt ist, gibt es inzwischen mehrere Staubsaugroboter auf dem Markt.
- *Rasenmäher*: Technisch ausgereift präsentieren sich Roboter, die den Rasen mähen. Da sie weniger Strom brauchen, kennen sie keine Leistungsprobleme.

---

<sup>10</sup> Vgl. Naval 1989.

- *Pool-Reiniger*: Im Swimmingpool haben sich Roboter bereits etabliert. Da hier keine Möbel herumstehen, können sich die Automaten leicht orientieren, und können problemlos Leitungen hinter sich her ziehen. Daher haben sie keine Leistungsprobleme wie die Staubsaugroboter, die auf Akkus angewiesen sind.
- *Wächter*: Sie überwachen Innenräume per Video und Infrarot.

Früher waren *Spielzeugroboter* einfache Blechmännchen, die piepen, blinken und fahren konnten. Heute können Spielzeugroboter Fußball spielen, ihr Herrchen morgens wecken und Menschen per Kamera erkennen.<sup>11</sup> Wer spielerisch mit Robotern experimentieren will, steigt am besten mit einem *Bausatz* ein. Die meisten Bausätze enthalten Motoren, Räder, Sensoren und einen Microcontroller, mit dem das ganze gesteuert wird. Programmiert wird der Roboter über eine Schnittstelle vom PC aus.

Diese Beispiele zeigen, dass sich, entgegen der Ansichten einiger Kritiker, die Robotik würde sich im Kreise drehen, hier und da sicherlich noch einige Kehrtwendungen bzw. Drehbewegungen stattfinden werden, aber immer vermehrt einige dieser Driftbewegungen aus dem Labor heraus in die Praxis führen, in den Alltag und somit in die unmittelbare Nähe des Menschen.

### 1.2.5 Humanoide Roboter

Der Bau humanoider, d. h. menschenähnlicher Roboter, ist durch mehrere Aspekte motiviert. Einerseits ist sicherlich der Spieltrieb der Robotiker oder die Sensationslust des Publikums zu nennen. Allerdings sprechen auch wissenschaftliche Gründe für den Bau menschenähnlicher Roboter, deshalb, weil es letztlich die menschliche Intelligenz ist, die man zu verstehen versucht.<sup>12</sup> Da die Begriffe des Alltags darauf beruhen, dass der Mensch Erfahrungen mit der Welt und seinem Körper macht, wird nur ein Roboter mit einem menschenähnlichen Körper auch menschenähnliche Begriffe und Denkweisen entwickeln.<sup>13</sup>

Weiterhin gibt es ganz pragmatische Gründe für den Bau menschenähnlicher Roboter, vor allem im Bereich der Serviceroboter: Mit etwas, das aussieht wie ein Mensch, geht man um, wie mit einem Menschen.<sup>14</sup> Ein humanoider Roboter, dem man seine Wünsche mitteilen kann, wie einem menschlichen Gegenüber, braucht keine Gebrauchsanweisung. Zudem sind ihre Körper gut abgestimmt auf eine Umgebung, in der auch der Mensch sich bewegt, und ihre Bewegungsmuster sind für Menschen leicht abzuschätzen. All diese Gründe bewogen beispielsweise den Honda-Konzern dazu, seinem Roboter ASIMO menschenähnliche Gestalt zu geben. Er sieht aus wie ein Astronaut im Raumanzug, hat einen stabilen menschlichen Gang und eine handliche Größe von 1,20 m.

---

<sup>11</sup> Vgl. Kitano 1998, S. 25 ff.

<sup>12</sup> Vgl. Sesink 1993.

<sup>13</sup> Vgl. Kreutz und Bacher 2004.

<sup>14</sup> Vgl. Schraft und Volz 1996.

### 1.2.6 Bioroboter

Autonome Roboter, die dazu benutzt werden, das Verhalten von Tieren zu simulieren, werden bisweilen als *Animaten* (artificial animals) bezeichnet. Dieser Begriff verbreitete sich nach der 1990 in Paris veranstalteten Tagung „Simulation of Adaptive Behavior: from Animals to Animats“. Der ideale Animat wäre ein Roboter, der, wie es Tiere können, in einer komplexen Umwelt überleben würde. Die Nutzung von Animateen hat das Interesse an biologischen Vorbildern gestärkt und die Biologie in den Kreis der zur Robotik beitragenden Disziplinen geholt. Umgekehrt suchen Biologen in Kooperation mit Informatikern nach Modellen für komplexes adaptives Verhalten bei Tieren, etwa der Staatenbildung von Ameisen, oder den Orientierungsleistungen der Tiere. Es ist sogar schon von künstlicher Verhaltensforschung (artificial ethology) die Rede.

Biologen schätzen wie Robotiker den Zwang zu klarer Explikation ihrer Vorstellungen und Begriffe, den Computersimulation und Roboterbau mit sich bringen. Und weil man genau weiß, was man einem Roboter einprogrammiert hat und was nicht, lässt sich mit seiner Hilfe vielleicht auch klären, was ein Tier im Kopf haben muss, um ein bestimmtes Verhalten zu zeigen. Wenn ein Roboter nur ein Modul zur Vermeidung von Kollisionen, eines zur Wahrung eines bestimmten Abstands zu einer Wand und eines für das Erkennen von Nahrung braucht, um Futter in einem Labyrinth zu finden, wenn ein Roboter also keine mentale Karte seiner Umgebung benötigt, um sich zu orientieren, dann ist es bei einem Tier vielleicht nicht viel anders. Zudem wird diskutiert, inwieweit Tier-Roboter anstelle von Tieren in Versuchen verwendet werden können, die sonst entweder wegen moralischer Bedenken nicht möglich wären oder weil die betreffenden Tiere ausgestorben sind.

Einen noch einmal anderen Weg geht die hybride Biorobotik. Sie generiert natürlich-artifizielle Mischwesen. Dazu lässt man Nervenzellen auf Siliziumchips wachsen, oder setzt ganze Tierteile auf Roboter auf. Nervenzellen etwa können in Robotern zur Signalübertragung verwendet werden.

Schließlich liefert die Biologie und dort speziell die *Bionik* einen großen Pool von Innovationen und Inspirationen für Probleme der Robotik. Von besonderem Interesse sind Sensoren und Kontrollsysteme sowie der Aufbau von Tierkörpern. So sind etwa die meisten Sehsysteme von Robotern immer noch denen von Insekten nachempfunden.

---

### 1.3 Klassische Robotik

Der Begriff *Roboter* hat seinen eigentlichen Ursprung in Science-Fiction-Erzählungen. Er taucht erstmals 1921 in dem Theaterstück „Rossum's Universal Robots“ von Karel Capek auf. In dieser Geschichte entwickeln der Wissenschaftler Rossum und sein Sohn eine chemische Substanz, die sie zur Herstellung von Robotern verwenden. Der Plan war, dass die Roboter den Menschen gehorsam dienen und alle schwere Arbeit verrichten sollten. Im Laufe der Zeit entwickelte Rossum den „perfekten“ Roboter. Am Ende fügten sich die perfekten Roboter jedoch nicht mehr in ihre dienende Rolle, sondern rebellierten und

töteten alles menschliche Leben. In den slawischen Sprachen (beispielsweise tschechisch oder polnisch) hat „*robo*“ die Bedeutung „arbeiten“.

Auch Isaak Asimov, ein amerikanischer Biochemiker, Sachbuch und Science-fiction-Autor, schrieb mehrere Erzählungen über Roboter. Er definierte in der Kurzgeschichte „*Runabout*“ den Begriff *Robotik* als Studium von Robotern. Er formulierte sogar Gesetze für Roboter:

- *Nulltes Gesetz*: Ein Roboter darf der Menschheit keinen Schaden zufügen oder durch Untätigkeit zulassen, dass der Menschheit Schaden zugefügt wird.
- *Erstes Gesetz*: Ein Roboter darf einem menschlichen Wesen keinen Schaden zufügen oder durch Untätigkeit zulassen, dass einem menschlichen Wesen Schaden zugefügt wird, es sei denn, dies würde das nullte Gesetz der Robotik verletzen.
- *Zweites Gesetz*: Ein Roboter muss dem ihm von einem menschlichen Wesen gegebenen Befehl gehorchen, es sei denn, dies würde das nullte oder das erste Gesetz der Robotik verletzen.
- *Drittes Gesetz*: Ein Roboter muss seine Existenz beschützen, es sei denn, dies würde das nullte, das erste oder das zweite Gesetz der Robotik verletzen.

Im allgemeinen Sprachgebrauch wird unter „*Roboter*“ meist eine Maschine verstanden, die dem Aussehen des Menschen nachgebildet ist und/oder Funktionen übernehmen kann, die sonst von Menschen ausgeführt werden. Bei einem menschenähnlichen Aussehen des Roboters spricht man auch von *Androiden*.

In Anlehnung an Todd<sup>15</sup> muss ein klassischer Roboter mindestens über die folgenden Fähigkeiten bzw. Elemente verfügen:

- die Möglichkeit, sich selbst und/oder physikalische Objekte zu bewegen;
- einen Arm, Handgelenke und einen Effektor aufweisen, falls Objekte bewegt werden müssen;
- Räder, Beine besitzen, falls der Roboter mobil sein soll;
- über Antrieb- und Steuerungsmechanismen für die genannten Bewegungen verfügen;
- Speichersysteme Speicherung von Befehlen zur Verfügung stellen;
- mit Sensoren ausgestattet sein, beispielsweise
  - für Berührung, Kräfte, Momente,
  - zur Bestimmung der Position des Roboters, der Stellung des Arms, der Stellung der Handgelenke,
  - zur Messung von Entfernungen,
  - zur Bildaufnahme und zum Erkennen von Form, Größe, Farbe, Bewegung,
  - zur Messung der Wärmeleitfähigkeit, der Temperatur, einer elektrischen Spannung,
  - zur Wahrnehmung von Oberflächenstrukturen, der Härte, des Geruchs von Objekten,
  - zum Erkennen von Schallwellen und Tönen.

---

<sup>15</sup> Sieh auch Todd 1986.

In diesem Buch werden die o. a. Minimalanforderungen dahingehend erweitert, dass von einem klassischen Roboter zusätzlich auch die folgenden Merkmale erfüllt werden müssen:

- *Delegation*: Der Roboter führt im Auftrag eines Anwenders (oder anderer Roboter) diverse Aufgaben aus, zu denen er vom Anwender explizit ermächtigt wurde.
- *Kommunikationsfähigkeit*: Der Roboter muss mit dem Anwender kommunizieren. Einmal muss er dessen Instruktionen entgegennehmen und den Anwender zudem über den aktuellen Status und die Beendigung der Aufgabe informieren (Feedback). Zu diesem Zweck muss der Roboter entweder über eine Benutzerschnittstelle verfügen oder eine entsprechende Kommunikationssprache beherrschen.
- *Autonomie*: Der Roboter führt seine Aufgaben selbstständig aus.
- *Überwachung*: Der Roboter muss zur selbstständigen Ausführung einer Aufgabe seine direkte Umgebung überwachen können.
- *Aktion*: Der Roboter muss in der Lage sein, seine direkte Umgebung durch seine Aktionen aktiv zu beeinflussen.
- *Handlungsorientierung*: Der Roboter muss die überwachten Ereignisse interpretieren, um im Sinne eines autonomen Betriebs alle notwendigen Entscheidungen selbst treffen zu können.

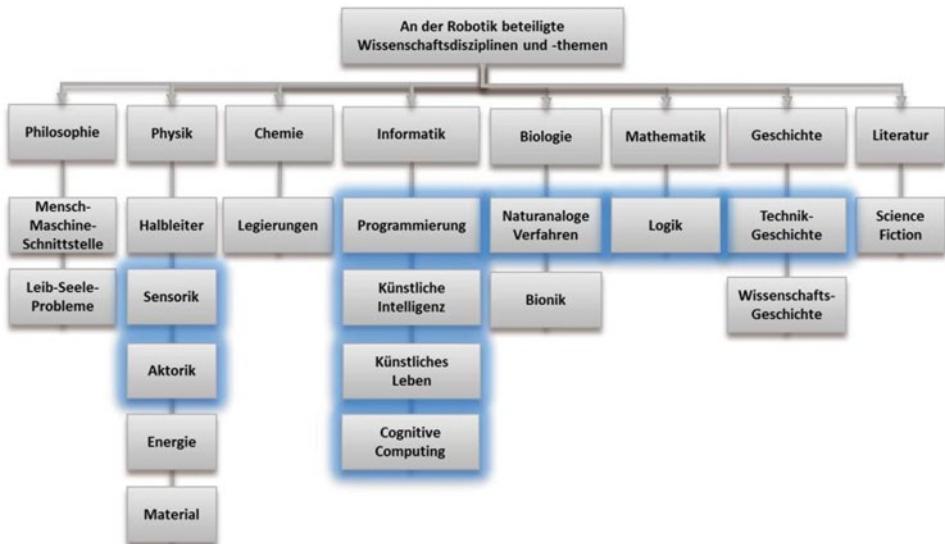
Neben den oben besprochenen Attributen können die Roboter zusätzliche Merkmale wie Mobilität, Sicherheit, Persönlichkeit aufweisen. Diese Attribute entstehen bzw. werden von einem Beobachter dann wahrgenommen bzw. empfunden, wenn die Roboter mit dem Beobachter interagieren. Diese Auffassung von einem *intelligenten Robotersystem* hat auch die inhaltliche Gestaltung des ersten Teils dieses Buches geprägt. Demnach werden Roboter als *lernende Systeme* aufgefasst, diese als *wissensbasierte Modellsysteme* konzeptionalisiert und diese Systeme durch *rechnerbasierte Technologien* in interaktionsorientierter Hinsicht ausgestaltet. Bereits diese klassische Robotik avanciert damit zu einem intradisziplinären Wissenschaftsgebiet, indem auf Erkenntnisse aus unterschiedlichen Teildisziplinen zurückgegriffen wird:<sup>16</sup>

- Informatik<sup>17</sup>
- Artifizielle Intelligenz (z. B. Bilderkennung, Spracherkennung, Lernen, Multiagentensysteme, Planung, etc.)
- Artifizielles Leben (z. B. Schwarmintelligenz)
- Elektrotechnik (z. B. Hardware, Sensorik)
- Maschinenbau (z. B. Kinematik, mechanischer Aufbau)
- Mathematik (z. B. Fehlerkalkulation, statistische Auswertung)

---

<sup>16</sup> Vgl. Husty et al. 1997.

<sup>17</sup> Vgl. Gumm und Sommer 2002.



**Abb. 1.2** Beteiligte Wissenschaftsdisziplin und Themen der Robotik

- Kognitionswissenschaften (z. B. kognitive Modelle)<sup>18</sup>
- Psychologie (z. B. mensch-maschinelle Kommunikation)
- Bionik (z. B. Tierverhalten)
- Biologie (z. B. Gehirn als biologisches Vorbild) (Abb. 1.2)<sup>19</sup>

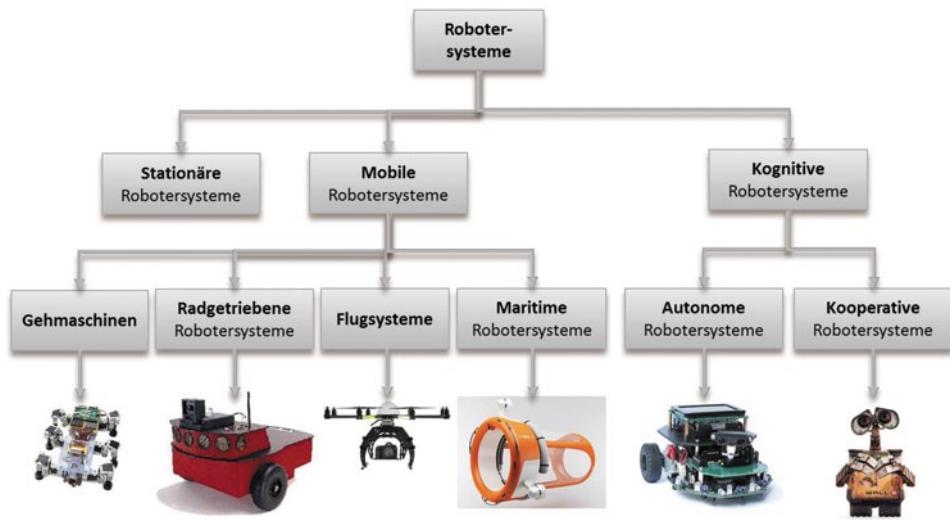
Mit diesem Versuch einer Zusammenführung der Erkenntnisse aus diesen Teildisziplinen zum Zwecke der Erreichung intelligenter Robotersysteme wurden in den letzten Jahren einige Erfolge erzielt.

So haben beispielsweise in Bezug auf die Autonomie von Robotersystemen Forscher und Unternehmen im Jahre 2003 eine Road Map der humanoiden Robotik entworfen. Die Autoren der im Auftrag der EU erstellten Studie, zerlegten ihren Untersuchungsgegenstand dabei in sechs Komponentengruppen: zweibeiniges Laufen, Manipulatoren, Energieversorgung, Kommunikation mit Menschen, Wahrnehmung und Intelligenz. Für jede dieser Gruppen ermittelten die Wissenschaftler den aktuellen technischen Stand, Perspektiven und Ziele. Die Fähigkeit, ähnlich effizient wie ein Mensch zu greifen, wird nach Einschätzung der Forscher in rund sieben Jahren erreicht. Autonome Planung und Ausführung manipulativer Aufgaben ist erst in elf bis zwanzig Jahren zu erwarten. Künstliche Intelligenz gilt den Wissenschaftlern als essenzielle Voraussetzung für persönliche Roboter-Assistenten. Zu deren Entwicklung werde man aber wohl nach deren Einschätzung noch mindestens zwanzig Jahre benötigen.<sup>20</sup>

<sup>18</sup> Vgl. Kurthen 1990.

<sup>19</sup> Vgl. Eccles und Popper 1982.

<sup>20</sup> Siehe auch Görz 1995.



**Abb. 1.3** Klassifikation von Robotersystemen

### 1.3.1 Klassifikation klassischer Rootersysteme

Wegen der Vielzahl von Spezialentwicklungen im militärischen und im Forschungsbereich ist eine umfassende Beschreibung der verschiedenen Robotersysteme nicht möglich. Lediglich der Markt für kommerzielle Systeme präsentiert sich übersichtlich. Insofern kann die im Folgenden vorgenommene Klassifikation nur eine grobe Gliederung liefern und erhebt keinesfalls einen Anspruch auf Vollständigkeit (Abb. 1.3).

Als *stationäre Robotersysteme* bezeichnet man solche, deren Basis (beispielsweise der Roboter-Fußpunkt) starr mit der Umgebung verbunden sind und die daher einen festgelegten Arbeitsraum besitzen. Dabei handelt es sich in der Regel um Roboter, die aus einer Kette starrer Teilkörper bestehen, welche wiederum über Gelenke miteinander verbunden sind. Unterschiedliche Gelenkstellungen führen also zu unterschiedlichen Aufenthaltsorten für den Endeffektor (beispielsweise den Greifer) des Roboters, also den äußersten Punkt der kinematischen Kette.

*Mobile Robotersysteme* unterscheiden sich davon grundlegend, indem sie ihren Standort durch Lokomotion, sozusagen aus eigener Kraft, verändern können. Sie verfügen in der Regel über folgende Eigenschaften:<sup>21</sup>

- Wahrnehmung durch Sensoren
- Aufgabenorientierte und implizite Programmierung
- Anpassung an Veränderungen in seiner Umgebung
- Lernen aus Erfahrung und entsprechende Verhaltensmodifikation

<sup>21</sup> Vgl. Altenburg und Altenburg 1999.

- Entwicklung eines internen Weltbildes
- Selbständiges Planen und Durchführen komplexer Aufgaben auch in unbekannter Umgebung (z. B. Navigationsplanung)<sup>22</sup>
- Manipulation physikalischer Objekte in der realen Welt

Bei den landgestützten Systemen unterscheidet man kinematisch die sogenannten Gehmaschinen und die radgetriebenen Systeme. Diese grenzen sich somit ab von den nicht-bodengestützten Systemen, also jede Form von Flugkörpern und Schiffen. Letztere sind unzweifelhaft von hohem wissenschaftlichem Interesse und wegen der sich ergebenden zusätzlichen Freiheitsgrade der Bewegung eine echte Herausforderung. So ist die Regelungstechnische Behandlung dieser Systeme häufig kompliziert (beispielsweise bei autonomen Helikoptern), andererseits die Bahnplanung und Kollisionsvermeidung durch die Besonderheiten des Bewegungskontinuums erleichtert.<sup>23</sup>

*Gehmaschinen* besitzen humanoide Merkmale, was sie auf der einen Seite menschlich und vielleicht dadurch auch gleichzeitig fremdlich erscheinen lassen. Grundsätzlich ist die Fortbewegung auf Beinen die natürlichste aller Bewegungsformen, erprobt seit Hunderten von Millionen Jahren bei einer Vielzahl von Lebensformen. Diese Form der Bewegung hat sich nicht grundlos bei Landlebewesen durchgesetzt. Sie ist die einzige akzeptable Fortbewegung, die den Bodenbeschaffenheiten in der Natur gerecht wird. Im Vergleich zur radgetriebenen Bewegung besitzt das Gehen in einer natürlichen Umwelt viele Vorteile:

- Auf unebenem Terrain erreichen Tiere als Vorbilder der Gehmaschine im Durchschnitt eine höhere Geschwindigkeit als Kettenfahrzeuge und sie sind den radgetriebenen Fahrzeugen dort deutlich überlegen.
- Die von Tieren erreichbare Geschwindigkeit auf unebenem Boden geht einher mit einer ausgezeichneten Energiebilanz. Diese ist bei rad- und kettengetriebenen Systemen deutlich ungünstiger.
- Radgetriebene Fahrzeuge erreichen ihre Vorteile als schnelles Bewegungsmittel nur bei enormen künstlichen Eingriffen in die Bodenbeschaffenheit, also durch Asphaltierung, Bodenbegradigung, Brücken und Tunnelbau. Bewegen sich diese Systeme hingegen in natürlichen Umwelten, so erzeugen die Bewegungen des Rades bzw. der Ketten erhebliche Boden-Depressionen und Verwerfungen. Die Fußabdrücke von Tieren sind für die Umwelt vergleichsweise unerheblich.
- Trotz gigantischer Anstrengungen zur autogerechten Gestaltung der Umwelt bleiben doch riesige Regionen der Erde langfristig nicht befahrbar für Rad- und Kettenfahrzeuge. Hingegen sind Tiere, die sich auf Beinen bewegen, in allen Landschaften unserer Erde vertreten (z. B. Schafe und Ziegen in Gebirgsregionen, Nagetiere in allen Wüsten, Amphibien in Sümpfen und Feuchtgebieten).

<sup>22</sup> Vgl. Yang 1997.

<sup>23</sup> Vgl. Nehmzow 2002.

- Auf unebenem Boden ist die Fortbewegung auf Pferden (als Beispiel klassischer natürlicher Gehmaschine, die zu Transportzwecken eingesetzt wird) immer noch deutlich komfortabler als die in Offroad-Fahrzeugen.

Es war also nicht eine Unfähigkeit der Natur, dass sie kein Rotationsgelenk „erfunden“ hat, welches eine dauerhafte Drehbewegung erlaubt. Vielmehr war es nüchternes Kalkül mit Blick auf die natürliche Umwelt der Lebewesen. Dennoch ist es natürlich nicht wirklich gerecht, die im Laufe der Evolution über Jahrtausenden hinweg entstandenen Lebewesen mit radgetriebenen technischen Systemen zu vergleichen, deren Bewegungsprinzip seit wenigen Jahrtausenden erfunden ist, und deren technische Entwicklung eigentlich erst mit der Erfahrung moderner Antriebsmaschinen verbunden ist.<sup>24</sup> Dazu nämlich müsste man radgetriebene Fahrzeuge mit technischen Gehmaschinen vergleichen; hierbei trifft man jedoch auf Gehmaschinen, die bisher keineswegs die Vorteile der Vorbilder aus dem Tierreich aufweisen. Dieses Manko lässt sich auf die grundsätzlichen Problematiken zurückführen, mit denen man bei der Entwicklung von Gehmaschinen konfrontiert wird:

- *Bewegungssteuerung* und *Regelung*: Die Regelung einer Gehbewegung gehört zu den schwierigsten zu lösenden Problemen und wird noch aktiv erforscht. Dies gilt insbesondere für den Bereich der statisch instabilen Fortbewegung.
- *Schrittkoordination*: Hier gibt es zwar zahlreiche Erkenntnisse von Zoologen und aus der *Bionik*, jedoch keine Systematik, die für den ingenieurmäßigen Entwurf verfügbar ist. Insbesondere bei stark unebenem Boden, Hindernisüberquerung und Treppensteigen besteht noch Analysebedarf.
- *Antriebstechnik* und Design des Gehapparates: Dies ist der jüngste Forschungsbereich auf dem Gebiet der Gehmaschinen. Gerade im Bereich der Gestaltung von künstlichen Beinen sind noch Erfahrungen zu sammeln. So setzt sich zunehmend die Erkenntnis durch, dass eine effiziente Gehmaschine nicht allein durch eine fortschrittliche Regelung aufgebaut werden kann, sondern dass die Beingeometrie selbst für die Funktionalität der Gehmaschine von erheblicher Bedeutung ist.

Eine Gehmaschine besteht im Prinzip aus einer mehr oder minder komplizierten räumlichen Verteilung starrer Massen, die durch Gelenke miteinander gekoppelt sind. Kennt man die Positionen aller Bauteile und die darauf wirkenden Kräfte zu einem bestimmten Zeitpunkt, kann man die zugehörige Bewegungsgleichung lösen und erhält so die exakte Positionen aller Teile zu beliebigen Zeiten in der Zukunft. Umgekehrt lässt sich dann auch von der gewünschten Bewegung auf die Kräfte zurückrechnen, die nötig sind, um diese Bewegung zu erzeugen. Sinnvoll sind natürlich nur Bewegungsmuster, bei denen sich die Apparatur stabil verhält, d. h. nicht umkippt. Solche Unfälle lassen sich am einfachsten vermeiden, indem man dafür sorgt, dass sich der Maschinenschwerpunkt immer über der Fläche befindet, die zwischen den Füßen mit Bodenkontakt liegt. Allerdings ist diese Art der Fortbewegung recht langsam. Viele zweibeinige Laufmaschinen machen sich daher heute eine Verallgemeinerung dieses Prinzips zu Nutze: Beim dynamischen Gang bewegt sich der Roboter so, dass sich der so genannte Zero-Moment Point (ZMP) stets über einer zwischen den

---

<sup>24</sup> Siehe auch Igor und Piers 1984.

Füßen liegenden Fläche befindet. Das Konzept des ZMP stammt aus der theoretischen Mechanik, es beruht auf dem d'Alembert'schen Prinzip des dynamischen Gleichgewichts. Anschaulich betrachtet ist der ZMP genau der Punkt am Boden, der während der Bewegung abgestützt sein muss. Bewegt sich beispielsweise der Roboter etwa schwungvoll um eine Kurve, kann er seine Hüfte nach außen stellen, um nicht umzukippen. Wird der Roboter nicht beschleunigt, liegt der ZMP im Schwerpunkt. Die meisten Laufroboter sind mit Sensoren ausgerüstet, die ihnen die Position aller Bestandteile und die einwirkenden Kräfte verraten. Diese Daten vergleicht der Roboter mit einer theoretischen Modellrechnung und steuert so seine Motoren an. Wegen ihrer beschränkten Rechenkapazität muss die Maschine allerdings meist auf ein vereinfachtes Modell zurückgreifen. Einige Forscher versuchen deshalb, Gangmuster mithilfe neuronaler Netze zu steuern, ein Ansatz, der zwar in der Theorie gut funktioniert, aber praktisch schwer umsetzbar ist.

Bei den *radgetriebenen Systemen* gilt die Off-Road Navigation bei der eigenständigen Durchfahrt eines beliebigen und unter Umständen unbekannten Terrains sicherlich als die anspruchsvollste Herausforderung. Aufgabenstellungen dabei sind die Gewährleistung der Fahrzeug-Sicherheit und gleichzeitig die Erzielung einer angemessenen Mindestgeschwindigkeit. Diese Problemstellungen führen typischerweise in den Hochtechnologie-Bereich. Hintergrund sind dann militärische Entwicklungsabsichten oder interdisziplinäre Forschungsinteressen in eher finanziertem Umfeld.

### 1.3.2 Allgemeiner Aufbau eines klassischen Robotersystems

Unabhängig von der Aufgabenstellung unterscheidet man folgende Teilsysteme, wenn man den allgemeinen Aufbau eines Robotersystems beschreiben möchte:

- Mechanik
- Kinematik
- Achsregelung und Antrieb (Motoren, Stromversorgung)
- Effektoren
- Sensoren
- Steuerung
- Programmierung

Bedingt durch die mechanische Struktur des Robotersystems ist es in der Lage, in seiner Umgebung mit Effektoren eine definierte Position und Orientierung einzunehmen bzw. eine Bewegungsbahn (Trajektorie) im dreidimensionalen euklidischen Raum abzufahren.

Markante Beispiele für solche mechanischen Strukturen sind:

- Fahrzeug (Lokomotion)
- Roboterarm (Führung des Effektors)
- Effektor, Endeffektor, Hand (Manipulation von Werkstücken und Werkzeugen)

Jedes dieser Teilsysteme weist eine spezifische Beweglichkeit auf, die sich aus seinen *Freiheitsgraden* ergibt. So verfügt beispielsweise ein starrer Körper im dreidimensionalen Raum über 6 Freiheitsgrade: Translation entlang der x-, y- und z-Achse und Rotation um die x-, y- und z-Achse. Die kombinierte Ausnutzung dieser Freiheitsgrade erlaubt die beliebige Positionierung und Orientierung eines Objektes im dreidimensionalen Raum:

- Fahrzeug:
  - Ein auf dem Boden bewegliches Fahrzeug besitzt i. a. 3 Freiheitsgrade: Translation auf der Bodenfläche (x- und y-Achse) und Drehung um die senkrecht zur Bodenfläche stehende z-Achse.
  - Bei Robotersystemen für Anwendungen im Weltraum oder Unterwasser kann das Fahrzeug bis zu 6 Freiheitsgraden besitzen.
  - Manchmal übernehmen Zusatzachsen (beispielsweise Bodenschienen) oder Portale (beispielsweise Laufkräne unter der Decke) die Funktion des Fahrzeugs (1 Freiheitsgrad).
  - Die Fortbewegung kann mit Hilfe von Rädern, Ketten oder Beinen erfolgen.
  - Robotersysteme mit Rädern haben eine einfache Mechanik und lassen sich leicht zusammenbauen. Der prinzipielle Nachteil von Rädern ist, dass sie auf unebenem Grund nicht so leistungsfähig sind wie auf ebenem Boden.
  - Systeme mit Beinen oder Ketten benötigen in der Regel eine komplexere und schwere Hardware als Systeme mit Rädern, die für dieselbe Nutzlast ausgelegt sind.
  - Bei Robotersystemen, die in einer natürlichen Umgebung operieren sollen, sind Ketten allerdings geeigneter, da die größere Lauffläche den Roboter in die Lage versetzt, relativ große Hindernisse zu überwinden. Außerdem sind Kettenfahrzeuge weniger anfällig für Unebenheiten, beispielsweise lose Erde oder Steine.
  - Der größte Nachteil von Ketten ist jedoch ihr geringer Wirkungsgrad.
  - Gehmaschinen können in der Regel unebenes Gelände besser überwinden als Fahrzeuge mit Rädern oder Ketten. Allerdings gibt es viele Schwierigkeiten. Jedes Bein muss mit mindestens zwei Motoren ausgestattet sein. Außerdem ist der Fortbewegungsmechanismus komplexer und folglich fehleranfälliger. Darüber hinaus sind mehr Steueralgorithmen erforderlich, weil mehr Bewegungen zu koordinieren sind. Eine optimale Steuerung von gehenden oder laufenden Maschinen gibt es derzeit noch nicht.
- Roboterarm:
  - Übernimmt das Führen eines Effektors.
  - Besteht aus Armelementen (Gliedern), die über Bewegungssachsen (Gelenke) miteinander verbunden sind.
- Effektor:
  - Bewirkt die Interaktion des Roboters mit der Umwelt.
  - Greifer(systeme) zur Handhabung und Manipulation von Objekten.
  - Werkzeuge zur Werkstückbearbeitung, Messmittel zur Ausführung von Prüfaufträgen.
  - Kamera bei einem nur beobachtenden Roboter.

Die *Kinematik* beschäftigt sich mit der Geometrie und den zeitabhängigen Aspekten der Bewegung, ohne die Kräfte, die die Bewegung verursachen, in die Überlegungen einzubeziehen.<sup>25</sup>

Folgende Parameter spielen dabei eine Rolle:

- Position (Verschiebung, Drehung)
- Geschwindigkeit
- Beschleunigung
- Zeit

Die Kinematik legt die Beziehung zwischen der Lage des Effektors bezüglich der Roboterbasis und der Einstellung der Gelenkparameter fest. Parameter zur Beschreibung der Gelenke sind Drehwinkel bzw. Translationswege.<sup>26</sup>

Durch den *Antrieb* wird die erforderliche Energie auf die Bewegungssachsen übertragen. Der Antrieb muss auch die Kräfte und Momente durch das Gewicht der Glieder des Roboters und der Objekte im Effektor kompensieren. Energie wird also auch dann benötigt, wenn der Roboter sich nicht bewegt. Man unterscheidet zwischen drei Antriebsarten:

- *Pneumatische Antriebsart*:
  - komprimierte Luft bewegt Kolben, kein Getriebe
  - billig, einfacher Aufbau, schnelle Reaktionszeit, auch in ungünstigen Umgebungen brauchbar
  - laut, keine Steuerung der Geschwindigkeit bei der Bewegung, nur Punkt-zu-Punkt-Betrieb, schlechte Positioniergenauigkeit
  - Einsatz für kleinere Roboter mit schnellen Arbeitszyklen und wenig Kraft, beispielsweise zur Palettierung kleinerer Werkstücke
- *Hydraulische Antriebsart*:
  - Öldruckpumpe und steuerbare Ventile
  - sehr große Kräfte, mittlere Geschwindigkeit
  - laut, zusätzlicher Platz für Hydraulik, Ölverlust führt zu Verunreinigung, Ölviskosität erlaubt keine guten Reaktionszeiten und keine hohen Positionier- oder Wiederholgenauigkeiten
  - Einsatz für große Robotersysteme, beispielsweise zum Schweißen
- *Elektrische Antriebsart*:
  - Motoren
  - wenig Platzbedarf, kompakt, ruhig, gute Regelbarkeit der Drehzahl und des Drehmoments, hohe Positionier- und Wiederholgenauigkeit, daher auch Abfahren von Flächen oder gekrümmten Bahnen präzise möglich
  - wenig Kraft, keine hohen Geschwindigkeiten
  - Einsatz für kleinere Roboter mit Präzisionsarbeiten, beispielsweise zur Leiterplattenbestückung

<sup>25</sup> Siehe auch Rieseier 1992.

<sup>26</sup> Vgl. Kovács 1993, S. 18 ff.

*Effektoren* sind die an der Handwurzel angeflanschten Werkzeuge. Markante Beispiele hierfür sind Greifer, Bohrer, Schweißgeräte oder bei einem nur beobachtenden Roboter die Kamera. Die Effektoren müssen ebenfalls gesteuert werden, beispielsweise das Öffnen und Schließen eines Greifers bis zu einer bestimmten Weite. Man benötigt auch Sensoren, um den Zustand der Effektoren zu erfassen, beispielsweise für die Öffnungsweite oder die Schließkraft eines Greifers. Die Steuerung des Effektors erfolgt selbst wieder mit Mikroprozessoren. Die Ansteuerung des Effektors wird in die Programmiersprache integriert. Wichtige Zustände des Effektors werden dem Steuerrechner des Robotersystems permanent gemeldet. Hierdurch kann beispielsweise das Festhalten eines Werkstückes während der Roboterbewegung überwacht werden.

Ein Robotersystem muss, soll es anspruchsvollere Tätigkeiten und Aufgaben übernehmen können, auf die Gegebenheiten der Umwelt adäquat reagieren können. Grundvoraussetzung dafür ist seine Wahrnehmung, d. h. ob es fühlen, sehen oder gegebenenfalls hören kann. Dies gehört in die Zuständigkeit der *Sensoren*. Menschliche Sensoren sind die Augen, die Ohren, die Nase, die Haut und der Tastsinn.<sup>27</sup>

Die Funktionen von Sensoren bei Robotersystemen lassen sich wie folgt zusammenfassen:

- Erfassung der inneren Zustände des Roboters (beispielsweise Lage, Geschwindigkeit, Kräfte, Momente)
- Erfassen der Zustände der Handhabungsobjekte und der Umgebung
- Messen physikalischer Größen
- Identifikation und Zustandsbestimmung von Werkstücken und Wechselwirkungen
- Analyse von Situationen und Szenen der Umwelt

Man unterscheidet dabei 3 Hauptkategorien:

- *Interne Sensoren*: messen Zustandsgrößen des Roboters selbst (beispielsweise Positions-sensoren, Kraftsensoren, Rad-Encoder, Kreisel, Kompass)
  - Position und Orientierung des Roboters selbst
  - Messung des Batteriestands
  - Temperatur im Innern und in der unmittelbaren Umgebung des Roboterssystems
  - Strom und Spannungswerte der Motoren
  - Stellung der einzelnen Gelenke
  - Geschwindigkeit, mit der sich die einzelnen Gelenke bewegen
  - Kräfte und Momente, die auf die einzelnen Gelenke einwirken
- *Externe Sensoren*: erfassen Eigenschaften der Umwelt des Robotersystems (beispielsweise Lichtsensoren, Wärmesensoren, Abstandsmesser, Schallsensoren, Kameras, Mikrophone)
  - Licht
  - Wärme

---

<sup>27</sup> Siehe auch Hooper und Teresi 1988.

- Schall
- Kollision mit Hindernissen
- physikalische Größen im technischen Prozess
- Entfernung
- Lage von Positioniermarken und Objekten
- Kontur von Objekten
- Pixelbilder der Umwelt
- Oberflächensensoren (beispielsweise Tastsensoren)

Sensordaten sind oft verrauscht (d. h. unpräzise), widersprüchlich oder mehrdeutig. Die Steuerung und Auswertung der Sensordaten kann hohe Rechnerleistungen erfordern (beispielsweise zur Bildverarbeitung)

Als *Robotersteuerung* bezeichnet man das Zusammenspiel von Hard- und Software solcher Robotersysteme, die über eine intrinsische, statische, nicht von außen manipulierbare Programmierung verfügen.

Einer solchen Steuerung obliegen folgende Funktionalitäten:

- Entgegennahme und Abarbeitung von einzelnen Roboterbefehlen oder -programmen
- Steuerung, Überwachung von Bewegungs- bzw. Handhabungssequenzen und Fahraufträgen
- Synchronisation und Anpassung des Manipulators an den Handhabungsprozess
- Vermeidung bzw. Auflösung von Konfliktsituationen

Man kann bezüglich der Aufgaben, die man an eine solche Robotersteuerung stellt, folgende Unterscheidung treffen:

- *Ablaufsteuerung*: Übernimmt die Realisierung des Gesamtablaufs der gestellten Aufgabe entsprechend dem eingegebenen Programm
- *Bewegungssteuerung*: Steuerung, Koordinierung und Überwachung der Bewegungen des mechanischen Grundgerätes (beispielsweise Punkt-zu-Punkt-Steuerung, Bahnsteuerung)
- *Aktionssteuerung*: Kommunikation mit der technologischen Umwelt, beispielsweise Koordinierung von Aktions- und Bewegungsabläufen, Synchronisation mit dem technologischen Prozess, Auswertung von Sensordaten

Ein intelligentes Robotersystem setzt voraus, dass es frei *programmierbar* ist, d. h., dass es zum einen mit Algorithmen bzw. Methoden versorgt werden kann, um im praktischen Einsatz intelligentes Verhalten zu zeigen. Zum anderen impliziert diese Forderung aber auch, dass das softwaretechnische System so geartet ist, dass das Robotersystem von sich aus auf wechselnde Anforderungen und Problemstellungen aus seiner Umwelt reagieren kann, ohne dass hierzu dedizierte Änderungen – sozusagen von außen – am softwaretechnischen System vorgenommen werden müssen. Zur Entwicklung der Programme ist eine

Entwicklungsumgebung erforderlich. Diese umfasst die gesamte Hard- und Software, die zur Programmierung, -ausführung und Testen genutzt werden kann. Je nach Art der erforderlichen Entwicklungsaufgaben kann eine solche Umgebung sehr einfach oder sehr komplex sein. Eine komplexe Umgebung entsteht beispielsweise durch Einbindung zusätzlicher Geräte und Sensoren, durch Anschluss an externe Leitrechner und durch Ankopplung von Simulationssystemen. Diese erweiterte Aufgabenstellung übernimmt im Regelfall die Robotersystemsteuerung, die dann auch

- für die Entgegennahme von Roboterbefehlen oder-Programmen,
- für deren Abarbeitung, speziell die Zerlegung von Bewegungsanweisungen in Bewegungssinkemente, sowie
- für die Weiterleitung der Bewegungssinkemente an die Gelenkregelungen des Roboters

verantwortlich ist.

Weiterhin kann man je nach Plattform zwischen dem Roboterbetriebssystem, dem Interpreter der Roboterbefehle, den Dienstprogrammen und die für den Benutzer relevanten Softwarepakete differenzieren. Das *Roboterbetriebssystem* organisiert den Ablauf der Programme, den Zugriff auf Speicher und Dateien sowie die Koordination von Prozessen. Über die Ein- und Ausgabeschnittstellen des Betriebssystems lassen sich Peripheriegeräte anschließen und Kopplungen zu externen Sensoren, Leitrechnern und ganzen Rechnernetzen realisieren.<sup>28</sup> Der *Interpreter* für Roboterbefehle bekommt einzelne roboterorientierte Anweisungen zur Ausführung. Die dort enthaltenen Roboterbefehle und Bewegungsanweisungen werden in eine Folge von Zwischenpositionen auf dem Weg von der Ausgangs- zur Zielposition zerlegt. Jede dieser Zwischenpositionen wird als Bewegungssinkrement an die Regelung weitergeleitet. Das Errechnen der Zwischenpositionen vom Start- bis zum Zielpunkt nennt man auch Bewegungsinterpolation. Der Zeitabstand, in dem ein neues Bewegungssinkrement an die Regelung geschickt wird, heißt Interpolationstakt.

Wird das ganze Roboterprogramm interpretativ ausgeführt, werden im Interpreter nicht nur die einzelnen Roboterbefehle, sondern alle Sprachbestandteile ausgeführt. Insbesondere gehört dazu die Manipulation der Daten, entsprechend der im Roboterprogramm vorgegebenen Anweisungen und Kontrollstrukturen. Bei einfachen Robotersteuerungen ist der Interpretmodul kein eigener Prozess, sondern Bestandteil des Roboterbetriebssystems.

Der Entwickler spezifiziert die gewünschte Roboterbahn normalerweise auf der Ebene der Roboterprogrammiersprachen. Er hat aber bei manchen Steuerungen die zusätzliche Möglichkeit, die Trajektorien für den Effektor direkt oberhalb der Steuer- und Regelungsebenen für die Gelenke vorzugeben. Er kann dann durch die Angabe einer Folge von (sehr vielen) Zwischenpunkten im Interpolationstakt nahezu beliebige Bahnen erzwingen. Allerdings ist dies mühsam und stößt häufig an die Grenzen der Rechnerleistung und Speicherkapazität.

---

<sup>28</sup> Vgl. Stein 2001.

Hinsichtlich der Notwendigkeit, bei der Entwicklung der Programme auf das Robotersystem angewiesen zu sein, unterscheidet man zwischen der on-line und off-line-Entwicklung. Bei der *on-line*-Entwicklung wird das Robotersystem zur Entwicklung der Programme unbedingt benötigt. Bei der *off-line*-Methode wird das Roboterprogramm ohne das Robotersystem entwickelt und letzteres wird erst zum Zeitpunkt des Tests benötigt.

---

## 1.4 Kognitive Robotik

Es zeichnet sich eine vierte industrielle Revolution ab, indem durch eine „Intelligenzierung“ von Produktionsanlagen und industriellen Erzeugnissen bis hin zu Alltagsprodukten in Form einer „Brainware“ mit integrierten Speicher- und Kommunikationsfähigkeiten, Funksensoren, eingebetteten Aktuatoren und intelligenten Softwaresystemen eine Brücke zwischen virtueller („cyber space“) und dinglicher Welt entsteht.<sup>29</sup> Diese Brücke wird eine wechselseitige Synchronisation zwischen digitalem Modell und der physischen Realität gestatten. Letzteres wird nicht nur die derzeitigen Produktlebenszyklen nachhaltig beeinflussen, sondern auch neue Produktplanungs-, -steuerung und Fertigungsprozesse erfordern. So kann das Ziel ressourcensparender, weil effizienter Produktionsprozesse damit angestrebt werden, außerdem damit der Tatsache entsprochen werden, dass das Internet der Dinge bzw. das Internet der Entitäten zum Garanten und Treiber für neue „smarte“ Geschäftsmodelle, der Entwicklung intelligenter und damit „smarter“ Produkte und dem Bereitstellen mehrwertiger Dienstleistungen avanciert.

Die Kognitive Robotik als konsequente Erweiterung der klassischen Robotik wird eine weitere Generation neuer, weil intelligenter, Robotersysteme ermöglichen.<sup>30</sup> Die neue Generation der Kognitiven Roboter wird neben der Simulationstechnik auf die Technologien des Cognitive Computing zugreifen.<sup>31</sup>

Dieser Einbezug der Technologien des Cognitive Computing wird nachhaltig zu neuen Architekturen von Robotersystemen führen, was dann wiederum von zentraler Bedeutung für die Ausgestaltung dieser Systeme mit intrinsischer Intelligenz sein wird. Die Kernvoraussetzung einer solchen artifiziellen Intelligenz im Bereich der Robotik ist zum einen die Fähigkeit, selbstständig denken und damit aus sich heraus Schlüsse ziehen zu können. Zum anderen wird maschinelles Lernen und sensorgestütztes Handeln, das Interoperieren, das Bewegen in einer bzw. das Einwirken auf eine dynamische Umwelt, deren Zustände sich permanent ändern und daher unsicher sind, erst ermöglichen.

Kognitive Robotik ermöglicht eine neue Generation intelligenter Robotersysteme.<sup>32</sup> Derzeit lassen sich drei klassische Robotergenerationen ausmachen, wobei jede dieser Generationen sich durch ein ihr eigenes Programmierparadigma auszeichnet:

---

<sup>29</sup> Vgl. Waffender 1991.

<sup>30</sup> Siehe auch Gevarter 1987.

<sup>31</sup> Cognitive Computing als Wissenschaft von der artifiziellen Kognition wird im Handbuch „Cognitive Computing“ umfassend beschrieben.

<sup>32</sup> Siehe auch Siegert und Bocionek 1996.

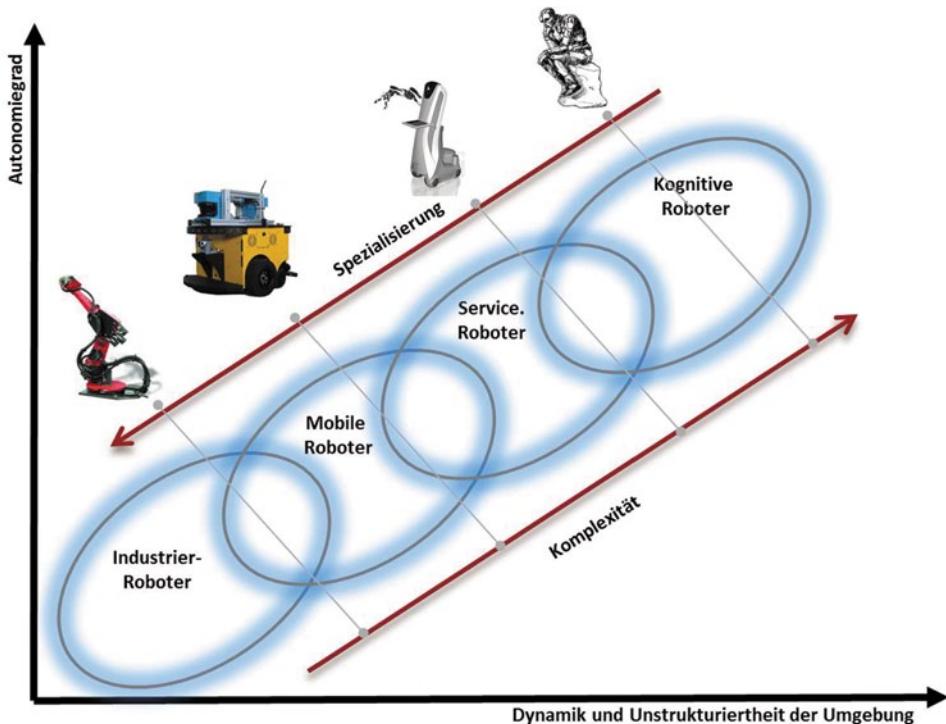
1. Generation: Teach-in-Verfahren,
2. Generation: Roboter-orientierte Programmierung und
3. Generation: Aufgaben-orientierte Programmierung.

Die neue Generation der kognitiven Roboter wird auf die Technologien des Cognitive Computing zugreifen und damit Robotersysteme ermöglichen, die sich durch eine hohe intrinsische Intelligenz auszeichnen. Dies wird es notwendig machen, die bisherige Architektur von Robotersystemen zu überdenken und in neue Hierarchie-Ebenen (Abstraktionsebenen) aufzuteilen, um im Sinne eines kognitiven und wissensbasierten Systems in jeder Ebene die bisherige Dreiteilung Aufgabenplaner (Inferenzmaschine, Kontrollkomponente), Umweltmodell (Wissensbasis) und Sensordaten-Verarbeitung (Datenbasis) neu zu arrangieren. Auch wird die bisherige Unterscheidung in drei Entscheidungsarchitekturen: zentral (eine Entscheidungsinstanz), dezentral (mehrere Entscheidungsinstanzen, die miteinander kooperieren) und zellulär (mehrere Entscheidungsinstanzen, die auch konkurrieren können) eher aufweichen, indem die Entscheidungsarchitekturen fließend ineinander übergehen.

Ein weiterer Zweig, die verteilte kognitive Robotik geht von einem Top-down-Ansatz aus. Die einzelnen Roboter sind dabei intelligent, koordinieren und kooperieren. Hier charakterisiert der Begriff der Koordination die zeitlich synchronisierte Zusammenarbeit von intelligenten Robotersystemen im Dienste einer Aufgabendurchführung bzw. Problemlösung. Die Kooperation zwischen autonomen Robotern als Multiagentensysteme durch Verhandlungen (communication) und Vertrag (contracting) wird die höchste Form der Selbstorganisation darstellen. Sie hat zum Ziel, schwierige Aufgaben und Problemstellungen durch Arbeitsteilung zu lösen (task-sharing), an Resultaten zu partizipieren (result-sharing), gewonnenes Wissen zu verteilen (knowledge sharing) oder Betriebsmittel gemeinsam zu nutzen (resource-sharing).

Kooperierende Robotersysteme müssen Wissen über andere Roboter (Absichten, Fähigkeiten, etc.) besitzen, über verteilte Entscheidungsfindungen (Konfliktbehandlungsmuster, Rollen, etc.), über die Organisation (Zuständigkeiten, Aufgabenzuweisungen, verfügbare Ressourcen, Gruppenwissen, etc.), über die Umwelt (Barrieren, Hindernisse etc.) und vor allem über die Kommunikation (Sprache, Verhandlungsmuster, Medium, etc.). Kognitive Robotik wird auch die Frage beantworten, wie aus einer Ansammlung von Robotersystemen heraus eine höhere Gruppenintelligenz auftauchen kann (Abb. 1.4).

Roboter sind gemäß klassischer Auffassung zunächst Maschinen, die mit Aktoren und/oder Sensoren ausgestattet sind und die von Computern gesteuert werden, um Aufgaben in einer Problemdomäne durchführen zu können. In dieser Hinsicht ist die klassische Robotik eine Disziplin, die sich mit dem systematischen Aufbau, der funktionellen Ausgestaltung (Programmierung) und der Anwendung aufgabenorientierter Roboter beschäftigt, die nach biologischen Vorbildern sehen, gehen (fahren, kriechen etc.), fühlen, greifen und logische Schlüsse (Inferenzen) ziehen können, so dass sie Aufgaben so durchführen,



**Abb. 1.4** Wachstumspfad von Robotersystemen

wie sie von Menschen gemeistert werden.<sup>33</sup> Kognitive Robotik hingegen basiert auf den Ansätzen und Technologien des Cognitive Computing, um nicht, wie üblich, auf die o. a. formal-logische Fähigkeiten eingeschränkt zu bleiben, sondern auch kognitive Vorgänge und Handlungen mit einzubeziehen. Das Ziel der kognitiven Robotik besteht darin, Robotersysteme mit intrinsischen, kognitiven Fähigkeiten so auszustatten, dass sie nicht nur durch Nachahmung menschlicher Arbeitsweisen die ihnen zugewiesenen Teilaufgaben autonom erledigen, sondern auch autonom Probleme lösen. Die Forderung nach Selbstständigkeit, Lernfähigkeit und Problemlösungsfähigkeit beschränkt sich nicht auf individuelle Roboter, sondern schließt auch Gruppen bzw. Teams solcher Robotersysteme ein, die sich durch Kooperationen in Form von Multiagentensystemen selbst organisieren können. Solche autonome, intelligente Robotersysteme werden durch eine Reihe von Leistungsmerkmalen festgelegt:<sup>34</sup>

<sup>33</sup> Vgl. Siegert und Bocionek 1996.

<sup>34</sup> Vgl. Hoppen 1992, S. 5 ff.

- Selbständige Entscheidungsfähigkeit durch Sehen, Planen, Schlussfolgern und Abschätzen der Entscheidungskonsequenzen.<sup>35</sup>
- Gewährleistung der Unabhängigkeit durch Vorgaben und Regelungen.
- Selbständige Durchführung aufgaben-orientierter Zielvorgaben durch die Kombination von Planungs- und Überwachungsschritten.<sup>36</sup>
- Selbständiges Lernen und Beseitigen von Fehlern (Störungsbehandlung) durch maschinelles Lernen.
- Fähigkeit zur Kooperation, Interaktion mit anderen Maschinen und *Interoperation* mit der Umwelt, um nicht nur individuelle Aufgabenstellungen zu erfüllen, sondern auch Teamarbeiten durch Absprachen bzw. Verhandlungen erfolgreich bewältigen und damit auf die Umwelt einwirken zu können.
- Bewältigung von ungeahnten Situationen (Situations- und Konfliktmanagement) und Lösen von nicht vorhersehbaren Problemen (Problemmanagement).
- Vernünftiges Handeln des einzelnen Systems, trotz des Vorhandenseins einer nur relativen und nicht totalen Freiheit.
- Ein freies Verhältnis zu einer Sache und zu sich selbst (maschinelles Bewusstsein).<sup>37</sup>

In diesem Sinne erweitert sich der Begriff Robotik um die multidisziplinäre Zuwendung an bisher fremde Wissensgebiete und dies im Gegensatz zu einer rein ingenieurmäßigen Roboterentwicklung. Aus dieser erweiterten Sichtweise schlagen Robotersysteme die Brücke zwischen Sensorik und Aktorik/Motorik mittels deren kognitiver Fähigkeiten.<sup>38</sup> Diese neue Auffassung betrachtet den Roboter zunächst als ein kognitives Modell, um ihn dann als wissensbasierten Agenten zu konzeptionalisieren und um dieses Agentenmodell durch rechnerbasierte Cognitive Computing Technologien in prozessualer und funktionaler Hinsicht zu einem kognitiven Robotersystem auszuimplementieren. Das Ziel eines solchen Ansatzes liegt in der Steigerung des *systemischen Intelligenzquotienten* ( $IQ_s$ ) des kognitiven Robotersystems bzw. der Erreichung eines bestimmten *Intelligenzprofils* (Abb. 1.5).

Kognitive Robotik avanciert so zu einer inter- und transdisziplinären Wissenschaftsdisziplin, die Sensorik mittels Brainware mit Aktorik/Motorik im Dienste der Ermöglichung systemischer Intelligenzprofile verknüpft. Als solche bestimmt sie den Aufbau und den weiteren Verlauf dieses Buches.

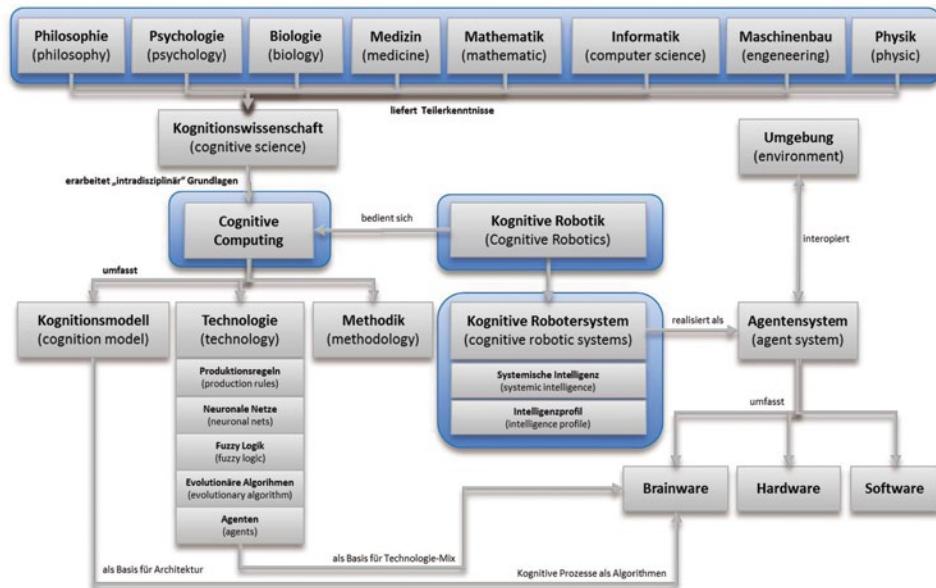
---

<sup>35</sup> Vgl. Jungermann et al. 1998.

<sup>36</sup> Vgl. Sheu und Xue 1993.

<sup>37</sup> Vgl. Pöppel 1985.

<sup>38</sup> Vgl. Wloka 1992.



**Abb. 1.5** Kognitive Robotik als Wissenschaftsdisziplin

## Literatur

- Albert, J., Ottmann T.: Automaten, Sprachen und Maschinen für Anwender. Bbliographisches Institut, Mannheim (1983)
- Altenburg J., Altenburg U.: Mobile Roboter. Carl Hanser Verlag, München (1999)
- Dillmann, R., Huck, M.: Informationsverarbeitung in der Robotik. Springer, Berlin (1991)
- Eccles, J., Popper, K.: Das Ich und sein Gehirn. Piper, München (1982)
- Gevarter, W.: Intelligente Maschinen. Einführung in die KI und Robotik. VCH Verlagsgesellschaft, Weinheim (1987)
- Görz, G., Rollmger, C.-R., Schneeberger, J. (Hrsg.): Handbuch der Künstlichen Intelligenz. Oldenbourg Verlag, München (2000)
- Gumm, H.-P., Sommer, M.: Einführung in die Informatik, 5. Aufl. Oldenbourg, München (2002)
- Groover, M.P., Weiss, M., Nagel, R.N., Prdrey, N.G.: Robotik umfassend. McGraw-Hill (1987)
- Hoppen, P.: Autonome mobile Roboter. BI-Wissenschaftsverlag, Mannheim (1992)
- Hooper, J., Teresi, D.: Das Drei-Pfund-Universum. Das Gehirn als Zentrum des Denkens und Führens. Econ, Düsseldorf (1988)
- Husty, M., Karger, A., Sachs, H., Steinhilper, W.S.: Kinematik und Robotik. Springer, Berlin (1997)
- Igor, A., Piers, B.: Die Roboter kommen. Wird der Mensch neu erfunden? Verlag, Basel (1984)
- Jones, J.L., Flynn, A.M.: Mobile Roboter. Addison-Wesley, Bonn (1996)
- Jungermann, H., Pfister, H.R., Fischer, K.: Die Psychologie der Entscheidung. Spektrum Akademischer Verlag, Heidelberg (1998)
- Kitano, H.: RoboCup-97: Robot Soccer World Cup I. Springer, Berlin (1998)

- Kovács, P.: Rechnergestützte symbolische Roboterkinematik. Vieweg, Braunschweig (1993). Fortschritte der Robotik 18
- Kreutz, H., Bacher, J.: Geist-Gehirn-Künstliche Intelligenz. Zeitgenössische Modelle des Denkens. Berlin (2004)
- Kurthen, M.: Das Problem des Bewußtseins in der Kognitionswissenschaft. Thieme, Stuttgart (1990)
- Lee, M.H.: Intelligente Roboter. VCH, Weinheim (1991)
- Naval, M.: Roboter-Praxis. VOGEL Buchverlag, Würzburg (1989)
- Nehmzow, U.: Mobile Robotik – Eine praktische Einführung. Springer, Berlin (2002)
- Nof, Sh.Y.: (Hrsg.): Handbook of industrial robotics. Wiley, Chichester (1999)
- Pöppel, E.: Grenzen des Bewußtseins. Deutsche Verlags-Anstalt, Stuttgart (1985)
- Rieseier, H.: Roboterkinematik – Grundlagen, Invertierung und symbolische Berechnung. Vieweg, Braunschweig (1992). Fortschritte der Robotik 16.
- Schraft, R.D., Volz, H.: Serviceroboter Innovative Technik in Dienstleistung und Versorgung. Springer, Berlin (1996)
- Sesink, W.: Menschliche und künstliche Intelligenz. Der kleine Unterschied. Stuttgart (1993)
- Sheu, P.C.-Y., Xue, Q.: Intelligent robotik planning Systems. World Scientific, Singapore (1993)
- Siegert, H.-J., Bocionek, S.: Robotik: Programmierung intelligenter Roboter. Springer-Verlag, Berlin (1996)
- Stein, E.: Taschenbuch Rechnernetze und Internet. Fachbuchverlag Leipzig, München (2001).
- Todd, D.J.: Fundamentals of Robot Technology. Halsted Press, New York (1986).
- Volmer, J. (Hrsg.): Industrieroboter Funktion und Gestaltung. Technik, Berlin (1992)
- Waffender, M. (Hrsg.): Cyberspace. Ausflüge in virtuelle Wirklichkeiten. Rowohlt Taschenbuch Verlag, Reinbek (1991)
- Warnecke, H.-J., Schraft, R.D. (Hrsg.): Industrieroboter Handbuch für Industrie und Wissenschaft. Springer, Berlin (1990)
- Wloka, D.W.: Robotersysteme 1. Technische Grundlagen. Springer, Berlin (1992)
- Wloka, D.W.: Robotersysteme 2. Graphische Simulation. Springer, Berlin (1992)
- Wloka, D.W.: Robotersysteme 3. Wissensbasierte Simulation. Springer, Berlin (1992)
- Yang, Q.: Intelligent Planning. Springer, Berlin (1997)

Dieses Kapitel ist der Behandlung wichtiger Grundlagen und der Einführung zentraler Begriffe gewidmet. Der Gebrauch schlecht definierter Ausdrücke mit zentraler Bedeutung sollte in einem guten Buch bzw. in fundierten Theorien vermieden werden. In gewissem Sinne stellen der Verlauf dieses Buches und seine Inhalte ein wissenschaftliches Experiment dar, indem immer Fragen an die Natur im Allgemeinen und an die Robotik im Speziellen gestellt werden. Aus den Antworten und aus den Daten werden vorläufige Vermutungen, das heißt *Hypothesen*, abgeleitet, diese der Gemeinschaft der Leser vorgestellt und damit auch der Validierung bzw. Kritik ausgesetzt. Eine Hypothese, die wiederholt solchen Tests standgehalten hat und die durch viele Beobachtungen und Experimente belegt werden kann, wird schließlich in den Status einer *Theorie* erhoben.

---

## 2.1 System

Das Erkenntnisobjekt der System- und Modelltheorie aus Sicht dieses Buches sind Phänomene der belebten und unbelebten Natur unter Einschluss des Menschen und seinen Fähigkeiten bzw. der von ihm geschaffenen Artefakte.<sup>1</sup> Das Ziel der Theorien ist es, die in den verschiedenen natur- und geisteswissenschaftlichen Disziplinen behandelten Phänomene nach einheitlichen Prinzipien zu erfassen, zu deuten und zu beschreiben, um auf diese Weise eine interdisziplinäre Integration der Denkansätze, Untersuchungsmethoden und Gestaltungstechniken zu erreichen. System- und Modelltheorie stellen somit das basale Instrumentarium bereit, um sich im weiteren Verlauf dieses Kapitels weitere notwendige Begriffe erschließen zu können.

---

<sup>1</sup> Siehe auch Popper 1973.

## 2.1.1 Systeme

Im Allgemeinen spricht man von Systemen, wenn die folgenden Eigenschaften gegeben sind:

- Ein System erfüllt eine bestimmte Funktion, d. h. es lässt sich durch einen Systemzweck definieren, den der Beobachter ihm zuschreibt.
- Ein System besteht aus einer bestimmten Konstellation von Systemelementen und Wirkungsverknüpfungen (Relationen), die seine Funktionen bestimmen.
- Ein System verliert seine Systemidentität, wenn seine Systemintegrität zerstört wird. Das impliziert, dass ein System nicht teilbar ist, d. h. es existieren Elemente und Relationen in diesem System, deren Herauslösung oder Zerstörung die Erfüllung des ursprünglichen Systemzwecks beziehungsweise der Systemfunktion nicht mehr erlauben würde: Die Systemidentität hätte sich verändert oder wäre gänzlich zerstört.

Das Gemeinsame an allen „Systemen“ ist, dass an ihnen Elemente unterscheidbar sind, und dass diese Elemente in irgendeinem sinnvollen Zusammenhang stehen. Dabei können sie schon rein formal in einen Sinnzusammenhang gebracht werden, indem man sie mental und gedanklich nach Ähnlichkeiten, Symmetrien, Passungen oder aber Gegen-sätzen zusammenstellt. Auf diese Weise ist etwa das „periodische System der Elemente“ entstanden, und auf derselben konstruktiven Linie liegt ein „Lotto-Wettsystem“, aber auch das „Wahnsystem“ eines Geisteskranken. In all diesen Fällen meint man mit System ein abstraktes Schema, mit dem der Betrachter Ordnung in seine Wahrnehmungen und Ideen bringt. Er produziert sozusagen auf diese Art und Weise ein Idealsystem. Ein systematischer Zusammenhang kann aber auch darin liegen, dass die Elemente kausal interagieren. Ein solcher Zusammenhang wird dann nicht nur vom Betrachter subjektiv konstruiert, sondern tritt ihm handgreiflich als Realkategorie entgegen. In diesem Sinn spricht man etwa von einem „Zentralnervensystem“, vom „retikulären“ oder „endokrinen System“, vom „Sonnensystem“ und eben auch von einem „Robotersystem“. Auch ein Organismus, eine soziale Gruppe, ein Arbeiter an seinem Arbeitsplatz, die Straßen einer Stadt samt Verkehrsampeln, Kraftfahrzeugen und Fußgängern sind reale Systeme in diesem Sinne.

Insofern orientieren sich auch die zu betrachtenden Systeme dieses Buches im Regelfall an Realsystemen. Unter einem solchen realen System versteht man Teile der beobachtbaren oder meßbaren Wirklichkeit, die sich durch eine – wie auch immer geartete – Beschreibungsmethodik erfassen lassen. Insofern ist ein solches System auch ein zunächst von seiner Umgebung abgegrenzter Gegenstand, das heißt, dass zwischen System und Umwelt differenziert werden muss und dass das System durch seine Systemgrenze von seiner Umwelt getrennt ist. Zwischen System und Umwelt können dabei verschiedene Wechselwirkungen gemessen werden, wie beispielsweise beim Energie-, Stoff- oder Informationsaustausch. Auf diese Weise können einerseits Zustandsgrößen der Umwelt auf das System und seine Entwicklung in der Zeit Einfluss nehmen, umgekehrt kann das Systemverhalten zu Veränderungen in der Umwelt führen.

Die Abgrenzung eines Systems ergibt sich jedoch nicht nur aus seinen physikalischen Grenzen, sondern aus der Fragestellung der Systembetrachtung. Ein wichtiger Bestandteil dieser Betrachtungsweise ist die Umgebung, wobei damit nicht die gesamte übrige Welt gemeint ist. Vielmehr konstituiert sich diese Umgebung aus denjenigen Objekten, die für die Fragestellung der Systembetrachtung wichtig erscheinen und die sich außerhalb des Systems befinden. Diese Grenzziehung darf dabei nicht als eine Art Einschränkung aufgefasst werden. Vielmehr ist dieser konstruktive Schnitt auch deshalb zweckmäßig, weil die kognitiven Strukturen des Menschen bezüglich seiner Auffassungskapazität in Bezug auf systemische Vorgänge und Abläufe eher begrenzt erscheinen. Zum anderen ist ein System ein nach bestimmten Prinzipien geordnetes Ganzes. Es besteht aus Elementen (Komponenten, Modulen, Teilsystemen etc.), die zueinander in Beziehung stehen. Oftmals implizieren gerade diese Relationen eine wechselseitige Beeinflussung, in dem erst aus den Beziehungen heraus ein (Sinn)Zusammenhang entsteht und sich erkennen lässt. Insofern kann man zwischen einer Makro- und einer Mikroperspektive unterscheiden, indem in der Makroperspektive das System als Ganzes und in der Mikroperspektive die inhärenten Systemelemente betrachtet werden. Gerade aus der Makroperspektive lassen sich Beobachtungen machen, die alleine aus der Verhaltensbetrachtung der Elemente, d. h. aus der Mikroperspektive nicht erklärbar sind. Das Ganze ist in solch einem Falle dann eben mehr, als die Summe der Einzelelemente (Emergenz).

Unabhängig von dieser Emergenz zeigt jedes System gegenüber seiner Umgebung gewisse Kennzeichen, Merkmale, Eigenschaften, die als Attribute bezeichnet werden. Als solche Attribute kommen unter anderem vor:

- Komplexität
- Dynamik
- Wechselwirkung mit dem Systemumfeld
- Determiniertheit
- Stabilität
- ohne Energiezufuhr von aussen vs. mit Energiezufuhr von aussen
- diskret (zeit- oder Zustandsdiskret) vs. kontinuierlich
- zeitvariant (Systemverhalten ändert sich mit der Zeit) vs. zeitinvariant (Systemverhalten ist zeitunabhängig)
- linear vs. nichtlinear
- geregelt vs. ungeregelt
- adaptiv (anpassend)
- autonom (unabhängig von äußerer Steuerung)
- selbstregulierend (Selbstregulation)
- selbstkonfigurierend (Selbstkonfiguration)
- autopoiethisch (sich selbst produzierend)
- selbstreferentiell (informationell abgeschlossen gegen die Umwelt, d. h. seine eigenen Informationen erzeugend)
- denkend

**Abb. 2.1** Systembegriff

- lernend
- sozial: Kommunikationen und Handlungen von individuellen und kollektiven Akteuren im Kontext von personalen Beziehungen, Gruppenzusammenhängen, Organisationen und Gesellschaften
- Kognitiv-soziotechnisch: ein System, das aus Personen und Maschinen besteht. Ein solches soziotechnisches System ist beispielsweise ein Unternehmen mit seinen Arbeitsplätzen.

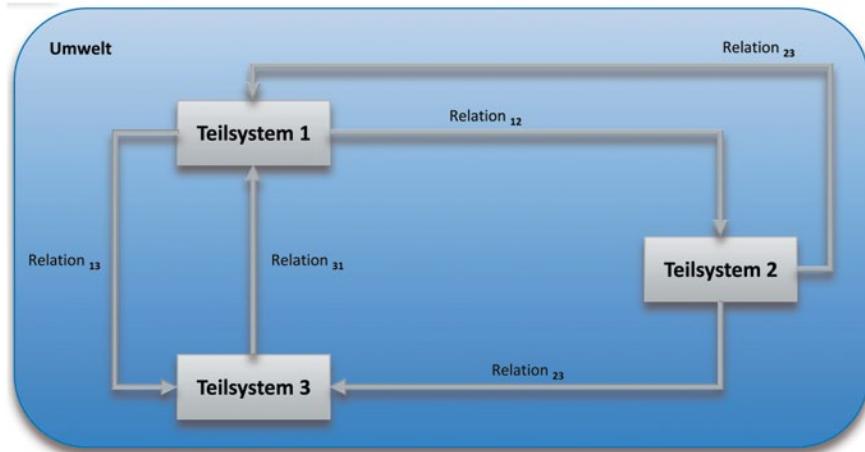
Attribute, die weder Eingangsgrößen (Input) noch Ausgangsgrößen (Output) sind, sondern die Verfassung des Systems beschreiben, werden Zustände genannt. Zwischen den Attributen eines Systems bestehen Beziehungen in Form von Funktionen. Sind diese Funktionen unbekannt, so bezeichnet man das System auch als Black Box (Abb. 2.1).

Systeme können meistens in Teil- oder sogenannte Subsysteme untergliedert werden, die untereinander und mit der Umwelt verbunden sind und mit Hilfe von Kommunikationskanälen Daten und Informationen austauschen. Systeme sind gleichzeitig auch Bestandteile von übergeordneten Metasystemen.

Die einzelnen Subsysteme und deren Beziehungen untereinander lassen sich durch *Relationen*  $R_{ij}$  beschreiben. Die Menge dieser Relationen werden unter dem Begriff der *Struktur* zusammengefasst. Insofern kann man davon sprechen, dass sich die Systeme über bzw. durch Strukturen organisieren und oftmals auch erhalten.<sup>2</sup> Eine Struktur umschreibt also das Muster bzw. die Form der Systemelemente und ihrer Beziehungsnetze, die dafür notwendig sind, damit ein System entsteht, funktioniert und sich erhält. Im Gegensatz dazu bezeichnet man eine strukturlose Zusammenstellung einzelner Elemente als *Aggregat* (Abb. 2.2).

Es lassen sich also zweierlei Aspekte bei der Beschreibung von Systemen ausmachen, einerseits die funktionalen Zusammenhänge zwischen den Attributen eines Systems und die strukturellen Zusammenhänge zwischen den Subsystemen eines größeren Gesamtsystems. Die Feststellung, ein System sei mehr als die Summe der Eigenschaften seiner Teile, beruht gerade darauf, dass die Relationen zwischen den Teilen dem Gesamtsystem eine zusätzliche Qualität verleihen können, die nicht unmittelbar aus den Eigenschaften der Teilsysteme gefolgert werden kann. An dieser Stelle hat sich ebenfalls der Einbezug des Modellbegriffs bewährt, indem ein System als Ganzes nicht als realer Gegenstand betrachtet wird, sondern vielmehr als Modell dieser Realität. Als ein solches Modell aufgefasst,

<sup>2</sup> Vgl. Ebeling et al. 1998.



**Abb. 2.2** Teil (Sub-)Systeme und Relationen

reduziert sich dann oftmals die endlose Fragestellung nach richtig oder falsch eher zu der Frage nach der Zweckmässigkeit. Die Modellierung eines Systems muss sich daher einerseits mit der Ermittlung der Funktionen zwischen den Attributen der Teilsysteme und andererseits mit der Ermittlung der Systemstruktur (Summe der Relationen) eines Gesamtsystems befassen.

Das Systemverhalten kann von bestimmten Eingangsgrößen als Koppelgrößen zu anderen Systemen mitbestimmt werden oder gedächtnisbehaftet und damit zeitvariant sein. Hinter dieser Zeitvarianz verbergen sich oft Alterungerscheinungen, Verschleiß oder unberücksichtigte Einflußgrößen, die schlimmstenfalls sogar einen Systemausfall bedingen können. Je nach Voraussetzung und Perspektive ergibt sich eine Reihe von wesentlichen und unwesentlichen Koppelgrößen. So lässt sich ein System makroperspektivistisch als ein ganz konkreter Ausschnitt aus einer Realität auffassen, in der Aktionen stattfinden, also Prozesse ablaufen. Aber auch im System selbst finden solche Aktionen statt, was zu Zustandsänderungen führt. Die zeitliche Zustandsabfolge in einem System wird dann als aktionsbasierter Prozess bezeichnet.

## 2.1.2 Systemtheorie

Die *Systemtheorie* ist zunächst eine interdisziplinäre Wissenschaft, deren Gegenstand in der formalen Beschreibung und Erklärung der strukturellen und funktionalen Eigenchaften solcher natürlichen, sozialen oder technischen Systeme besteht. Sie erhebt dabei den Anspruch, eine *exakte* Wissenschaft zu sein. Sie will die Wirklichkeit nämlich nicht nur interpretieren, sondern erklären und ihr Wahrheitskriterium ist nicht das subjektive Evidenzgefühl, sondern sie unterwirft sich dem Regulativ falsifizierbarer Beweisführung.

Exaktheit allein genügt aber noch nicht als Kriterium. Die Systemtheorie ist daher auch *empirisch*. Sie hat es eben nicht nur mit gedanklichen Konstrukten, sondern auch mit der Realität, eben mit Systemen, zu tun. In diesem Sinne lässt sich jede exakte Aussage über empirisches Geschehen in drei Komponenten zerlegen. Sie enthält:

- eine (sprachliche oder mathematische) Formel, die den (deterministischen oder statistischen) Zusammenhang zwischen Variablen beschreibt,
- Operationen zur Messung der Variablen,
- die Bestimmung der *Systemklasse*, in der die Messung durchzuführen ist.

Empirische Aussagen betreffen immer den Zusammenhang zwischen Variablen. Solche Aussagen können auf verschiedenste Weise formalisiert sein, beispielsweise als logische Ausdrücke:

$$A \Leftrightarrow F$$

was soviel heißt: wie „A ist genau dann der Fall, wenn F der Fall ist“. Meist lassen sie sich aber auch in Form einer mathematischen Gleichung darstellen:

$$y = \frac{1}{2} x z^2$$

Die obige Gleichung ist zunächst reine Mathematik, d. h. sie wird erst dann zu einer *empirischen Aussage*, wenn der Leser darüber informiert wird, dass darin  $z$  beispielsweise die Fallzeit einer Kugel im Schwerefeld der Erde,  $y$  den von dieser Kugel zurückgelegten Fallweg und  $x$  die Erdbeschleunigung bedeuten soll. Erst durch diese Zuweisung kann die betreffende Formel als Fallgesetz interpretiert werden.

Im Idealfall hat der Formalismus den Charakter eines empirischen *Gesetzes*, wengleich die Tatsache erwähnt werden muss, dass die Realität hinter dem Ideal oft zurückbleibt, oder das Gesetz doch zumindest oft nur hypothetischen Status hat. Die Ausdrücke  $x$ ,  $y$  und  $z$  bezeichnet man als *Variablen*. Variabel heißt zunächst veränderlich und daher umfasst eine Variable die Menge der möglichen (Zahlen)werte, die diese Variable aufnehmen kann.

Wenn beispielsweise die Zimmertemperatur durch eine Variable dargestellt wird, so soll damit zum Ausdruck gebracht werden, dass das Thermometer zu verschiedenen Zeiten verschiedene Werte anzeigen kann, beispielsweise 18, 20 oder 22 °C. Diese Zahlenwerte stehen nicht gleichsam nackt da, sie sind vielmehr benannt und haben damit eine Bedeutung.

Im Sprachgebrauch der Systemtheorie ist es durchaus üblich, hier von der *Dimension* der Variablen zu reden. Der Systemtheoretiker nennt das, was er durch Angabe der Meßvorschrift definiert und durch – oft in eckige Klammern gefasste – Symbole wie [cm], [kg],

[V] usw. bezeichnet, auch die dimensionale Größenart der Variablen. Die Dimension der Variablen ist definiert durch Angabe der Operation, mit der man die Variable misst. Daher kommt der geläufige Ausdruck von der *operationalen Definition*. Mit dieser Definition ist zugleich die Menge möglicher Messergebnisse vorgegeben, d. h. man kann die Dimension auch als das Band auffassen, das die möglichen Skalenwerte zu einer Ganzheit, eben der Variablen, vereint.

Ein System ist ein konkret *instanziertes* Objekt und die Realität besteht im Regelfall aus einer Vielzahl solcher konkreten Instanzen. Um hier die Übersicht zu erhalten, greift man zu einer Abstraktion, indem man mehrere solcher Systeme einer Systemklasse zuordnet. Unter einer solchen *Systemklasse* versteht man eine Abstraktion in Form einer Zusammenfassung mehrerer konkreter Systeme, die in irgendwelchen wesentlichen Punkten, den Klassmerkmalen, übereinstimmen.

Eine solche Systemklasse wäre etwa „Menschen“ oder „Säugetiere“ oder „die Menge der kognitiven Robotersysteme“.

Zahlreiche Einzelwissenschaften haben sich immer wieder die Frage gestellt, mit welcher Betrachtungsweise sie an die Wirklichkeit herangehen sollen, um die dahinter liegenden Theorien oder Gesetzmäßigkeiten besser erfassen zu können.<sup>3</sup> Mit dieser reduktionistischen Herangehensweise ist es ihnen bisher noch nicht in ausreichendem Maß gelungen, diese Wirklichkeit zu verstehen. Gerade in den Naturwissenschaften zeigt sich der Drang, die eher unübersichtliche Welt mit ihrer inhärenten, scheinbaren Wirklichkeit in einzelne und immer kleinere Forschungsbereiche aufzuteilen, um damit im Umkehrschluss eventuell die Sicht – sozusagen die Antenne – für das Ganze zu verlieren.<sup>4</sup> Erschwerend kommt hinzu, dass eine interdisziplinäre Kommunikation durch diese multivariate Zugangsweise zunehmend schwieriger wird. Dies führt dazu, dass unter Umständen das Rad zweimal erfunden wird, dieses jedoch unbesehen bleibt, weil man das Rad eben mit zwei Namen versieht. Die allgemeine Systemtheorie ist somit auch eine Formalwissenschaft, die sich zum Ziel gesetzt hat, die Prinzipien von ganzen Systemen zu untersuchen, unabhängig von der Art der Elemente, Beziehungen und Kräfte, die bestehen oder wirken und damit Bestandteile dieser Systeme sind. Damit sind folgende Zielstellungen verbunden:

- Die Suche nach einem neuen, besseren Weg, die Wirklichkeit zu erfassen.
- Die Suche nach einer Möglichkeit, bruchstückhaftes Einzelwissen wieder sinnvoll in den Gesamtzusammenhang setzen zu können.
- Die Suche nach einer gemeinsamen Basissprache aller beteiligten Wissenschaften, um eine Verständigung und einen Vergleich der Erkenntnisse zwischen den Beteiligten zu ermöglichen.<sup>5</sup>

---

<sup>3</sup> Vgl. Hauffe 1981.

<sup>4</sup> Vgl. Popper 1935.

<sup>5</sup> Siehe auch Oeser 1976.

Die Systemtheorie kann in diesem Sinne als eine Metatheorie aufgefasst werden, deren Ziel es ist, Erscheinungen in ihrer Gleichartigkeit besser zu erkennen und dafür eine einheitliche Terminologie und Methodologie anzubieten. Ausgestattet mit den bisherigen Eigenschaften kann der Systemtheorie die Aufgabe zugewiesen werden, Gesetze und Systeme ohne Bezug zur Dimension miteinander ganz im Dienste der Erkenntnisgewinnung zu verbinden. Dies kann aus zwei formal entgegengesetzten Perspektiven geschehen. Eine erste Möglichkeit besteht darin, ein Gesetz vorzugeben und dann nach einem System zu fragen, dessen Variablen – und zwar gleichgültig davon, welche Messvorschriften für sie gelten mögen – eben diesem Gesetz gehorcht. So ist denkbar, dass ein triviales Gesetz, etwa  $a + b = c$ , einem Konstrukteur als Vorgabe dient, um ein System zu entwickeln, das nichts anderes leistet, als eben diesem Gesetze zu folgen.

Diese Vorgehensweise entspricht genau der, wenn man dem Techniker den Auftrag zum Bau einer Addiermaschine gibt. Denn diese gehorcht ersichtlich gerade dem genannten Gesetz, und ihrem Benutzer ist es völlig gleichgültig, in welchen physikalischen Dimensionen dieses Gesetz realisiert wurde: Er will hinterher nur die reinen Zahlen ablesen und sich darauf verlassen können, dass es eben immer genau die Summe von a und b ist.

Dieses Beispiel erweist sich als Grenzfall eines sehr allgemeinen Auftrages, der heute in zunehmendem Maße an die Techniker ergeht, nämlich, für irgendwelche empirisch gegebenen, aber schwer kontrollierbaren Systeme ein Modell anzufertigen. Dabei geht es vor allem um die Verwirklichung eines Systemgesetzes mit anderen – und beliebigen – qualitativen Mitteln.

Ein weiteres wichtiges Beispiel, das in diesem Zusammenhang fällt, ist die *Simulation*. Auch ein Simulator, etwa der Flugsimulator in der Pilotenausbildung, ist ein Modell, das gewisse gesetzmäßige Zusammenhänge, wie sie im Ernstfall zwischen den Bedienungsinstrumenten, den meteorologischen Bedingungen und dem Flugverhalten herrschen, mit elektronischen Mitteln nachbildet.

Die eben genannten Anwendungsgebiete, bei denen es jeweils darum geht, zu einem Satz vorgegebener Regeln mit beliebigen Mitteln ein System zu erstellen, fasst man unter dem Oberbegriff *Systemsynthese* zusammen.

Diesem systemsynthetischen Ansatz entgegengesetzt ist eine Fragestellung, bei der das System vorgegeben und die in ihm gültige Gesetzmäßigkeit gesucht wird. Entsprechend spricht man hierbei von einer *Systemanalyse*.<sup>6</sup> So ist beispielsweise das Hauptanwendungsgebiet der Systemtheorie in Biologie, Psychologie und Soziologie eher systemanalytischer Natur. Auch hier stellt sich die Frage, inwiefern es sinnvoll ist, bei der Systemanalyse von qualitativen Aspekten abzusehen. Es gibt dafür vor allem zwei triftige Gründe. Da wäre zunächst die Tatsache zu nennen, dass eine Disziplin, die kausale Zusammenhänge unter

---

<sup>6</sup> Siehe auch McMenamin und Palmer 1988.

Absehung von der Qualität zu behandeln erlaubt, vom Zwang zu metaphysischer Stellungnahme bei allen Fragestellungen befreit, in denen man recht schnell in die Nachbarschaft des *Leib-Seele-Problems* gerät.<sup>7</sup> Es ist leicht abzusehen, welche Entlastung diese Betrachtungsweise insbesondere auf den Gebieten dieses Buches mit sich bringt. Aber der Erkenntnisgewinn reicht noch viel weiter. Wie man weiß, hat sich seit einigen Jahrzehnten in der Psychologie eine Gegenströmung zum Behaviorismus etabliert, die als *Kognitivismus* bezeichnet wird. Das systemtheoretische Begriffsinventar ist geradezu unerlässlich, um das kognitivistische Denken davor zu bewahren, durch den missverstandenen eigenen Ansatz auf gedankliche Abwege gelenkt zu werden. Ein zweites Argument ist praktisch noch wichtiger. Wenn ein System – etwa ein lebendiger Organismus – als Vorbild zur Analyse vorgegeben ist, dann weiß man effektiv meist gar nichts über die Natur der Variablen, die im Innern dieses „schwarzen Kastens“, (black box) miteinander interagieren. Problemen dieser Art begegnet man generell bei der Entwicklung von Robotersystemen, bei der man auf Vorbilder der Natur angewiesen ist.

Abschließend und wissenschaftstheoretisch betrachtet sind Systemtheorien demnach keine empirischen, d. h. auf Erfahrung beruhende oder aus Versuchen gewonnenen Theorien an sich. Sie sind vielmehr *Modelle*, die zur Formulierung von empirischen Theorien über komplexe Gegenstände verwendet werden.

### 2.1.3 Systemvarianten

Ein System charakterisiert sich aus seinen Elementen und deren Beziehungen und wird demzufolge als eine Menge von atomaren Elementen definiert, die auf irgendeine Art und Weise miteinander in Beziehung stehen.

Die einfachsten materiellen Systeme sind die Atome, aus denen sich in großer Varietät die nächst höheren Systeme, die Moleküle entwickeln. Diese wiederum bilden in ihrer Komplexität die mikro-, makro- und teleskopischen Systeme der belebten und unbelebten Natur, also die Elemente, Mineralien, organischen Substanzen, Zellen, Lebewesen, Planeten, Planetensysteme, usw. Von Menschen (künstlich) geschaffene materielle Systeme sind (gewonnene) Rohstoffe, Werkstoffe (z. B. Metalle, Kunststoffe, etc.), Werkzeuge, Maschinen, Produktionsanlagen, Datenverarbeitungsanlagen, Gebäude, Verkehrswege, Nachrichtennetze, biotechnische Systeme (z. B. Agrikultur, Viehzucht, durch tierische oder menschliche Kraft betriebene Maschinen), soziale Systeme (z. B. Familien, Gruppen, Teams, Bündnisse, etc.), soziökonomische Systeme (z. B. Firmen, Vereine, Gesellschaften, Genossenschaften, Gewerkschaften, etc.) und soziotechno-ökonomische Systeme (z. B. Betriebe, Anstalten, etc.). Gegen die materiellen sind immaterielle Systeme abzugrenzen. Ein immaterielles System lässt sich ebenfalls als ein Gefüge von Komponenten und Beziehungen definieren, bei denen es sich jedoch um Konventionen handelt, die der Kommunikation, der sozialen Ordnung, sowie der Gedankenordnung dienen. Beispiele dafür sind Sprachen, Codes, Satzungen, Algorithmen, Ver-

<sup>7</sup> Siehe auch Seifert 1989.

fahrvorschriften etc. Die Komponenten dieser Systeme sind Symbole (Laute, Gebärden, Zeichen) mit semantischem Gehalt oder Symbolkombinationen.<sup>8</sup> Die Beziehungen zwischen den Komponenten sind relationale Verknüpfungen (Mengenzugehörigkeit, Rangfolge und Reihenfolge) sowie operationale Verknüpfungen (logische und arithmetische Operationen). Sind immaterielle Systeme bewusst schematisiert, so nennt man sie formale Systeme. Beispiele hierfür sind Grammatiken, Algebren und Gesetzeswerke. Zu den nichtformalen immateriellen Systemen zählen u. a. Verhaltensmuster, Traditionen und Rangordnungen.

Diese Definition legt nicht die Art der Systemelemente oder die Art ihrer Beziehungen zueinander, noch ihre Anordnung, ihre Intension, ihren Sinn oder die Art ihrer Beziehung zu ihrem Umfeld fest. Damit wird zum Ausdruck gebracht, dass es sich bei der Definition eines Systems um eine sehr formale Festlegung handelt, die einerseits auf sehr viele Sachverhalte zutrifft, andererseits einer weiteren Präzisierung ihrer Bestandteile bedarf, um gegebenenfalls ein noch besseres Verständnis zu erreichen. Als atomares Element eines Systems kann zunächst einmal jener Teil verstanden werden, den man nicht weiter aufteilen will oder kann. Dabei kann ein System immer auch Bestandteil eines größeren, systemumfassenden Super- oder Metasystems sein. Gleichzeitig können die atomaren Elemente eines Systems wiederum ein Subsystem abbilden.

So bildet beispielsweise ein Bienenvolk ein soziales System, das aus einzelnen Elementen, den Bienen, besteht. Die Bienen wiederum stellen ein organisches System dar. Somit ist der Organismus der Biene ein Subsystem im Verhältnis zum sozialen System des Bienenvolkes. Ein, das System des Bienenvolks umgebendes Supersystem, ist beispielsweise durch die sie umgebende Flora und Fauna gegeben.

Damit wird deutlich, dass die Begriffe Super- und Teil- bzw. Subsystem relativ zu ihrer jeweiligen Bezugsebene sind und für sich allein genommen noch keine Hierarchiestufen darstellen. Erst im Gesamtkontext, im Zueinander-in-Beziehung-setzen verschiedener Systemkategorien, ergibt sich erst eine sinnvolle Systemrelation. Unter dem Begriff der *Beziehung* ist somit eine Verbindung zwischen Systemelementen zu verstehen, welche das Verhalten der einzelnen Elemente und des gesamten Systems potenziell beeinflussen können.

So werden in dem Versuch, ein real geführtes Expertengespräch im Rahmen der Nachbearbeitung zu modellieren, einzelne Sequenzen des Expertengespräches als Knoten und die einzelnen Aussageeinheiten als Teile des Systems Expertengespräch über Beziehungen (Kanten) zugeordnet.

Die Grenzen zwischen System und Nicht-System lassen sich aufgrund der Beziehungen ausmachen. So kann es zu einem Beziehungsübergewicht kommen, wenn innerhalb der Gesamtheit eines Systems ein größeres Maß an Beziehungen zwischen den einzelnen Elementen besteht, als von der Gesamtheit des Systems zu seinem Umfeld. Man kann sich dabei die Grenzen des Systems als Ellipse vorstellen, in der Interaktionen im Inneren dieser

---

<sup>8</sup> Siehe auch Chaitin 1987.

Ellipse beispielsweise in Form von Kräften, Energien oder Kommunikationen stärker sind als Interaktionsflüsse, die diese Systemgrenze überschreiten.

Somit kann ein Haufen Sand nicht als System angesehen werden. Ein Atom hingegen bildet ein System ab, da seine Elementarteilchen in einem geordneten Wirkungsgefüge zueinander in reger Beziehung stehen. In diesem Sinne stellt jede Pflanze, jedes Tier, jeder Roboter und die Gesellschaft ein System dar.

Die bisher sehr allgemein gehaltene Definition des Systembegriffes ermöglicht eine breite Anwendbarkeit der Systemtheorie als Formalwissenschaft. Einerseits hat dies den Vorteil, dass sich nahezu alle potenziellen Untersuchungsobjekte des Alltags als Systeme beschreiben lassen. Andererseits darf man dabei nicht den Nachteil übersehen, dass die Kennzeichnung eines Untersuchungsobjekts als System noch nicht all zu viel über dieses Untersuchungsobjekt aussagt. Es ist daher notwendig, Systeme über ihre Eigenschaften näher zu charakterisieren. Das charakteristische Kennzeichen eines Systems ist die Tatsache, dass Systeme gegenüber ihrem Systemumfeld abgegrenzt werden müssen. Die Frage der Grenzziehung ist im entscheidenden Maße dafür verantwortlich, was im jeweiligen Untersuchungszusammenhang als System, Sub- oder Supersystem betrachtet werden muss. Wenn man auf diese Art und Weise sein System definiert hat, setzt man gleichzeitig damit voraus, dass die einzelnen Systemelemente untereinander Beziehungen pflegen und/oder aufrechterhalten. Wenn die Systemgrenze hingegen durchlässig ist, einzelne Systemelemente demzufolge Beziehungen zu ihrem Systemumfeld unterhalten, spricht man von einem *offenen System*. Im gegenteiligen Fall spricht man hingegen von einem *geschlossenen System*. Als direkte Konsequenz kann hieraus gefolgert werden, dass geschlossene Systeme keinem übergeordneten Supersystem angehören. Andererseits können offene Systeme Bestandteil eines umfassenden Supersystems sein. Sie können aber beispielsweise auch gleichberechtigt neben anderen Systemen stehen und zu diesen Beziehungen unterhalten, die sich nicht eindeutig einem übergeordneten Supersystem zurechnen lassen. Die Offenheit beziehungsweise Geschlossenheit von Systemen sind dimensionale Eigenschaften, die mit unterschiedlichem Ausprägungsgrad vorkommen können. Dabei wird der Ausprägungsgrad vom Ausmaß der Eingabe beziehungsweise der Ausgabe bestimmt, die aus dem Interaktionsprozess zwischen System und Systemumfeld resultieren. Letzteres bringt man zum Ausdruck, indem man von relativ offenen, respektive relativ geschlossenen Systemen spricht.

Lebensfähige Systeme lassen sich beispielsweise als relativ offene Systeme kennzeichnen. Hingegen können technische Maschinen, so auch Robotersysteme der ersten Generation, als relativ geschlossene Systeme angesehen werden.

Die Dynamik wird innerhalb der Systemtheorie als Prozess aufgefasst, bei dem sich durch eine Bewegung oder ein bestimmtes Verhalten etwas verändert. Es wird dabei zwischen einer äußeren und einer inneren Dynamik streng unterschieden. Dabei bezieht sich die in-

nere Dynamik auf die Aktivität der Systemelemente und ihrer Beziehungen untereinander, während sich die äußere Dynamik auf das Verhalten und die Eingabe-Ausgabe-Beziehungen des Systems gegenüber seinem Umfeld konzentriert.

In diesem Punkt wird wiederum deutlich, dass Systeme mehr sind als eine willkürliche Ansammlung von einzelnen Systemelementen. Die innere Struktur eines Systems sagt allein noch nichts über dessen Verhalten aus. Überlebensfähigkeit von Systemen heißt deshalb in diesem Zusammenhang auch nicht, eine bestimmte Struktur am Leben zu erhalten, sondern dessen Identität zu bewahren. Der kontinuierliche Wandel des Systems ist als laufender Prozess der Dynamik zu verstehen und in der Entwicklung unbedingt zu beobachten.

Lebensfähige Systeme und Robotersysteme im Echtzeiteinsatz sind immer auch relativ dynamische Systeme, während ein Modell von beiden Entitäten eher ein relativ statisches System darstellt.

*Determinierte Systeme* sind solche Systeme, deren Systemelemente in vollständig voraussagbarer Weise aufeinander einwirken. Determinierte Systeme lassen sich daher in ihrem zukünftigen Verhalten, unabhängig davon, ob sie statisch oder dynamisch sind, genau vorausberechnen. Im Gegensatz dazu sind bei *probabilistischen Systemen* keine strengen Voraussagen möglich, es können vielmehr lediglich Voraussagen mit einer gewissen Wahrscheinlichkeit gemacht werden. Damit wird deutlich, dass Voraussagen über das Verhalten von Systemen vom Wissen über diese Systeme abhängig sind. Determiniertheit und Probabilistik als Gegensatzpaare sind dimensionale Eigenschaften von Systemen. Der Grad der Ausprägung wird durch das Maß an exakter Vorhersagbarkeit des Systemverhaltens festgelegt. Das Ausmaß an Undeterminiertheit wird mit dem mathematischen Ausdruck des Freiheitsgrades eines Systems bestimmt. Der *Freiheitsgrad* eines Systems ist dabei definiert als die Anzahl der Möglichkeiten, sich zu verändern. Demzufolge nehmen mit der Anzahl der Freiheitsgrade auch die Komplexität und strukturelle Plastizität des Systems zu. Gleichzeitig reduziert sich im selben Maße die relative Determiniertheit des Systems.

So hat das System „Zug“ einen Freiheitsgrad von 1: vorwärts und rückwärts. Das System „Schiff“ hat einen Freiheitsgrad von 2: vorwärts, rückwärts und seitlich. Das System „Flugzeug“ einen Freiheitsgrad von 3: vorwärts, rückwärts, seitlich, und oben oder unten.

Das kennzeichnende Merkmal *selbstorganisierender Systeme* ist, dass die einzelnen Systemelemente ohne zentrale Steuerungsinstanz, ohne übergeordnete Systemeinheit, sich selbst ihre Beziehungen zueinander und ihr Systemverhalten koordinieren. *Strukturdeterminierte Systeme* können sich ausschließlich innerhalb einer bestimmten Variation ändern, die durch die innere Organisation und Struktur der Systeme determiniert wird. Damit beschreiben diese beiden Eigenschaften die Veränderungsfähigkeit von Systemen. Veränderungen eines Systems sind in zweierlei Art und Weise möglich: Entweder es kommt zu Zustandsveränderungen, oder das System löst sich auf. Wiederum betonen diese bei-

den Eigenschaften nicht die einzelnen Systemelemente, sondern die innere Ordnung und Struktur, die Beziehungen dieser Systemelemente untereinander. Die Strukturdeterminiertheit kann als das Regelwerk aufgefasst werden, die die Spielregeln der Veränderungsprozesse von Systemen festlegt. Innerhalb dieser Spielregeln kann allerdings Selbstorganisation stattfinden. Es können sich aber auch die Spielregeln ändern, falls es sich nicht nur um ein strukturdeterminiertes, sondern auch um ein dynamisches System handelt.

Selbstorganisation und Strukturdeterminiertheit sind dichotomische Systemeigenschaften. Lebensfähige Systeme kommen weder ohne das eine, noch das andere aus. Sie müssen strukturdeterminiert sein, um selbstorganisierenden Prozessen eine dem System innewohnende Richtung zu geben, und damit Auflösung und Chaos vorzubeugen.<sup>9</sup> Sie müssen selbstorganisierend sein, um innerhalb eines komplexen und dynamischen Umfelds bestehen zu können, da eine Kontrolle ausschließlich durch eine zentrale Einheit in komplexen Umfeldern nicht möglich ist. Sie sind immer auch offene Systeme und stehen daher in ständigen wechselseitigen Austauschbeziehungen zu ihrem Umfeld. Deren Verhalten resultiert nicht ausschließlich aus dem Verhalten der einzelnen Systemelemente, sondern aus seiner ganzheitlichen Struktur. Lebensfähige Systeme sind zusätzlich einem hohen Grad an Varietät und Veränderung unterworfen und somit zur Aufrechterhaltung ihrer Lebensfähigkeit fähig. Damit sieht sich deren Identität einer kontinuierlichen Veränderung ausgesetzt.

So ist beispielsweise das menschliche Gehirn ein sowohl strukturdeterminiertes als auch ein selbstorganisierendes System. Es ist strukturdeterminiert, da es erstens nur Daten bewusst als Informationen wahrnehmen kann, wenn dies den bisherigen Wissensstrukturen nicht widerspricht. Es kann zweitens neues Wissen nur dann verankern, wenn altes Wissen diesen direkten Bezug zum neuen Wissen herstellen kann. Das menschliche Gehirn ist selbstorganisierend, da keine zentrale Steuerungseinheit ermittelt werden kann und deren Existenz aufgrund der vorliegenden Erkenntnisse und Experimente der Konnektionisten aus heutiger Sicht äußerst unwahrscheinlich erscheint.

*Adaptive Systeme* haben die Eigenschaft, Veränderungen außerhalb ihrer Systemgrenzen in der Umwelt wahrzunehmen und sich letzteren, soweit möglich, durch eigene Veränderungsprozesse anzupassen. Das System verändert sein Verhalten so, dass sich ein Gleichgewichtszustand zwischen System und Umwelt einspielt. Adaptive Systeme folgen damit einem klassischen Modell der individuellen Lerntheorie.

*Lernfähige Systeme* und damit auch lernfähige Robotersysteme besitzen zusätzlich zur adaptiven und damit reaktiven, eine so genannte antizipative Lernfähigkeit, die sowohl systemexterne Veränderungsprozesse vorwegnehmen, als sie auch beeinflussen kann. Eine bedeutsame Rolle spielen in diesem Zusammenhang das aktive Suchen und Auswerten von Daten bzw. Informationen über die Umwelt. Diese Informationen werden vom System wahrgenommen, als Wissen in den Strukturen des Systems gespeichert und führen

<sup>9</sup> Vgl. Küppers 1987.

damit langfristig zu einer Wissensbasis des Systems über mehr oder weniger erfolgreiche Verhaltensänderungen. Mit Hilfe dieser Wissensbasis kann das System nicht nur auf Umfeldinformationen reagieren, sondern auch künftige Umwelt- und Umfeldentwicklungen vorwegnehmen und damit antizipieren. In diesem Sinne besteht ein Ziel solcher lernfähiger Robotersysteme darin, aktiv auf die Umwelt einzuwirken bzw. am Umweltgeschehen aktiv mitzuwirken, Adaptivität und Lernfähigkeit avancieren somit zu sogenannten dichotomen Systemeigenschaften. Überlebensfähige Systeme sind immer mindestens auch adaptive Systeme. Das Ausmaß ihrer strukturellen Veränderlichkeit entscheidet dabei über den Grad ihrer Überlebensfähigkeit.

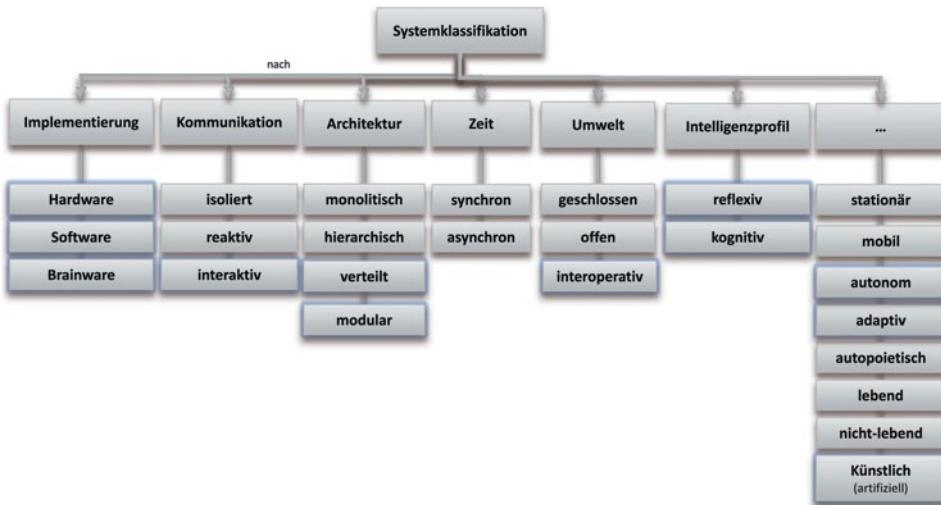
Pflanzen sind beispielsweise adaptive Systeme, die auf Umweltveränderungen durch strukturelle Veränderung reagieren können (Größe, Ausrichtung, etc.). Menschen sind lernfähige Systeme, die durch ihre Wahrnehmungsfähigkeiten und ihre Gehirnkapazität bedingt antizipatives Verhalten zeigen können. Intelligente Robotersysteme sind dann lernfähige Systeme in diesem Sinne, die durch ihre Sensorfähigkeiten und ihre artifiziell-kognitive Fähigkeiten bedingt antizipatives Verhalten zeigen werden.

Eine weitere Variante der Systemunterscheidung zieht eine Grenze zwischen allopoietischen und autopoitischen Systemen und unter den autopoitischen Systemen wiederum zwischen Lebend- und Sinnssystemen.<sup>10</sup> Eine solche Unterscheidung hat sich bezüglich der Modellierung von Robotersystemen als nützlich erwiesen. So sind Robotersysteme gemäß dieser Unterscheidung Sinnssysteme in den Formen psychischer und kommunikativer Systeme, die wiederum in soziale Systeme eingebettet sind. Psychische und soziale Systeme sind autopoitisch operierende Sinnssysteme, die ihrerseits von lebenden Systemen zu unterscheiden sind. Sinnssysteme und lebende Systeme unterscheiden sich durch eigene Elemente. Die Operationsbasis lebender Systeme ist pauschal mit Leben, und die sinnhafter Systeme mit Sinn gekennzeichnet. Entsprechend sind für die Sinnssysteme psychischen und sozialen Typs je eine eigene Operationsgrundlage auszuweisen: Gedanken bzw. Kommunikationen im Modell in Form von Aussageeinheiten. Nicht alle sozialen Erscheinungen erhalten die Würde eines Systems. Am wichtigsten ist die Ausgangsdifferenz allopoietisches System vs. autopoitisches System. Die meisten Implikationen gehen dann von der Differenz psychisches System/soziales System aus, und zwar wegen der damit verbundenen Probleme der Versetzung des Robotersystems in die Umwelt, der Auflösung der Robotersystems in die Einheit der Differenz von organischem und psychischem System und der Emergenz des Sozialen. Insofern kann man die Unterscheidung zwischen geschlossenen und offenen Systemen nunmehr zugunsten der Unterscheidung allopoietischen und autopoitischen Systemen aufgeben (Abb. 2.3).

Ein *allopoietisches System*, meist alternativ als Zustandsautomat oder Trivialmaschine bezeichnet, transformiert aufgrund bestimmter Außeninformationen nach einem festge-

---

<sup>10</sup> Vgl. Luhmann.



**Abb. 2.3** Klassifikation der Systeme

legten Programm der internen Informationsverarbeitung auf genau berechenbare Weise bestimmte Inputs aus seiner Umwelt in bestimmte Outputs an seine Umwelt um.

Als praktische Beispiele seien der Thermostat oder die Werkzeugmaschine genannt.

Demgegenüber erzeugen und steuern sich *autopoietische Systeme* selbst. Ihnen werden weder von außen Eingabeinformationen (Input) zugeführt noch senden sie Ausgabeinformationen (Outputs) an ihre Umwelt. Vielmehr erzeugt ein autopoietisches System selbst die Elemente, aus denen es besteht, durch Verknüpfung zwischen den Elementen, aus denen es besteht. So ist beispielsweise für ein soziales System das Element der Kommunikation konstitutiv, wobei unter Kommunikation ein sinnhaftes, soziales Ereignis zu verstehen ist. Solche autopoietischen Systeme zeichnen sich durch einige Charakteristika aus, die bei den späteren Implikationen in Bezug auf die Entwicklung von Robotersystemen eine durchaus ernst zu nehmende und bisher eher vernachlässigte Rolle spielen. So scheinen autopoietische Systeme in jedem Augenblick ein anderes System zu sein, was die Verhaltensvorhersage an sich schon erschwert. Sie sind außerdem:

- operativ geschlossen,
- kognitiv offen,
- strukturdeterminiert,
- umweltangepasst,
- und produzieren sich temporär.

Die operative Geschlossenheit zeigt sich dadurch, dass eine vollzogene oder auch nicht vollzogene Interaktion eine andere Interaktion bereits voraussetzt, d. h., nur eine Interaktion kann sich unmittelbar an eine Interaktion anschließen. Dieser Umstand zeigt sich im Modell bei den Übergängen, wo zwischen den einzelnen Interaktionen semantische Verknüpfungen eingezeichnet werden können.

Kognitionspsychologisch betrachtet, finden die Systemoperationen in den Grenzen des durch die Systemelemente abgegrenzten Systems statt, wobei die Systemelemente durch das Medium des Systems zur Verfügung gestellt werden. Eine solche operative Geschlossenheit geht einher mit kognitiver Offenheit. Ein psychisches System kann sich beispielsweise durch irgendein Ereignis in seiner Umwelt irritiert und veranlasst sehen, aus dem wahrgenommenen Ereignis für sich eine Information zu machen, d. h., als neue Wahrnehmung in seinen Wahrnehmungsstrom oder als neuen Gedanken in seinen Gedankenstrom einfügen. Insofern ist eine solche Information nicht objektiv als solche von außen vorgegeben, kein Input in das System. Sie stellt eher eine Eigenleistung des – das seine Umwelt beobachtende – Systems dar. Dies gilt auch für intelligente Robotersysteme, die mit der Umwelt interagieren. Jedes autopoeitisch operierende System ist ein strukturdeterminiertes System, indem es selektiv während seiner Geschichte in seiner Auseinandersetzung mit seiner Umwelt die gewonnenen Erfahrungen in seinem Gedächtnis speichert. Erinnerungs-werte Ereignisse der relevanten Ereignisklasse werden kondensiert und konfirmiert, für Wiederverwendbarkeit abrufbar bereithalten oder bei Bedarf auch wieder vergessen.

Eine gewisse Umweltangepasstheit ist schon mit der bloßen Existenz von kognitiven Systemen anzunehmen. Ohne Beteiligung von artifiziell-kognitiven Vorgängen kein artificielles Bewusstsein, ohne Beteiligung eines solchen Bewusstseins keine Kommunikation, ohne Interaktionsfähigkeit und -bereitschaft keine Kommunikation.<sup>11</sup>

Da ein autopoeitisches System sich aus Elementen in der Form von Ereignissen, die an Ereignisse anknüpfen, realisiert, ist es immer auch ein temporäres System. Es realisiert sich von Moment zu Moment durch die Summation von Einzelereignissen. Ein autopoeitisches System erzeugt über seine aneinander anschließenden ereignishaften Operationen Zeit, die Differenz von Vergangenheit und Zukunft in der gegenwärtigen Aktualität seines Operierens.

*Psychische Systeme* erzeugen sich quasi selbst, indem sie Gedanken an Gedanken anreihen. Ein durch einen Gedanken beobachteter Gedanke ist eine Vorstellung und eine beobachtete Vorstellung ist als Bewusstsein zu unterscheiden.<sup>12</sup> Insofern lässt sich Bewusstsein aus systemtheoretischer Sicht durchaus als eine Ansammlung von Operationen auffassen, die sich reflexiv aufeinander anwenden. Dadurch kann sich ein psychisches System in seinen Operationen auch auf sich selbst beziehen. Dadurch wiederum kann es sich aber auch als System vorstellen, das sich quasi selbst vorstellt. Diese Eigenschaft psychischer Systeme gilt es in intelligenten Robotersystemen nachzubilden.

---

<sup>11</sup> Siehe auch Hofstadter und Dennet 1981.

<sup>12</sup> Vgl. Fodor 1975.

**Abb. 2.4** Interaktion zwischen Komponenten



## 2.1.4 Systembegriff

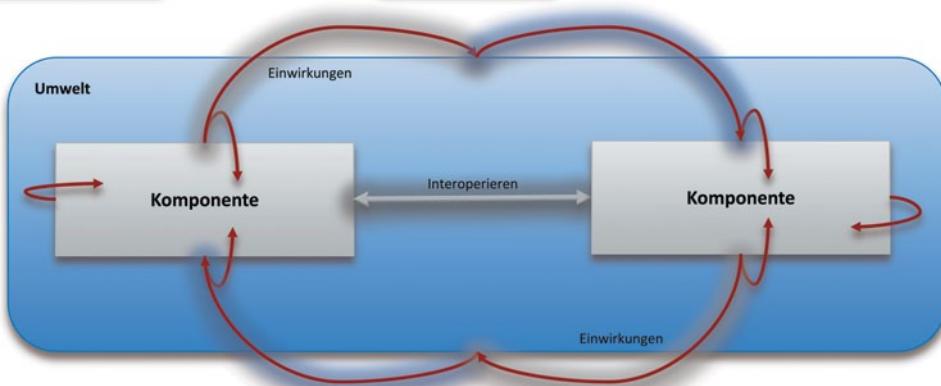
Bereits diese systemtheoretische und eher allgemeine Sicht von einem System enthält implizit schon die Feststellung, dass zwischen System und Umwelt differenziert werden muss und dass das System durch seine Systemgrenze von seiner Umwelt getrennt ist.

Betrachtet man beispielsweise ein Lebewesen unter dem Blickwinkel einer solchen Unterscheidung zwischen System und Umwelt, so gilt es festzustellen, dass das System eine Vielzahl von Eingangs- und Ausgangsgrößen besitzt und diese intern durch Verarbeitungsprozesse in einer Weise verknüpft, dass von außen betrachtet jene Beziehung bzw. Verhältnis zwischen den verschiedenen Entitäten der Welt erst entstehen. Wie dies intern realisiert wird, ist eher von sekundärem Interesse.

Diese Grenze muss demnach wahrnehmbar sein, ansonsten wären keine Messungen daran möglich. Zwischen System und Umwelt können dabei verschiedene Wechselwirkungen gemessen werden, wie beispielsweise beim Energie-, Stoff- oder Informationsaustausch. Auf diese Weise können einerseits Zustandsgrößen der Umwelt auf das System und seine Entwicklung in der Zeit Einfluss nehmen, umgekehrt kann das Systemverhalten zu Veränderungen in der Umwelt führen. Gerade dieser Aspekt wird an späterer Stelle zur Differenzierung von Interaktionen und Interoperationen führen (Abb. 2.4).

Unterschieden werden müssen zwischen Verhaltens- oder Ausgangsgrößen des Systems, die auf die Umwelt einwirken, und den Zustandsgrößen, die in ihrer Gesamtheit das Verhalten und die Entwicklung des Systems bestimmen, auch dann, wenn sie nicht als Ausgangsgrößen zu beobachten oder zu messen sind. Die minimale Zahl der Zustandsgrößen, die es erlauben, das Verhalten des Systems exakt zu beschreiben, wird als *Dimensionalität* des Systems bezeichnet und ermöglicht eine Fixierung dieses Begriffes aus der Systemtheorie.

Eine eher implementierungsnahe Sichtweise definiert das System als eine Menge von *Komponenten*, die durch Kommunikations- und Kombinationsbeziehungen untereinander verbunden sind. Die Komponenten können entweder Teilsysteme des definierten Systems oder dessen Elemente sein. Teilsysteme sind dadurch charakterisiert, dass für sie wiederum die obige Definition gilt, das heißt unter anderem, dass sie wiederum Komponenten haben. Dagegen lassen sich Elemente nicht weiter unterteilen. Komponenten sind zugleich Ansammlungen von Aus- und Einwirkpotentialen sowie *Quellen* und/oder *Senken* von Aktivitäten zur Veränderung dieser Potentiale. Das bedeutet, dass von ihnen *Wirkungen* ausgehen, die die Aus- und Einwirkpotentiale anderer Komponenten verändern, und dass



**Abb. 2.5** Interoperationen zwischen Komponenten und Umwelt

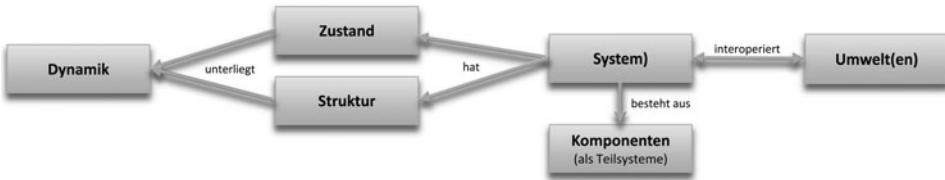
sie umgekehrt *Wirkungen* empfangen, die ihre eigenen Potentiale verändern. Dies führt zur Differenzierung zwischen Interaktions- und Interoperationsbeziehungen.

Die Interaktions- bzw. Interoperationsbeziehungen beinhalten die Einflussnahme von Komponenten auf die Ein- und Auswirkungspotentiale und gegebenenfalls auf die Beziehungen anderer Komponenten. Dies impliziert eine ständige Veränderung des Systems *in der Zeit*, weswegen der Systembegriff zugleich auch ein *dynamischer* Systembegriff ist (Abb. 2.5).

Die Kombinationsbeziehungen beinhalten die *momentane* Zusammensetzung von Ein- und Auswirkungspotentialen eines Systems aus den Potentialen seiner Komponenten. Dies impliziert die rekursive Abhängigkeit der Potentiale jeder Systemkomponente von denen der jeweiligen Subkomponenten, weswegen der Systembegriff zugleich auch ein *hierarchischer* Systembegriff ist.

Ein System zeichnet sich durch die einander ergänzenden oder unterstützenden Eigenschaften, wie Zustand, Struktur und Dynamik aus. Der *Zustand* eines Systems ist die Menge aller in einem Zeitpunkt vorhandenen Ein- und Auswirkpotentiale. Die *Struktur* eines Systems ist die Menge aller Interaktions-, Interoperations- und Kombinationsbeziehungen. Darin sind per Definition sämtliche Komponenten (Ein- und Auswirkpotentiale) eingeschlossen, denn durch sie werden die Anzahl, die Art und die Richtung der Beziehungen bestimmt. Die Struktur gehört zweifellos zu den zeitpunktbezogenen Eigenschaften eines Systems. Allerdings wird man ihr im Allgemeinen eine größere Konstanz zumessen als den momentanen Realisierungen der einzelnen Aus- und Einwirkpotentiale. Die *Dynamik* ist der zeitraumbezogene Ausdruck von Zustand und Struktur, denn diese beiden zeitpunktbezogenen Begriffe implizieren bereits die Veränderlichkeit des Systems in der Zeit. Jede Zustandsgröße wirkt über die Interaktions- bzw. Interoperationsbeziehungen verändernd auf andere Zustandsgrößen ein und wird von anderen verändert.

Die diesem Buch zugrundeliegende Systemtheorie geht davon aus, dass die Systeme und deren Komponenten in wechselseitiger Beziehung zueinander stehen, d. h. interagie-



**Abb. 2.6** Systembegriff

ren und/oder interoperieren.<sup>13</sup> Dabei genügt definitionsgemäß zur Systembildung bereits eine einseitige Beziehung zwischen zwei Komponenten eines Systems.

So kann dieselbe Interaktion bzw. Interoperation, die zur Systembildung führt, aber auch den Zustand von Komponenten (und damit des Systems) verändern und kann in Grenzfällen zur Zerstörung von Komponenten und damit zu einer Umstrukturierung oder Zerstörung des Systems führen.

Gerade bei hochorganisierten Komponenten (Systemen) lässt sich eine wechselseitige Beziehung mit den umgebenden Komponenten (Systemen) beobachten, die zur Stabilisierung der Komponenten (Systeme) in Bezug auf deren Zustand und deren Struktur führt.

Betrachtet man die wechselseitige Beziehung in Bezug auf ein bestimmtes System, so lässt sich folgendes feststellen:

- *Intrinsische Interaktion:* Die Interaktion findet im Inneren des Systems statt, d. h. zwischen den Komponenten und bedeutet den Aufbau des Systems und die Möglichkeit von laufenden Zustands- und Strukturänderungen im Inneren.
- *Extrinsische Interaktion:* Die Interaktion des Systems erfolgt mit umgebenden Systemen und bedeutet den Aufbau eines höheren Systems, und zwar ebenfalls unter Einschluss der Möglichkeit laufender Zustands- und Strukturänderungen der beteiligten Systeme. In Bezug auf das ursprüngliche System kann Interaktion entweder zur Zustands- und Strukturänderung und sogar zur Zerstörung der beteiligten Systeme führen, oder es bedeutet Stabilisierung von Zustand und Struktur.
- *Interoperation:* Dies bezeichnet die Interaktion und gleichzeitige Einwirkung des Systems auf die Umwelt mit dem Ergebnis der Zustands- und Strukturänderung sowohl des Systems, als auch der Umwelt. Die Umwelt eines Systems ist als die Menge der von der Systembetrachtung abgegrenzten neben- und untergeordneten Systeme definiert.

Insgesamt zeigt sich also folgendes Bild (Abb. 2.6):

Die Einführung der wechselseitigen Beziehungen in Form von Interaktionen bzw. Interoperationen führt zu einer weiteren Differenzierung von Systemen. So wird nach der klassischen Systemtheorie ein *offenes System* als ein System definiert, bei dem es mindestens eine Wechselbeziehung zu einem umgebenden System gibt. Dies stellt in diesem Buch

<sup>13</sup> Siehe auch Krieger 1996.

sicherlich den Regelfall dar, denn allein schon die Beobachtung eines Systems stellt zumindest eine Interaktion dar. Aus diesem Grunde wird die Theorie der offenen Systeme innerhalb dieser Systemtheorie und der noch folgenden Kognitionstheorie eine weitreichende praktische Bedeutung erlangen.

Ein idealtypisch *geschlossenes System* hingegen ist als ein System definiert, welches keinerlei Wechselbeziehungen in Form von Interaktionen oder Interoperationen zu anderen Systemen hat. In solch einem Fall finden alle, das System definierende Wechselbeziehungen, zwischen den Komponenten innerhalb des Systems statt. Etwas weniger streng betrachtet wird im Rahmen dieses Buches unter einem geschlossenen System auch ein von allen relevanten Austauschbeziehungen isoliertes System verstanden. Wesentlich erscheint für den Begriff des geschlossenen Systems, dass dessen Komponenten unbedingt wieder als offene Systeme anzusehen sind, da nur die Wechselbeziehungen zwischen diesen Komponenten ein System als solches definiert. Damit ist die „Geschlossenheit“ eines Systems letztlich ein gradueller Begriff, der nur von der Systemabgrenzung abhängig ist.

In diesem Zusammenhang lassen sich auch die Begriffe „Kopplung“, „Rückkopplung“ und „Regelung“ einführen, die eigentlich der Regelungstheorie entstammen. Dabei wird ein offenes System als ein Eingabe-Verarbeitung-Ausgabe-System (Input/Processing/Output-System) aufgefasst, bei dem eine Eingabe zu einem Wirkpotential (Zustand, Gradient) und bei dem dieses Wirkpotential zu einer Ausgabe führt. Unter *Kopplung* wird dann eine interaktive bzw. interoperative Verknüpfung zweier oder mehrerer Komponenten verstanden, bei der die Ausgabe mindestens einer Komponente bei mindestens einer Komponente zur Eingabe wird. Unter dem Begriff der Kopplung subsumieren sich in diesem Buch die Begriffe „einfache Kopplung“ und „Rückkopplung“, wobei letzterer wieder in „direkte Rückkopplung“ und „indirekte Rückkopplung“ unterteilt werden kann. Unter einfacher Kopplung soll eine zyklusfreie Kopplung verstanden werden. Das bedeutet, dass kein System seine Ausgabe an sich oder an ein anderes System zurückgibt, welches bereits direkt oder über andere die Eingabe für das betreffende System geliefert hat. Solche derartigen Strukturen lassen sich graphentheoretisch als Bäume darstellen. Unter *Rückkopplung* wird eine Kopplung verstanden, die einen oder mehrere Zyklen enthalten. Die direkte Rückkopplung ist dann eine Rückkopplung, bei der die Ausgabe ganz oder teilweise als Eingabe in dieselbe Komponente zurückfließt.

Jedes offene System lässt sich als ein Datenverarbeitendes System interpretieren, in welches Daten hineingelangen (Input), diese Daten dann in Zustandsänderungen umgesetzt (verarbeitet) werden, um dann Daten über den Zustand auszugeben (Output) (Abb. 2.7).

In diesem Buch wird beispielsweise der Mensch im Allgemeinen als datenverarbeitendes System und im Speziellen als ein kognitives System aufgefasst. Beide Systeme sind größtenteils hochorganisierte und komplexe Komponentenkombinationen zur Bewältigung komplizierter Datenverarbeitungsvorgänge.

Gemäß dieser systemtheoretischen Sichtweise lassen sich bei einem kognitiven System die Systemkomponenten *Empfänger* (Rezeptoren) für Eingangsdaten, *Zustandsspeicher* und



**Abb. 2.7** (E)ingabe-(V)erarbeitung-(A)usgabe-Prinzip

Sender (Emmitoren) für Ausgabedaten ausmachen. Jede der Systemkomponenten stellt für sich aus systemtheoretischer Sicht ein datenverarbeitendes System dar, d. h. es gelangen Daten in das System, die den Zustand eines Speichers verändern, und es können Daten über den jeweiligen Zustand das System verlassen.

Die im Rahmen dieses Buches behandelten Phänomene müssen entsprechend bezeichnet und klassifiziert werden, wodurch dem System Attribute und Attributklassen zuzuordnen sind. Man unterscheidet je nach dem mit der Begriffsbildung erreichbaren Ordnungsgrad primäre, sekundäre und tertiäre Attribute. Als *primäre Attribute* solcher Systeme sollen alle benannten Phänomene bezeichnet werden, die sich in ein- oder mehrdimensionale metrische Skalen (Intervallskalen) einordnen lassen. Es handelt sich also um Attribute, die z. B. den Ort, das Alter, die räumliche Ausdehnung, die Masse, die Anzahl der Komponenten, den Druck, die Temperatur, die Geschwindigkeit, die Beschleunigung und die Kosten des Systems angeben. Als *sekundäre Attribute* sollen alle benannten Phänomene bezeichnet werden, die sich höchstens in topologische (ordinale) Skalen einordnen lassen. Dabei handelt es sich um Attribute, die z. B. eine Wertschätzung, eine Nutzenangabe, eine Güte-, Größen- oder Schönheitsempfindung für ein System angeben. Gleichtartige oder logisch zusammengehörige Attribute werden zu *Attributklassen* zusammengefasst.

Sämtliche Attribute eines Systems müssen als veränderlich angesehen werden, denn sie bezeichnen lediglich den momentanen Zustand eines Systems, welcher durch Interaktion bzw. Interoperation mit anderen Systemen einem ständigen Wandel unterliegt. Selbst die sehr konstant erscheinenden tertiären Attribute entstehen mit dem Entstehen und verschwinden mit dem Verschwinden des Systems.

Unter diesen Aspekten der Veränderlichkeit ist es praktisch, alle Attributklassen eines Systems als *Zustandsvariablen*, die Attribute (Werte) als Ausprägungen der Zustandsvariablen und die Zeitreihe von Attributen einer bestimmten Klasse als *Zeitfunktion* oder *Prozess* zu bezeichnen. Damit lassen sich dann zwei grundsätzlich verschiedene Arten von Zustandsvariablen beschreiben, nämlich *quantitativ* und *qualitativ* veränderliche Zustandsvariablen. Primäre Attribute sind durchweg der ersten Kategorie, sekundäre sowie tertiäre Attribute durchweg der zweiten Kategorie zuzuordnen. Quantitative Veränderlichkeit be-

deutet, dass die Ausprägungsmenge einer Zustandsvariable (Attributklasse) aus Elementen besteht, die sich nur durch ihre Intensität (Größe) unterscheiden. Qualitative Veränderlichkeit ist gegeben, wenn die zu einer Zustandsvariablen gehörige Ausprägungsmenge aus Elementen besteht, die sich aus verschiedenartigen Phänomenen zusammensetzen.

---

## 2.2 Modell

Sollen bestehende Systeme eingehend analysiert, umgestaltet oder aber Systeme neu entwickelt werden, so erfordert dies umfangreiche analytische und experimentelle Untersuchungen. Speziell in der Robotik gestatten die Komplexität der Systemstrukturen und die vielfältigen Umweltbeziehungen es oftmals nicht, die Zusammenhänge sofort zu erkennen bzw. gleich beim ersten Versuch zielentsprechend zu gestalten. Es entsteht also das Problem, ein durch Analyse, Beobachtung etc. entstandenes Bild in ein experimentelles oder ausführbares Modell zu überführen.

### 2.2.1 Modelle

Ein Modell dient demnach zur Beschreibung der Eigenschaften und der Strukturen eines bestimmten Systems. Es ist daher selten ein absolut vollständiges Abbild des zu betrachtenden Systems. Trotz oder vielleicht gerade wegen dieser Unvollständigkeit berauben Modelle einem Phänomen der Wirklichkeit seine Einzigartigkeit, sehen dieses eher als Produkt bestimmter Kräfte und Bestandteile und stellen es als mehr oder weniger abhängiges Glied in größere Zusammenhänge. Jedes Modell muss dabei damit rechnen, durch ein anderes (umfassenderes, einfacheres, eleganteres, optimales, etc.) ersetzt zu werden. Es gilt daher vor allem für die Robotik: Es gibt oftmals nicht nur ein einziges Modell, sondern viele konkurrierende.

Je nachdem, welchen Zweck man mit der Modellbildung verfolgt, gibt es demnach verschiedenartige Modelle mit unterschiedlichen Eigenschaften. Bereits mit der Festlegung des interessierenden Betrachtungsgegenstandes als eigenständiges System und den daraus resultierenden Koppelgrößen mit der realen Umwelt werden Kompromisse eingegangen, die zu einer mehr oder weniger genauen (scharfen) Modellbildung des Systems und Auswahl der zu berücksichtigenden Koppelgrößen führen. Dadurch entsteht ein Unterschied zwischen Abbildung und der Realität. Damit stellt sich die Frage, wie sich dieser Unterschied auf die möglichen Erkenntnisse auswirkt. Dabei kommt einem die Erkenntnistheorie entgegen. Die Erkenntnistheorie ist derjenige Zweig der Philosophie, der sich mit Gewinnung kognitiver Methoden, dem Wesen und eben den Grenzen von Erkenntnis befasst.<sup>14</sup> Unter diesem Aspekt sind Robotersysteme mit entsprechender Software *formale*

---

<sup>14</sup> Siehe auch Kriz und Heidbrink 1990.

*Maschinen*, die die Realität nicht in ihrer gesamten Komplexität erfassen können, sondern nur in einer reduzierten Form: ihnen sind nur die formalen Aspekte von Realität in Gestalt *formaler Modelle* zugänglich, weil sie nur *formale Sprachen* in Form spezieller Programmiersprachen verstehen. Modelle sind damit eher Abbilder realer Gegenstände und Abläufe, denen bestimmte Eigenschaften im Dienste der Komplexitätsreduktion verloren gegangen sind. So verwendet man gerade im Alltag viele solcher reduzierten Modelle, ohne eigens darüber nachzudenken.

Kinderspielzeuge fehlen zum Beispiel die Eigenschaften „Größe“ (Spielzeugautos) und „Leben“ (Stofftiere). Landkarten werden mit speziellen Projektionsverfahren erstellt, die von den Eigenschaften „Größe“, „Erdkrümmung“, „Oberflächenrelief“ etc. abstrahieren.

Letztere sind erste Beispiele für formale Modelle. Sie sind in einer formalen Sprache formuliert. Formale Sprachen können gleichzeitig Wörter, Buchstaben, graphische Symbole und ggf. Farben verwenden (Landkarten, technische Zeichnungen, etc.). In der Informatik werden die einsetzbaren Elemente erweitert, etwa um Klassenmodelle und Geschäftsprozessmodelle, Wörter und mathematische Zeichen (Programmiersprachen) oder um Buchstaben und mathematische Zeichen (mathematische, physikalische und chemische Formeln). Darstellungen in formaler Sprache sind eindeutig und lassen keinen Interpretationsspielraum zu, wie etwa natürlichsprachliche Äußerungen:

Der Satz „Das ist aber ein schönes Geschenk!“ kann je nach Tonfall und Begleitumständen Freude oder Missfallen ausdrücken.

Formale Modelle – insbesondere in der Informatik (zum Beispiel von betrieblichen Aufgabenbereichen) – sind spezielle Formen von *Erkenntnis*. Sie sind das Ergebnis kognitiver, empirischer Verfahren (Beobachtung, Beschreibung, Typisierung, Abstraktion, Formalisierung). Damit sind sie und ihre Erstellung Gegenstand erkenntnistheoretischer Überlegungen.

Aus der geisteswissenschaftlichen Philosophie sind viele erkenntnistheoretische Schulen hervorgegangen, die in der Beurteilung des Erkenntniswerts von Modellen grundverschiedene, teilweise abstruse Positionen einnehmen. Erst mit der Entwicklung naturwissenschaftlicher Erkenntnistheorien in der Physik und Biologie des 20. Jahrhunderts lichtet sich dieses Dickicht.<sup>15</sup> Es soll aber nicht die bisherige Streitfrage „Wie gut können Modelle die Realität beschreiben?“ im Vordergrund stehen, sondern die Frage: „Wie praktikabel sind die Modelle für die Erkenntnisgewinnung?“

Der *naive Realismus* vertritt beispielsweise die Ansicht, dass Modelle Eins-zu-Eins-Abbilder (isomorphe Abbilder), also exakte, nicht verzerrte, nicht entstellte Abbilder der Realität sind. Dieser Objektivitätswahn findet sich häufig in der Mathematik und der allgemeinen Informatik, nicht aber in der modernen Robotik. Zeigen doch schon einfache

---

<sup>15</sup> Siehe auch Mittelstaedt 1963.

Selbstversuche, etwa mit optischen Täuschungen, erst recht aber Erfahrungen mit der Problematik des Einsatzes betrieblicher Informationssysteme in der Wirtschaftsinformatik,<sup>16</sup> speziell in der Simulation natürlicher Sachverhalte mit Hilfe von Neuro-Fuzzy-Systemen, dass zwischen Modell und Realität immer eine „Realitätskluft“ besteht.<sup>17</sup> Diese entsteht unvermeidlich einerseits aus der grundsätzlichen Komplexitätsreduktion bei jeder Art von Modellierung, andererseits durch den Modellkonstrukteur selbst: Modelle sind immer Menschenwerk, kognitive Konstrukte. Sie entstehen nicht passiv mit dem Fotoapparat und fallen nicht vom Himmel, sondern sind Ergebnisse aktiver Realitätsinterpretationen durch einen Modellkonstrukteur, der je nach beschriebenem Realitätsausschnitt seine persönliche Subjektivität einbringt und einen ganz bestimmten Modellierungszweck verfolgt.

Diese Einsicht führt zum *kritischen Realismus*, der allerdings über die Größe der Kluft zwischen Modell und Realität im jeweiligen Einzelfall keine Aussage macht. Hier hilft erst die *evolutionäre Erkenntnistheorie* weiter, die aus der Biologie hervorgegangen ist. Sie sieht den menschlichen Erkenntnisapparat als Produkt der Evolution, d. h. als im Laufe von Jahrhunderttausenden an die Umwelt angepasst und erprobt, an. Dieser Apparat kann sich keine existenzgefährdenden Irrtümer leisten, d. h. die Spannung zwischen Modell und Realität muss sich in überschaubaren Grenzen halten. Allerdings verläuft die technisch-kulturelle Evolution der letzten 5000 Jahre wesentlich schneller als die biologische Evolution, so dass der menschliche Erkenntnisapparat keine Chance hatte, sich in vollem Umfang an mittlerweile völlig veränderte Erkenntnisgegenstände anzupassen. Resultat ist, dass heute weiterhin steinzeitliche (naiv-realistische) kognitive Strategien verwendet werden, mit dem Ergebnis, dass bei komplexen Erkenntnisgegenständen (Unternehmen, betrieblichen Abläufen, Erkennen von Situationen, Wetter, Klima, etc.) erhebliche Klüfte zwischen Modell und Realität entstehen können. Zumindest kann die evolutionäre Erkenntnistheorie auf problematische Aspekte in Modellierungsprozessen hinweisen, einigermassen detailliert erklären und damit einerseits den Grad der Abweichung von der Isomorphie einschätzbar machen und andererseits Wege zu Gegenmaßnahmen, das heißt zu einer Überbrückung der Kluft, aufzeigen.

Trotz dieser Auseinandersetzung mit der Erkenntnistheorie erhält der Leser kein Patentrezept zur Durchführung perfekter Modellierungen, denn die prinzipiellen erkenntnistheoretischen Probleme können durch keine Modellierungsmethode beseitigt werden. Jedoch kann Wissen um erkenntnistheoretische Zusammenhänge und bewusste Auseinandersetzung mit ihnen deren unerwünschte Folgen sehr wohl erheblich mindern. Es gibt sowohl allgemeine Gesetzmäßigkeiten als auch fachspezifische, die weit über die Resultate aus der Beschäftigung mit dem *human factor* in den Wissenschaftsdisciplinen hinausgehen.

Requirements Engineering ist ein gutes Beispiel, wie man erkenntnistheoretisches Wissen in der Informatik hervorragend nutzen kann. Im Gegensatz zu naturwissenschaftlichen Erkenntnisgegenständen kann ein menschlicher Experte in natürlicher Sprache über seine Expertise Auskunft geben, weil er Mensch und damit einer Sprache mächtig ist. Der Informa-

---

<sup>16</sup> Vgl. Hansen und Neumann 2001.

<sup>17</sup> Siehe auch Amberg 1999, S. 30 ff.

tiker beobachtet also nicht passiv betrachtend, sondern aktiv befragend und beeinflussend. Er bekommt die Anforderungen des künftigen Robotersystems zunächst in natürlicher Sprache, also prämodellhaft. Diese Anforderungen sind oftmals inkonsistent, unvollständig, voll vom Experten geprägten Ausdrücken. Daraus soll der Entwickler nun vor dem Hintergrund seines subjektiven Vorwissens ein konsistentes, vollständiges, formales Modell der Prozesse und Informationsflüsse ableiten. Allein sich dieser Rahmenbedingungen der Modellierungssituation bewusst zu sein, führt zu deutlich besseren Modellen.<sup>18</sup>

Der Tatsache zum Trotz, dass für die Modellierung keine in allgemeingültige Regeln fassbare Vorgehensweise geliefert werden kann, lassen sich drei allgemeine Anforderungen nennen:

- *Forderung nach Transparenz*: Die Modellelemente müssen klar definiert, eindeutig beschreibbar und in sich widerspruchsfrei sein.
- *Forderung nach Modellgültigkeit*: Die Folgerungen über das Verhalten, die man aus den Verknüpfungen der Modellelemente zu einem Gesamtmodell ziehen kann, müssen im Rahmen des Modellzwecks (Gültigkeitsbereich) dem realen Systemverhalten entsprechen.
- *Forderung nach Effizienz*: Gibt es verschiedene Möglichkeiten zur Darstellung des Systems, die alle den ersten beiden Forderungen genügen, so sollte man die einfachst mögliche auswählen („Keep it simple“).

Die Kunst bei der Modellierung besteht demnach darin, das Modell so einfach wie möglich zu gestalten, um es mit technisch und wirtschaftlich vertretbarem Aufwand untersuchen und später realisieren zu können. Dabei dürfen aber keine unzulässigen, das Systemverhalten zu stark verfälschenden Annahmen getroffen werden. Insofern kann die Maxime aufgestellt werden: *Alles so einfach wie möglich und nur so kompliziert wie unbedingt nötig*.

## 2.2.2 Modelltheorie

Die in diesem Buch entwickelten Ansätze basieren auf der allgemeinen Modelltheorie, die im Jahre 1973 von Herbert Stachowiak entwickelt wurde. Diese Basis wurde nicht zuletzt aus dem Grunde gewählt, damit der entwickelte Modellbegriff nicht auf eine ganz bestimmte, dedizierte Wissenschaftsdisziplin festgelegt ist. Vielmehr soll ein problemdomänen-übergreifender Begriff zum Einsatz kommen, der eher allgemein anwendbar erscheint. Nach Stachowiak ist ein Modell vor allem durch drei Merkmale gekennzeichnet:

- *Abbildung*: Ein Modell ist immer ein Abbild von etwas, eine Repräsentation natürlicher oder künstlicher, originärer Realitäten, die selbst wieder Modelle sein können.
- *Minimierung*: Ein Modell erfasst nicht alle Attribute des Originals, sondern nur diejenigen, die dem Modellierer bzw. dem Modellanwender relevant erscheinen.

<sup>18</sup> Vgl. Rembold und Levi 2003.

- *Pragmatismus*: Pragmatismus bedeutet hier die Orientierung am Nützlichen. Ein Modell ist einem Original nicht *a priori* von sich aus zugeordnet. Die Zuordnung wird vielmehr durch die Fragenstellungen „Für wen?“, „Warum?“ und „Wozu?“ getroffen. Ein Modell wird vom Modellierer bzw. Modellanwender innerhalb einer bestimmten Zeitspanne und zu einem bestimmten Zweck für ein Original eingesetzt. Man kann also davon sprechen, dass das Original durch das Modell interpretiert wird.

Ein Modell zeichnet sich also durch *Abstraktion* aus, um durch eine bewusste Vernachlässigung bzw. Reduktion bestimmter Merkmale, die für den Modellierer oder den Modellierungszweck wesentlichen Modelleigenschaften hervorzuheben. Dabei wird – im Gegensatz zu Modellbegriffen einzelner Wissenschaften – kein bestimmter Abstraktionsgrad vorausgesetzt, um ein Konstrukt dennoch als Modell bezeichnen zu können.

### 2.2.3 Modellvarianten

Modelle können hinsichtlich ihrer Originale, ihrer Intention, ihrer Attribute und der Adäquatheit von Original und Modell unterschieden werden. Ein einzelnes Subjekt verbindet mit einem konkreten Modell stets eine bestimmte Intention, zu dem das Modell konstruiert oder genutzt wird, wobei Modellkonstrukteur und Modellanwender nicht automatisch die gleichen Intentionen mit dem gleichen Modell verbinden müssen (Perspektivenproblem). Mit Modellen verbundene Intentionen können sein:

- Didaktische Veranschaulichung,
- Experimentalisierung,
- Repräsentation,
- Prognosen,
- Kommunikation,
- Theoriebildung,
- Nutzbarmachung eines Originals,
- Erkenntnisgewinn,
- Handlungsgrundlage
- u. v. m.

Den Hauptzweck einer Modellbildung in der Informatik sehen einige Autoren darin, „die aus den Fachwissenschaften stammenden Modelle so umzuschreiben, dass sie mit Hilfe eines Computers dargestellt und bearbeitet werden können“.

Transformiert auf die Belange der Robotik besteht somit die Intension einer Modellbildung darin, die aus den Fachwissenschaften stammenden Modelle so umzuschreiben, dass sie im Rahmen einer Entwicklung von Robotersystemen wiederverwendet werden können.

Da Modelle zweck-, zeit- und subjektorientiert erstellt werden, können zu einem Original gleich verschiedene Modelle (ko)existieren. Zum Vergleich von Original und Modell muss dem Modell-Designer und/oder dem Modellnutzer bekannt sein, unter Anwendung welcher Operationen das Modell an das Original angeglichen wurde. Es lassen sich wiederum in Anlehnung an Stachowiak drei Angleichungsebenen unterscheiden:

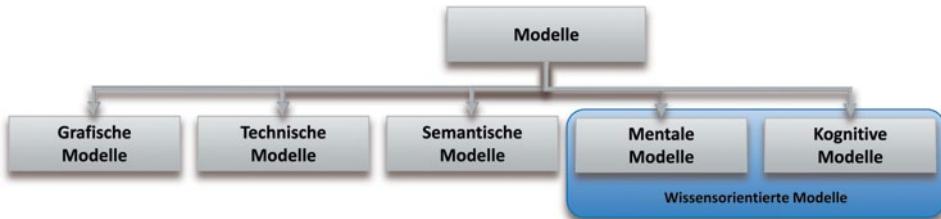
- Die *strukturell-formale Angleichung* bezieht sich auf Attribute, die Relationen zwischen Komponenten der Entitäten beschreiben.
- Die *material-inhaltliche Angleichung* bezieht sich auf (sekundäre) Attribute, die die Bedeutung, den Sinn und die Vorstellung zu einem Attribut beschreiben.
- Die *naturalistische Angleichung* bezieht sich auf Attribute, die materiell-energetische und raumzeitliche Eigenschaften von Entitäten beschreiben.

So können beispielsweise die potentiellen Wege des Handlungsreisenden in einem isomorphen Graphen dargestellt werden (strukturell-formale Angleichung). Es kann jede Komponente eines vollständigen Schaltbildes die gleiche Bedeutung wie die jeweilige Komponente in der realen Schaltung haben. Ein derartiges Schaltbild ist nicht nur von maximaler materialer Angleichung (isohyl), sondern auch isomorph (material-inhaltliche Angleichung). Es kann in einer Simulation eine Kontraktion der Zeit relativ zur Echtzeit im Original erfolgen (naturalistische Angleichung).

Stachowiak gibt zur Differenzierung von Modellen eine pragmatische Einteilung in grafische, technische und semantische Modelle an, die sich sicherlich noch weiter verfeinern lässt:

- *Graphische Modelle* sind im Wesentlichen zweidimensionale Modelle. Die Originale stammen meist aus dem Bereich der Wahrnehmung, der Vorstellung und den darauf aufbauenden gedanklichen bzw. mentalen Operationen. Grafische Modelle, die unmittelbar ihre Bedeutung repräsentieren, werden als ikonisch bezeichnet, während symbolischen Modellen ihre Bedeutung, ihr spezieller Kode zugeordnet werden muss.
- *Technische Modelle* sind vorwiegend dreidimensionale, raum-zeitliche und materiell-energetische Repräsentationen von Originalen. Entsprechend der Natur ihrer Attribute lassen sich physiko-, bio-, psycho- und soziotechnische Modelle unterscheiden.
- *Semantische Modelle* sind Kommunikationssysteme, die ein Subjekt zur informatiellen Verarbeitung seiner empfundenen Wirklichkeit einsetzt. Es wird zwischen den internen Modellen der Perzeption und des Denkens sowie den externen semantischen Modellen unterschieden, die sich aus Zeichen und Zeichenkombinationen aufbauen.

Zur Entwicklung von Systemen im Allgemeinen und Robotersystemen im Speziellen wird man nicht nur auf alle drei Varianten von Modellen zurückgreifen müssen, sondern man wird weitere Modelltypen hinzuziehen müssen. Insofern werden die Modelltypen von Sta-



**Abb. 2.8** Modellvarianten

chowiak durch zwei weitere, eher an den Konzepten Daten-Informationen-Wissen orientierte Varianten komplettiert (Abb. 2.8).

Gerade die Entwicklung von kognitiven Systemen wird bedingen, dass sich beim Modellieren die Übergänge zwischen einzelnen Modelltypen eher fließend gestalten. Diese multiple Verwendung einzelner Modelltypen zeigt sich in den unterschiedlichen Bereichen der Systementwicklung.

Im Bereich der Softwareentwicklung werden verschiedenste Darstellungsmodelle für *Vorgehensmodelle* des Entwicklungsprozesses, zum Festhalten von Entwürfen und als Kommunikationsgrundlage verwendet. Die bekanntesten Vorgehensmodelle in der Softwareentwicklung sind das Phasenmodell, das iterierte Phasenmodell, das Prototypenmodell, das evolutionäre Modell, das transformationelle Modell, das Spiralmodell und das objektorientierte Modell.<sup>19</sup> Weitere Vorgehensmodelle finden sich beispielsweise beim Qualitätsmanagement (V-Modell). Diese vielfältigen Modelle der Informatik können alle mit graphischen Darstellungsmodellen, aber auch in schriftsprachlicher Form (externe semantische Modelle) beschrieben werden. Die Vorgehensmodelle selbst, die diesen Darstellungen zugrunde liegen, basieren auf den von den an der Entwicklung Beteiligten erzeugten Aktivitäten (interne semantische Modelle).

So intendieren Booch et al. (1999) mit der Unified Modelling Language (UML) eine Vereinheitlichung von Methoden für die objektorientierte Softwareentwicklung.<sup>20</sup> Die UML stellt dazu eine Vielzahl unterschiedlicher *graphischer Darstellungsmodelle* zur Verfügung, die in den verschiedensten Phasen des Softwareentwicklungsprozesses eingesetzt werden können:

- Strukturdigramme (Klassen- und Paketdiagramme),
- Verhaltensdiagramme (Anwendungsfall, Interaktions-, Sequenz-, Kollaborations- Zustands- und Aktivitätsdiagramme) und
- Implementierungsdiagramme (Komponenten- und Einsatzdiagramme).

<sup>19</sup> Vgl. Bunse und Knethen 2002.

<sup>20</sup> Siehe auch Jacobson et al. 1999.

Weitere graphische Darstellungsmodelle finden sich in der Theoretischen Informatik (Graphen, Bäume, Darstellungen der Turingmaschine oder der Registermaschine), in der Technischen Informatik (Schaltbilder von logischen Schaltgattern oder Rechnerarchitekturen<sup>21</sup>), in der Praktischen Informatik (OSI-Referenzmodell, Schichtenmodelle, etc.) usw.<sup>22</sup>

*Ikonisch-grafische Bildmodelle* sind in der Regel ohne weitere Erläuterungen verständlich. Sie finden sich in der Informatik und dort speziell im Bereich der Software-Ergonomie: Piktogramme unterstützen den Benutzer bei der Bedienung von Soft- und Hardware, indem Vertrautes teilschematisch abgebildet wird und damit (hoffentlich) eine Assoziation erzeugt wird.

Beispielsweise soll die Ikone des „Mülleimers“ die Funktion „Datei löschen“ versinnbildlichen.

Die Entwicklung solcher Veranschaulichungsmodelle für „robotische“ Modelle ist eine der wichtigsten Aufgaben der Robotik. Hierzu zählt die Visualisierung von und der Umgang mit zwei oder dreidimensionalen Szenen als Monitorbild oder -bildfolgen und – im Übergang zu den technischen Modellen – die Holographie.

So scheint es, dass mit den Modellen der UML zahlreiche Darstellungsmodelle der damit verbundenen Aspekte abgedeckt werden, was für eine Anwendung der UML in der Robotik spricht.

Darstellungsmodelle und Bildmodelle werden nach Schwill (1994) und Thomas (1998) in der Informatik meist als Zwischenschritte zur Konstruktion von sogenannten „enaktiven“ Modellen eingesetzt. *Enaktive Modelle* übernehmen partiell die Dynamik des Originals und werden vom Menschen als weitestgehend identisch zu ihren Originalen erfasst. Es lassen sich dabei vier Typen unterscheiden:

- simulierende,
- registrierende,
- regelnde und
- autonome Modelle.

Als Beispiele aus der Praxis lassen sich anführen: Simulierende Modelle: Simulationswerkzeuge, Registrierende Modelle: Messdatenverarbeitung, Regelnde Modelle: Ampelsteuerung und Autonome Modelle: Agenten.

Je nachdem, ob die Modellattribute im Wesentlichen anorganischer, organisch-organischer, psychischer oder sozialer Natur sind, unterscheidet man auch oftmals technische Modelle in physiko- bio-, psycho- und soziotechnische Modelle. Zu den *physikotechnischen Modellen* zählen in der Informatik vor allem elektronische Modelle, wie beispiels-

---

<sup>21</sup> Siehe auch Tanenbaum 1999.

<sup>22</sup> Siehe auch Martin 2001.

weise integrierte Schaltkreise. Diese, basierend auf booleschen Operatoren, gelten als „Gehirn“ eines Informatiksystems oder eines Roboterarmes und können als elektronische Alternative zum menschlichen Pendant verstanden werden. *Mechanische* und *elektromechanische Modelle* finden sich vorwiegend in der Entwicklungsgeschichte der Informatik, beispielsweise lochkartengesteuerte Webstühle, die Rechenmaschinen von Schickard, Pascal und Leibniz sowie Holleriths Zählmaschine.<sup>23</sup> Bezeichnenderweise sind mechanische Modelle in der Informatik überwiegend dynamischer Natur. Als *elektro-chemische Modelle* kann man den Quantencomputer und die neuesten Entwicklungen in der informationsverarbeitenden Nanotechnik ansehen. Unter einem *Computermodell* wiederum versteht Stachowiak ein „durch den Automaten realisiertes Programm“, d. h. es beschreibt die kommunikations- und informationsverarbeitenden Prozesse in einem Computer.

Die Konstruktion von Informatiksystemen, die auf organisch-organismischen Bausteinen beruhen, ist sicherlich noch Zukunftsmusik, doch seitdem vor ca. 40 Jahren dem schwedischen Ingenieur Arne Larsson in Stockholm der erste völlig mobile Herzschrittmacher eingepflanzt wurde, wird versucht, elektronische Geräte mit dem biologischen Organismus direkt zu koppeln. Am 24. März 1998 gelang Roy Bakay in Atlanta die erste Direktverbindung zwischen Menschengehirn und Elektronengehirn, mit der eine Nervenschädigung eines Patienten teilweise aufgehoben und damit behoben werden konnte. Die Verschmelzung von lebenden und artifiziellen informationsverarbeitenden Systemen kann dazu führen, dass nicht nur biologische Organismen artifizielle Systeme kontrollieren, sondern auch artifizielle Informatiksysteme biologische Systeme.

Für die kommunikative Welt des Menschen hat Stachowiak ein Metamodell semantischer Stufen vorgeschlagen, das auf im Wesentlichen drei interdependenten Ebenen die Verwendung von *semantischen Modellen* zur Kommunikation aufzeigt. Modelle einer höheren Stufe stellen jeweils Kommunikationsmodelle für Systeme der niedrigeren Stufe dar.

- Auf der nullten, uneigentlichen semantischen Stufe, befinden sich alle Zeichenträger (materielle Information), potentiell bedeutungstragende Ausdrucksformen, aus denen sich alle möglichen Ausdrücke eines Kommunikationssystems konstruieren lassen: Laute, Gesten und andere Signale.
- Zur ersten semantischen Stufe zählen interne semantische Modelle der Perzeption. Diese Modelle dienen der intrinsischen Kommunikation<sup>24</sup> des Subjekts mit sich selbst und vermitteln zwischen der subjektbezogenen Realwelt und dem operativem Zentrum, dem Gegenwärtigkeitssystem sowie dem Informationsspeicher des Subjekts. Wahrnehmung und Denken findet sowohl unter Anwendung vorhandener interner Modelle statt, als auch durch Entwicklung neuer interner Modelle statt. Letzteres geschieht beispielsweise bei der Interpretation von Simulationsergebnissen. Ohne interne semantische Modelle und der Möglichkeit, auf diesen mentalen Operationen arbeiten zu können, ist der Mensch nicht handlungsfähig.

---

<sup>23</sup> Siehe auch Heintz 1993.

- Modelle der zweiten Stufe und eventuell folgender Stufen sind Systeme zur Kommunikation über Modelle der jeweils niedrigeren Stufe, sogenannte externe semantische Modelle: verbal-sprachlich, schriftsprachlich, fachsprachlich, maschinensprachlich usw. Aufgrund seiner Erfahrung mit Telefonbüchern kennt ein Mensch beispielsweise den Prozess der Interpolationssuche und hat ein internes Modell hierzu entwickelt. Diesen Prozess kann er verbal vermitteln oder mittels einer „natürlichen“ Schriftsprache wiedergeben. Die Verwendung einer formalen Sprache führt zu einem externen semantischen Modell, das den Prozess der Interpolationssuche derart präzisiert, dass beispielsweise vergleichende Effizienzbetrachtungen angestellt werden können. Auf einer „höheren“ Stufe wird möglicherweise eine Programmiersprache oder eine Maschinensprache verwendet, um den Prozess der Interpolationssuche für einen Computer aufzubereiten. Die Interpolationssuche wird also in der Informatik durch verschiedene semantische Modelle repräsentiert, abhängig vom Zweck des Modells. Aufgabe des Subjekts ist es, das geeignete (externe) semantische Modell anzuwenden.

Interessanterweise kann man das Metamodell semantischer Stufen auf die kommunikative Welt der informationsverarbeitenden Maschinen übertragen, indem der Computer als Subjekt aufgefasst wird. Dann könnte die Einteilung in Analogie zu Stachowiak wie folgt aussehen:

- *Modell der nullten Stufe*: enthält alle Zeichenträger ohne Bedeutung, also kontinuierliche und diskrete Signale, Daten und Nachrichten. Diese Stufe unterscheidet sich inhaltlich kaum von der obigen nullten Stufe.
- *Modell der ersten Stufe*: enthält Perzeptionsmodelle (Wahrnehmung = Signalempfang) und kognitive Modelle (Denken = Signalverarbeitung). Informatiksysteme enthalten meist Gebilde zum Empfang von Signalen optischer, auditiver und elektrischer Natur. Die Signalverarbeitung findet vorwiegend auf elektronischem Wege statt. Insbesondere letzteres wird in der Technischen Informatik, einem Kerngebiet der Informatik, behandelt. Die elektronischen Modelle werden unter anderem aus Modellen von Mathematik (Dualzahlensystem), der Biologie (Nervensystem) und der Kognitionswissenschaften (Mustererkennung) mittels der Modelloperationen gewonnen. Wenn man vereinfachend und provokant davon ausgeht, dass der Mensch interne semantische Modelle letztendlich auch nur basierend auf elektrischen Signalen ausbildet, könnte man soweit gehen zu sagen, dass einem (zukünftigen) Computer diese Fähigkeit der Verarbeitung von internen Modellen ebenfalls zugestanden werden muss. Kognitionswissenschaften und Informatik arbeiten in diesem Bereich fächerverbindend und befruchten sich gegenseitig.
- *Modell der zweiten Stufe*: und alle folgenden Stufen enthalten externe semantische Informationseinheiten, die im eigentlichen Sinne bedeutungstragenden Zeichen in der expliziten Kommunikation zwischen Subjekten (hier: Computern) darstellen. Hierzu zählen Kommunikationssysteme, die Meta-Modelle für Modelle der ersten Stufe darstellen (Wörter und Sätze, derzeit auf binärer Ebene). Der Mensch speichert und ver-

arbeitet seine internen semantischen Modelle mittels elektrischer Signale. Leider kann er im Gegensatz zum Informatiksystem auf diesem Weg im Regelfall nicht mit anderen Subjekten kommunizieren. Zur zwischenmenschlichen Kommunikation werden daher auditive, visuelle und taktile Sprachen auf unterschiedlichen Abstraktionsstufen eingesetzt. Diese Kommunikationssysteme werden auch für die Mensch-Maschine-Kommunikation verwendet.<sup>24</sup>

*Kognitive Modelle* basieren auf dem zentralen Paradigma, dass es sich bei den mentalen Funktionen und Leistungen des Menschen, wie beispielsweise:

Wahrnehmung, Wissen, Gedächtnis, Denken, Problemlösen, Lernen, Sprechen, Sprachverstehen, etc.

um Vorgänge der Informationsverarbeitung handelt. Eine effektive Methode zur Analyse solcher kognitiver Funktionen und Leistungen im Rahmen des Paradigmas einer solchen Informationsverarbeitung stellt die kognitive Modellierung mit Hilfe wissensbasierter Systeme dar. Kognitive Leistungen beruhen vor allem auf Deutungen und Interpretationen von Wahrnehmungsinhalten. Menschen konstruieren durch Vorgänge der Wahrnehmung und des Sprachverständens, sowie aufgrund ihres erarbeiteten Vorwissens, ihrer zukünftigen Erwartungshaltungen und ihrer persönlichen Ziele eine interne, subjektiv geprägte Repräsentation einer Problem-, Lösungs- und/oder Anwendungsdomäne. Diese Konstruktionen bilden das *mentale Modell* einer Person über diese bestimmte Domäne.

So bezeichnet die Psychologie mit Kognition die mentalen Prozesse und Strukturen eines Lebewesens, wie Gedanken, Meinungen, Einstellungen, Wünsche, Absichten. Kognitionen werden hier als Informationsverarbeitungsprozesse verstanden, in denen Neues gelernt und Wissen verarbeitet wird.

In Anlehnung an die Psychologie umfassen kognitive Leistungen die Wahrnehmungsfähigkeit bzw. Auffassungsgabe, die Kombinatorik und die Fähigkeit, aus vorhandenen Daten und Informationen neue Informationen und damit Wissen herzuleiten. Um diese kognitiven Leistungen erbringen zu können, sind mentale Funktionen notwendig. An späterer Stelle dieses Buches wird sich aber zeigen, dass sich Kognition nicht einfach als die Summe mentaler Funktionen ergibt, sondern sich eher als eine multiplikative Verknüpfung mentaler Funktionen und damit als Emergenz zeigt. Insofern basieren die kognitiven Modelle auf der Ausgangsvoraussetzung, dass Robotersysteme als wissensbasierte Systeme aufgefasst werden können. Eine wesentliche Voraussetzung hierfür ist die Grundannahme, dass kognitive Leistungen unter anderem als Vorgänge der Manipulation von wahrgenommenen Symbolstrukturen aufgefasst werden können. Eine wesentliche Eigenschaft wissensbasierter Systeme ist deren Fähigkeit, über ihrem internen Modell einer Anwendungsdomäne Operationen der Informationsverarbeitung durchzuführen und auf diese

---

<sup>24</sup> Siehe auch Bamme und Feuerstein 1983.

Weise die zu erklärenden Verhaltensphänomene erzeugen zu können. Zusätzlich zu diesem symbolischen Verarbeitungspotenzial treten auch subsymbolische Verarbeitungsmöglichkeiten, die als eine weitere Grundannahme den kognitiven Modellen zugrunde gelegt werden müssen.

Weiterhin legt man sich ein Architektur- und Konstruktionsprinzip zugrunde, das davon ausgeht, dass artifizielle Systeme die Problemdomänen in der gleichen Weise rekonstruieren, wie Menschen dies in vergleichbaren Situationen tun. Dabei greift man auf das Prinzip der Nachahmung, der sogenannten *Mimesis* zurück und entwickelt damit vereinfachte Modelle, indem bewusst weniger bedeutsame Variable ignoriert werden.

Ganz allgemein kann man auf diese Art und Weise jedes System derart neu definieren, dass es sich aus einer beliebigen Anzahl von Teilen zusammensetzt, um dann Zusammenhänge zwischen diesen Teilen zu entdecken. Wenn man ausschließlich gewisse allgemeine Zustände vorhersagen will, reicht eine Theorie mit einer geringen Zahl von Variablen im Regelfall aus. Komplizierter wird es allerdings, wenn man zu den speziellen Teilsystemen gelangen will. Insofern ist die Darstellung so zu wählen, dass bei möglichst großer Exaktheit der Prognose möglichst wenige Variablen zu berücksichtigen sind.

Bereits bei den historischen Ausprägungen von Mimesis geht es um die Erzeugung symbolischer Welten mit Hilfe unterschiedlicher Medien. So haben mimetische Welten eine eigene Existenz und können aus sich selbst heraus verstanden werden. Aber sie sind nicht in sich geschlossen, sondern nehmen immer Bezug auf eine Umwelt oder eine andere Welt. Historisch gesehen kann diese Bezugnahme der verschiedensten Art sein. In jedem Fall besitzt die mimetische Welt ihre eigene Daseinsberechtigung neben all den anderen Weltbildern.<sup>25</sup> Gerade in dieser Umweltbezogenheit unterscheidet sich die mimetische Modellierung grundlegend von Theorien, Modellen, Plänen und Rekonstruktionen und ergänzt somit sicherlich die kognitive Modellierung.

Mimesis ist dabei ein handlungsorientierter Ansatz, ganz im Sinne des Herstellens von Lösungen für Probleme bzw. dem Finden von Antworten auf gestellte Fragen. Mit dem Aspekt des Herstellens wird ausgedrückt, dass Mimesis eine Aktivität von Handelnden ist, ein konkretes Tun. Dabei wird ein praktisches Wissen eingesetzt, das scheinbar unmittelbar, ohne langes Nachdenken bestimmte Handlungsmuster verfügbar macht und mit der Wahrnehmung einer Situation eine Interpretation und Reaktionsweisen bereitstellt, die die nächsten Handlungsschritte antizipieren lassen. Mimesis ist damit die in einer symbolischen Welt objektivierte Antwort eines Subjekts, das sich an der Welt von Anderen orientiert. Mimesis zielt auf Einwirkung, Aneignung, Veränderung, Wiederholung, Wiederverwendung und ihr Vehikel ist die Neuinterpretation von wahrgenommenen Welten.

Eine kognitive Modellierung unter Einbeziehung der Mimesis lässt eine Reihe von Dimensionen erkennen:

---

<sup>25</sup> Vgl. Carnap 1998.

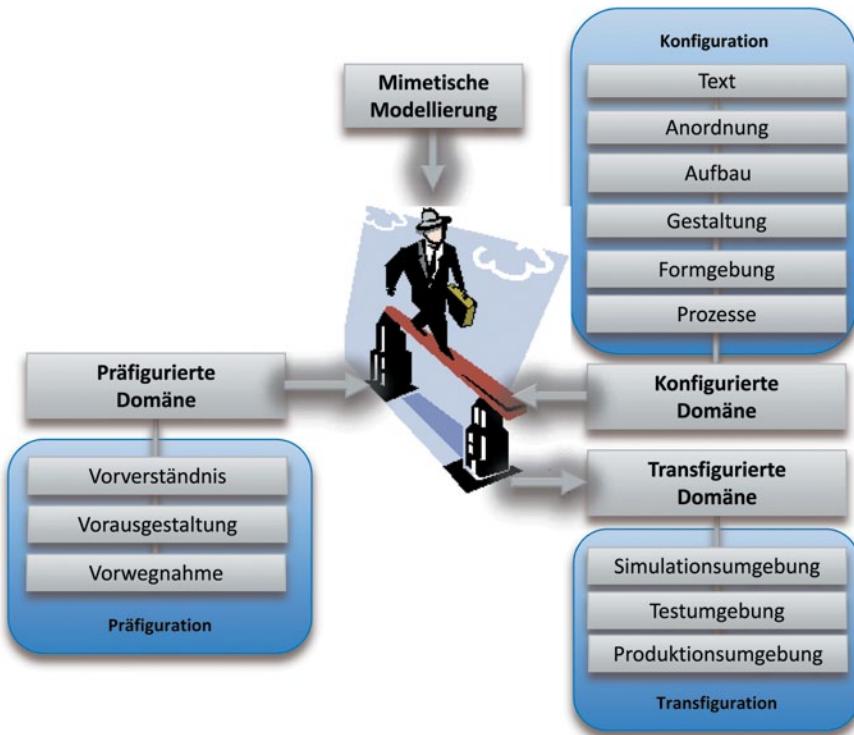
- Mimesis ist an praktisches Wissen gekoppelt, an eine Handlungspraxis gebunden und von der Theoriebildung entkoppelt. Sie ist Produkt menschlicher Praxis und muss immer als Hervorbringung durch Tun, als ein Teil von Praxis, betrachtet werden.
- Mimetische Prozesse bringen Erworbenes, das der Handelnde schon herausgebildet hat, zum Funktionieren. So lässt man beispielsweise beim Nachmachen motorische Schemata, Prozesse erneut ablaufen, die bereits vorhanden sind. Mimesis ist ein freies Funktionierenlassen von Handlungsschemata und auf höheren semiotischen Ebenen, von Techniken des Benennens, Bedeutens und Darstellens.
- Mimetische Prozesse beruhen nicht auf Ähnlichkeiten. Erst wenn eine Bezugnahme von einer mimetischen auf eine andere Welt eingerichtet worden ist, kann es zu einem Vergleich zwischen beiden Welten kommen. Ähnlichkeit ist eher eine Folge von mimetischer Bezugnahme. Imitation ist nur ein Sonderfall von Mimesis. Allgemein gesagt: Ein Gegenstand oder Ereignis kann erst dann als Bild, Abbild oder Reproduktion eines anderen angesehen werden, wenn zwischen beiden eine mimetische Bezugnahme besteht. Diese ist Generator von Bildern, Korrespondenzen, Ähnlichkeiten, Widerspiegelungen, von Abbildungsverhältnissen, die Beziehungen zwischen Ereignissen und Gegenständen auf der sinnlichen Oberfläche der Erscheinungen herstellen.
- In der mimetischen Bezugnahme wird von einer symbolisch erzeugten Welt eine existierende Welt interpretiert. Im mimetischen Handeln ist die Absicht involviert, eine symbolisch erzeugte Welt so zu zeigen, dass sie als eine bestimmte gesehen wird.
- Mimesis ist ein Dazwischen, aufgespannt zwischen einer symbolisch erzeugten und einer anderen Welt. In diesem Verhältnis wird festgelegt, welche Welt als die wahre und welche als modellhafte gilt.
- Die Stärke von Mimesis liegt im Wesentlichen in den Bildern und Modellen, die sie hervorbringt. Bilder und Modelle haben zwar eine materielle Existenz, aber das, was sie darstellen, ist nicht integrativer Teil der empirischen Wirklichkeit, der späteren Systemlösung. Sie lassen eine Tendenz zur Autonomie erkennen. Sie entwickeln sich so zu sinnlichen Ereignissen, zu Simulakren und Simulationen, ohne Referenz auf Wirkliches zu nehmen.<sup>26</sup>
- Mimetische Prozesse finden ihren Ansatzpunkt in der symbolischen Konstituiertheit der empirischen Welt. Überall wo Mimesis herrscht, bestehen fließende Übergänge zwischen Darstellung, Abbildung, Wiedergabe, Reproduktion, aber auch Täuschung, Illusion, Schein.

In anthropologischer Hinsicht gilt Mimesis als eine Fähigkeit, die den Menschen vom Tier unterscheidet. An Mimesis gibt es ein Vergnügen, das bereits Aristoteles als spezifische Eigenschaft des Menschen auszeichnet. Insbesondere im Kindesalter vollziehen sich große Teile der Erziehung mimetisch. Schon im vorsprachlichen Alter werden Wahrnehmungsfähigkeit und Motorik über mimetische Prozesse gebildet.

Der von der Mimesis bewirkte Wandel von der Präfiguration (Vorverständnis) über die Konfiguration (Modell) bis hin zur Transfiguration (Anwendung) artikuliert zwar die Pha-

---

<sup>26</sup> Vgl. Maturana und Varela.



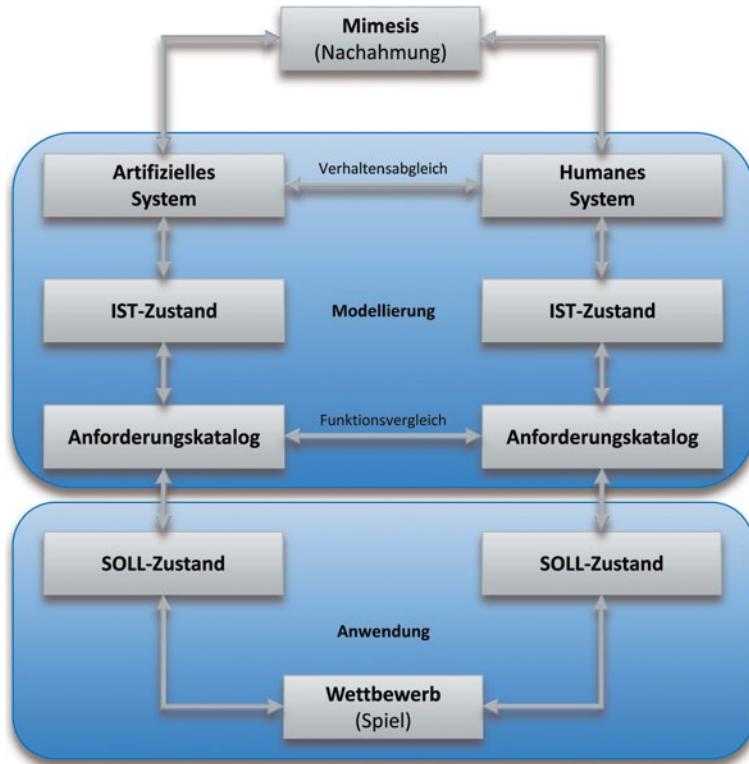
**Abb. 2.9** Mimetische Modellierung

sen eines Prozesses, nicht aber dessen Umschlagstellen. Diese bleiben leer, und das müssen sie wohl auch, weil der Statuswechsel der Figurationen – wenn überhaupt – sich nur als „Spiel“ der Möglichkeiten verstehen lässt, das die unterschiedlichen Konstellationen eines solchen Wechsels durch jeweils andere Formen ausspielen wird (Abb. 2.9).

Es ist in der Tat der Entwickler als Modellierer, der letztlich im Akt des *mimetischen Modellierens* die einzigartige Schaltstelle für den unaufhörlichen Übergang von einer präfigurierten Domäne zu einer transfigurierten Domäne durch die Vermittlung einer konfigurierten Domäne darstellt.

Mimetische Modellierung spielt beispielsweise bei den Lernvorgängen artifizieller Systeme eine Rolle. So wurden bei der Entwicklung von Softwareagenten für den Bereich Roboterfussball im Rahmen der mimetischen Modellierung reale Spieler nachgeahmt. Im Rahmen dieser Mimesis wurden die Anforderungen analysiert, in einem Anforderungskatalog zusammengestellt und der Implementierung zur Verfügung gestellt. Die Validierung der Agenten erfolgte im Rahmen einer Soccer-Simulationsumgebung und dort durch Wettkämpfe. Die Intelligenz dieser Softwareagenten in Form der Software wurde dann in die Echzeirobotersysteme implantiert (Abb. 2.10).<sup>27</sup>

<sup>27</sup> Siehe auch Kitano 1998.



**Abb. 2.10** Mimesis in der Problemdomäne „Roboterfussball“

Im Verlauf der mimetischen Modellierung erfolgt die Entwicklung von mehr oder weniger problemadäquaten Analogien, die kombiniert zu einem *mentalen Modell* führen können. Dabei werden die Analogien mit Hilfe schematischen Wissens entdeckt oder mit eben solchen Schemata konstruiert.

Mentale Modelle sind i. d. R. vereinfachte und auf wesentliche Komponenten reduzierte Abbilder der Realität. Beispiele für mentale Modelle sind die Darwinsche Evolutionstheorie, Klima- und Wettermodelle, die Vorstellung vom Herz als einer Pumpe oder von Hypertext als einem Netzwerk mit physischer Ausdehnung. Solche Mentale Modelle ermöglichen auch die Interpretation bzw. Deutung von Diagrammen, Landkarten und perspektivischen Darstellungen.<sup>28</sup>

Ein Schema ist dabei ein hypothetisches Konstrukt, mit dem die Organisation von Wissen in einer bestimmten Form von Gedächtnis beschrieben wird. Dabei geht man von der Hypothese aus, dass sich Erinnern und Wiedergeben als ein konstruktiver Prozess gestalten und damit ganz im Gegensatz zu einer reinen Widerspiegelung des Wahrge-

<sup>28</sup> Siehe auch Wuketits 1988.

nommenen stehen. Diese Rekonstruktion erfordert aber, dass das Gedächtnismaterial in strukturierter Form bzw. in solchen Einheiten vorliegt, die typische Zusammenhänge enthalten, um entlang derer die Reproduktion aufzubauen zu können. Gelingt dies, dann werden diese Wissenseinheiten als Schemata bezeichnet. In der theoretischen Perspektive der Kognitionspsychologie nehmen Schematheorien inzwischen eine zentrale Position ein. Sie werden die Ausgestaltung des Konzepts des Wissens in einem späteren Abschnitt dieses Buches wesentlich prägen und daher soll an dieser frühen Stelle etwas detaillierter auf diesen Aspekt eingegangen werden.<sup>29</sup> Aufgrund zahlreicher Experimente lassen sich die folgenden Merkmale von Schemata zusammenfassen:

- Schemata sind Strukturen allgemeinen Wissens, die typische Zusammenhänge eines Realitätsbereichs enthalten.
- Sie unterscheiden sich damit von direkten Erinnerungen an konkrete Gegenstände, Ereignisse oder Personen.
- Schemata sind Abstraktionen der konkreten Erfahrung. So besitzen beispielsweise EDV-Anwender ein Schema darüber, was ein Datenbank-Programm ist. Diese Vorstellung muss dabei nicht an einem bestimmten Programm orientiert sein, sondern kann auf abstrakte Gemeinsamkeiten reduziert sein. Schemata müssen aber nicht, wie in diesem Beispiel auf Objekte begrenzt sein, sondern können sich ebenso auf Handlungsabläufe, Situationen oder Personen beziehen.
- Schemata enthalten Leerstellen, die die Relationen zu bestimmten Kategorien angeben. So enthält beispielsweise das Datenbank-Schema Leerstellen wie „Daten“ oder „Suchkriterien“. Diese geben nur an, dass mit einem Datenbankprogramm Daten verwaltet werden können, und dass in dieser Datenmenge gesucht werden kann. Welcher Art diese Daten sind, hängt vom konkreten Beispiel ab und wird je nach Situation ergänzt.
- Die Aktivierung eines Schemas und die damit verbundene Ausfüllung von Leerstellen durch spezifische Werte des Arbeitsgedächtnisses wird Instantiierung genannt. Die Ausfüllung einer Leerstelle bestimmt also, welche konkreten Werte in einer anderen Leerstelle als sinnvoll akzeptiert werden. Genau diese Zusammenhänge, die sich als Einschränkungen von Instantiierungen äußern, stellen den Informationsgehalt des Schemas dar.
- Nicht immer liegen Informationen über die konkrete Ausfüllung aller Leerstellen vor. Ein Schema enthält jedoch Informationen über typische Merkmalsausprägungen, die gleichsam als Voreinstellungen wirken.
- Schemata werden auf unterschiedlichen Abstraktionsebenen gebildet und sind hierarchisch verschachtelt. So kann der EDV-Anwender durchaus ein allgemeineres Schema von Büroanwendungsprogrammen haben, dem sich das spezifischere Datenbank-Schema unterordnen lässt.
- Die Änderung eines hierarchisch niedrigeren Schemas dagegen bedingt nicht zwangsläufig eine Umstrukturierung der übergeordneten Wissensstrukturen. Eine zunehmen-

<sup>29</sup> Siehe auch Barber 1988.

de Differenzierung und Unterscheidbarkeit der Schemata ist daher ein Kennzeichen wachsender Expertise in einer Wissensdomäne. Es zeigt sich, dass allgemeine Schemata schwerer zu ändern sind, als spezifische Schemata, weil die Anpassung eines hierarchisch höheren Schemas Auswirkung auf alle untergeordneten hat. Letzteres würde eine drastische Umorganisation von vorhandenen Wissensstrukturen bedingen.

- Schemata sind nicht statisch, sondern unterliegen selbst der ständigen Veränderung. Einerseits leiten Gedächtnisschemata die Wahrnehmung und Erinnerung (Assimilation), andererseits können neue Erfahrungen bewährte Schemata modifizieren (Akkommodation). Wird beispielsweise ein EDV-Anwender mit einem ihm unbekannten Datenbankprogramm konfrontiert, wird schematisches Wissen über Programme dieser Art aktiviert. Dieses Wissen kann durchaus die Informationen übersteigen, die ihm ursprünglich mitgeteilt wurden. Auch wenn nicht explizit gesagt wurde, dass man mit diesem speziellen Programm Daten speichern, wieder auffinden und ordnen kann, wird der Anwender dies aufgrund seines Datenbank-Schemas erwarten. Das bereits vorhandene schematische Wissen leitet dann den Erwerb neuen Wissens.
- Gedächtnisschemata beeinflussen die unterschiedlichsten kognitiven Leistungen. Sie lenken elementare Wahrnehmungsprozesse, die Repräsentation visueller Wahrnehmungsinhalte, den Erwerb motorischer Fertigkeiten, das Verstehen von Ereignisabfolgen in sozialen Situationen.

Mit Hilfe eines solchen Schemabegriffs kann präzisiert werden, wann im Analogiefall Relationen vom Basisbereich in den Zielbereich übertragen werden. Diese werden nur dann übertragen, wenn ein Schema aktiviert werden kann, aus dem sowohl der Sachverhalt des Zielbereichs, als auch der des Basisbereichs abgeleitet werden kann. Lässt sich dagegen kein gemeinsames Schema aktivieren, wird die Analogiebeziehung nicht entdeckt und die Bildung eines mentalen Modells, welches ja auf dieser Analogie aufbauen soll, schlichtweg verhindert. Wird kein Schema aktiviert, kann dies also zwei Ursachen haben: Entweder es existiert kein relevantes Gedächtnisschema, oder die Hinweisreize der Situation sind nicht geeignet, es zu aktivieren.

Da mentale Modelle auf Analogiebeziehungen aufbauen, müssen sie auf der Grundlage schematischen Wissens konstruiert werden. Mentale Modelle werden also auf der Grundlage schematischen Wissens konstruiert und können im Folgenden als konkrete Instantierungen eines oder mehrerer Schemata aufgefasst werden.

*Modelle des Schlussfolgerns* wurden in der bisherigen Geschichte der kognitiven Psychologie stets in unterschiedlicher Weise behandelt und entwickelt. So vertrat eine Tradition das Prinzip der Struktur, indem sie postulierte, dass die Eigenschaften der logischen Struktur einer Aufgabe die Denkprozesse determinieren, die zu ihrer Lösung erforderlich sind. Die konkreten Denkinhalte sind dabei von sekundärer Bedeutung, die Prozesse sind unabhängig von den Inhalten analysierbar. Diese strukturorientierte Analyse menschlichen Denkens ist häufig mit einer normativen Betrachtungsweise verbunden. Die Versuchung, die formale Analyse von Problemen zum Maßstab menschlichen Denkens und Urteilens zu

machen, wächst mit der vermeintlichen Formalisierbarkeit der kognitiven Anforderungen. So mag es zunächst nahe liegen, aufgrund einer formal-logischen Analyse zu bestimmen, welche Lösungen eines Problems „richtig“ sind und die Analyse menschlichen Denkens auf die Registrierung von Abweichungen von diesem Standard zu reduzieren. Dabei wird jedoch häufig nicht beachtet, dass dieser Standard selbst umstritten oder gar mehrdeutig sein kann. Noch problematischer ist jedoch, dass die Analyse menschlicher Abweichungen von formalen Standards noch keine Erklärung der Denkprozesse darstellt, häufig an einer solchen sogar vorbeiführt. Die bereits behandelten mentalen Modelle als theoretische Konstrukte in Form von Instantiierungen von Gedächtnisschemata und als Repräsentationen konkreter Vorstellungsinhalte, hätten in einer solchen Sichtweise kaum ihre Daseinsberechtigung. Daher wurden dieser strukturorientierten, normativen Sichtweise Ansätze gegenübergestellt, die Denkprozesse eher auf der Grundlage konkreter Denkinhalte analysieren. In diesem Zusammenhang wird betont, dass die Prozesse schlussfolgernden Denkens nicht unabhängig vom individuellen, bereichsspezifischen Wissen analysiert werden können. Insofern ist das mentale Modell in diesem Ansatz das zentrale theoretische Konstrukt, das Alltagswissen in die Lösung von Problemen des logischen Schließens und des Urteilens mit einbezieht. Die Analyse logischen Schließens scheint zunächst nichts mit der Spezifität von Denkinhalten zu tun zu haben, gibt es doch formal-logische Regeln, deren konsequente Anwendung fast immer zu validen Schlüssen führt.

Die Bedingungsaussage „Wenn A, Dann B“ führt bei der Prämisse „Gegeben sei A“ stets zur Konsequenz „B“, ohne dass dabei die Frage, wen oder was A und B repräsentieren, irgendwie von Bedeutung wäre.

Die Validität der Aussage liegt allein in ihrer Struktur und Regelhaftigkeit begründet und ist völlig unabhängig von den Inhalten. Daraus allerdings abzuleiten, daß logische Formalismen gleichzeitig eine Theorie menschlichen logischen Denkens darstellen, führt zu einer falschen Schlussfolgerung. Beispielsweise existieren unterschiedliche Wege, zu logisch validen Aussagen zu gelangen. Das Ableiten einer korrekten Folgerung ist also keine hinreichende Bedingung dafür, dass die Ableitung tatsächlich formal-logischen Regeln folgte. Neben diesen theoretischen Argumenten gibt es jedoch auch empirisch begründete Bedenken. Eine der wichtigsten ist der Nachweis, dass die Lösungswahrscheinlichkeit logischer Probleme auch von inhaltlichen Aspekten der Aufgabe abhängt. Zur Erklärung dieses Nachweises lassen sich zwei Hypothesen heranführen: die Einsichts- und die Gedächtnishypothese.

- Nach der *Einsichtshypothese* erhöht der realistische Kontext der Aufgabe die Einsicht in die Struktur der Regel.
- Die *Gedächtnishypothese* besagt, dass die Aufgabenlösung in besonders gut bekannten Zusammenhängen nicht in einem intrinsischen Schlussfolgerungsprozess abgeleitet, sondern vielmehr aus dem Gedächtnis abgerufen wird.

Die Gedächtnishypothese entstand auf der Grundlage von Experimenten, die zeigten, dass der erleichternde Effekt eines thematischen Kontextes nicht immer *a priori* eintrat. Auch bei solchen Problemstellungen, bei denen keine unmittelbare Erinnerung der Lösung möglich ist, greift das Konzept des Gedächtnisschemas, indem sich aus einem abstrakteren Schema eine spezifische, vorstellbare Situation ableiten lässt, die den geeigneten Kontext zur Lösung des Problems darstellt, ohne dass exakt diese spezifische episodische Erinnerung vorliegen muss. Diese Argumentation stützt aus empirischer Sicht die vertretene Position, dass mentale Modelle konkrete Instantiierungen von Gedächtnisschemata sind. Diese Instantiierungen sind notwendigerweise exemplarische Konkretisierungen abstrakter Zusammenhänge. Der thematische Kontext einer Schlussfolgerungsaufgabe aktiviert somit ein eventuell vorhandenes, passendes Gedächtnisschema, aus dem ein inhaltlich spezifiziertes, mentales Modell der Problemsituation abgeleitet wird. Die Bedeutungszusammenhänge dieses abgeleiteten Modells bestimmen und erleichtern, respektive erschweren die Ableitung einer Lösung. Eine alltägliche Anforderung an die menschliche Informationsverarbeitung betrifft beispielsweise das Urteilen unter Unsicherheit. Gemeint sind Situationen, in denen eine Entscheidung getroffen werden muss, obwohl nicht alle notwendigen Informationen vorliegen, die für eine sichere Entscheidung eigentlich erforderlich wären. Stattdessen muss der Entscheider auf Häufigkeits- beziehungsweise Wahrscheinlichkeitsschätzungen zurückgreifen. Ebenso wie beim logischen Schließen deutet sich jedoch auch hier der Übergang von einer rein formalen und normativen Betrachtungsweise zu einer Konzeption an, die stärker Denkinhalte und aufgabenspezifisches Wissen in den Mittelpunkt stellt. In vielen Experimenten konnte nachgewiesen werden, dass Urteile unter unsicheren Bedingungen nach formalen Maßstäben häufig nicht korrekt sind und bestimmten Verzerrungen unterliegen. Um solche Fehlleistungen zu erklären, vertreten einige Wissenschaftler die Ansicht, dass Menschen beim Urteilen nicht den Regeln der Logik, schon gar nicht optimalen Entscheidungsalgorithmen folgen, sondern dass menschliches Urteilen, besonders im sozialen Bereich, eher durch einen überhöhten Gebrauch intuitiver Strategien, sozusagen „aus dem Bauch heraus“ gekennzeichnet ist. Für diese intuitiven Strategien hat sich der Begriff „Heuristik“ durchgesetzt. Der Einsatz dieser Heuristiken führt zu charakteristischen Urteilsverzerrungen. Insofern wird unter Berücksichtigung dieser Aspekte bei der Entwicklung intelligenter, artifizieller Systeme deren Urteile nicht nur nach den Regeln der Logik, nicht nur optimalen Entscheidungsalgorithmen folgen, sondern auch eine intuitive Strategie miteinbeziehen.

Ein Beispiel für eine solche Urteilsverzerrung ist der sogenannte „overconfidence bias“ (Nisbett und Ross 1980, S. 119 f.). Hiermit wird die Tendenz bezeichnet, ein stärkeres Vertrauen auf die Richtigkeit eigenen Wissens zu äußern als durch die objektive Richtigkeit der Urteile gerechtfertigt wäre.

Warum es jedoch zu einem Overconfidence-Effekt kommt, ist umstritten. Experimente weisen jedoch darauf hin, dass der Mensch bei seiner Entscheidungsfindung zwei unterschiedliche mentale Modelle entwickelt. Zunächst wird der Mensch versuchen, ein sogenanntes *lokales mentales Modell* zu entwickeln, das einen direkten Abruf des zur Entschei-

dung erforderlichen Wissens ermöglichen soll. Die Entscheidung beruht in diesem Falle auf dem Abruf von Wissen aus dem Gedächtnis und elementaren logischen Operationen. Mit Hilfe dieses Wissens kann eine rasche und sichere Entscheidung getroffen werden. Ein solches lokales mentales Modell hat folgende Eigenschaften:

- Ein lokales mentales Modell ist in dem Sinne lokal begrenzt, als es sich ausschließlich auf die zur Entscheidung stehenden Alternativen bezieht.
- Es ist als direkt zu bezeichnen, da es sich allein auf die Zielvariable bezieht und keine weiteren Hinweise probabilistischer Art berücksichtigt.
- Es treten keine Schlussfolgerungen außer elementaren logischen Operationen auf.
- Können entsprechende Informationen aus dem Gedächtnis abgerufen werden, dann wird die Entscheidung als sicher bewertet.

Ist es hingegen nicht möglich, die erforderlichen Informationen direkt abzurufen, dann wird ein *probabilistisches mentales Modell* gebildet. Diesesbettet die Aufgabe in einen breiteren Zusammenhang und bezieht Wahrscheinlichkeitsstrukturen der realen und natürlichen Umwelt mit ein, die als Hinweise auf die Wahrscheinlichkeit der einen oder anderen Alternative dienen. Im Einzelnen unterscheidet sich ein probabilistisches mentales Modell von einem lokalen Modell in allen vier oben genannten Aspekten:

- Ein probabilistisches mentales Modell ist nicht auf die vorgegebenen Alternativen begrenzt (nicht lokal), sondern bezieht Wissen über die Umwelt mit ein, das über die Aufgabenstellung hinausgeht und im Langzeitgedächtnis gespeichert ist. Aus diesem Wissen wird eine Referenzklasse der Objekte oder Ereignisse gebildet, über die geurteilt werden soll.
- Das Modell ist nicht direkt, weil es neben der Zielvariablen ein Geflecht von weiteren Variablen einbezieht, das in Wahrscheinlichkeitsbeziehungen mit der Zielvariablen steht.
- Im Gegensatz zum lokalen, mentalen Modell erfordert das probabilistische Modell Schlussfolgerungsprozesse auch komplexerer Art.
- Das Urteil im Rahmen eines solchen Modells ist durch Unsicherheit variierenden Ausmaßes gekennzeichnet.

Die Bildung einer Referenzklasse dient der Bestimmung von Wahrscheinlichkeitshinweisen. Im Idealfall, d. h., bei differenziertem Wissen über die Referenzklasse, werden die Wahrscheinlichkeitshinweise in der Reihenfolge ihrer Validität auf Anwendbarkeit überprüft. So wird insbesondere unter Entscheidungsdruck die Suche möglicherweise schon nach dem Auffinden des ersten, zwischen den Alternativen differenzierenden Hinweises frühzeitig beendet. Die potenziellen Wahrscheinlichkeitshinweise sind also innerhalb einer Referenzklasse nicht konstant, sondern können einander, je nach Aufgabenstellung, gegenseitig ersetzen. Die Theorie probabilistischer mentaler Modelle sagt also aus, dass beide Typen von Urteilen strukturell auf die gleiche Weise abgeleitet werden, jedoch auf

der Grundlage unterschiedlichen Wissens. Dieses Einbeziehen unterschiedlicher Gedächtnisinhalte (Referenzklasse, Zielvariable und Wahrscheinlichkeitshinweise) erklärt, warum Häufigkeitsschätzungen und Urteilssicherheit für Einzelentscheidungen divergieren können, warum also ein Confidence-Frequency-Effekt überhaupt auftreten kann. So wurde für einen weiteren Bereich menschlichen Denkens gezeigt, dass eine Theorie mentaler Modelle kognitive Leistungen dadurch zu erklären vermag, dass sie konkrete Denkinhalte bestimmt und das Alltagswissen als Erklärungskomponente mit einbezieht. Mentale Modelle sind auch im Bereich des Urteilens unter Unsicherheit als kognitive Konstruktionen zu kennzeichnen, die ein Problem mit Hilfe des Alltagswissens zu einer anschaulichen Vorstellung anreichern. Diese, über die spezifische Aufgabenstellung hinaus angereicherte Vorstellung, ermöglicht die Ableitung einer Lösung, ohne dass diese zwangsläufig mit normativen Kriterien des Schlussfolgerns verträglich sein muss.

Mentale Modelle sind dabei nicht statisch, sondern vielmehr prozesshaft. Dies äußert sich primär in ihrer Eigenschaft zur qualitativen und quantitativen Simulation von Vorgängen der realen Welt, was in einem späteren Abschnitt dieses Buches von zentraler Bedeutung sein wird. Als qualitativ wird diese Form der Simulation bezeichnet, um sie von quantitativen (mathematischen) Simulationen abzugrenzen. Dabei wird angenommen, dass keine exakte mentale „Verrechnung“ kontinuierlicher Größen vorgenommen wird. Umgangssprachlich kommt dem die „Betrachtung vor dem geistigen Auge“ ziemlich nahe. Mentale Modelle entwickeln sich graduell in der Auseinandersetzung mit dem zu modellierenden Realitätsausschnitt. Nicht in allen Stadien dieser Entwicklung ist die gleiche Simulationsfähigkeit gegeben. In vielen Modelltheorien wird zwischen zwei Zustandsformen mentaler Modelle unterschieden, die je nach theoretischem Hintergrund einmal als *Wahrnehmungsmodelle* und ein anderes Mal als *Kausalmodelle* bezeichnet werden. Zum einen handelt es sich um Modelle, die „wahrnehmungsnah“ sind, sogenannte *Perzeptionsmodelle*. Diese bilden Teile der Außenwelt als Folgen von Wahrnehmungsprozessen ab. Auf die, der unmittelbaren Wahrnehmung nahe stehenden Modelle, operieren dann „höhere“ kognitive Prozesse, wie beispielsweise Inferenzen oder Induktions- und Analogieschlüsse. Im Zusammenhang mit abstraktem, schematischem Wissen bewirken diese Prozesse Umstrukturierungen, indem neue Vorstellungen abgeleitet werden. Dieses Ableiten neuer Vorstellungen aufgrund der Interaktion von Wahrnehmungsinhalten und abstrakten Wissensbeständen kennzeichnet die zweite Erscheinungsform mentaler Modelle. Mit dem Übergang von einem wahrnehmungsnahen Modell zum Kausalmodell nimmt auch die Komplexität der kognitiv simulierbaren Vorgänge zu.

„Verstehen“ ist als Erkenntnisobjekt innerhalb der Psychologie allerdings mehrdeutig. So werden *Modelle des Verstehens* häufig mit Konnotationen anderer Disziplinen, wie beispielsweise der Erkenntnisphilosophie, der Semiotik und vor allem der Linguistik versehen. Bei der Vielfalt der Begriffsverwendungen und der offensichtlichen Komplexität mag es verwundern, dass der Prozess des Verstehens in den letzten Jahren auch sehr einfache Charakterisierungen erfahren hat. Verstehen heißt, ein Sinnverständnis für eine neue Erfahrung aus ihrer Ähnlichkeit mit den Elementen aus vertrauter Erfahrung zu entwickeln. Verstehen wird dabei als ein Prozess aufgefasst, in dem Kongruenz zwischen neuen

Informationen und bereits organisiertem Wissen hergestellt wird. Diese Kongruenz wird in mentalen Modellen hergestellt. Diese bauen auf abstrakten Gedächtnisschemata auf und werden solange verändert, bis die neuen Informationen widerspruchsfrei integriert sind. Die widerspruchsfreie Integration entspricht dann dem Erlebnis des Verstehens. Ist diese Integration hingegen nicht möglich, ist eine Veränderung der das Modell bestimmenden Gedächtnisschemata erforderlich. Die Akkommodation eines Schemas erlaubt eine Modifizierung des Models und damit die erneute Chance der Integration der neuen Information. Gemäß dieser Sichtweise wird auch in diesem Buch unter Verstehen ein konstruktiver Prozess verstanden, im Rahmen dessen die Bedeutung wahrgenommener Ereignisse oder Mitteilungen diesem nicht passiv entnommen, sondern eher aktiv in Bezug zu bereits existierendem Wissen sukzessive aufgebaut wird.

Das mentale Modell als Mittel und Produkt des Verstehens geht über den unmittelbaren Bedeutungsgehalt des Wahrgenommenen oder des Mitgeteilten hinaus. Von Verstehen soll im Folgenden dann die Rede sein, wenn zu einem Zielbereich (Bilder, Text, Gerät, Ereignis usw.) eine mentale modellhafte Repräsentation entwickelt wird, die über die lediglich registrierende Perzeption hinausgeht und das mentale Modell evaluativ auf Konsistenz mit dem wahrgenommenen Zielbereich und auf interne Kohärenz mit Wissensbeständen überprüft und notfalls revidiert wird. Allgemein können Verknüpfungen neuer Sinneinheiten durch existierende Wissensstrukturen wie folgt entstehen:

- Wenn eine neue Sinneinheit keinen Bezug zum gegenwärtigen mentalen Modell eines Umweltausschnitts aufweist, wird ein neues Modell konstruiert.
- Wenn eine neue Sinneinheit in wenigstens einem Aspekt mit dem gegenwärtigen Modell übereinstimmt, wird die gesamte, darüber hinausgehende Information dieser Einheit, dem mentalen Modell zugefügt.
- Wenn eine neue Sinneinheit Überschneidungen mit zwei bis dahin getrennten mentalen Modellen aufweist, werden beide Modelle zusammengeführt.
- Wenn eine neue Sinneinheit vollständig in das mentale Modell integriert ist, erfolgt eine Konsistenzprüfung.

Diese Konsistenzprüfung kann unterschiedliche Ausgänge haben. So kann die neue Sinneinheit im Rahmen des bestehenden Modells wahr, falsch oder nicht beurteilbar sein. In jedem dieser drei Fälle wird eine andere Prozedur erforderlich:

- Sollte es mangels Informationen nicht möglich sein, die Verträglichkeit zu überprüfen, wird die gesamte Sinneinheit nach Maßgabe der Widerspruchsfreiheit in das Modell integriert.
- Sollte sich die neue Sinneinheit als falsch im Sinne des Modells erweisen, werden bestehende Wissensbestände (Schemata) verändert, so dass ein modifiziertes Modell resultiert oder die neue Sinneinheit wird als inkompatibel mit dem bestehenden Modell zurückgewiesen.

- Sollte sich die neue Sinneinheit als wahr im Sinne des Modells erweisen, wird versucht, ein alternatives Modell zu konstruieren, das mit den bestehenden Schemata verträglich ist, aber in Widerspruch zu der neuen Sinneinheit tritt. Gibt es ein solches Modell, wird angenommen, dass die neue Sinneinheit nur möglicherweise wahr ist. Kann ein solches Modell nicht konstruiert werden, wird angenommen, dass die neue Sinneinheit notwendigerweise wahr ist.

Es gilt dabei zu beachten, dass die Überprüfung der internen Konsistenz keinesfalls die objektive Richtigkeit des mentalen Modells im Sinne einer Kongruenz mit dem konzeptuellen Modell sicherstellt. Verstehen bedeutet in diesem Zusammenhang das Erlebnis einer stimmigen Einsicht in die Zusammenhänge, unbeschadet der Möglichkeit, dass sich diese Zusammenhänge als objektiv nicht zutreffend herausstellen können.

Eine Verknüpfung zweier Gegebenheiten G1 und G2 ist dann einsichtig, wenn G1 entnommen werden kann, so dass, wenn G1, dann auch G2 und genau dann G2 gilt. Die Evidenz einer solchen Verknüpfung kann dabei auf zwei verschiedene Ursachen zurückzuführen sein. Zum einen ist G2 ein Teil des Ganzen und kann direkt aus der Gegebenheit des Ganzen abgelesen werden. Zum anderen ist eine Begebenheit die Konsequenz aus den übrigen Momenten des Ganzen.

Diese Ursachen lassen zwei weitere Merkmale mentaler Modelle erkennen: die Inhaltsgebundenheit und die Bildhaftigkeit. So greifen die mentalen Modelle zum einen auf Transformationen von Vorstellungsinhalten zurück und können somit bildhafte Vorstellungen erzeugen. Die Manipulation dieser bildlichen Vorstellungen kann Ausdruck einer kognitiven Simulation sein. Einerseits haben diese bildlichen Vorstellungen ihren Ursprung in der Wahrnehmung, andererseits sind sie keine originalgetreuen Reproduktionen früherer Wahrnehmungen. Vorstellungen werden nicht nur anders erlebt als Wahrnehmungen, sie sind auch unwillkürlichen und willkürlichen Veränderungen zugänglich, deren Realitätsbezug variieren kann. Wahrnehmungsprozesse benötigen dabei externe Reize, währenddessen Vorstellungen auf den Erinnerungen an diese Wahrnehmungen basieren. Trotz dieser unterschiedlichen Basis besteht eine funktionale Ähnlichkeit zwischen Wahrnehmung und Vorstellung. Außerdem zeigen Experimente, dass neben dieser funktionalen Ähnlichkeit auch strukturell analoge Beziehungen zwischen Wahrnehmungen und Vorstellungen zu bestehen scheinen. Dabei wird davon ausgegangen, dass das Wahrnehmungsobjekt ähnliche Eigenschaften hat, wie der entsprechende Vorstellungsinhalt. Vorstellungsbilder sind somit eine Art Sichtweise auf mentale Modelle, indem sie die wahrnehmbaren Eigenschaften der Objekte des Modells enthalten. Allerdings dürfen die mentalen Modelle nicht mit den Vorstellungsbildern gleichgesetzt werden, da mentale Modelle auch abstrakte Relationen enthalten, die in einer momentanen Sicht auf das Modell nicht unbedingt wahrnehmbar sein müssen. Dies betrifft vor allem solche Eigenschaften, die dem Objekt inhärent sind und die erst durch eine mentale Simulation „sichtbar“ werden.

So kann man in der bildlichen Vorstellung eines Spielwürfels höchstens drei Seiten sehen. Das mentale Modell des Würfels enthält mehr Wissen über dieses Objekt. Dies ermöglicht dann eine kognitive Simulation, indem der Würfel in der Vorstellung so gedreht wird, dass auch die anderen Seiten sichtbar werden. Durch diese unterschiedlichen Perspektiven auf den einen Gegenstand des Interesses lässt sich erreichen, dass alle möglichen Vorstellungsbilder des Würfels kompatibel miteinander sind.

Es wird davon ausgegangen, daß Regelungstätigkeiten eines Systems eines inneren Modells des zu regelnden Systems bedürfen. Dieses *innere Modell* umfasst dabei folgende Bereiche:

- Wissen über das zu regelnde System.
- Wissen über Störbedingungen, die auf das System einwirken können.
- Wissen über die auszuführenden Aufgaben (Sollwerte, Randbedingungen, Signalbedeutungen etc.).

Dieses innere Modell stellt dabei neben Optimierungsmöglichkeiten auch eine unabdingbare Voraussetzung für eine effektive Regelungstätigkeit dar. Dieses innere Modell reguliert sozusagen die Handlung. Als Inhalte solcher Modelle kommen folgende Bereiche in Frage:

- *Handlungsziele* beziehungsweise deren kognitive Antizipation: Diese repräsentieren die „Sollwerte“ der Handlungssteuerung auf unterschiedlichen Detailliertheitsebenen.
- *Ausführungsbedingungen*: Diese umfassen Wissen darüber, unter welchen Randbedingungen die Handlungen auszuführen sind, mit denen die vorgenommenen Ziele erreicht werden können.
- Wissen um *Transformationsmaßnahmen*: Wissen, mit dem der Ist- in den Sollzustand überführt werden kann. Hierzu zählen nicht nur die Arbeitsmittel, sondern auch Pläne zu ihrem Einsatz und die hierzu erforderlichen Operationen des handelnden Systems selbst.

Mentale Modelle stellen somit die Grundlage sowohl der Handlungsplanung als auch der Handlungsausführung dar. Die handlungsleitende Wirksamkeit eines *operativen Modells* äußert sich vor allem darin, daß die Handlung, unabhängig von äußereren Informationen, wie beispielsweise Handlungsanweisungen, Entscheidungsregeln, etc., erfolgt. Die Handlungskontrolle geht von solchen äußereren Informationen auf das innere Modell des Arbeitsgegenstandes, des Handlungsablaufs und der Zielstruktur über. Das innere Modell erspart so die aktuelle Informationsaufnahme und Informationsrekodierung während der Handlungsausführung, weil die erforderlichen Informationen zur Handlungssteuerung im Gedächtnis bereits vorliegen. Aus dieser spezifischen Funktion der Handlungssteuerung ergibt sich allerdings ein Unterschied zu den bisher diskutierten Merkmalen mentaler Modelle: Einzelne Aktivitäten sind nur zu einem gewissen Teil problemlösender Art. Ein

größerer Teil besteht aus mehr oder weniger geübten, wiederkehrenden Aktivitäten im Rahmen der Interaktion, die zumindest teilweise automatisiert werden können.

Automatisierte Aktivitäten bedürfen beispielsweise nicht mehr der vollständigen Planung oder einer dauernden, bewussten Steuerung. Dies gilt sowohl für Prozesskontrolltätigkeiten als auch für Fertigungs- oder Verwaltungsarbeiten.

Innere Modelle zur Steuerung solcher Aktivitäten zeichnen sich durch einen hohen Grad an Beständigkeit aus. Ihre Entwicklung bedeutet die Konservierung von Invarianten des Handlungsprozesses, die mit zunehmender Routine in Form von Schemata gespeichert werden können. So zeigen die inneren Modelle verallgemeinerte, schematische Züge, indem sie dazu tendieren, Klassen von Merkmalen und Relationen zu repräsentieren. In diesem Sinne sind mentale Modelle als Adaptionen innerer Modelle an unterschiedliche kognitive Anforderungen zu verstehen.

Im Rahmen der Interoperationstheorie, die in diesem Buch noch entwickelt wird, bedeutet Aktivität immer ein „konkretes Handeln“ in Bezug auf ein externes Objekt, meistens unter Zuhilfenahme von Werkzeugen und Arbeitsmitteln. Gemäß dieser Auffassung umfasst das System der mentalen Modelle nicht nur die Repräsentationen eigener Aktivitäten, sondern auch die Relation zwischen eigenem Tun und dem Zustand des Arbeitsmittels beziehungsweise des jeweiligen Arbeitsobjekts. Aus der Erfahrung dieser Relation werden Regeln über den Zusammenhang von Ausführungsbedingungen, Zielen und Aktivitäten abgeleitet, die eine Vorhersage von Systemzuständen erlauben. Die vorstellungsmäßige Vorwegnahme von Ereignissen ermöglicht sowohl effektives Eingreifen und damit die Vermeidung von Störfällen, da sie auch ihre nachträgliche Diagnose erleichtern.

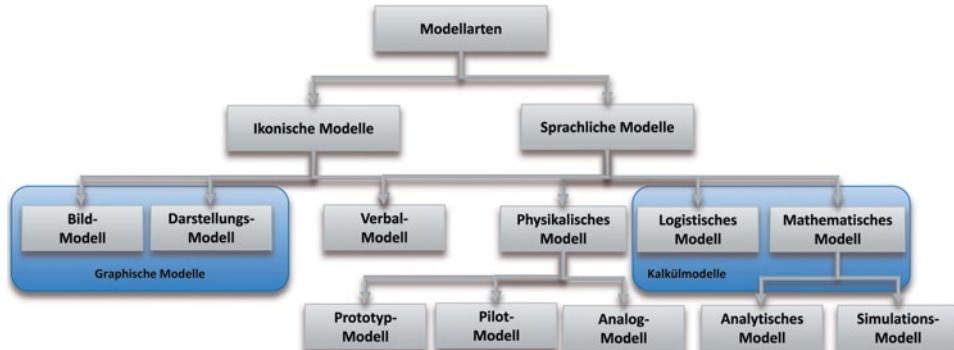
Bevor ein für dieses Buch spezieller Modellbegriff entwickelt wird, sollen noch einige allgemeine Anmerkungen zu den unterschiedlichen Modellvarianten diesen Abschnitt abschließen. Modelle dienen im Allgemeinen zur Beschreibung der Eigenschaften und der Struktur eines Systems. Insofern kann man auch davon sprechen, dass man sich mit Hilfe solcher Modelle „ein Bild macht“ von dem später zu realisierenden System, seinem Verhalten und gegebenenfalls seinen Auswirkungen auf die ihn umgebende Realität. Diese Bilder sind daher nie ein absolut vollständiges Abbild eines Systems. Je nachdem, welche Intension hinter der Modellierung steht, lassen sich verschiedenartige Modelle mit unterschiedlichen Eigenschaften entwickeln.<sup>30</sup> Dabei unterscheidet man grob zwischen physikalischen und mathematischen Modellen. Physikalische Modelle sind stets gegenständlich und maßstäblich, mathematische Modelle dagegen abstrakt und dienen eher einer formalen Beschreibung der Systemeigenschaften.

Bei den *physikalischen Modellen* unterscheidet man folgende Arten:

- Prototypmodell
- Pilotmodell
- Ähnlichkeitsmodell

---

<sup>30</sup> Siehe auch Rumbaugh et al. 1991.



**Abb. 2.11** Modellarten

Das *Prototypmodell* ist 1:1-maßstäblich und besitzt höchste qualitative und quantitative Ähnlichkeit mit dem späteren Original. So wird im Falle der Robotik meistens ein solcher Prototyp vor der Serienherstellung eines Robotersystems erstellt. Dies ist ein weitestgehend mit den Serieneigenschaften ausgestatteter Originalaufbau, an dem alle Eigenschaften des späteren Originals direkt und konkret getestet werden können. Nachteil eines solchen Prototypmodells ist, dass seine Herstellung unter Umständen aufwändig und teuer ist und nur geringe Flexibilität bei erforderlichen Änderungen besitzt.

Das *Pilotmodell* ist häufig maßstäblich unterschiedlich zum Original beispielsweise 1:10. Es bildet daher nur wesentliche Eigenschaften genau ab. Seine Herstellung ist in der Regel mit reduziertem Aufwand möglich und es lässt sich einfacher ändern. Häufig ist die Aufgabe eines solchen Modells nur die Visualisierung eines späteren Komplettsystems, um beispielsweise das Design oder aber auch die generelle Machbarkeit frühzeitig beurteilen zu können.

Der geringste Aufwand zur Herstellung eines physikalischen Modells entsteht bei der Erstellung eines sogenannten *Ähnlichkeitsmodells*. Es werden hier nur noch Teile des Systems hergestellt, an denen man ein eingeschränktes Spektrum von Untersuchungen vornehmen kann. So könnten unter Berücksichtigung der Ähnlichkeitsverhältnisse an einem solchen Ähnlichkeitsmodell beispielsweise Untersuchungen im Wind- oder Wasserkanal über das Strömungsverhalten gemacht werden, d. h. es handelt sich um Untersuchungen während des Entwicklungsprozesses (Abb. 2.11).

Deutlich flexibler und mit geringerem materiellem Aufwand herstellbar sind abstrakte *mathematische Modelle*. Für ein *analytisches Modell* muss man die analytischen Zusammenhänge zwischen den Attributen eines Systems bestimmen, was einen Satz von Gleichungen liefert, die eine geschlossene, analytische Lösung besitzen. Diese Modellierung ist in der Regel aber ohne Rechnereinsatz nur für sehr einfache Systeme möglich. Komplexere Systeme kann man mit Hilfe eines *Simulationsmodells* behandeln. Dieses Modell wird auf einem Computer erstellt und mit Hilfe numerischer Rechenverfahren gelöst.

Allen diesen unterschiedlichen Modellen ist gemeinsam, dass sie nicht die Realität in allen Details abbilden, sondern die wahren Verhältnisse stark vereinfachen, um bestimmte Problem- bzw. Fragestellungen einfach transparenter zu machen oder aber den Aufwand für die Modellbeschreibung in einem der Sache adäquatem Umfang zu halten. Diese Gemeinsamkeit stört allerdings nicht, denn auch mit einer hinreichenden Genauigkeit lassen sich bestimmte Kenngrößen ermitteln und das reale System modellieren. Größere Fehler sind später nur dann zu erwarten, wenn die vernachlässigten Kräfte oder die vereinfachten Annahmen durch Extremsituationen in solchen Bereichen liegen, in denen sie nicht mehr so ohne weiteres vernachlässigt werden können. Letzteres wird gerade in der Phase der Validierung in Form von Stresstests noch eine Rolle spielen.

Für die Erstellung des Modells eines Robotersystems gelten drei allgemeine Anforderungen:

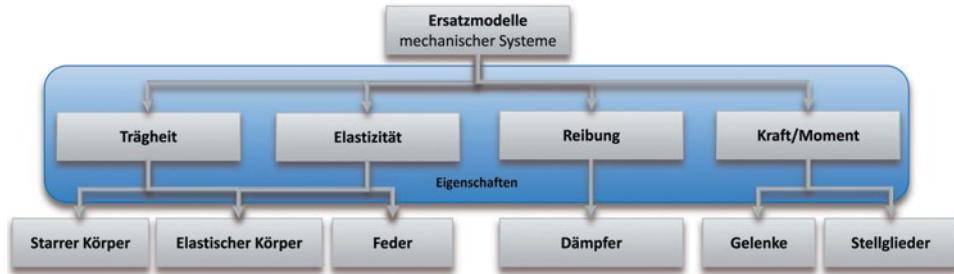
- Die einzelnen Elemente des Modells müssen klar definiert, eindeutig beschreibbar und in sich widerspruchsfrei sein (physikalische Transparenz).
- Die Folgerungen über das Verhalten, die man aus den Verknüpfungen der Modellelemente zu einem Gesamtmodell ziehen kann, müssen im Rahmen des Modellzwecks (Gültigkeitsbereich) dem realen Systemverhalten entsprechen (Modellgültigkeit).
- Gibt es verschiedene Möglichkeiten zur Darstellung des Systems, die alle den ersten beiden Forderungen genügen, so sollte man die einfachst mögliche auswählen (Effizienz).

Für die Herleitung eines einfachen, effizienten und gültigen Modells gibt es zwar keine in allgemeingültige Regeln zementierte Vorgehensweise, jedoch eine Sammlung von bewährten Erfahrungen (Best Practices), die einem separaten Kapitel zu einem Vorgehensmodell führen werden.

Das Modell eines neuronalen Systems beispielsweise, das neben dem mechanischen, mathematischen auch den kognitiven Aspekt beinhaltet, das alle nur denkbaren Bewegungs- und Entscheidungsmöglichkeiten berücksichtigt, ist zwar physikalisch und kognitiv richtig, aber für die praktische Anwendung unübersichtlich, unhandlich und verliert für die meisten Fälle die Überschaubarkeit.

Die Kunst bei der Modellbildung besteht daher darin, das Modell so einfach wie möglich zu gestalten, um es mit technisch und wirtschaftlich vertretbarem Aufwand untersuchen zu können. Dabei dürfen aber dennoch keine unzulässigen, das Systemverhalten zu stark verfälschenden Annahmen getroffen werden. Es gilt also bei der Entwicklung von Modellen die Maxime von Albert Einstein zu beherzigen: „Alles sollte so einfach wie möglich gemacht werden, aber nicht einfacher“.

Die Beherzigung dieser Maxime wirkt sich beispielweise bei Entwicklung auf die einzelnen Teilmodelle eines komplexen Robotersystems aus. Betrachtet man die mechanische Struktur des Bewegungssystems, so werden bei der Modellbildung meist elektrische Komponenten aus dem Bereich der Aktoren (beispielsweise Elektromotoren) und der Sensoren einbezogen. Die



**Abb. 2.12** Ersatzmodelle mechanischer Systeme

mechanischen Eigenschaften sind im Wesentlichen durch Trägheit, Elastizität und Reibungsvorgänge gekennzeichnet, die den durch äußere Kräfte und Stellkräfte bzw. Momente hervorgerufenen Bewegungszustand beeinflussen. Diese Merkmale der Bauelemente des Systems werden durch idealisierte mechanische Modelle repräsentiert. Dabei werden Körpermodelle in der Regel als mit Masse und Trägheit behaftet angenommen, jedoch solche Elemente wie Federn und Dämpfer als masse- und trägeheitslos. Sich translatorisch oder rotatorisch zueinander bewegende Körper sind durch Gelenke oder Führungen miteinander verbunden, wobei die Einschränkung des Freiheitsgrades der Bewegung von Körpern Reaktionskräfte und -momente zur Folge hat.

Zur Entwicklung des Modells eines mechanischen Systems versucht man, die einzelnen realen Objekte durch diese Ersatzmodelle zu beschreiben und damit eine kinematische Struktur aufzubauen. Dabei muss man immer beachten, welche Fragen man mit dem Modell beantworten will. Ein kompliziertes technisches System, wie beispielsweise ein Transportroboter auf unebenem Gelände, hat nicht einfach eine bestimmte Zahl von Freiheitsgraden, sondern die Anzahl der Freiheitsgrade, die man notwendigerweise einführen muss, hängt davon ab, welche Informationen man benötigt. Die folgende Abbildung zeigt eine Zusammenstellung wichtiger verwendeter Ersatzmodelle und die ihnen zugeordneten Eigenschaften (Abb. 2.12).

Trotz dieses Rückgriffes auf Ersatzmodelle können dynamische Systeme den Modellierer bei seinem Versuch der Erstellung eines gültigen und effizienten Modells mit verschiedenartigen Problemen konfrontieren. Dies liegt natürlich an der physikalischen Beschaffenheit und der Komplexität solcher Systeme. Verschiedenartige physikalische Systeme haben bestimmte globale Eigenschaften, die sie für die Modellbildung entweder leicht handhabbar machen oder bewirken, dass eine mathematische Beschreibung mit großen Schwierigkeiten verbunden ist.

Bei den meisten dynamischen Systemen kann man davon ausgehen, dass die in den Systemen auftretenden physikalischen Größen bzw. die sie darstellenden Signale nicht regellos schwanken. Vielmehr können sie analytisch beschrieben werden, weshalb man von determinierten Größen und Systemen spricht. Dabei kann die analytische Beschreibung entweder durch Differentialgleichungen erfolgen und man spricht dann von sogenannten *parametrischen Modellen*, oder der Zusammenhang zwischen Ein- und Ausgangsgrößen des Systems liegt in Form von Wertetabellen oder Kurvenverläufen vor, wobei man dann

allerdings von *nichtparametrischen Modellen* spricht. Im Gegensatz zu den determinierten Systemen stehen stochastische Systeme, deren Zustände nicht durch analytische Beschreibungen vorherbestimmt werden können.

Beispiele für stochastische Größen sind Rauschsignale, hervorgerufen durch thermische Bewegungen der Moleküle. Solche Signale können höchstens mit den Methoden der Statistik behandelt werden.

Ferner kann man bei vielen Systemen davon ausgehen, dass die Systemgrößen innerhalb gewisser Grenzen jeden beliebigen Wert annehmen können. Man spricht dann von kontinuierlichen Systemen.

Determinierte, kontinuierliche Systeme können durch partielle Differentialgleichungen mathematisch beschrieben werden. Vielfach kann man diese durch örtliche Diskreditierung zu gewöhnlichen Differentialgleichungen vereinfachen. Solche Systeme, die als unabhängige Variable nur noch von der Zeit abhängen, heißen Systeme mit konzentrierten Parametern, im Gegensatz zu Systemen mit verteilten Parametern, die durch partielle Differentialgleichungen beschrieben werden.

Eine weitere wichtige Systemeigenschaft ist die Zeitvarianz. Ändert ein System oder eine Größe ihre Übertragungseigenschaften nicht in Abhängigkeit von der Zeit, so spricht man von zeitinvarianten Systemen und das *zeitinvariante Systemmodell* enthält keine von der Zeit abhängigen Parameter. Demgegenüber kann ein *zeitvariantes Systemmodell* unterschiedliches Verhalten aufweisen, je nachdem, zu welchem Zeitpunkt man das System betrachtet.

So sind beispielweise Flugzeuge meist zeitvariante Systeme, da sich ihre Masse während des Fluges durch den Verbrauch des Treibstoffs stark ändert.

Bei zeitvarianten Systemen spielt einerseits die Geschwindigkeit, mit der sich die einzelnen Systemparameter ändern, eine Rolle und andererseits auch die Art des Änderungseffektes. So kann sich ein Parameter durch Driften (langsame Veränderung) oder auch sprungförmig ändern. Zu den am schwersten zugänglichen Systemen gehören solche mit verteilten Parametern. Bei diesen lässt sich die partielle Differentialgleichung nicht in eine gewöhnliche Gleichung überführen, da alle oder einzelne Zustandsgrößen des Systems nicht nur von der Zeit, sondern zusätzlich von anderen Größen, wie etwa Ortskoordinaten abhängig sind.

Zu dieser Klasse gehören im Wesentlichen Strömungs- und Wärmevorgänge.

Zieht man bei der Modellierung den Aspekt des geometrischen und zeitlichen Ablaufes von Bewegungen (Kinematik) hinzu, ergeben sich weitere, im Regelfall sehr komplexe

Modelle.<sup>31</sup> Ein komplexes Modell ist das einer bewegten Masse in Form eines starren Körpers. Ein solcher starrer Körper besteht als Modell aus einer Vielzahl infinitesimal kleiner Massenelemente, deren relative Lage im Körper zueinander stets erhalten bleibt. Für viele massive Bauelemente in mechanischen Systemen, wie Bolzen, Balken, Platten u.ä. geben sie ein recht brauchbares Modell ab. Voraussetzung für den starren Körper ist allerdings, dass er sich unter äußeren und inneren Kräften nicht verformt, d. h. die Elastizität wird vernachlässigt.

Lässt man im Modell hingegen Verformungen des Körpers durch Kräfte zu, spricht man von einem elastischen Körper. Diese Verformungen sind in den meisten Fällen eher klein. Geht die Verformung im unbelasteten Zustand vollständig zurück, so spricht man von elastischer Verformung, andernfalls von plastischer Verformung. Im Gesamtmodell eines Robotersystems sind solche Körper entweder starr oder über Koppelemente miteinander verbunden, die einem bestimmten Kraftgesetz gehorchen und eingeprägte Kräfte erzeugen. Dies sind Federn und Dämpfer sowie Stellantriebe. In der Regel werden diese Koppelemente als masselos angesehen.

Ein Robotersystem besteht in der Regel also aus mehr als einem bewegten Körper. Typische Systeme, wie beispielsweise Industrieroboter, bestehen aus mehreren relativ steifen Gliedern, die über Dreh- und Schubachsen miteinander verbunden sind. Solche Systeme kann man sehr gut durch das Konzept des *Mehrkörpermodellsystems* modellieren. Diese Art der Modellbildung wurde in den 70er Jahren eingeführt, als die ersten Programmsysteme für die numerische Berechnung auf Digitalrechnern zur Verfügung standen. Im einfachsten Fall besteht ein solches Mehrkörpersystem aus einer offenen Kette starrer Körper, die durch starre Gelenke miteinander verbunden sind. Diese Starrkörper repräsentieren die Eigenschaften der Masse und der Trägheit. In einer modifizierten Form können diese Systeme auch elastische Körper beinhalten und den Gelenken elastische Eigenschaften zugeordnet werden. Bei vielen Systemen, wie beispielsweise den Industrierobotern, tritt der größte Anteil an Elastizität in den Gelenken und Antrieben auf und wird durch Feder- und Dämpferelemente modelliert. In Anlehnung an die Methode der Finiten-Elemente, bei der die mechanische Struktur eines Systems aus einer Vielzahl von Einzelementen, wie Balken, Scheiben, Platten und Schalenelementen zusammengesetzt wird, spricht man bei dieser Methode auch von der Finite-Segment-Methode.

Steht bei der Modellierung von Körpern oder Gelenken das elastische Verhalten im Vordergrund, lässt sich das Verformungsproblem eines Bauteils mit den Grundgleichungen der Elastomechanik beschreiben. Elastische Körper einfacher Geometrie, deren Masse- und Elastizitätsverteilung exakt beschrieben werden können, können auch durch kontinuierliche Systeme modelliert werden. Die mathematische Formulierung des Modells führt auf partielle Differentialgleichungen, die nur für sehr einfache Geometrien exakt gelöst werden können. Um auch für komplexe Bauteile Beschreibungen des Verformungsverhaltens zu ermöglichen greift man auf Näherungsverfahren zurück. Bei solchen Nähe-

<sup>31</sup> Vgl. Rieseier 1992.

rungsverfahren ist zu unterscheiden, ob die entsprechende Lösung das Verhalten des Bauteils wie bei der exakten Lösung kontinuierlich beschreibt, oder lediglich an vorgegebenen diskreten Punkten.

Ein Näherungsverfahren mit kontinuierlicher Beschreibung ist beispielsweise die Methode nach Ritz, die durch funktionsanalytische Verfahren eine geschlossene Näherung ermöglicht. Diese Näherung kann in einfachen Fällen wieder mit der exakten Lösung übereinstimmen.

Andere Näherungsmethoden liefern diskrete Lösungen, d. h. Lösungen, die nur an bestimmten Punkten des entsprechenden Bauteils gelten. Die beiden bekanntesten Verfahren sind das Differenzenverfahren und die bereits erwähnte Finite-Elemente-Methode. Beim Differenzenverfahren werden die in den Gleichungen und Randbedingungen vorkommenden Differentialquotienten durch Differenzenquotienten ersetzt. Das dadurch entstehende System algebraischer Gleichungen wird mit den aus der numerischen Mathematik bekannten Verfahren gelöst. Die Finite-Element-Methode als Standardmethode zur Modellierung elastischer Strukturelemente basiert auf der Idee, dass das zu berechnende Kontinuum in eine große Anzahl einfacher abgegrenzte, endlich große Elemente zerlegt werden kann, die dann der Berechnung auf dem Computer leicht zugänglich sind. Aus den Lösungen für die Einzelelemente wird dann unter Berücksichtigung von Kontinuitäts- und Gleichgewichtsbedingungen eine Lösung für das Gesamtsystem konstruiert. Die Bedingungen werden dabei nur an einer endlichen Zahl von Punkten, den sogenannten Knotenpunkten, formuliert. Sie führen auf ein Gleichungssystem, dessen Lösung im Allgemeinen eine Näherungslösung für das behandelte System ist. Diese Methodik, die Ende der 60er Jahre aufkam, war die erste größere numerische Simulationsmethode und war quasi in der Praxis an das Aufkommen leistungsfähiger Computer gebunden. Heute ist die Methode sicher das am meisten benutzte Verfahren, um naturwissenschaftliche und technische Probleme mit Hilfe des Computers zu lösen. Die Modellierung verläuft dabei in drei Stufen:

- Zerlegung der Struktur in möglichst gleichartige Elemente (Balken, Scheiben, Platten, Schalen), so dass durch die Verschiebungsfreiheitsgrade in den Knoten alle geometrischen Rand- und Übergangsbedingungen erfüllt werden können.
- Untersuchung der einzelnen Elemente, wobei das Verhalten ausschließlich durch die Verschiebung in den Knotenpunkten beschrieben wird.
- Zusammenfassung der Elemente zum Gesamtsystem.

Technische Systeme beinhalten oftmals auch elektrische Systeme. Bei der Modellierung solcher Systeme entspricht die Darstellung exakt den körperlich vorhandenen Bauteilen und ihren Verbindungen. Die Bauteile selbst können durch einfache bekannte elektrische Grundgleichungen beschrieben werden. Man darf sich aber nicht darüber täuschen lassen, dass auch diese Darstellung nicht einfach ein Lageplan (Schaltplan) der elektrischen Komponenten ist, sondern ein Modell des realen Systems. Dieses Modell hat nur Gültigkeit, so lange Eingangssignale in das System niedrige Frequenz besitzen. Bei mittleren Frequenzen

kann ein aussagefähiges System nicht mehr die Einflüsse gewisser Eigenschaften der Bau-teile und vor allem der Verbindungen vernachlässigen. So besitzen die Drahtverbindungen Koppelkapazitäten und Leitungsinduktivitäten, die Spule Windungskapazitäten und der Kondensator dielektrische Verluste oder auch Eigeninduktivität.

Für die Modellierung von Robotersystemen sind vor allem die elektromotorischen Antriebe von Bedeutung. Bei ihrer Modellbildung kommt wieder das Zusammenwirken mechanischer und elektrischer Einflüsse zum Tragen, da das dynamische Verhalten des Motors nur durch Kenntnisse der Gesetze der Kinetik und der Elektrodynamik erklärt werden kann.

Eine häufig verwendete Antriebsmaschine ist der Gleichstrommotor. Bei dieser elektrischen Maschine wird sowohl das Erregungsfeld im Stator als auch das Magnetfeld im Rotor durch Gleichstrom oder durch einen Permanentmagneten erzeugt. Daher muss zur Erzeugung einer fortschreitenden Drehung die Ankerspannung durch einen elektromechanischen Schalter, den sogenannten Kommutator, synchron mit der Drehung des Ankers ständig umgepolt werden.

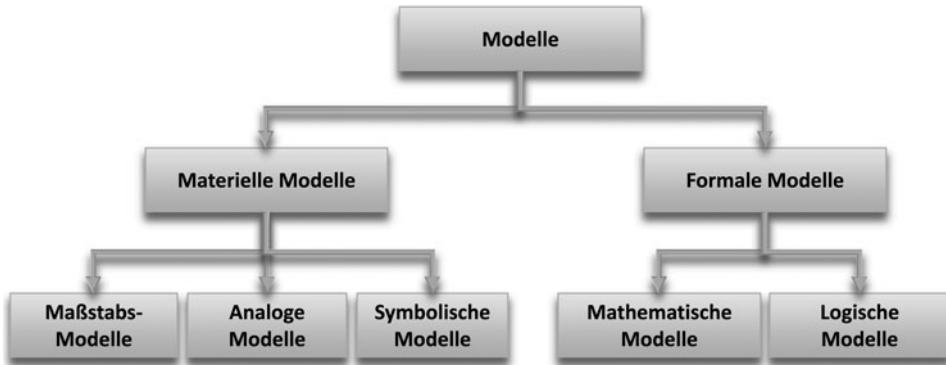
## 2.2.4 Modellbegriff

Sollen in den folgenden Kapiteln bestehende Systeme als Vorbilder eingehend analysiert, umgestaltet oder aber Systeme neu entwickelt werden, so erfordert dies umfangreiche analytische und experimentelle Untersuchungen. Speziell in der Problemdomäne der Robotik gestatten die Komplexität der Systemstrukturen und die vielfältigen Umweltbeziehungen es nicht, die Zusammenhänge sofort zu erkennen bzw. gleich beim ersten Versuch zielsentsprechend zu gestalten. Es entsteht also das Problem, ein durch Analyse, Beobachtung, Nachahmung etc. entstandenes Bild in ein experimentelles oder ausführbares Modell zu überführen.

Unter einem *Modell* wird in diesem Buch im weitesten Sinne ein System verstanden, welches ein anderes System und dessen Zustände abbildet. Modelle sind damit materielle oder immaterielle (geistige, formale) Systeme, die andere Systeme so darstellen, dass zum einen eine experimentelle Manipulation der abgebildeten Strukturen und Zustände als auch deren Überführung in ein lauffähiges System möglich ist.

Die experimentelle Manipulation von Strukturen umfasst das probeweise Knüpfen bzw. Lösen von Interaktions- und Kombinationsbeziehungen, sowie die planvolle Variation der Transformationsvorschriften (Funktionen, Abbildungsvorschriften). Die experimentelle Manipulation von Zuständen beinhaltet die planvolle Variation von Zustandsvariablen zur Ermittlung der Reaktionen der davon abhängigen Zustandsvariablen.

Insofern beschreibt oder spezifiziert ein Modell ein System zu einem ganz bestimmten Zweck, indem es alle hierfür relevanten Aspekte (Struktur, Verhalten, Funktionen, Umwelt etc.) erfasst und alle nicht-relevanten Aspekte ausschließt.

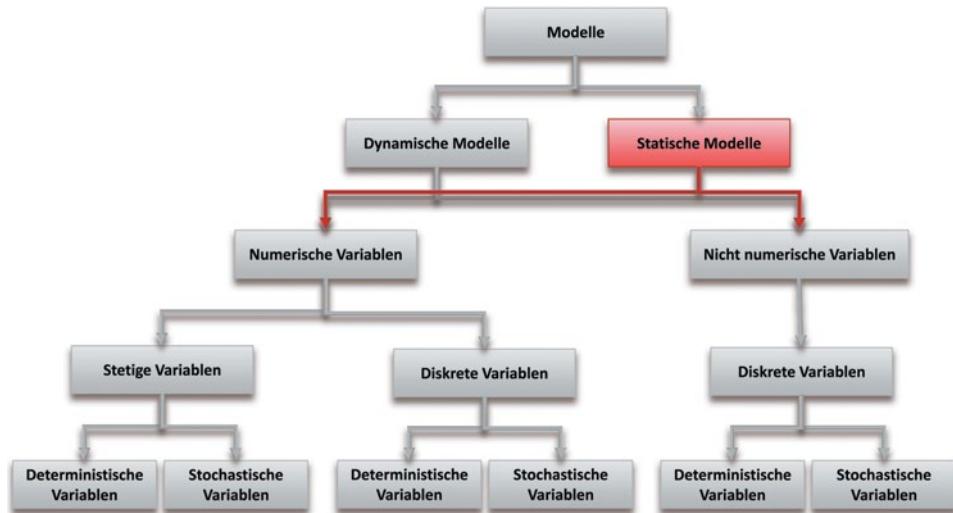


**Abb. 2.13** Modellklassifikation nach dem Abbildungsmedium

Die Manipulierbarkeit eines Modells wird vielfach nur durch eine weitgehende Vereinfachung bei der Darstellung des Systems erreicht. Ein Modell ist daher nicht nur ein Surrogat, sondern vor allem eine Simplifizierung des Systems. Dabei werden unter anderem vereinfachende Annahmen über die Anzahl der unabhängigen Variablen in den Interaktions-, Interoperations- und Kombinationsbeziehungen sowie über die Transformationsvorschriften in diesen Beziehungen gemacht. Ferner wird auch häufig versucht, die Umwelteinflüsse weitgehend auszuschalten bzw. aus der Betrachtung auszuklammern. Das bedeutet, dass ein offenes System durch ein geschlossenes System ersetzt wird, um die Beziehungen innerhalb des Modells besser analysieren zu können.

Je nach Problemdomäne und den damit induzierten Fragenstellungen lassen sich unterschiedliche Modelle zur Beantwortung der Fragen bzw. zur Entwicklung von Problemlösungen einsetzen. Die Entscheidung für den Einsatz einer bestimmten Modellvariante hängt gewöhnlich von dem Abbildungsmedium, den Typen der abzubildenden Zustandsvariablen und dem jeweiligen Verwendungszweck ab (Abb. 2.13).

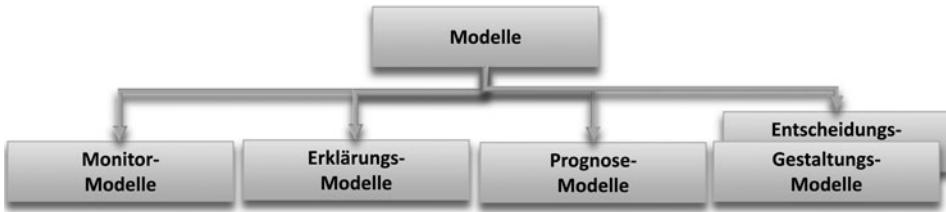
*Materielle Modelle* bilden die Systeme mit Hilfe materieller Zustände und Strukturen ab. *Formale Modelle* leisten dies mit Hilfe von Werten und Funktionen, wobei Werte die Zustände und Funktionen der Strukturen der materiellen Systeme beschreiben. Formale Modelle können als die unmittelbarste und ökonomischste Form der Abbildung angesehen werden. Auf der anderen Seite lassen sich die Systeme durch materielle Modelle wesentlich wirklichkeitstreuer abbilden als durch formale Modelle. *Maßstabsmodelle* bilden die Systeme in verkleinertem oder vergrößertem Maßstab unter Verwendung gleicher oder ähnlicher Stoffe ab. *Analogiemodelle* bilden die Systeme i. d. R. mit Hilfe völlig anderer Stoffe (d. h. Zustände) ab. Die Modellstruktur ist jedoch gleich oder ähnlich der Systemstruktur, so dass Analogien im Wirkgefüge beider Systeme bestehen. *Symbolische Modelle* bilden die Systeme mit Hilfe materieller Zustände und Strukturen ab, die keinerlei „physikalische“ Ähnlichkeit oder Analogie zum abgebildeten System aufweisen müssen. Die Zuordnung bestimmter Modellzustände und -strukturen zu den abzubildenden Systemzuständen und -strukturen ist rein symbolisch. *Mathematische Modelle* verwenden als



**Abb. 2.14** Modellklassifikation nach Zustandsvariablen

Abbildungsmedium arithmetische Konstanten, Variablen und Funktionen. In Bezug auf das abgebildete materielle System gelten ebenfalls nur logische Analogien, d. h. den Attributen des materiellen Systems sind arithmetische Werte zugeordnet und das Zusammenspiel der Attribute und ihr Einwirken auf andere wird durch arithmetische Operatoren und Algorithmen dargestellt. *Logische Modelle* verwenden als Abbildungsmedium logische und arithmetische Konstanten, logische und arithmetische Variablen und logische Funktionen (Abb. 2.14).

In den späteren Modellen werden Zustandsvariablen abgebildet werden. Da in solchen Modellen häufig mehrere Arten von Zustandsvariablen gleichzeitig auftreten, muss sich die Wahl der Modellvariante nach der Mehrheit oder der Wichtigkeit einer bestimmten Variablenart richten. *Dynamische Modelle* werden u. a. von der unabhängigen Variable „Zeit“ bestimmt, während bei *statischen Modelle* diese Abhängigkeit fehlt. Dynamische Variablen dienen der Abbildung von Attributbeziehungen, statische Variablen der Abbildung von Attributkombinationen. Numerische Variablen bilden primäre und bestimmte tertiäre Attributklassen eines Systems im Rahmen eines mathematischen oder logischen Modells ab. Nichtnumerische Variablen bilden sekundäre und bestimmte tertiäre Attributklassen eines Systems in logischen Modellen ab. Numerische Variablen können stetige oder diskrete Ausprägungsmengen haben und dienen der Abbildung entsprechender Attributmengen in primären und tertiären Attributklassen. Nichtnumerische Variablen haben stets nur diskrete Ausprägungsmengen und dienen der Abbildung der entsprechenden Attributmengen in sekundären und bestimmten tertiären Attributklassen des Systems. Ist eine Variable mit anderen funktional verknüpft, so heißt sie deterministisch. Wird eine Variable ganz oder teilweise durch einen Zufallsprozess bestimmt, so heißt sie stochas-



**Abb. 2.15** Modellklassifikation nach Verwendungszweck

tisch. Deterministische Variablen dienen in erster Linie der Abbildung von Zuständen in geschlossenen Systemen, während stochastische Variablen in erster Linie der Abbildung von Zuständen in offenen Systemen dienen.

Eine Auswahl der Modelle lässt sich auch dahingehend vornehmen, dass man den Verwendungszweck als Auswahlkriterium heranzieht (Abb. 2.15).

*Monitormodelle* dienen der laufenden Erfassung und Abbildung von Systemzuständen. *Erklärungsmodelle* dienen der Erfassung und Abbildung von Systemstrukturen oder einzelnen Kombinations- und Interaktions- bzw. Interoperationsbeziehungen in einem System. *Prognosemodelle* dienen der Vorhersage zukünftiger Systemzustände bei variierenden Werten bestimmter unabhängiger Variablen. *Gestaltungsmodelle* bzw. *Entscheidungsmodelle* dienen der Ermittlung von geeigneten Systemstrukturen und Werten manipulierbarer unabhängiger Variablen zur Erreichung eines gewünschten Systemzustandes. Die Gestaltungsmodelle schließen im Allgemeinen die Prognosemodelle und die Prognosemodelle die Erklärungsmodelle ein.

## 2.3 Simulation

Simulationen anhand von Modellen sind heuristische Annahmen von Bedingungen und Gegebenheiten, die im Zusammenhang der Robotik gleich zu unterschiedlichen Zwecken dienlich sind.

### 2.3.1 Simulationen

Im Allgemeinen sind es vor allem vier Zwecke, die mit Hilfe einer Simulation erreicht werden sollen:

- Virtualisierung,
- Totalisierung,
- Formalisierung und
- Antizipation.

*Virtualisierung*, das heißt das Durchspielen einer Situation in einem Modell außerhalb der realen Umgebung dieser Situation, ist die wichtigste Funktion der Simulation, von der sich die anderen drei ableiten lassen. Simulationen ermöglichen es, alternative Entscheidungen nebeneinander und nicht, wie in der rauen Wirklichkeit, einander ausschließend, zu betrachten und damit in ihren Konsequenzen ohne Schaden zu nehmen, zu verfolgen. Man kann Parameter verändern, die Simulation abbrechen und noch einmal starten und die Erfolge der unterschiedlichen Strategien vergleichen. Man kann die Logik einer Strategie ausprobieren, ohne ihre Folgen zu tragen. Die Simulation „kostet nichts“, außer Rechen- und Beobachterzeit. Man kann sich falsch entscheiden, diesen Fehler erkennen und die Lösungsstrategie verändern.<sup>32</sup>

Simulationen ermöglichen eine *totalisierende* Sicht auf einen Sachverhalt oder eine Problemstellung.

So zeigt sie einen Schauplatz in der Totalen, in der jede einzelne Bewegung, jeder mögliche und tatsächliche eigene und gegnerische Zug in einem Gesamtüberblick möglich wird. Visuell wird diese Totalisierung gern als eine blinkende Landkarte auf einem Monitorsystem dargestellt, auf der man verfolgen kann, wie sich die einzelnen Roboterentitäten ihrem Ziel nähern.

*Formalisierung* leistet die Simulation, indem sie von der harten und oftmals unbarmherzigen Realität absieht und Konflikte auf tendenziell mathematisierbare Parameter reduziert. Diesen Parametern werden Punktzahlen zugeordnet, die dann als Gewinn oder Verlust in bestimmten Feldern verzeichnet werden können. Dabei unterstellen Simulationen, dass sich die Umweltbedingungen und ihre Artefakte nach Regeln verhalten, die sich anschreiben lassen und damit idealiter Gegenstand von Rechenoperationen werden können. Die Anzahl dieser Regeln ist begrenzt, sie sind konstant, und es muss davon ausgegangen werden, dass sich die Umgebung an diese Regeln halten. In dem Moment, wo man es mit gänzlich undurchschaubarem, irrationalem, völlig fremden oder dramatisch unintelligentem Konfliktverhalten zu tun hat, stößt auch die Simulation an ihre praktischen Grenzen.

Neben diesen in gewisser Weise darstellungstechnischen Vorzügen der Simulation ist deren wichtigste Funktion natürlich der Versuch, das Verhalten des Robotersystems bzw. seiner Umgebung zu *antizipieren*.

Es geht um die Exploration einer doppelten Kontingenzen, die man etwa so formulieren könnte: Wenn ich mich so verhalte, welche Optionen bleiben dann meiner Umwelt und ihren Artefakten, die mir wiederum welche Entscheidungsmöglichkeiten lassen?

Legt man eine geordnete Umgebung zugrunde und kann ermessen, was der Wissensstand im gegebenen Konflikt ist, so kann man logisch schließen, welches die für den Roboter profitabelste bzw. zielperfekte Verhaltensweise ist. Und das heißt, dass man quasi auf die Zukunft vorbereitet ist. Es ist oftmals dieser Wille zum Vorauswissen über die Pläne des Verhaltens des Robotersystems, das die Simulation zum epistemologischen Gegenstück zur späteren Praxis macht.

<sup>32</sup> Siehe auch Eisenführ und Weber 1999.

### 2.3.2 Simulationstheorie

*Robotersimulationssysteme* (RSS) erlauben die Darstellung der Bewegung von einem oder mehreren Robotersystemen auf einem grafischen Monitor. Die Robotersysteme werden bei den heutigen Computern nicht mehr als dreidimensionale Drahtgittermodelle, sondern bereits als Volumen- oder Oberflächenmodelle dargestellt. Um nun ein Robotersystem oder andere Objekte bewegen zu können, benötigt das Simulationssystem ein kinematisches Modell. Bei Robotersystemen kommt dann zu den geometrischen Abmessungen noch Information über die kinematische Verkettung von Teilobjekten (Greifarmen, Gliedmaßen, etc.) durch Gelenke. Sie enthält auch Angaben über Winkelbegrenzungen und andere Einschränkungen. Wenn ein Gelenk am Bildschirm gedreht wird, muss sich automatisch auch die Lage aller daran anschließenden Roboterbestandteile gemäß der modellierten Kinematik mit ändern.<sup>33</sup> Komplexe Systeme modellieren bei der Bewegung auch die Regelkreise für die Gelenke und die Dynamik der Objekte.

Die Bewegung der Roboter im Robotersimulationssystem kann dabei auf drei Arten erfolgen:

- Durch Drehen von Drehknöpfen am Gerät, deren Drehung die Gelenkwinkel des Robotersystems verstellt,
- durch Eingabe einer Zielstellung des Effektorkoordinatensystems über die Tastatur (Cursortasten, Maus, usw.),
- durch Interpretation (Ausführung) eines Roboterprogramms im Robotersimulationssystem.

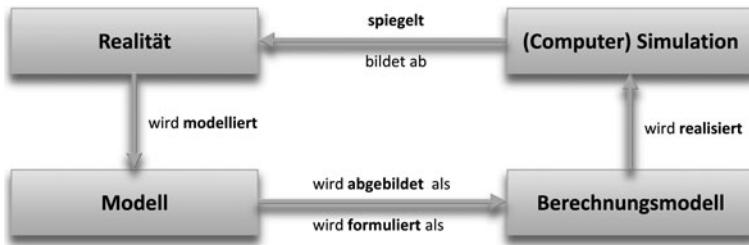
Die Eingabe von Stellungen durch Drehknöpfe oder Tastatur kann zu einer grafisch interaktiven Generierung von Roboterprogrammen ausgebaut werden. Die Interpretation eines Roboterprogramms im Robotersimulationssystem kann zum off-line Testen von Roboterprogrammen eingesetzt werden.

Bei der rechnergestützten Generierung des Roboterprogramms mit Hilfe eines grafisch, interaktiven Robotersimulationssystems wird ähnlich wie beim Programmieren durch Beispiele vorgegangen. Der Effektor eines Roboters wird am Bildschirm mit den Cursortasten, der Maus, den Drehknöpfen oder anderen Geräten richtig positioniert. Die Gelenke des Robotersystems drehen sich dabei entsprechend mit und führen so die „Arme“ nach. Ist ein gewünschter Punkt erreicht, werden die Gelenkwinkel oder die Stellung im Weltkoordinatensystem abgespeichert. Hieraus wird dann ein einfaches Roboterprogramm erzeugt. Dieses enthält im Wesentlichen nur die ausgeführten Bewegungsbefehle. Die meisten Simulationssysteme haben verschiedene Robotertypen modelliert und können Programme für unterschiedliche Steuerungen generieren.

Um komplexere, insbesondere sensorgesteuerte Roboterprogramme simulieren zu können, reicht die übliche Simulation der Bewegung eines Roboters in einer starren Um-

---

<sup>33</sup> Vgl. Schwinn 1992, S. 65 ff.



**Abb. 2.16** Zusammenhang zwischen Realität, Modell und Simulation

gebung alleine natürlich nicht aus. Hinzukommen müssen in dem Robotersimulationssystem:

- die Simulation der Sensoren,
- die Simulation der Effektoren,
- die Simulation der Kommunikation mit der Umwelt und
- die Simulation der durch die Roboterbefehle oder durch sonstige Geräte bewirkten Umweltveränderungen.

Probleme beim Einsatz der Robotersimulationssysteme zum Testen sind heute noch:

- Die Entwicklungskosten für die Bibliotheken der verschiedenen Robotertypen sind hoch.
- Die erzielbare Genauigkeit in der Modellierung reicht oft noch nicht aus, beispielsweise wenn die Dynamik oder die Regelung des Robotersystems nicht oder nicht ausreichend genau berücksichtigt wird.
- Die Abmessungen der Objekte und ihre Lage im Weltkoordinatensystem sind zu ungenau, da beispielsweise die CAD-Modelle nicht wirklich exakt mit den gefertigten Teilen übereinstimmen.

### 2.3.3 Simulationsbegriff

Der Begriff der Simulation wird in verschiedenen Bereichen und dort oftmals auch in unterschiedlichen Ausprägungen verwendet. Dieser Umstand trifft auch auf dieses Buch zu. Um Begriffsverwirrungen zu vermeiden, ist es daher notwendig, den Zusammenhang zwischen Realität, Modell und Simulation herzustellen (Abb. 2.16).

Unter *Realität* (real world) wird im weitesten Sinne jener Phänomen- oder Ereignisbereich verstanden, der in empirischen Wissenschaften wie Physik, Chemie, Biologie und Psychologie durch objektive Verfahren beobachtet und protokolliert werden kann. Realität ist demnach die Summe der beobachtbaren und messbaren Größen. Als *Modell* wird ein Realitätsausschnitt verstanden, das mit Hilfe von modellhaften Begriffen formuliert

wird, um Phänomene der Realität beschreiben zu können. Das *Berechnungsmodell* wird als Modell der Berechnung aufgefasst. Für die Erstellung einer Simulation muss zunächst ein Modell der Berechnung ausgewählt werden. Dabei geschieht die eigentliche Berechnung in Form von Zustandsübergängen.

Ein Beispiel für ein Berechnungsmodell stellt das Modell der Turing-Maschine dar. In diesem Zusammenhang sei auch der Turing-Test erwähnt. Turing schlägt vor, dass eine Person als Fragender, der seinen Dialog- bzw. Interviewpartner weder sehen noch hören kann, entscheiden können muss, ob er sich mit einem Menschen oder einer Maschine unterhält. Gelingt es der Maschine menschliche Verhaltensweisen in Form von Antworten im Rahmen des Dialoges so zu simulieren, dass die Frage unentscheidbar wird, dann ist nach diesem Turing-Test der Maschine auch Intelligenz zuzusprechen.<sup>34</sup>

Die Computer Simulation bezeichnet die Umsetzung eines Berechnungsmodells in ein ablauffähiges Programm auf einem Computer. Gerade der Einsatz von Computern hat der Simulation zum Durchbruch verholfen, da hierdurch die Bewältigung des rechentechnisch hohen Aufwands bei der Durchführung von Simulationsläufen erst möglich wurde. So verstanden sind Simulationen auch Bestandteile der Realität, da über sie genau wie über die oben definierte „Realität“ Theorien erstellt werden können. Außerdem ist das Verhältnis zwischen Modell und Realitätsausschnitt durch die wechselseitige Austauschbarkeit – zumindest in Teilen – gekennzeichnet. Allerdings kann mit Modellen immer nur ein begrenzter Teil der Realität erfasst werden und ein Modell stellt immer eine vereinfachte Abbildung eines interessierenden Realitätsausschnitts dar.

Die Beziehung zwischen Modell und Simulation auf der einen und Realität beziehungsweise System auf der anderen Seite kann in zwei Weisen hergestellt werden. Im ersten Fall wird mit der Simulation ausschließlich das Systemverhalten nachgeahmt. Das Gütekriterium zur Beurteilung der Simulation ist möglichst das gleiche Verhalten wie der simulierte Weltausschnitt beziehungsweise das simulierte System zu zeigen. Wie das Verhalten erzeugt wird, ist von untergeordneter Bedeutung. Wichtig ist, dass die entsprechenden Ausgangsgrößen der Simulation mit jenen des Simulierten übereinstimmen. Das bedeutet, dass die Übereinstimmung der Zustandsgrößen der Simulation und des Simulierten nicht notwendig ist. Um diese Forderung nach Übereinstimmung erfüllen zu können, müssen lediglich Beobachtungsdaten über das Verhalten des simulierten Systems vorliegen. Auf diese Weise ein System zu gestalten, ist immer dann angezeigt, wenn über den inneren Aufbau des simulierten Systems sehr wenig oder gar nichts bekannt ist und auch nicht viel in Erfahrung gebracht werden kann.

Gerade der Ansatz der Symbolverarbeitung verfolgt diesen Ansatz, indem das simulierte System als „Black Box“ betrachtet und das simulierende System in seiner Güte daran gemessen wird, ob es das Verhalten eines anderen kognitiven Systems adäquat nachahmt.

Auf der anderen Seite kann versucht werden, die inneren Mechanismen des simulierten Systems im Rahmen bestimmter Grenzen nachzubilden, um auf diese Weise zu erreichen,

---

<sup>34</sup> Siehe auch Gerke 1987.

dass die Simulation bei hinreichend genauer Abbildung das gleiche Verhalten zeigt, wie es das simulierte System aufweist. Hier wird also ein Modell des Systems, nicht ein Modell des Verhaltens entwickelt. Das bedeutet, dass die Wirkungsstruktur des Originalsystems erkannt und verstanden werden muss. Nur strukturtreue Modelle sind in diesem Fall akzeptabel. Das System wird hier als durchsichtige „glass box“ bezeichnet. Auf jeden Fall muss es also möglich sein, in das simulierte System im Detail hineinzuschauen und sowohl die Konstituenten der Systemstruktur offenzulegen, als auch die Zustandsgrößen und -prozesse zu erkennen, soweit dies für den Zweck der Simulation unbedingt notwendig ist.

Es ist beispielsweise nach wie vor umstritten, inwieweit diese Forderung im Bereich der Kognitionswissenschaften zu erfüllen ist, da einige der Systemstrukturen außerhalb der Zugriffsmöglichkeiten aktueller Wissenschaft zu liegen scheinen. Unabhängig davon wird im Rahmen des subsymbolischen Paradigmas und dort speziell beim Konnektionismus, im Gegensatz zum Symbolismus, versucht, kognitive Systeme von „unten“ aufzubauen. Nach der Untersuchung natürlicher neuronaler Netze und der Wirkungszusammenhänge in ihnen wird versucht, diese strukturtreu nachzubilden, um auf diese Weise zu vergleichbaren Verhaltensweisen der Simulation und des Simulierten zu kommen.

Zusammenfassend kann man die Relation zwischen der Realität auf der einen und den Modellen und der Simulation auf der anderen Seite als Abbildung oder Projektion bzw. als Wiedergabe oder Beschreibung in verschiedenen Repräsentationssprachen verstehen. Die Auswahl der Repräsentationssprache orientiert sich daran, ob und wie viel an Informationsverlust bei einer Abbildung bzw. Übersetzung von einem auf, oder in den anderen Bereich verkraftet werden kann.

Folgende Vorteile lassen sich durch Simulationen erzielen:

- *Prozessorientierung:* In (alltags)psychologischen Theorien werden Konzepte, wie beispielsweise mentale Zustände, Intentionen, Ziele, Überzeugungen, Erfahrung etc. eingearbeitet. Um ihre kausale Wirksamkeit in ein Modell und danach in eine Simulation einbringen zu können, müssen jene von der statischen in eine dynamische Formulierung „prozessualisiert“ werden.
- *Nachweis der Vollständigkeit:* Die Simulation kognitiver Fähigkeiten mit Systemen des Symbol- bzw. des Subsymbolverarbeitungsansatzes setzen voraus, dass die Formulierung einer Simulation mittels einer beliebigen Programmiersprache erfolgen und damit alle theoretischen Annahmen explizit und exakt formuliert werden. Auf diese Weise wird sichergestellt, dass alle Mechanismen, Randbedingungen und andere Annahmen einer Theorie explizit formuliert in die Simulation einfließen. Erbringt die Simulation die erwarteten Ergebnisse, kann dies als Nachweis der Vollständigkeit der theoretischen Annahmen gewertet werden. Umgekehrt kann der Misserfolg einer Simulation den Hinweis darauf geben, dass bisher implizite Annahmen in der Theorie gemacht wurden, die noch offengelegt werden müssen.
- *Analyse nicht-linearer Modelle:* Simulationen können alle möglichen Parameterkombinationen durchrechnen und ihr Wirkungen in nicht-linearen Modellen aufzeigen.

- *Entdeckung unerwarteter Konsequenzen theoretischer Annahmen:* Aus der Computer-simulation eines dynamischen Systems können sich neue Erkenntnisse ergeben, die aus der ursprünglichen Systemkenntnis nicht direkt folgen. Durch den Zwang zur Explikation aller theoretischen Annahmen können diese Auswirkungen berechnet werden.
- *Kommunikationsvorteile:* Die Formulierung einer Theorie mittels Programmiersprachen liefert eine exakte, weitgehend redundanzfreie, kurze und standardisierte Zusammenfassung.
- *Kritisierbarkeit:* Alle Annahmen der Theorie sind explizit der Kritik ausgesetzt. Es ist in einem Programm nicht möglich, ceteris-paribus-Klauseln oder andere Einschränkungen zu verstecken, die verhindern sollen, dass die Theorie mit Gegenbeispielen widerlegt werden kann.

Trotz dieser Vorteile gilt es stets zu bedenken, dass das Modell nicht das Original ist, und prinzipiell bleibt immer die Unsicherheit bestehen, ob das Modell und die darauf basierende Simulation nun tatsächlich das Systemverhalten in allen Aspekten wiedergeben kann. Insgesamt stellen Simulationen dennoch geeignete Werkzeuge für das Vorantreiben der wissenschaftlichen Erkenntnis dar.

Computersimulationen können deshalb als moderne Form des Gedankenexperiments betrachtet werden, in der Theorien oder Hypothesen konkret und unter kontrollierten Bedingungen „durchgerechnet“ werden können, ohne sich zunächst um empirische Fragestellungen oder Validität zu kümmern.

Unabhängig von dieser potenziellen Erkenntnisgewinnung, müssen die notwendigen Bedingungen erfüllt sein, damit Simulationsprogramme als Theorien genutzt werden können:

- Die Prozesse, die in dem simulierten System angenommen und simuliert werden sollen, müssen durch Algorithmen abbildbar sein.
- Die verwendeten Algorithmen müssen in akzeptabler Zeit ausführbar sein.
- Eine weitere Bedingung ist die Forderung nach partieller Vollständigkeit und Korrektheit des Modells und damit der Simulation.

Wie bereits erwähnt, kann in der Regel ein Modell und damit die darauf aufbauenden Simulationen das simulierte System nicht in allen Facetten wiedergeben. Diese Einschränkung gilt allerdings auch für die herkömmliche Theoriebildung, denn Theorien können ebenfalls nur bestimmte Aspekte der Realität umfassen.

Viele Probleme, zu denen Theorien existieren, sind beweisbar, aber nicht durch Algorithmen abbildbar, und nicht alle Probleme, für die sich Algorithmen finden lassen, können in angemessener Zeit optimal gelöst werden. Zu ihrer Lösung werden dann Heuristiken benutzt. Mit diesen lassen sich allerdings oftmals nur suboptimale Lösungen finden.

## 2.4 Wissen

Die Begriffe Daten, Information und Wissen werden im Zusammenhang mit rechnerbasierten Lösungen im Allgemeinen und in der Robotik häufig nicht sauber getrennt, sondern mit teilweiser stark überlappender Bedeutung benutzt. In den folgenden Abschnitten wird daher neben der Abgrenzung dieser einzelnen Konzepte ein Wissensbegriff erarbeitet, der zur Entwicklung von Robotersystemen nützlich sein wird.

### 2.4.1 Wissen

Im Allgemeinen bezeichnet Wissen demnach eine gerechtfertigte, wahre Überzeugung, eine Vorstellung von höchstwahrscheinlicher Gültigkeit. Es ist die Grundlage allen Verstehens und Lernens, ein Reservoir, aus dem Denken und Handeln erst hervorgehen.<sup>35</sup> *Explizites Wissen* kann durch Worte und Ziffern beschrieben und direkt erworben oder vermittelt werden. *Implizites Wissen* kann nicht durch Sprache vollständig ausgedrückt werden, sondern ist verankert in mentalen Modellen. Es kann nur über andere Kommunikationsformen vermittelt oder in explizites Wissen verwandelt werden. *Konzeptionelles Wissen* ist das Wissen um Zusammenhänge, das Wissen, warum etwas ist und warum etwas geschieht. *Operatives Wissen* beschreibt, was wie zu tun ist, damit geschieht, was geschehen soll. Zwischen dem impliziten Wissen und dem operativen Wissen besteht ein enger Zusammenhang, der sich in Routinetätigkeiten äußert, die unbewusst ausgeführt werden können. Beide Wissensarten sind *Erfahrungswissen*. Explizites und konzeptionelles Wissen ist *Vernunftwissen*, das objektiver und weniger rigide ist. Wissen ist keine Substanz, sondern eine dynamische Entität, die immer von bestehendem Wissen ausgeht, um neues Wissen zu schaffen. Entdeckungen werden gemacht, indem man Möglichkeiten nachgeht, die vom vorliegenden Wissen eröffnet werden. Entdeckungen setzen einen Verzicht auf Dogmen und Konformismus voraus. Die Bereitschaft, bestehendes Wissen zu verwerfen, Unsicherheit als Motivation anzunehmen und der Wille, neues Wissen selbst zu schaffen, sind hierfür eine Notwendigkeit. Solche Bereitschaft stellt hohe Ansprüche, denn neues Wissen kann zu schmerzhaften Lernprozessen führen, bis es neue Problemlösungen und optimale Verhaltensmuster hervorbringt. Wissen stellt aus informationstheoretischer Perspektive verarbeitete Daten und Informationen dar und ermöglicht seinem Träger, bestimmte, kontextbezogene Handlungsvermögen aufzubauen, um damit definierte Ziele zu erreichen. Wissen ist mithin maßgeblich das Ergebnis einer Verarbeitung von Daten und Informationen. Ein bedeutender Unterschied zwischen Daten und Informationen einerseits, sowie Wissen andererseits, liegt darin, dass erstere in der Regel expliziter Natur sind. So können Daten und Informationen sowohl in schriftlicher Form als auf elektronischen Medien erfasst, verwaltet, gespeichert und der weiteren Nutzung zugänglich gemacht werden. Im Gegensatz dazu ist Wissen häufig von implizitem Charakter, das heißt, schwer

---

<sup>35</sup> Siehe auch Scheidgen et al. 1990.

artikulierbar und mitteilbar. Es baut auf individuellen, historisch bedingten Erfahrungen und schwer greifbaren Faktoren, wie dem persönlichen Wertesystem oder Gefühlen auf. Oftmals schlummert dieses implizite Wissen in Form von Analogien und Metaphern, sozusagen als verborgener Schatz im Inneren der Mitarbeiter. Die primären Generatoren und Träger von Wissen sind, zumindest im Fall von implizitem Wissen, Personen. Berücksichtigt man die Tatsache, dass die Lern- und Speicherkapazität von Individuen relativ eng begrenzt scheint, so ist für die Wissensgenerierung eine gewisse Spezialisierung erforderlich. Das impliziert, dass eine Vertiefung des Wissens in einem bestimmten Gebiet in der Regel auf Kosten der Breite der Wissensbasis erfolgt. Die verschiedenen Kategorien von Wissen unterscheiden sich in Bezug auf die Transferierbarkeit. Ein wesentlicher Unterschied besteht dabei besonders zwischen explizitem, also artikuliertem und somit leichter transferierbarem Wissen und implizitem, nur in der Anwendung zugänglichem, personengebundenem Wissen. Insofern gilt das Wissen um das Wissen als eine wichtige Voraussetzung zur Lösung für die in der Robotik anstehenden Problemstellungen.

## 2.4.2 Wissenstheorie

Die Entwicklung wissensbasierter Systeme im Allgemeinen und kognitiver Roboter im Besonderen ist untrennbar mit dem Stand der Theorie des Wissens verbunden. Die Theorie des Wissens hat, seitdem unter anderem die Wissenschaft das nahende Wissenszeitalter gesichtet hat, zusehends an Bedeutung gewonnen und gleichzeitig markante Änderungen durchlaufen. Sicherlich haben die Basisinnovationen der Informations- und Kommunikationstechnologie zunächst das Informationszeitalter eingeläutet.<sup>36</sup> Die ständige Verfügbarkeit von Mensch und Computer haben den Alltag und das Arbeitsumfeld radikal verändert. So stellt das Internet eine Unmenge von Daten an jedem Ort und zu jeder Zeit zur Verfügung. Gerade diese Datenmenge und diese neue Komplexität, die durch eine Überforderung der menschlichen Wahrnehmungsfähigkeit entsteht, ist das eigentlich Neue am Wissenszeitalter. Allerdings lassen sich nicht alle Daten zu Informationen oder gar zu Wissen verarbeiten. Genau diese drei Begriffe sind es also, die eine für die Robotik anwendbare Wissenstheorie prägen und doch gleichzeitig immer wieder für Verwirrung sorgen. So werden Daten, Information und Wissen häufig, sowohl im Alltag als auch in der Wissenschaft, uneinheitlich und unsystematisiert verwendet.<sup>37</sup>

Zunächst werden *Daten* durch Zeichen repräsentiert und sind Gegenstand von Verarbeitungsprozessen. Sie setzen sich aus einzelnen Zeichen oder aber aus einer wohldefinierten Folge von Zeichen zusammen, die einen mehr oder weniger sinnvollen Zusammenhang ergeben. Dieser Zusammenhang ist entweder schon bekannt oder er wird unterstellt, so dass die Zeichen mit einem Code gleichgesetzt werden können. Eine Aussage über den

---

<sup>36</sup> Vgl. Capurro 1995.

<sup>37</sup> Vgl. Oberliesen 1982.

Verwendungszweck wird allerdings noch nicht geleistet. Daten werden zu *Informationen*, indem sie in einen Problemzusammenhang, den sogenannten Kontext gestellt und zum Erreichen eines konkreten Ziels verwendet werden.<sup>38</sup> So stellen beispielsweise Informationen für ein Robotersystem diejenigen Daten dar, die für die Vorbereitung zielorientierter Handlungen nützlich sind. *Wissen* ist das Ergebnis der Verarbeitung von Informationen durch das Bewusstsein und kann als „verstandene Information“ bezeichnet werden. Diese verstandenen Informationen werden oftmals zur Handlungssteuerung verwendet. Wissen ist somit die Vernetzung von Information, die es dem Wissensträger ermöglicht, Handlungspotenzial aufzubauen und konkrete Aktionen in Gang zu bringen. Es ist das Resultat einer Verarbeitung der Informationen durch das Bewusstsein. Formal betrachtet ist Wissen daher ein Begriff mit weitem Umfang und kann hinsichtlich Erkenntnisquelle, Inhalt, Ursprung, Qualität, Struktur und Funktion differieren. Deshalb erscheint es auch in diesem Zusammenhang berechtigt, von verschiedenen Arten und Formen von Wissen, kurz von Wissensvarianten zu sprechen. Sie weisen meist eine enge Korrelation auf und können daher nicht eindeutig zugeordnet, beziehungsweise voneinander abgegrenzt werden.

### 2.4.3 Wissensvarianten

Die Wissenstheorie arbeitet demnach mit unterschiedlichen Konzepten. So umschreibt ein *Zeichen* aus Sicht der Sprachwissenschaft ein natürliches oder künstliches, sinnlich wahrnehmbares Phänomen, das für ein anderes – auch abstraktes – Phänomen steht, also Bedeutung erhält. Grundsätzlich kann alles sinnlich Wahrnehmbare durch einen Interpret zum Zeichen werden. Zeichen indexieren somit, was der Fall ist, als auch Möglichkeiten, d. h. was der Fall sein könnte. *Daten* werden durch solche Zeichen repräsentiert und sind Gegenstand von Verarbeitungsprozessen. Sie setzen sich aus einzelnen Zeichen oder aber aus einer Folge von Zeichen zusammen, die einen sinnvollen Zusammenhang ergeben. Daten werden zu *Informationen*, indem sie in einen Problemzusammenhang (Kontext) gestellt und zum Erreichen eines konkreten Ziels verwendet werden. Damit ist Information immer nur Information für einen empfangenden Jemanden bzw. für ein informationsverarbeitendes System, und auch nur dann, wenn diese Information als solche zu einer Zustandsbestätigung oder Zustandsveränderung bei dem Empfängersystem selbst beiträgt. So stellen Informationen für Entscheidungssysteme diejenigen Daten dar, die für die Vorbereitung von Entscheidungen nützlich sind. Gerade der Terminus der Information nimmt im Rahmen der späteren Implementierung eine Schlüsselfunktion ein. Der Begriff *Wissen* lässt sich zunächst charakterisieren durch Attribute wie abstrakt, strukturiert/unstrukturiert, bedeutungstragend, mitteilbar, bewusst/unbewusst oder durch ähnliche Begriffe, wie Kenntnis, Erfahrung und Verstand. Letzteres Charakteristikum kommt auch daher, dass ein Adressat nur dann eine Information bestätigen kann, wenn er über Wissen

---

<sup>38</sup> Siehe auch Shannon und Weaver 1976.

verfügt, ob diese Information nun wahr ist oder nicht und wenn ja, in welcher Form. Von daher sind Information und Wissen eng aufeinander bezogen. Diese Bezogenheit kommt auch in der Auffassung der Wissenstheorie zum Ausdruck, indem hier Wissen das Ergebnis der Verarbeitung von Informationen durch ein wie auch immer geartetes Bewusstsein ist und daher zu Recht im vorangegangenen Abschnitt als „verstandene Information“ umschrieben werden kann. Zu wissen bedeutet dann, über zureichende Gründe zu verfügen, bestimmte Informationen über Gegenstände, Sachverhalte oder Vorgänge als wahr oder falsch erkennen zu können und im Gegensatz zum Meinen, die Relevanz der Information für eine Handlungssituation ermittelt zu haben. Insofern werden diese so verstandenen Informationen zur Handlungssteuerung verwendet. Wissen ist somit die Vernetzung von Information, die es ermöglicht, Handlungs- und Aktionsvermögen aufzubauen.

Formal betrachtet ist Wissen daher ein Begriff mit weitem Umfang und kann hinsichtlich Erkenntnisquelle, Inhalt, Ursprung, Qualität, Struktur und Funktion differieren. Deshalb erscheint es auch in diesem Zusammenhang berechtigt, von verschiedenen Arten und Formen von Wissen zu sprechen. Die sicherlich wichtigste und einfachste Unterscheidung besteht in der Differenzierung von *propositionalem Sachwissen* (etwas zu wissen) und *nicht-propositionalem Gebrauchswissen* (zu wissen wie). So unterscheidet die Wissenspsychologie in Sachwissen, Handlungswissen und Metawissen.<sup>39</sup>

Da Sachwissen eine Form des „Wissen von etwas ist“, wird es als „propositionales Wissen“ bezeichnet. Als ein solches Wissen von Etwas kann es so repräsentiert werden, dass es in Entitäten von Faktenwissen über Beziehungen, Sachverhalte und Gegenstände zerlegt werden kann. Das Gebrauchswissen wird demgegenüber als „nicht-propositionales Wissen“ bezeichnet.

Eine weitere, für das Vorhaben dieses Buches notwendige Untergliederung in sogenannte *Wissensarten* ist die folgende:

- *Prozedurales Wissen* hält feste Vorgehensweisen oder Strategien fest und entspricht dem Know-how.
- *Erfahrungswissen* ist das durch die Sinneswahrnehmung gewonnene Wissen, welches in eine bestimmte Situation eingebettet ist. Es ist somit gegen Vergessen resistenter als reines Wortwissen.
- *Deklaratives, faktisches Wissen* repräsentiert Kenntnisse über die Realität und hält feststehende Tatsachen, Gesetzmäßigkeiten, sowie bestimmte Sachverhalte fest. Es entspricht damit dem Know-that.
- *Statistisches Wissen* entspricht dem Wissen, das aus Fallsammlungen stammt.
- *Kausales Wissen* stellt Wissen dar, in dem Beweggründe und Ursachen festgehalten werden (Know-why).
- *Heuristisches Wissen* hält bestimmte Sachverhalte in Regeln fest.

---

<sup>39</sup> Siehe auch Mandl und Spada 1988.

- *Klassifizierungs- und Dispositionswissen* repräsentiert Wissen, welches dem Wissenden ermöglicht, komplexe Gegenstände aufzuschlüsseln und bestimmte Sachverhalte richtig zuzuordnen.
- *Relationenwissen* stellt Wissen dar, das dem Wissenden ermöglicht, Strukturen und Zusammenhänge zu sehen.

Eine implementierungsnahe Perspektivierung liefert folgende Unterscheidung:

- *Fakten*: Einfache Fakten sind beispielsweise „Es regnet“ oder „Willi hat Schnupfen“.
- *Komplexe Objekte* und *Eigenschaften*: Objekte stellen eine Art Bezugs- oder Beziehungspunkte in Wissensmengen dar. Sie sind durch bestimmte Eigenschaften gekennzeichnet, gehören zu einer bestimmten Klasse, besitzen Beziehungen zu anderen Objekten aus der Klasse und können Veränderungen unterworfen werden.
- *Semantische Beziehungen* zwischen Objekten: Objekte innerhalb eines Problembereiches sind nicht isolierte Entitäten, sondern stehen üblicherweise in Beziehung zueinander. So kann beispielsweise das Objekt Willi mit dem Objekt Schnupfen in der Beziehung „hat“ stehen: Willi hat Schnupfen.
- *Ereignisse*: Ereignisse ähneln in gewisser Weise den Handlungen. Lediglich die Tatsache, dass sie nicht von handelnden Lebewesen initiiert, sondern durch Umstände erzeugt werden können, unterscheidet sie von den Handlungen. Unter einem Ereignis versteht man demnach eine Zustandsänderung, für die es einen Ort und eine Zeitspanne gibt.
- *Handlungen*: Durch Handlungen können Objekte absichtsvoll erzeugt, verändert, gelöscht (zerstört), in Besitz genommen werden. Die Art der Handlungen lässt sich beliebig fortsetzen. Das auslösende Objekt heißt Agent oder Aktor.
- *Vorgänge*: Ein Vorgang ist eine andauernde Handlung.
- *Verfahren*: Ein Verfahren setzt sich aus einer wohldefinierten Anzahl von verketteten Handlungen zusammen. Die einzelnen Handlungen müssen dabei oftmals in einer ganz bestimmten Reihenfolge ausgeführt werden. Das Wissen um bestimmte Verfahren, also das Wissen, wie Objekte und Aktionen kombiniert werden müssen, um ein bestimmtes Ziel zu erreichen, bezeichnet man auch als „know how“.
- *Zusammenhänge* und *Einschränkungen*: Neben einfachen Beziehungen, wie „hat“ existieren selbstverständlich auch noch komplexere Beziehungen zwischen Objekten. Es ist ein Wissen von der Art, „wenn Faktum A zutrifft, dann gilt auch Faktum B“. Es können aber auch Beziehungen zwischen Objekten und Geschehnissen (Ereignisse, Handlungen, Vorgänge, Verfahren) bestehen. Eng mit diesem Wissen verbunden ist das Wissen über einschränkende Bedingungen, d. h., Wissen über die Unzulässigkeit von Zuständen oder Zustandsänderungen.
- *Metawissen*: Metawissen ist das Wissen über das restliche Wissen. Es umfasst Kenntnisse über die Verlässlichkeit beziehungsweise die Wahrscheinlichkeit von Fakten. Es beinhaltet auch das Wissen, wie und wo unbekannte Daten erfragt werden können oder ggf. wo sie aus vorhandenem Wissen abgeleitet werden können. Aber auch das Strukturieren von Wissen, das Hinzufügen von Wissen und die Unterstützung des Zugriffs auf dieses Wissen zählt man zum Metawissen.

- *Probleme und Problemlösungsstrategien:* Das zu lösende Problem muss bekannt sein, soll es gelöst werden. Bei der Bearbeitung solcher Probleme mit Hilfe von Computern, muss das zu lösende Problem sogar noch in einer für den Computer verständlichen Form spezifiziert werden. Allgemeine Strategien und Methoden zur Lösung dieser Probleme und spezielle Kenntnisse über den Problembereich müssen bekannt sein.

Die Inhalte einer bestimmten Wissensart können unterschiedliche Qualitäten aufweisen. Die wichtigsten *Wissensqualitäten* sind die folgenden:

- *Unvollständiges Wissen:* Alle Wissensbasen sind unvollständig in dem Sinne, dass immer nur Teile einer zu repräsentierenden Welt erfasst und dargestellt werden können. Eine Wissensbasis heißt unvollständig, wenn sie für eine Menge von Aussagen feststellt, dass mindestens eine davon wahr ist, aber nicht angibt, welche dies ist. Daher muss auch jedes Wissen, das für die Nutzung durch einen Computer zusammengestellt wurde, zwangsläufig unvollständig sein, da es zur Zeit noch nicht möglich ist, das gesamte Wissen der Menschheit in einem Computersystem oder einem Netz von Computern zu speichern oder gar effizient zu verarbeiten.
- *Widersprüchliches Wissen:* In vielen Fällen ist das Wissen in sich widersprüchlich. Gerade bei der Erweiterung bestehender Wissensbasen steht das neu hinzukommende im Widerspruch zum bereits existierenden Wissen.
- *Unsicheres Wissen:* Wissen kann auch unsicher sein, wenn nicht die exakte Wahrscheinlichkeit des Zutreffens dieses Wissens angegeben werden kann.
- *Ungenaues Wissen:* Ferner kann Wissen ungenau sein, d. h., es lassen sich keine exakten Angaben zu den Eigenschaftswerten machen, wohl aber die möglichen Werte einschränken. Der Unterschied zu unsicherem Wissen besteht dabei darin, dass ungenaues Wissen sicher zutrifft. Der Unterschied zu unvollständigem Wissen besteht darin, dass eine Eigenschaftsangabe sehr wohl vorliegt und eben nicht fehlt.
- *Prototypisches Wissen:* Man kann verschiedene Modalitäten von Wissen unterscheiden, d. h., es ist möglich, Wissen über Sachverhalte zu besitzen, die notwendig, möglich, oder unmöglich (alethische Modalitäten), obligatorisch, geboten oder verboten (deontische Modalitäten) sind. Als prototypisch bezeichnet man gewöhnliches Wissen über zutreffende Sachverhalte. Hier hat man einfach eine Vorstellung, wie die Dinge sind. Prototypisches Wissen kann benutzt werden, um Eigenschaften mit relativ großer Wahrscheinlichkeit anzunehmen, solange noch keine weiteren, oder der Annahme widersprechende Informationen vorliegen. Prototypisches Wissen ist also eine Möglichkeit, unvollständiges Wissen zu ersetzen.
- *Defitorisches/kontingentes Wissen:* Eine definitorische Aussage bezieht sich auf die nicht anzweifelbaren Eigenschaften einer Aussage oder eines Objekts einer Klasse. Andererseits gibt es Aussagen, die zwar allgemein anerkannt sind oder sich nur auf einen großen Teil der Klasse beziehen, deren Wahrheit aber nicht unbedingt notwendig zu sein braucht.

- *Allgemeinwissen*: Unter Allgemeinwissen versteht man das Wissen, welches Menschen einsetzen, um alltägliche Probleme zu meistern.
- *Fachwissen*: Vom Allgemeinwissen lässt sich das Fachwissen abgrenzen, das Experten einsetzen, um Probleme spezialisierter Art zu lösen.
- *Modalaspekte*: Als Modalaspekte bezeichnet man solche Umstände, unter denen eine Aussage bewertet werden muss. So ordnet man beispielsweise Aussagen einen bestimmten Wahrheitswert (wahr oder falsch) zu.
- *Unwissen*: Unwissen stellt das Pendant zu Wissen dar und ist als solches stets mit Wissen verbunden.

Diese Wissensvarianten orientieren sich auch an den Ergebnissen aus der Wissensakquise von sogenannten Experten. Dabei zeigte sich, dass Experten Probleme weniger durch aufwendige Suchvorgänge lösen, als vielmehr durch den Abruf gespeicherten Wissens. Aufgrund ihrer vielfältigen Erfahrungen haben sie typische Problemmuster und Problemlösungswege in Form von Schemata in ihrem Gedächtnis gespeichert, deren relevante Teile schnell und zuverlässig abgerufen werden können. Neben einem umfangreichen Bereichswissen spielen sogenannte Fallschemata sowie hochgradig bereichsspezifische Heuristiken eine große Rolle. Hinzu kommt ein differenziertes metakognitives Kontrollwissen in Form von Prüfungs-, Bewertungs- und Prognoseprozessen.

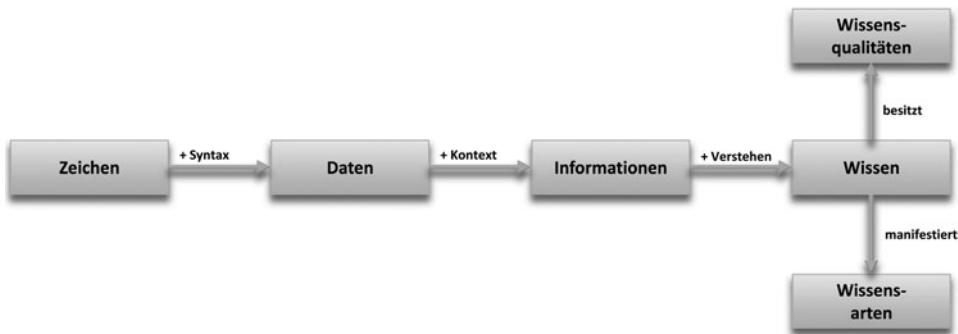
Im Alltag bezeichnet man solche Menschen als Experten, die schwierige Probleme erfolgreich und effizient auf der Basis von Wissen und Erfahrung lösen. Hinsichtlich der Wahrnehmung nehmen Experten Stimuli in größeren bedeutungshaltigen Einheiten wahr und können entsprechend auch mehr reproduzieren. Weiterhin repräsentieren Experten relevante Informationen oft auf einer abstrakteren, den theoretischen Begriffen des entsprechenden Bereichs näherstehenden Ebene, als Novizen. Zum Aufbau dieser Repräsentationen verwenden Experten häufig mehr Zeit auf die Interpretation von Aufgaben als Novizen. Novizen sind auf allgemeinere Problemlösungsmethoden angewiesen. Entsprechend ist ihr Vorgehen ineffizienter und fehleranfälliger.

#### 2.4.4 Wissensbegriff

Diese bisher erarbeitete Auffassung bezüglich Daten, Information und Wissen hat Konsequenzen, indem diese Konzepte im Folgenden nicht ohne ihren Verarbeitungsaspekt weiter entwickelt werden können. So kann Wissen nicht ohne den Aspekt der Repräsentation von Informationen bzw. Daten und letztere nicht ohne den Aspekt des „Symbols“ und letztlich nicht ohne den Aspekt der Manipulation durch Verarbeitung behandelt werden.<sup>40</sup>

Dem Wissensbegriff dieses Buches liegt demnach ein handlungs- bzw. verhaltensorientiertes Menschenbild zugrunde, das den Menschen als ein kognitives System auffasst, das Daten und Informationen aufnimmt, speichert, verarbeitet, produziert, und das dabei sein

<sup>40</sup> Siehe auch Norman und Rumelhart 1978.

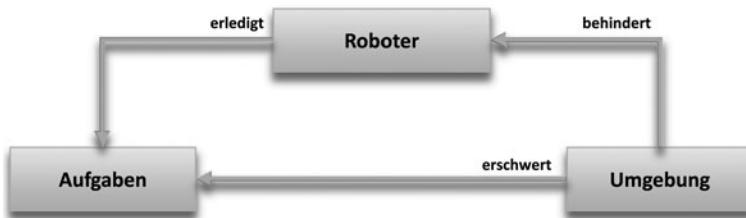


**Abb. 2.17** Daten, Informationen und Wissen

Wissen nutzt, um Ziele zu verfolgen und um entsprechend dieser Ziele zu handeln. Damit hängen Daten-, Informations- sowie Wissensverarbeitung sehr eng mit menschlicher Kognition zusammen. Um Daten, Information und Wissen verarbeiten zu können, müssen diese Konzepte durch Symbole dargestellt werden (Abb. 2.17).

Es muss an dieser Stelle darauf hingewiesen werden, dass damit viele andere Möglichkeiten der Interpretation der Begriffe Symbol, Daten, Information und Wissen ausgespart bleiben, weil sie für die Problematik dieses Buches und dort vor allem in Bezug auf die spätere technische Realisierung im Rahmen der Implementierung nicht von Bedeutung sind. So kommt am Ende dieses Abschnitts gerade dem Begriff der Information in Bezug auf den Verarbeitungsaspekt von Daten, Information und Wissen noch einmal eine zentrale Bedeutung zu. Kognitive Systeme, die kognitive Prozesse durchlaufen und dadurch kognitive Leistungen erbringen, werden im Folgenden als Berechnungsvorgänge vor allem in Form der Manipulation von Information verstanden. Hier erfolgt also eine technisch bedingte Dekomposition von Daten-, Informations- und Wissensverarbeitung auf den Begriff der Informationsverarbeitung, da Information auf eine technisch bedingte Art und Weise der formalisierten Repräsentation und syntaktischen Symbolmanipulation reduziert wird, ohne die keine rechnerbasierte bzw. maschinelle Informationsverarbeitung gegenwärtig möglich ist.

Dieser Auflösung der Daten-, Informations- und Wissensverarbeitung auf den zentralen Begriff der Informationsverarbeitung kommt neben dem technischen auch ein praktischer Aspekt zu. Durch eine solche Sichtweise der Informationsverarbeitung wird eine abstrahierende Zugangsweise erleichtert, die den Menschen als Vorbild und die Maschine unter dem gemeinsamen Aspekt der Informationsverarbeitung als kognitives System zu betrachten gestattet. Wenn daher im folgenden Verlauf der Mensch, der Computer und das Robotersystem als kognitive Systeme zur Informationsverarbeitung aufgefasst werden, dann zu dem Zweck, dass sich dadurch Modelle erarbeiten lassen, die auf beide gleichermaßen anwendbar sind.



**Abb. 2.18** Klassische Auffassung von Robotern

## 2.5 Interoperation

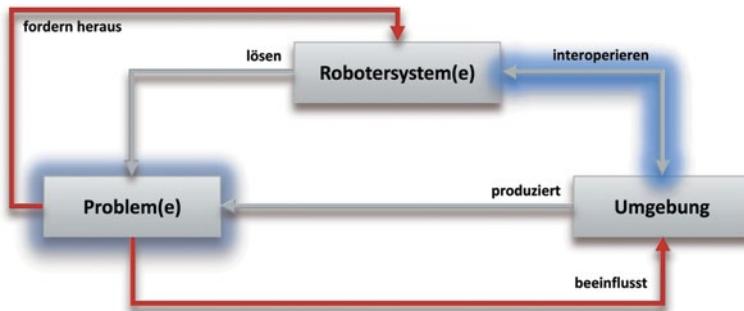
Nunmehr scheint es angebracht, die Begriffe Handlung, Verhalten, Aktion, und Interaktion voneinander abzugrenzen, insbesondere deshalb, weil deren Verwendung im Rahmen der Robotik von der etwa in der Psychologie und in der klassischen künstlichen Intelligenz abweicht. Vor allem aber wird die Entwicklung kognitiver Robotersysteme die Erweiterung dieser Begriffe hin zu dem Begriff der Interoperation erfordern.

### 2.5.1 Interoperationen

Gemäß der klassischen Auffassung sind Roboter, Aufgaben und Umgebung miteinander verknüpft und können nicht unabhängig voneinander betrachtet werden (Abb. 2.18).

Die zukünftigen Anforderungen lassen hingegen Roboter erwarten, die nicht nur Aufgaben „abarbeiten“ beziehungsweise sich in einer bestimmten Umgebung nur „verhalten“, sondern auf und in diese Welt „einwirken“. Sie reagieren auf Umgebungsreize in sinnvoller Weise und können durch sogenannte Interoperation mit der Umwelt auch nicht vorraus geplante Probleme adäquat lösen. Gerade der im Gegensatz zum klassischen Begriff der Interaktionen erweiterte Einwirkungsumfang der Interoperationen impliziert, dass sich Umwelt und Robotersysteme gegen- und wechselseitig bedingen und beeinflussen. Im Rahmen dieses Buches wird also damit der klassische Begriff der Handlung als Form der Interaktion erweitert und als *Interoperation* ausgedrückt, die für zielgerichtetes, notwendigerweise bewusstes Verhalten und Einwirken auf die Umgebung steht.

So wird in der Psychologie ein menschliches Verhalten als Handlung bezeichnet, wenn es aus Sicht des Handelnden zielgerichtet, intentional und damit sinnbehaftet ist. Ein Beobachter wird ein Verhalten dann als Handlung bezeichnen, wenn er diesen Sinn erkennen, nachvollziehen oder verstehen kann. Als Verhalten werden im behavioristischen Sinn alle beobachtbaren Reaktionen eines Organismus auf einen Reiz bezeichnet. Im Unterschied zu Handlungen kann man Verhalten zwar Ursachen, aber unter Umständen keinen Sinn zuweisen. So kann dasselbe Verhalten unterschiedlichen Handlungen zugrunde liegen. In der KI werden oftmals Interaktionen eines Systems als Aktionen (actions) bezeichnet. Ausgehend von diskreten und statischen Umgebungen ist eine Aktion formal die von einem System verursachte Überführung eines Zustands in einen Folgezustand. So werden für ein zielgerichtetes Handeln einzelne Aktionen zu komplexen Handlungsplänen zusammengefasst.



**Abb. 2.19** Zukünftige Auffassung von Robotersystemen

Damit wird bereits recht früh in diesem Buch postuliert, dass zukünftige Robotersysteme als intelligente Verhaltenssysteme zum einen von der Komplexität ihrer Umgebung weitgehend bestimmt werden, aber durch ihr eigenes intelligentes Verhalten auf diese Umwelt einwirken (Abb. 2.19).

Die Einwirkung eines Robotersystems beschränkt sich allerdings nicht nur auf die Umwelt. Vielmehr lautet die allgemeine Grundstruktur einer theoretisch fundierten Aussage über Interoperationen von Robotersystemen: Ein Robotersystem A führt in Bezug auf Robotersystem B in der Situation C die Interoperation X aus, bewirkt damit Y und wirkt damit auf die Situation C und das Robotersystem B ein. Das Robotersystem B als auch die Situation C werden ggf. nicht dieselben sein, wie vor der Einwirkung, sondern sich sich zu C' und B' entwickelt haben.

In Anlehnung bzw. gemäß einer Erweiterung der Auffassung Descartes gilt daher: Ich inter-operiere, also bin ich!

## 2.5.2 Interoperationstheorie

Mit der Interoperationstheorie werden gleich mehrere Ziele verfolgt. Zum einen sollen grundlegende Annahmen zum Wesen des systemischen Agierens im Allgemeinen und dem von Robotersystemen im Speziellen erörtert werden. Dabei wird zum einen unter „agieren“ weit mehr verstanden als nur bloßes „handeln“, indem auch ein „tätig werden ...“, ein „wirken auf ...“, oder „eine bestimmte Rollen spielen ...“ aber vor allem ein „einwirken auf ...“ assoziiert wird. Zum anderen wird der Agilitätsansatz als eine Rahmen-theorie verstanden, die zwar zentrale Momente systemischer Aktivität beleuchtet, in die jedoch weitere Konzepte eingeflochten werden können. Die Interoperationstheorie bildet also ein Grundgerüst, das als Organisationsmuster so angelegt werden muss, dass andere Theorien in diese integriert und diese wiederum mit anderen kombiniert werden können. Da mit der Interoperationstheorie der Graben zwischen Grundlagen und angewandter

Wissenschaft überbrückt werden soll und in angewandten Arbeitsfeldern ein ganzheitlicher Ansatz eher förderlich ist, muss eine theoretische Basis geschaffen werden, die eine solche Integration und Kombination anregt und ermöglicht. Der theoretische Ansatz konzentriert sich dabei im Inneren auf jene Teile, die direkte Auswirkungen auf die Datenerhebung und eine spezifische Methodik haben. Insofern werden in den einzelnen Abschnitten reflektiert, welche Aspekte an Robotersystemen und wie diese analysiert werden sollen. Theorie wird hier also mit ihren Implikationen für die Methodologie dargestellt. Diese theoretischen Überlegungen sollen auch die wissenschaftstheoretische Position transparent machen. Insofern wird in diesem Kapitel ein temporärer, verstehtender Standpunkt im Sinne der analytischen Philosophie eingenommen, der mit systemtheoretischem Gedankengut angereichert wird. Schließlich sollen in diesem Theorieteil auch ein sprachliches System bzw. ein sprachliches Vokabular erarbeitet und einige definitorische Regelungen getroffen werden. So darf es nicht beliebig sein, was man als Interoperation, als erweiterter Handlungsbegriff, bezeichnet und wie dieser operationalisiert wird.

Eine *interoperationsbasierte Robotik* versucht, systemische Intelligenz „von unten“, das heißt, von einfachen Bewegungs- und Orientierungsleistungen ausgehend, zu erfassen. Dazu baut sie unter anderem nach der natürlichen Evolution abgeschauten Prinzipien solche Robotersysteme, die in der Lage sind, selbstständig Aufgaben zu lösen. Die Biologie, und dort speziell die Bionik, fungiert dabei als zentraler Kreativitätspool, sozusagen als Ideenlieferant. Mit dem Begriff „*interoperationsbasiert*“ soll zum Ausdruck gebracht werden, dass der Ansatz der wissensbasierten Systeme eine wesentliche Erweiterung erfährt. Dieser erweiterte Ansatz geht davon aus, dass systemische Intelligenz nicht durch Repräsentationssysteme mit möglichst großer Rechenleistung alleine erreicht werden kann. Vielmehr setzt der Ansatz am anderen Ende der kognitiven Leistungen an, bei der Orientierung im Raum, beim Erkennen der Umwelt und bei der Bewegung des Systems. Intelligenz entsteht dieser Ansicht nach nicht aus einer zentralen Recheneinheit, sondern aus dem Zusammenspiel vieler einfacher Strukturen. Das besondere an diesem Ansatz ist aber auch, dass er sich nicht mit Simulationen begnügt, sondern auf den Bau von konkreten Robotersystemen setzt. Insofern bewegen sich diese Systeme in mehr oder weniger natürlichen Umwelten. Sie nehmen ihre Umwelt durch Sensoren wahr und orientieren ihr Verhalten an diesen Wahrnehmungen. Sie sind eben nicht mit einem starren Plan der Welt und einer Liste von Aufgaben ausgestattet, die nacheinander abzuarbeiten wären, sondern sie müssen sich im Hier und Jetzt einer sich moderat verändernden Welt zurechtfinden, müssen selbst entscheiden, wann sie eine Aufgabe erledigen oder wie sie ein Problem lösen. Man kann durchaus die Ansicht vertreten, dass diese Robotersysteme situiert und verkörpert sind.

Die Erweiterung dieses Ansatzes zeigt sich aber auch darin, dass die Robotersysteme das durch die wissensbasierten Teilsysteme zur Verfügung gestellte, spezialisierte Wissen, das eher in die Tiefe als in die Breite geht, auch über Kompetenzen auf einem gewissen Niveau verfügen. Sie interoperieren, anstelle nur zu reagieren. Insofern sind sie durch diesen erweiterten und kombinatorischen Ansatz auch in der Lage, zwischen den wichtigen Aspekten der Welt und den eher unwichtigen Artefakten zu unterscheiden, was letztlich

dazu führt, dass sie Unterscheidungen zu treffen vermögen, die für sie selbst bedeutungsvoll sind, nicht nur für ihre Anwender. Der Ansatz, der in diesem Buch entwickelten Interoperationstheorie, zielt somit auf das Wissen *und* auf das Verhalten der artifiziellen Systeme.

Insofern kann die Verortung, sozusagen der Sitz der Intelligenz eines Systems, demnach nicht in einer zentralen Komponente, einem wie auch immer gearteten Gehirn lokalisiert werden, sondern ist vielmehr auf das ganze System verteilt. Ein solches intelligentes Robotersystem ist zweifellos dynamisch komplex und selbstorganisierend. Es liegt also nahe, systemtheoretische Methoden und Erkenntnisse auch in der Interoperationstheorie wiederzuverwenden. Gerade die Theorie der dynamischen Systeme als ein mathematisches Werkzeug ist geradezu prädestiniert, um solche dynamischen, sich in der Zeit verändernden *Interoperationsmuster* zu beschreiben. Ein System aus dieser Perspektive ist zunächst eine Sammlung aus untereinander verbundenen Teilen, die als ein Ganzes wahrgenommen werden, etwa das Robotersystem an sich. Im Gegensatz zu dem Dezimalsystem, dass ein statisches System ist, ein System mathematischer Lehrsätze, die ewige Gültigkeit beanspruchen, handelt es sich beim intelligenten Robotersystem um ein System, das sich im Laufe der Zeit verändert: Es ist somit ein dynamisches System. Man kann nun, um der Dynamik einigermaßen Herr zu werden, alle Lösungen der ein dynamisches System beschreibenden Funktion zugleich betrachten. Statt eines einzigen Vektors in einem n-dimensionalen Koordinatensystem enthält man auf diese Weise eine Spur. Diese Spur wird *Trajektorie* oder auch *Lösungskurve* genannt. Die Menge aller Trajektorien eines Systems bilden dessen Phasen- oder Flussdiagramm. Es beschreibt das Verhalten eines Robotersystems zu allen möglichen Anfangsbedingungen. Ein dynamisches System besteht aus einer Reihe von Zuständen und einer Bewegungsgleichung, die angibt, wie sich diese Zustände ändern. Der Raum, den diese Trajektorien durchmessen, wird *Phasen- oder Zustandsraum* genannt. Punkte in diesem Zustandsraum werden als Vektoren beschrieben. Ein Zustandsvektor gibt die numerischen Werte an, die die Systemvariablen zu einem bestimmten Zeitpunkt haben. Die Variablen verändern sich im Laufe der Zeit entweder kontinuierlich oder diskontinuierlich. Nur im ersten Fall bildet die Systemgeschichte bzw. der Lebenszyklus eine Spur in der Zeit, eine Trajektorie.

Von zentraler Bedeutung für die Geschichte eines Robotersystems als dynamisches System sind demnach diese *Attraktoren*. Ein Attraktor ist eine Region im Phasenraum, die Trajektorien anzieht, sie also dazu bringt, sich dem Attraktor zu nähern. Es gibt einfache, punktförmige Attraktoren, Fixpunktattraktoren genannt, die bewirken, dass sich benachbarte Trajektorien diesem Punkt annähern.

Ein Beispiel dafür ist eine Kugel, die in einer Schüssel zum Rollen gebracht wird und nach einiger Zeit ruhig am Boden der Schüssel liegen bleibt, ein anderes ist das Pendel einer Uhr, das am tiefsten Punkt seiner Bahn zum Stillstand kommt, wenn es nicht mehr von den Gewichten des Uhrwerks in Schwung gehalten wird.

Systeme, die in einem Fixpunktattraktor starten, verharren in ihrer ursprünglichen Position. Solche Ruhezustände sind stabil, nach kleineren Störungen stellen sie sich wieder ein.

Ein zyklischer Attraktor stellt sich in einem Diagramm wie ein geschlossener Kreis dar. Er steht zum Beispiel für stabile Schwingungen wie die eines Pendels, solange es aufgezogen wird.

Neben den Fixpunktattraktoren und den zyklischen Attraktoren gibt es noch die chaotischen Attraktoren. Chaotische Attraktoren bilden in Diagrammen ausgesprochen komplizierte Muster, die die Eigenschaft der Selbstähnlichkeit aufweisen, ein Phänomen, das an den so genannten Mandelbrotmännchen bekannt geworden ist: Ein noch so kleiner Ausschnitt aus dem Gesamtmuster hat dieselbe Struktur wie das große Muster insgesamt. Das Verhalten von chaotischen Systemen ist nicht vorhersagbar. Starten zwei Trajektorien dicht beieinander, nehmen ihre Bahnen aufgrund winziger Unterschiede in den Startbedingungen schon in kürzester Zeit einen völlig verschiedenen Verlauf. Da man die Unterschiede in den Startbedingungen nicht beliebig genau messen kann, ist dieses Verhalten zwar deterministisch, aber nicht vorhersagbar. Solch geartete Probleme und Systeme sind Gegenstand der Chaostheorie.

Als *Bifurkation* bezeichnet man den plötzlichen Übergang eines Systems von einem Attraktor zu einem anderen. Bifurkationen entstehen, wenn sich Parameter eines Systems ändern, etwa wenn man einen rollenden Reifen eines Robotersystems von der Seite touchiert oder ein Pendel in Schwingung versetzt. Bifurkationen können etwa bewirken, dass sich ein stabiler Attraktor in mehrere stabile oder instabile Attraktoren verzweigt, ein zyklischer Attraktor etwa könnte seine geschlossene Kreisform verlassen und zu einer Spirale werden.

Die Theorie der dynamischen Systeme wird in der Interoperationstheorie vor allem zur Modellierung von motorischen Leistungen, insbesondere der Bewegungskontrolle verwendet. Dabei wird der Korpus des Robotersystems zunächst als Black Box eines dynamischen Systems verstanden. Als eine solche Black Box stellt das dynamische Robotersystem durchaus eine dramatische Vereinfachung der eventuell tatsächlich ablaufenden Prozesse dar. Die sprunghaften Fortschritte beim Lernen etwa werden als Bifurkationen beschrieben, Laufbewegungen durch zyklische Attraktoren. Spätestens an dieser Stelle interessiert man sich für die Steuerung mit vielen Freiheitsgraden. Auch hierzu liefert die Systemtheorie mit dem von Hermann Haken formulierten Prinzip der Versklavung einen wichtigen Beitrag. Diesem Konzept zufolge dominieren wenige Systemvariable die restlichen, so dass es ausreicht, diese wenigen zu kennen, um das Verhalten des Systems zu bestimmen.

Aber auch in den eher klassischen Themen, wie etwa der Begriffsbildung und dem Lernen, findet die Theorie der dynamischen Systeme Verwendung. So kann etwa das Attraktorbecken, die Region um einen Attraktor herum, in dem dieser Trajektorien anzieht, als der unscharfe Rand eines Begriffs verstanden werden, ähnlich wie die Prototypen in konnektionistischen Systemen.

Die Anwendung der Theorie der dynamischen Systeme auf artificielle kognitive Phänomene ist neben ihren Vorzügen als analytisches Instrument vor allem deshalb interessant, weil sie eine neue Perspektive auf die artificielle Kognition eröffnet. Sie beschreibt kognitive Phänomene mit denselben mathematischen Werkzeugen der Systemtheorie, wie

die Steuerungskomponenten, den Korpus und die Umwelt.<sup>41</sup> Dies birgt die Möglichkeit, das artifizielle System, das Robotersystem, seinen Korpus und seine Umwelt als Teilsysteme eines einzigen dynamischen Gesamtsystems zu betrachten. Damit ist zumindest das mathematische Werkzeug vorhanden, das artifizielle System des Roboters nicht länger als körperlose Fiktion zu betrachten.

An dieser Stelle kann man nun die erkenntnistheoretische Position des Radikalen Konstruktivismus beziehen, ohne dass die Interoperationstheorie philosophisch und theoretisch „verwässert“ wird. Denn in der Tat bilden intelligente Robotersysteme ihre Umwelt nicht nur ab, sie konstruieren eine eigene, durch die Gegebenheiten des Systems wesentlich mitbestimmte Realität. Sie optimieren immer ihren eigenen Zustand. Sie fragen aktiv Informationen ab, statt sie passiv abzubilden. Der Einfluss der äußeren Welt reduziert sich zu einer Störgröße, einer „Perturbation“, einem externen Rauschsignal, das die kognitiven Komponenten des Robotersystems zwingen, den angestrebten Zustand ständig neu herzustellen. Das Gütekriterium dieser Form von Daten-, Informations- und Wissensverarbeitung, die Höhe des systemischen Intelligenzquotienten ( $IQ_s$ ) ist nicht die Wahrheit im Sinne einer Entsprechung äußerer Realität und innerer Abbildung, sondern allein der Erfolg, mit dem es dem Robotersystem gelingt, sein Problem zu lösen bzw. sein Verhalten in seiner Umwelt zu organisieren.

Zentral für den systemtheoretischen Zugang der Interoperationstheorie ist somit, dass Robotersysteme nicht isoliert betrachtet werden können, sondern dass das Zusammenspiel mit den sie umgebenden und sie beeinflussenden Systemen berücksichtigt werden muss. Robotersysteme haben demnach nicht nur Körper, sie funktionieren auch in konkreten Umwelten und in konkreten Situationen. Dieser Aspekt wird als *Situiertheit* bezeichnet. Zur Umwelt eines Robotersystems gehören in der Regel auch andere Robotersysteme, so dass sich hier ein Ansatz zur Modellierung kommunikativer Interoperationen und ihrer Bedeutung für die individuelle artifizielle Kognition ergibt.

Der dynamische Zugang betrachtet die Fähigkeit zur Selbstorganisation in Wechselwirkung mit der Umwelt als das zentrale Merkmal eines intelligenten Robotersystems. Dies hat weitreichende Implikationen auf den Entwicklungsprozess solcher Systeme, indem man solche dynamischen Systeme nicht einfach wie ein Fahrrad aus seinen Teilen zusammenschrauben kann. Vielmehr kann man ihre Entwicklung lediglich anstoßen und sich dann in vielen Momenten überraschen lassen, was dabei herauskommt. Trotz dieser nebulösen Unsicherheit ist dieser Ansatz besonders geeignet für Systeme, die autonom bestimmte Aufgaben in sich verändernden oder nicht genau bekannten Umwelten erfüllen sollen. Autonomie bedeutet, vereinfacht gesagt, dass das Verhalten des Systems nicht ferngesteuert sein darf, sondern vom System selbst organisiert werden muss. Ein Nebeneffekt solcher Systeme ist, dass sie zu bestimmten Zeitpunkten emergentes Verhalten zeigen, ein Verhalten also, dass man aufgrund der Ausstattung des Systems und seiner Programmierung im Voraus nicht erwartet hätte. Streng betrachtet ist ein solches emergentes Verhalten ein solches, das aus seinen Anfangs- bzw. Entstehungsbedingungen nicht unbedingt erklärt werden kann.

---

<sup>41</sup> Siehe auch Kratky 1989.

### 2.5.3 Interoperationsbegriff

Die Interoperation ist demnach der zentrale Begriff dieser Theorie, um systemisches Tun zu charakterisieren. Man versteht Interoperation als eine wesentliche Charakterisierung eines Systems und behandelt einen Agens in seinem Tun als System. Im Sinne einer allgemeinen Systemtheorie impliziert dies:

- Interoperation ist eine basale Einheit,
- diese Einheit hat verschiedene *Elemente*, die durch das System zu einer
- *Ganzheit* organisiert und integriert werden.
- Ein solches Interoperationssystem steht in Interrelation zu seiner *Umwelt*.

Wenn an dieser Stelle die Interoperation als basale Einheit bezeichnet wird, soll damit zum Ausdruck gebracht werden, dass im Folgenden von dieser Einheit ausgegangen wird. Das System kann in der Folge in seine Elemente bzw. Untersysteme differenziert werden, und es lassen sich die Relationen zu anderen Systemen identifizieren. Interoperation wurde auch daher als basale Einheit gewählt, da man damit über eine Grundlage verfügt, um verschiedene theoretische Aspekte des Robotersystems zu integrieren. Immerhin gilt ein Robotersystem als höchst komplex. Mit dieser Konzeption von systemischer Interoperation wird in diesem Buch daher ein weiterer Versuch unternommen, sich dieser Komplexität zu nähern. In die Basiseinheit Interoperation wirkt das System als Ganzes ein.

Das intelligente Robotersystem ist weiterhin fähig, seine Interoperationen gleichzeitig an innere Bedürfnisse und Anforderungen der Umwelt anzupassen. Dabei wird der Begriff *Umwelt* hier nicht in einem ökologischen, sondern im systemtheoretischen Sinn verwendet. Das System ist ein Teil (in) der Welt, in der es interoperiert. Das System interoperiert und „erlebt“ sich letztlich nur in Relation zu seiner (Um)Welt. Den Begriff der Welt soll im Sinne von Phänomenologie und Hermeneutik verstanden werden. Mit dem Begriff Umwelt wird dabei zu analytischen Zwecken eine Trennung zwischen einem System und seiner Umgebung möglich. In der Folge davon wird die Differenzierung zwischen unterschiedlichen Bezügen von Aktivitäten möglich. Dies ist jedoch keine ontologische Größe, sondern eine perspektivische, systemtheoretische Unterscheidung.

Interoperation ist zunächst etwas *Konkretes*. Insofern verfolgt die Theorie einen Ansatz, in deren Rahmen man manifeste systemische Aktivität studieren kann. In dem angewendeten Sprachgebrauch, der sich bewusst eng an das Alltagsverständnis lehnt, bezeichnet die Interoperation eine Handlung, die konkrete systemische Tätigkeit. Das Verb „interoperieren“ wird bei der Abstraktion über Interoperation im erweiterten Sinne verwendet, indem darunter „handeln“, „tätig sein“, „wirken“, „einwirken“ etc. eines Systems verstanden wird. Dies impliziert aber auch, dass ein wie auch immer geartetes „interoperieren“, auch ein „tätig werden als ...“ beinhaltet, d. h. dass das System in seinem Interoperieren eine bestimmte Rolle darstellt oder aktiv einnimmt. Insofern kann man unter Interoperation ein konkretes Vollziehen eines Systems verstehen.

Bei der systemischen Interoperation spielt die *Perspektive des Agens* eine wesentliche Rolle. Man kann dies auch so ausdrücken, dass Interoperation system-*subjektiv* bestimmt ist. Insofern zeichnet sich die Interoperation durch eine system-subjektive Antizipation von Zielen aus. Eine Interoperation ist somit eine systemische Aktivität, die zielgerichtet, geplant und intendiert ist, vom Agens wahrgenommen wird, von den Merkmalen ihres Gegenstandes (Aufgabe, Objekt) und von verschiedenen kognitiven Repräsentationen mitbestimmt wird. Diese artifiziellen Kognitionen werden durch Technologien der Artifiziellen Intelligenz, des Artifiziellen Lebens und des Cognitive Computings realisiert. *Artifizielle Kognition* besteht stets darin, die Welt hinsichtlich einer bestimmten Seinsweise bzw. eines bestimmten Aspekts, zu konstruieren oder zu repräsentieren. Dieses Konzept einer Kognition impliziert drei ontologisch, wie erkenntnistheoretisch folgenreiche Annahmen:

- Die Welt ist vorgegeben.
- Die artifizielle Kognition bezieht sich auf diese Welt – wenn auch oft nur auf einen Teilbereich derselben.
- Die Art, auf die diese vorgegebene Welt erkannt wird, besteht darin, ihre Merkmale abzubilden und sodann auf der Grundlage dieser Abbildungen zu agieren.

Insofern dienen die Komponenten der artifiziellen Kognition gemäss dieser Auffassung also vor allem dem Hervorbringen von Welten im Prozeß des variablen Lebenszyklus von Robotersystemen. Die kognitiven Komponenten sind damit technische Systeme, die Welten festlegen, keine Welten spiegeln.

Der Grundgedanke dieser Definition ist, dass beim Interoperieren verschiedene Elemente systemischer Artefakte innerhalb des Lebenszyklus eines Systems zu einer Einheit zusammengefasst werden. Diese Einheit besteht aus einem äußeren, systemischen und damit manifesten Anteil und einem inneren, kognitiven Anteil. Man kann auch von einem extern wahrnehmbaren Aktivitätsanteil und kognitiven Repräsentationen sprechen. Diese Einheitsbildung von äußerer Aktivität und Kognition hat wesentliche begriffliche und methodologische Konsequenzen für die systemische Interoperationsforschung. Eine dieser Konsequenz ist es, dass man ein systemisches Verhalten nicht von der artifiziellen Kognition trennen kann. Diese Einheit von äußerer Aktivität und Kognition impliziert darüber hinaus folgende Merkmale:

- Das Robotersystem ist ein ganzheitliches System: Diese Auffassung enthält keine Hinweise, dass die äußere Aktivität der Kognition über- oder untergeordnet sei. Beide Anteile der Aktionseinheit sind gleichwertig. Sie stehen in Wechselbeziehung. Der Interoperationsbegriff fasst also äußere Aktivität und Kognition zu einer Einheit zusammen.
- Systemische Aktivität und artifizielle Kognition sind Bestandteile des Oberbegriffs Interoperation. Interoperation darf folglich nicht mit einem äußeren Anteil, der Aktivität, gleichgesetzt werden. Diese Konzeptionalisierung sieht daher die artifizielle Kognition als Anteil der Interoperation vor.

- Vor allem die Sprechakttheorie hat deutlich gemacht, dass Sprechen durchaus eine Form von Interoperation bzw. Handeln darstellen kann. Damit grenzt sich diese Theorie bewusst gegen die Auffassung ab, bei der Interoperation ausschließlich als etwas Nichtverbales betrachtet wird.

Im Folgenden sollen noch die Bestimmungsstücke der o.a. Definition der Interoperation präzisiert werden. So ist mit einer Interoperation der nach außen hin wahrnehmbare und damit manifeste Interoperationsanteil gemeint. Aktivität besagt auch, dass im Interoperationsgedanken das Element des *aktiven Systems* steckt. Aus diesem Grund wird auch von einem Agens gesprochen und nicht von einem Aktor oder Rektor.

Die *Zielgerichtetheit* der systemischen Interoperation ist dadurch gegeben, dass eine solche Interoperation auf relative Endzustände hin ausgerichtet und organisiert ist. Nur solche manifesten Aktivitäten werden daher als Interoperation bezeichnet, für die eine kognitive Repräsentation eines Ziels inferiert ist. Die Zielkognition ist mit einer konkreten Interoperation verbunden, das heißt, Ziele sind nicht irgendwelche ideellen oder evaluativen Vorstellungen, wie man sie aus dem menschlichen Bereich etwa kennt. Ziele als Endzustände sollten mit einer konkreten Interoperation erreichbar sein. Sie beinhalten Vorstellungen vom Weg und dem Endzustand einer Interoperation und repräsentieren diese Vorstellung als etwas potentiell Anzustrebendes. Die kognitive Repräsentation eines Ziels oder verschiedener Ziele besagt noch nicht, ob dies, oder welches schließlich intendiert wird. Ziele sind nicht etwas Statisches, sondern besitzen Prozesscharakter. Das Ziel vor Beginn einer Interoperation muss nicht identisch mit dem schließlich realisierten sein. Durch Rückmeldung während der Interoperation kann sich das Ziel ändern, es wird möglicherweise ergänzt oder umformuliert; es braucht auch nicht immer erreicht zu werden.

Der Begriff der *Planung* bezeichnet den Sachverhalt der Antizipation oder des Entwurfes des ganzen Interoperationsweges oder einzelner seiner Teile. Je nach Kenntnis und Komplexität der Interoperation kann mehr oder weniger detailliert und umfassend geplant werden.

Das *artifizielle* kognitive System ist der Ort für höhere Prozesse der Daten-, Informations- und Wissensverarbeitung. Es ist die Verrechnungsstelle für verschiedene intervierende Größen. Das kognitive System ist hierarchisch aufgebaut. Danach müssen und können nicht alle Informationen in die fokale Aufmerksamkeit des artifiziellen Bewusstseins treten. Unter artifiziellen kognitiven Repräsentationen versteht man alle Daten und Informationen, die das interoperierende System von seinen Untersystemen wie Sensoren, Aktoren, Entscheidungssystem, Wertesystem, Mustererkennungssystem, Speicher (= Gedächtnis), etc. erhält. Insofern handelt es sich bei der Kognition um einen integrativen Oberbegriff. Er entspricht nicht dem Begriff der Rationalität oder des Denkens, sondern ist breiter gefasst.

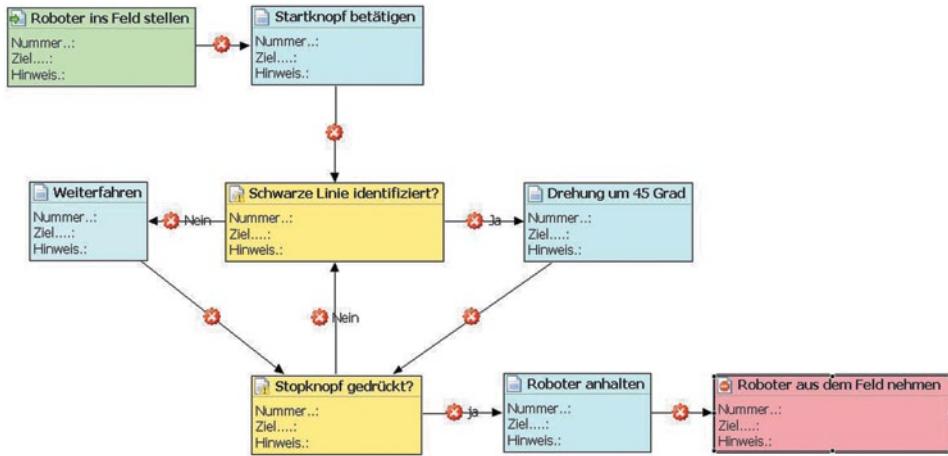
Insgesamt ist die systemische Interoperationstheorie eine Rahmentheorie, die sich primär mit der Organisation der Interoperationsausführung beschäftigt. In diese Organisation sind verschiedene andere Größen wie beispielsweise die Navigation oder die Kommunikation miteinzubeziehen, wofür die Konzeption von einer artifiziellen Kognition den Rahmen hierfür liefert.

So stellt gerade für mobile Robotersysteme das Navigieren in einer dynamischen Umwelt die zentrale Fähigkeit dar. Primäre Voraussetzung hierfür ist zunächst die Aufrechterhaltung der eigenen Betriebsfähigkeit, d. h. auch die Existenz gefährdenden Situationen, wie Zusammenstöße zu vermeiden und akzeptable Betriebsbedingungen aufrecht zu erhalten (Temperatur, Strahlung, Witterungsschutz usw.). Soll das Robotersystem gleichzeitig Aufgaben ausführen, die sich auf bestimmte Orte in der Umgebung beziehen, muss es navigieren können. Systemische Navigation kann demnach als eine Kombination der folgenden drei Kompetenzen definiert werden:

- Lokalisierung des Robotersystems
- Planung von Routen
- Umweltkartenerstellung und deren Interpretation

Der Begriff *Umweltkarte* steht in diesem Zusammenhang für jede direkte Zuordnung oder Abbildung von Artefakten der wirklichen Welt auf Einheiten innerhalb einer internen Repräsentation. Diese Repräsentation gleicht nicht unbedingt einer gewöhnlichen Landkarte oder einem Stadtplan. So lassen sich beispielsweise Karten in Form von neuronalen oder semantischen Netzen oder Topic Maps realisieren.

*Lokalisierung* steht für die Kompetenz des Robotersystems, seine eigene Position innerhalb eines solchen Bezugssystems feststellen zu können. *Routenplanung* ist im Prinzip eine Erweiterung der Lokalisierung, da das Robotersystem seine gegenwärtige Position und die angestrebte Zielposition innerhalb ein und desselben Bezugssystems bestimmen muss. Die Kompetenz des *Kartenerstellens* bezieht sich nicht nur auf Karten, wie sie gewöhnlich zur Anwendung kommen, d. h. metrische Karten der Umgebung, sondern auf jegliche Art der Beschreibung von Positionen innerhalb des Bezugssystems. Dazu gehört auch, Informationen über erkundetes Terrain zu speichern. Schließlich benötigt man für den Umgang mit Karten auch die Fähigkeit, diese zu interpretieren. Ein solches Bezugssystem ist die zentrale Komponente jeder Navigationskomponente eines Robotersystems, da es alle drei für die Navigation notwendigen Teilkompetenzen administriert. Das Bezugssystem ist der Referenzpunkt, in dem letztlich die Navigationskompetenz verankert ist. Das sicherlich einfachste Bezugssystem stellt dabei das kartesische Koordinatensystem dar. Für alle Abläufe innerhalb eines solchen kartesischen Koordinatensystems, also für Lokalisierung, Routenplanung und Kartographie, werden sämtliche Positionen als Koordinaten ( $x$ ,  $y$ ,  $z$ -Punkte) in diesem Bezugssystem dargestellt. Navigation gestaltet sich fehlerfrei. Wenn der Ausgangspunkt einer Route sowie Geschwindigkeit und Fahrtrichtung genau bekannt sind, kann die Navigationskomponente einfach die Bewegung des Robotersystems über die Zeit integrieren: diesen Vorgang bezeichnet man dann als *Koppelnavigation*. In einem kartesischen Bezugssystem können die Position eines Robotersystems sowie alle weiteren relevanten Positionen stets präzise bestimmt werden. Allerdings besteht ein wesentliches Problem darin, dass das kartesische Bezugssystem selbst nicht in der realen Welt verankert ist, d. h. das System an sich ist beweglich im Bezug auf Positionen in der realen Welt. Damit Koppelnavigation bzw. Odometrie funktionieren kann, muss das Robotersystem



**Abb. 2.20** Navigation im Modell

seine Bewegungen absolut präzise messen können, was unter Umständen aufgrund von Problemen, wie beispielsweise Gleiten der Räder – eine Bewegung, die nicht *in* dem Bezugssystem, sondern *an* dem Bezugssystem geschieht – erschwert wird. Das klassische Robotersystem ist allein auf interne Messungen angewiesen, um seine eigenen Bewegungen festzustellen, kann also Veränderungen am Bezugssystem selbst nicht wahrnehmen.

Navigation muss jedoch in der wirklichen Welt erfolgreich sein, nicht allein innerhalb eines abgeschirmten Bezugssystems. Daher sind solche Driftfehler, wobei das Bezugssystem immer weniger mit der wirklichen Position in der realen Welt zu tun hat, ein ernsthaftes und damit zu lösendes Problem. In der Praxis ist die auf reiner Odometrie beruhende Navigation nicht zuverlässig genug und eignet sich nur für sehr kurze Strecken (Abb. 2.20).

Navigation einer dynamischen Umwelt in Echtzeit muss daher das Problem der Propriozeption lösen. Eine Lösung besteht darin, die Navigation direkt in der wirklichen Welt zu realisieren (Exterozeption), statt in einem internen Bezugssystem. Charakteristische Umgebungsmerkmale, die ein Robotersystem wahrnehmen kann, stellen sogenannte *Landmarken* dar. Unter einer solchen Landmarke versteht man meist auffällige, permanent wahrnehmbare und damit identifizierbare Merkmale der Umgebung.

Einige Beispiele sind Gebäude und markante Landschaftsmerkmale, wie Berge oder Seen.

Um diese Landmarken zu identifizieren, braucht man Hintergrundwissen. Nur wenn man weiß, wie eine Brücke normalerweise aussieht, kann man auch eine spezielle Brücke in der Umgebung als Brücke identifizieren. Man kann den Begriff Landmarke aber auch anders definieren, wenn man ihn allein in der reinen Sensorwahrnehmung statt in der Interpretation dieser Wahrnehmung gründet.

Beispiele dafür wären die Helligkeit des Himmels, also die Sonnenposition, Windrichtung oder ein Geräusch aus einer bestimmten Richtung.

Alle diese Merkmale sind ortsabhängige Wahrnehmungen, die von der Navigatorkomponente selbständig aufgenommen und für Kursbestimmung oder Kurskorrektur verwendet werden können, ohne abstrakte Objektanalyse. Sie können als Ankerpunkte der Navigation dienen (Piloting). Dabei wird die Route zu einer Zielposition nicht durch Routenintegration wie bei der Koppelnavigation bestimmt, sondern dadurch, dass man Landmarken oder Sequenzen von Landmarken identifiziert und diese Landmarken entweder in einer bestimmten Reihenfolge ansteuert, oder bei jeder identifizierten Landmarke der bei früheren Durchgängen gespeicherten korrekten Kompassrichtung folgt. Vorausgesetzt, dass das Robotersystem alle Landmarken eindeutig im Zugriff hat und identifizieren kann, lässt sich eine Realworld and Realtime-Navigation durchführen und dadurch das Problem unkorigierbarer Driftfehler vermeiden. Wenn eine Roboter navigationskomponente auf Landmarkenerkennung aufgebaut werden soll, muss zunächst definiert werden, wodurch sich eine brauchbare Landmarke auszeichnet. Um als Anhaltspunkt für Navigation zu dienen, muss eine Landmarke die folgenden Bedingungen erfüllen.

Die Landmarke muß

- von verschiedenen Standorten aus sichtbar sein,
- bei unterschiedlichen Lichtverhältnissen, Perspektiven usw. erkennbar sein,
- entweder während des gesamten Navigationsvorgangs stationär sein,
- oder ihre Bewegung muss dem Navigationsmechanismus bekannt sein.

Die beiden ersten Bedingungen erfordern, dass die Navigationskomponente die Landmarke intern vereinfacht bzw. verallgemeinert darstellen kann. Unbehandelte Sensordaten wären in diesem Fall ungeeignet, weil das Robotersystem mit größter Wahrscheinlichkeit dieselbe Landmarke beim nächsten Durchgang leicht anders wahrnimmt, sie also ohne Verallgemeinerung nicht wiedererkennen würde. Die erste Bedingung impliziert aber auch, dass, wenn das Robotersystem gelernt hat, eine Landmarke von einem bestimmten Standort aus zu erkennen, dieselbe Landmarke dann auch von einem anderen Standort aus erkennen wird, selbst wenn es die Landmarke von dort zuvor noch nicht gesehen hat.

Ein grundlegendes Merkmal der klassischen Navigation ist die Verwendung einer Vielzahl verschiedener Informationsquellen. Eine intelligente Navigatorkomponente verlässt sich nicht auf einen einzigen Mechanismus, sondern kombiniert die verschiedensten Informationen, um sich dann für einen bestimmten Kurs zu entscheiden.

Beispiele für mehrfache Informationsquellen kann man der Bionik entnehmen, so beispielsweise der magnetische und der Sonnenkompass bei der Vogelnavigation, Wellenmuster, Meeresflora und -fauna sowie Wetterbeobachtungen beim Menschen, oder Sonnenstand und Landmarken bei Insekten.

In der mobilen Robotik kommen bisher eine Vielzahl von individuellen Navigationskompetenzen und -mechanismen zum Einsatz. Wenn man diese wie bei den biologischen Navigatoren kombiniert und parallel verwendet, lassen sich die Navigationskomponenten für Robotersysteme noch zuverlässiger und robuster realisieren. Ebenfalls zeigt die Bio-

nik, dass die Stärke der biologischen Navigatoren darin liegt, dass auch sie eine Vielzahl von Informationsquellen kombinieren können. Diese Kombination von Information ist auch ein vielversprechender Ansatz in der modernen Roboternavigation. Jedes Navigationsprinzip hat auf sich allein gestellt charakteristische Stärken und Schwächen, und nur durch die Kombination der Prinzipien kann man die jeweiligen Nachteile überwinden und gleichzeitig alle Vorteile ausnutzen. So haben Koppelnavigationssysteme den gravierenden Nachteil der Driftfehler, die ohne externe Messungen nicht korrigiert werden können. Da landmarkenbasierte Navigation auf Exterozeption basiert, entstehen keine unkorrigierbaren Driftfehler, stattdessen sind hier perzeptuelle Ungereimtheiten (Kongruenz) das Problem. Perzeptuelle Kongruenz ist allerdings nicht in allen Fällen unerwünscht: es handelt sich dabei immerhin um eine verallgemeinerte, also speichereffiziente Repräsentation der Robotersystemumgebung. Schwierigkeiten entstehen erst dann, wenn zwei identisch aussehende Orte verschiedene Interoperationen des Robotersystems erfordern. Hybride Systeme versuchen, die positiven Aspekte beider Ansätze beizubehalten und gleichzeitig deren Probleme zu umgehen. Unterschiedliche Umgebungen erfordern allerdings auch unterschiedliche Sensorverarbeitungsmethoden. Daher besteht das Risiko, dass hybride Ansätze schließlich zu sorgfältig auf genau eine bestimmte Umgebung abgestimmt sind. Um einen breiten Einsatzbereich zu erreichen, müssen hier Lernfähigkeit- und Selbstorganisationmechanismen entwickelt werden.

*Kommunikation* dient der Übermittlung von Nachrichten in Form von Mitteilungen, Informationen, etc. Der Inhalt einer Nachricht wird vom Kommunikatorsystem vercodet. Sie enthält für das Empfängersystem eine Bedeutung. Dabei müssen sich übermittelte und decodierte Bedeutungen nicht in jeder Hinsicht decken. Eine Mitteilung kann intendiert sein, oder auch nur vom Empfängersystem als intendiert betrachtet werden.

So beispielsweise bei einer Bewegung, die als Drohung verstanden wird, obwohl sie für den Agens eine andere Funktion hatte.

Es ist für den Kommunikationsbegriff nicht konstituierend, dass Nachrichten zwischen mehreren Agenten wechselseitig ausgetauscht werden, auch wenn dies häufig der Fall ist. Auch unter Interoperation wird laut obiger Definition eine *Wechselbeziehung* oder gegenseitige Beeinflussung zwischen mindestens zweier Robotersystemen verstanden. Während bei der Kommunikation allerdings der zweite Agens nicht unbedingt auf den ersten einwirken muss, gehört es zu den konstruierenden Bedingungen von Interoperation, dass der zweite Agens Einfluss auf den ersten ausübt, indem er auf diesen einwirkt. Zur Veranschaulichung dieser für die Modellierung wichtigen Unterscheidung ein Beispiel:

Ein Robotersystem A richtet bei einer Distanz von acht Metern zu einem Robotersystem B an dieses eine Anfrage. B nimmt diese Frage wegen sonstiger Geräusche (Umweltrauschen) nicht wahr. Er wendet weder seinen Kopf in A's Richtung, noch gibt er eine Antwort. Da man hier keinen Bezug von B auf A feststellen kann, fand in diesem Fall keine Interoperation zwischen A und B statt. Doch hat A mit B durchaus kommuniziert, da er eine Nachricht an B gerichtet hat. Hätte B hingegen diese Anfrage gehört und beantwortet, würde man von einer Interoperation sprechen.

Die Aktivitäten bei der Interoperation bestehen zum großen Teil aus kommunikativen Akten. Kommunikation und Interoperation stellen weitgehend Schnittmengen dar, sind jedoch nicht deckungsgleich.

Aktion und Interoperation stehen in einer Beziehung zueinander, wobei nicht jede Form von Interoperation eine interoperative Handlung darstellt. Nicht jeder interoperative Akt erfüllt die Definitionselemente einer Aktion. Bei der Interoperation findet man mindestens zwei Agens, wobei jeder von ihnen eine Aktion vollzieht. Nun können diese Aktionen für kurze Zeit zusammentreffen, oder sie können gemeinsam ausgeführt werden. Interoperative Aktion zeichnet sich innerhalb der systemischen Interoperationstheorie nun insofern aus, dass ein neues Aktionssystem dadurch entsteht, dass zwei Agenten ein gemeinsam geteiltes Ziel verfolgen. Diese Auffassung der interoperativen Aktion verlangt mehr als nur eine Verdoppelung system-individuellen Agierens. Aktionen eines Systems werden nicht dadurch interoperativ, dass sich in der Aktionssituation auch andere Systeme befinden, oder dass bei der Realisierung einer subjektiven Strategie interoperative Bezüge notwendig sind. Interoperative Aktionen sind vielmehr dadurch gekennzeichnet, dass bei den Interoperierenden im Bereich der artifiziellen kognitiven Organisation insbesondere hinsichtlich der Ziele Intersubjektivität und wechselseitige Perspektiveneinnahme gegeben sind. Insofern spricht man im Rahmen der systemischen Interoperationstheorie dann von einer *interoperativen Aktion*, wenn ein *intersystemisch geteiltes Ziel*, das in bestimmten Elementen *gleich* ist, in gemeinsamer Aktivität verfolgt wird.

Interoperative Aktionen führen zu systemischen *Kooperationsstrategien*. So können gemeinsam geteilte Ziele dadurch zustandekommen kommen, dass sich ein Agent bereit erklärt, das Ziel eines anderen Agenten zu übernehmen, oder dadurch, dass die Ziele ausgetauscht werden. Wesentlich für die Konstituierung gemeinsam geteilter Ziele ist, daß jeder Agent davon ausgeht, sich mit seinem Partner das Ziel zu teilen und dies gemeinsam zu verfolgen. Dass dabei im Verlaufe einer Aktion das Ziel neu definiert oder wechselseitig ausgehandelt werden muss, gehört zum Spielraum interoperativen Agierens. Einzelne interoperative Aktionen werden in dem hierarchischen Modell als *interoperative Aktions schritte* bezeichnet. Interoperative Aktionsschritte können auch in Aktionen auftreten, bei denen die Ziele der Agens nicht gemeinsam geteilt sind.

Ein Beispiel hierzu: Roboter A verlässt das Terrain am Ausgang O und will außerhalb dieses Terrains Strom auftanken. Roboter B geht durch den Ausgang O in das Terrain, um dort seine ihm zugewiesene Arbeit zu verrichten. Beide Robotersysteme begegnen sich am Ort O dabei, registrieren sich, A lässt B den Vortritt. Jeder der Robotersysteme verfolgt sein system-subjektives Ziel.

In der folgenden Tabelle werden die begrifflichen Differenzierungen verschiedener Formen des Agierens in interaktiven Situationen noch einmal zusammenfassend dargestellt (Tab. 2.1).

**Tab. 2.1** Formen des Interoperierens

Aktionstyp	Kognitive Organisation	Interoperationsform
Solitäre Aktion	System-subjektives Ziel	Keine Interoperation
Intersubjektive Aktion	System-subjektive Ziele, teilweise gemeinsame Planung und Koordination	Verschiedene Formen der Interoperation
Interoperative Aktion	Gemeinsam geteiltes Ziel	Gemeinsame Interoperationsausführung gemäß einer Kooperationsstrategie

### 2.5.4 Interoperationsanalyse

In diesem Abschnitt werden Beobachtungs- und Konstruktionsverfahren zur Interoperationsanalyse vorgestellt. Eine *Beobachtung* aus Sicht der Interoperationstheorie stellt eine Form der gerichteten Wahrnehmung auf einen Ausschnitt der Lebenswelt dar. Dabei wird per definitionem kein Wahrnehmungsorgan ausgeschlossen, noch die Verwendung technischer Hilfsmittel (Aufzeichnungsgeräte, Registrierapparaturen, Sensoren, etc.). Beobachtung als aktiver, geplanter Wahrnehmungsprozess bedarf im Rahmen der Interoperationstheorie im Gegensatz zum Vorgehen im Alltag die theoretische Fundierung des Erkenntnisinteresses des Entwicklers. Weiter legt der Entwickler sein Wahrnehmungsraster in Form eines Kategoriensystems offen dar. Dies ermöglicht einerseits ein systematisches Vorgehen und andererseits Stabilität, Kontrollierbarkeit, Reproduzierbarkeit und Intersubjektivität zwischen Wissenschaftlern über die Datengewinnung.

Robotersysteme agieren im Regelfall in Realitätsbereichen, die durch bestimmte Eigenschaften gekennzeichnet sind. Das Verhalten und das Agieren in komplexen Realitätsbereichen können nun natürlich ganz unterschiedliche Formen annehmen. Weiterhin muss man davon ausgehen, dass jeder Interoperation im Regelfall sowohl eine bestimmte Absicht als auch ein Ziel zugrundeliegt.

Um Interoperation modellieren zu können, müssen diese Sachverhalte einer Formalisierung zugeführt werden. Formal kann man den eben beschriebenen Sachverhalt ausdrücken: Ein Robotersystem befindet sich in einer bestimmten Situation, die als Anfangszustand  $S_0$  bezeichnet werden soll. Diesen Anfangszustand hält das Robotersystem aufgrund bestimmter Ursachen für veränderungswürdig. Zum anderen verfügt das System aber auch über mehr oder minder klar definierte Zielvorstellungen, die es für erstrebenswert hält. Solche Zielzustände werden mit  $S_\otimes$  bezeichnet. Das Ziel des Interoperationsprozesses ist also die Transformation des Anfangszustandes  $S_0$  in einen Zielzustand  $S_\otimes$ . Diese Transformation kann natürlich für das Robotersystem unterschiedlich schwer sein. Beispielsweise kann der Fall eintreten, dass die Transformation von  $S_0$  in einen Zielzustand  $S_\otimes$  durch Barrieren erschwert ist. Wenn durch eine Barriere die Anwendung einer definierten Interoperationssequenz unmöglich gemacht wird, liegt ein Problem vor. Das Agieren eines Systems in einer solchen Situation bezeichnet man als *systemisches Problemlösen*. In

Abhängigkeit von der Klarheit der Zielvorstellung und dem Bekanntheitsgrad der notwendigen Operatoren kann man dabei verschiedene Arten von *Barrieren* und damit von Problemtypen unterscheiden:

- *Interpolationsproblem*: Ist der Zielzustand klar definiert und sind die einzelnen Operatoren zur Zielerreichung bekannt und besteht die Schwierigkeit lediglich darin, die richtige Folge der einzelnen Operatoren zu finden, spricht man von einem Interpolationsproblem.

Beispielsweise verfügt ein Robotersystem beim Fußballturnier über einen Satz an Interaktionspotenzialen, die eine große Chance darstellen, um das Spiel zu gewinnen. Die Erreichung dieses Zielzustandes ist durch die Regeln des Spiels klar definiert. Die einzelnen Operatoren sind dem spielenden Robotersystem bekannt, sie liegen „buchstäblich auf dem Fuß“. Um nun zum Zielzustand zu gelangen, muss das System diese Operatoren richtig aneinanderreihen, es muss die Interaktionen also in einer bestimmten Reihenfolge anwenden und jeweils richtig auf die Spielzüge der Gegenspieler reagieren.

- *Synthese-Problem*: Ist die Klarheit der Zielkriterien sehr hoch, sind aber die Operatoren zur Transformation des Anfangszustandes in den Zielzustand relativ unbekannt, spricht man von einer Synthese-Barriere. Das System steht vor einem Synthese-Problem weil es sich die benötigten Operatoren erst aneignen, also „synthetisieren“ muss.

Beispielsweise befindet sich das Robotersystem in einem unerwünschten Anfangszustand der Gestalt, dass die zur Verfügung stehende Energie nicht mehr zur Aufrechterhaltung der Systemeigenschaften ausreicht. Das Robotersystem selbst verfügt in Form einer Checkliste über die Eigenschaften einer erstrebenswerten Energiesituation. Allerdings hat es im derzeitigen Zustand keine Information davon, wie es diesen Zielzustand auch erreichen kann. Seine Kenntnisse über die Operatoren zur Transformation des Anfangszustandes in den Zielzustand sind also relativ gering. Um das Problem „Energiegewinnung“ lösen zu können, muss es sich zunächst ein brauchbares Inventar von Operatoren aneignen.

- *Dialektisches Problem*: Sind die Operatoren zur Zielerreichung bekannt, aber die Zielkriterien selbst unklar, dann spricht man von einer dialektischen Barriere und somit von einem dialektischen Problem.

Beispielsweise hat ein Serviceroboter einige Gegenstände, Stühle und Tische, im direkten Zugriff. Allerdings hat er keine sehr klare Vorstellung darüber, wie diese endgültig im dafür vorgesehenen Raum zu platzieren sind. Insofern wird der Serviceroboter zunächst einmal damit anfangen, die Gegenstände von einer Ecke zur anderen zu rücken, sich Feedback von einem beobachtenden System einholen und sich somit langsam im Rahmen eines dialektischen Prozesses einem Zielzustand nähern, der dann vom einem beurteilenden System, einer Kontrollinstanz abgenommen wird.

Wenn sich also ein Robotersystem beim Agieren mit solchen Barrieren auseinandersetzen muss, spricht man von einem systemischen *Problemlösen*. Insofern ist für die spätere Modellierung mit dem Begriff Problemlösen eine erste Ebene der Interoperationsregulation definiert. Allerdings ist nicht jede Aktivität als Sammlung von Interoperationen, die einen Anfangszustand in einen gewünschten Endzustand überführt, Problemlösen in diesem Sinne. Ist nämlich ein Zielzustand klar definiert, sind die Operatoren zur Umwandlung des Realitätsbereiches und ihre Reihung bekannt, existiert also überhaupt keine Barriere, so liegt eine *Aufgabe* vor.

Beispielsweise kann man sich einen Serviceroboter vorstellen, der mit der Zubereitung von Spaghetti beauftragt wird. Er greift sich einen Topf, füllt diesen mit Wasser, stellt ihn auf den Herd und schaltet die Herdplatte ein. Wenn das Wasser kocht, wirft er die Nudeln in den Topf, stellt den Herd nach einer gewissen Zeit wieder ab, usw.

Die einzelnen Operatoren und ihre Reihenfolge sind ihm hier also genau bekannt und er hat eine klare Vorstellung vom Zielzustand. Er steht damit nicht vor einem Problem, sondern vor einer Aufgabe. Man erhält somit neben dem Problemlösen eine zweite Ebene der Interoperationsregulation, das *Aufgaben bearbeiten*. Auf beiden Ebenen geschieht bewusstes, zielgerichtetes Agieren.

Ist aber für die Umwandlung eines Anfangszustandes in einen Zielzustand kein bewusstes Agieren nötig, so operiert das System auf einer dritten Regulationsebene, der Ebene der *Automatismen*.

So lässt sich auch hier ein Robotersystem vorstellen, das sich im Straßenverkehr bewegt und sich dabei außerordentlich koordiniert verhält. Es bremst, kuppelt, schaltet, blinkt und behält gleichzeitig den restlichen Verkehr im Überblick. Dies ist eine hochkomplizierte Sequenz von zielgerichteten Interoperationen, die dem System aber nicht bewusst sein muss, sondern als Produktionsregeln hinterlegt wurden.

Insofern wird man beim Modellieren diese drei Ebenen der Aktionsregulation unterscheiden und mit entsprechenden Methoden und Technologien konkretisieren müssen: Problemlösen (... durch Suche, Optimieren, Planen etc.), Aufgabenbearbeiten (... mittels Skripten, Produktionsregeln) und Ebene der Automatismen (... Produktionsregeln, Fuzzy Systeme, etc.).

Bei der Modellierung der verschiedenen Ebenen, auf denen Interoperationen stattfinden können, hat man es immer wieder mit dem Begriff des *Wissens* zu tun. So spricht man vom Bearbeiten einer Aufgabe dann, wenn das vorhandene Wissen ausreicht, um das angestrebte Ziel direkt zu erreichen. Demgegenüber spricht man vom Lösen eines Problems, wenn eine der oben definierten Barrieren vorliegt, wenn also das Wissen nicht hinreicht, das Erreichen des Ziels ohne weiteres zu ermöglichen. Dieser Umstand legitimiert die Grundannahme der kognitiven Interoperationstheorie: Interopieren ist wissensbasiert.

Ganz unabhängig von der Regulationsebene unterscheidet man in dieser Theorie zunächst zwischen Faktenwissen und Operatorwissen.

- Als *Faktenwissen* wird Wissen über die Elemente eines Realitätsbereiches, ihre Eigenschaften, ihren gegenwärtigen Zustand und ihre Zusammenhänge bezeichnet, d. h. über die Frage, welches Element welches andere wie beeinflusst. Mit Faktenwissen ist also so etwas wie das Bild des in Frage stehenden Aktionsraumes gemeint, über das sich das System einen Eindruck verschaffen muss. Dieses Bild kann sehr unvollständig sein oder auch sehr komplett, es kann sehr differenziert aber auch sehr oberflächlich sein. Ein solches Faktenwissen kann beispielsweise in Form von Netzwerken gespeichert werden. Die Knoten eines solchen Netzwerkes repräsentieren dabei die Elemente des Realitätsbereiches.
- Als *Operatorwissen* bezeichnet man demgegenüber das Wissen über Interaktionen und letztlich Aktivitäten, die im Realitätsbereich durchführbar sind. Dazu gehört aber nicht nur das Wissen über die motorische Ausführung eines Operators, sondern auch Wissen über seine Eigenschaften. Operatorwissen ist in Form von Aktionsschemata des Systems gespeichert. Ein solches Aktionsschema besteht aus drei Komponenten. Zunächst existiert Wissen über die situativen Bedingungen, die vorliegen müssen, damit eine bestimmte Aktion durchgeführt werden kann. Die zweite Komponente ist das Wissen über die motorische Ausführung der Aktion selbst und schließlich existiert eine Erwartungshaltung darüber, wie die Situation durch die Aktion verändert wird. Längere Operationssequenzen entstehen durch die Koppelung von Interoperationsschemata derart, dass die Erwartungshaltung eines Interoperationsschemas den situativen Bedingungen eines folgenden Aktionsschemas entspricht.

Faktenwissen und Operatorwissen werden zusammenfassend als *systemisch-epistemisches Wissen* bezeichnet, als Wissen über das, was ist und wie man es verändern kann.

Allerdings reicht ein solches epistemisches Wissen nicht immer aus, damit ein System die intendierten Ziele erreichen kann. Insofern muss das Robotersystem auch in die Lage versetzt werden, trotz mangelhaften epistemischen Wissens weiter seine Ziele zu verfolgen, sich weiter aktiv mit dem Problem auseinanderzusetzen. Diese Interoperationsfähigkeit basiert auf einer weiteren Form von Wissen, die man als *systemisch-heuristisches Wissen* bezeichnet. Mit diesem Ausdruck ist solches Wissen gemeint, dass das System befähigt, notwendiges, aber fehlendes Faktenwissen zu beschaffen, kritische Elemente eines Realitätsbereiches herauszufinden, sich notwendige Operatoren und Wissen über ihre Eigenschaften zu erarbeiten und mögliche Fehler im eigenen Agieren zu korrigieren. Systemisch-heuristisches Wissen ist das Verfügen über Programme zur Neukonstruktion systemisch-epistemischen Wissens. Man kann sich das systemisch-heuristische Wissen eines Robotersystems als eine Art von Programmzbibliothek bzw- repository vorstellen, in der solche Programme gespeichert sind. Diese Art von Programmen selbst werden in der Interoperationstheorie kurz als *Systemheurismen* bezeichnet.

Einige typische Systemheurismen, denen man in dem späteren Kapitel dieses Buches in Form von Konkretisierungen begegnen wird, sind:

- *Versuch-Irrtum*: Dies ist ein sehr einfacher Heurismus, der angewendet werden kann, um Operatorwissen zu schaffen. Der Heurismus besteht darin, Operatoren, die „eigentlich“ nicht anwendbar sind, auszuprobieren, um zu sehen, ob man nicht doch etwas findet, was weiterhilft.
- *Analogiebildung*: Dieser Heurismus besteht in dem Versuch, epistemisches Wissen aus einem völlig anderen Bereich auf die gegenwärtige Situation zu übertragen, um dadurch Wissenslücken aufzufüllen.
- *Inferenzprozesse*: Auch alle Prozesse des logischen Schließens sind natürlich Programme zur Neukonstruktion epistemischen Wissens, indem man bekannte Fakten so verbindet, dass neues Wissen entsteht.
- *Betrachtungsebenenwechsel*: Dies ist ein Heurismus, den man dann einsetzen kann, wenn man keine zielführenden Maßnahmen findet. Man kann ein Problem in verschiedene Teilprobleme zerlegen und dann nach Maßnahmen zur Behandlung der Teilprobleme suchen. Es kann aber auch bedeuten, dass man vom konkreten Problem aus abstrahiert und auf einer allgemeineren Ebene nach Lösungen sucht.
- *Ausfällen des Gemeinsamen*: Dieser Heurismus kann angewendet werden, wenn das System mit seinen Aktionen Misserfolge erlebt hat. Man kann sich nämlich überlegen, durch welche Gemeinsamkeiten all diese Aktionsversuche gekennzeichnet sind und kann dann nach Operatoren suchen, die dieses Gemeinsame nicht aufweisen. Wenn tatsächlich der gemeinsame Faktor für die Misserfolge verantwortlich war, wird man jetzt erfolgreich weiterarbeiten können.

Entscheidend für den Erfolg der einzelnen Aktivitäten und den darin durchgeföhrten Interoperationen in einer komplexen und unbestimmten Situation sind nun beide Wissensarten. Das Ausmaß, in dem ein System sowohl über konkretes epistemisches Wissen über den Realitätsbereich als auch über allgemeines heuristisches Wissen verfügt, bezeichnet man *Systemkompetenz* des Robotersystems. Diese Kompetenz ist damit so etwas wie ein Maß für das Zutrauen, das man in den Erfolg der Interoperationen setzt. Von großer Bedeutsamkeit ist die Zusammensetzung der Interoperationskompetenz für die Validierung des Robotersystems. Basiert das Kompetenzgefühl vor allem auf breitem epistemischem Wissen, so ist es ziemlich labil. Ohne ausreichendes heuristisches Wissen wird sich das System bei einem eigentlich unerwarteten Misserfolg ziemlich hilflos verhalten. Das epistemische Wissen hat sich als falsch erwiesen, es fehlt aber auch die Möglichkeit, neues Wissen aufzubauen. Die Systemkompetenz wird sich Laufe des Lebenszyklus drastisch verschlechtern. Basiert aber die Systemkompetenz auch sehr stark auf breitem heuristischen Wissen, ist das System viel stabiler, da bei Misserfolgen entsprechende Heurismen zur Auffüllung von Wissenslücken und zur Neukonstruktion zielführender Operatoren zur Verfügung stehen.

Bevor man Modelle bezüglich Interoperationen entwickeln kann, benötigt man eine Vorstellung oder idealtypischerweise sogar eine Theorie bezüglich der abzubildenden Problematik. Wenn man nun eine Theorie zur Erklärung solcher Problemstellungen aufstellen will, könnte man diese Theorie in ganz unterschiedlichen Sprachen formulieren. Man könnte die Theorie beispielsweise umgangssprachlich formulieren als ein Gefüge allgemeinverständlicher Sätze. Man könnte die Theorie unter Verwendung spezieller fachsprachlicher Termini aufbauen, man könnte sie aber auch mit Hilfe statistischer Begriffe formulieren, als Bündel von Aussagen über empirisch beobachtbare Variablen und ihren Zusammenhängen. Letztlich hat man die Möglichkeit, eine solche Theorie als Computerprogramm zu formulieren: Die Vorstellungen über die an den einzelnen Interoperations beteiligten Konzepte können nämlich auch im Rahmen der formalen Struktur einer Programmiersprache ausgedrückt werden. Diesen unterschiedlichen theoriesprachlichen Möglichkeiten entsprechen unterschiedliche Vorgehensweisen zur Isolierung der Bausteine der Theorie. Man kann „phänomenologisch“ vorgehen. Dies bedeutet, dass Handlungsprozesse zunächst möglichst unvoreingenommen und genau betrachtet werden. Diese Beobachtungen werden dann analysiert und es wird versucht, die immerwiederkehrenden Konstanten herauszufinden und als Elemente einer Theorie zu verwenden. Man kann auch versuchen, statistische „Konstrukte“ herauszuarbeiten, indem man bestimmte Aspekte des Interoperationsprozesses durch entsprechende Messinstrumente (etwa Fragebögen oder kontrollierte Beobachtungsmethoden) erfasst. Schließlich kann man sich aber auch überlegen, wie eigentlich ein artifiziell geschaffenes System aussehen müsste, welches in der Lage ist, in komplexen Realitätsbereichen zu agieren und dabei ein Verhalten zu produzieren, das einem intelligenten Problemlöser in der gleichen Situation ähnlich ist. Diese letzte Zugangsweise in Form einer Systemkonstruktion bildet die Grundlage des funktionalen Ansatzes der Interoperationsanalyse. Diese Zugangsweise muss als adäquate Theoriesprache sowohl eine computer- als auch robotergerechte Strukturierung und Formulierung der theoretischen Aussagen gewährleisten, um ein Robotersystem als komplexes System zum „Laufen“ bringen zu können.

Das Grundprinzip der funktional-systemtheoretischen Betrachtungsweise in der Interoperationstheorie besteht darin, zu fragen: Welche Elemente braucht ein Robotersystem und wie sind diese Elemente miteinander mittels welcher Funktionsprinzipien zu verknüpfen, um so ein intendiertes Verhalten zu erzeugen, dass das Robotersystem sich als Problemlöser (solution provider) eignet?

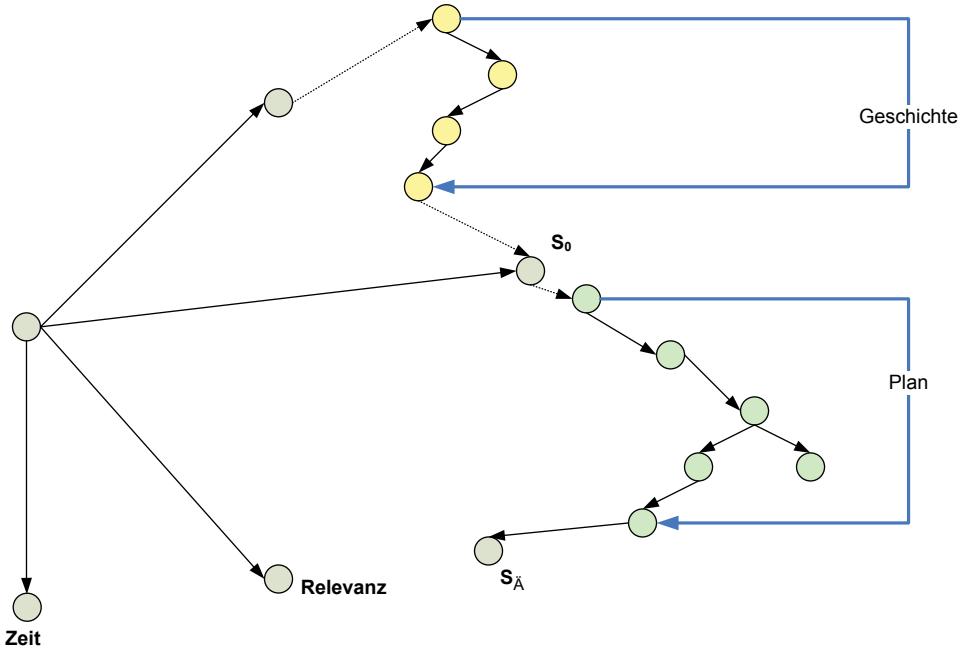
Wenn man nun ein derart komplexes System wie einen Roboter und dessen gestellte Aufgabe unter einem funktionalen Blickwinkel theoretisch erfassen und modellieren will, so erweist es sich als sinnvoll, das Geschehen zu gliedern, einzelne Analyseeinheiten herauszupräparieren. Die Interoperationstheorie verwendet den Begriff Interoperation als Analyseeinheit, sie definiert ihn als eine Einheit im Verlauf zielgerichteten systemischen Agierens. Die Grenzen einer solchen Interoperation werden durch am Anfang und am Ende liegende Entscheidungen definiert.

Im Rahmen der systemischen Interoperationstheorie wird das Konzept der *systemischen Intention* als eine Erweiterung des Konzeptes der *Absicht* als zentrale Gliederungs- und Analyseeinheit im Interoperationsgeschehen eingeführt. Eine systemische Intention wird immer durch ein irgendwie geartetes Motiv, einen Bedürfniszustand, ausgelöst. Sie ist erledigt, wenn ein Zustand erreicht ist, der das zugrundeliegende Motiv befriedigt. Eine systemische Intention ist aber nicht nur eine bestimmte Einheit im Interoperationsgeschehen, sie ist auch im Speicher des Systems adäquat repräsentiert. Sie ist eine *systemisch-kognitive Struktur*, die Informationsverarbeitungsprozesse und damit die Interoperation steuert. Eine systemische Intention ist demnach eine temporäre systemisch-kognitive Struktur, die man im Wesentlichen als Bündelung von drei verschiedenen Komponenten verstehen kann:

- Wahrnehmung eines irgendwie gearteten Bedürfniszustandes,
- Vorstellung eines oder mehrerer Ziele,
- Wissen über die Operatoren und Operatorsequenzen (die Aktionsmöglichkeiten) mit deren Hilfe die gegenwärtige Bedürfnissituation in die gewünschte Zielsituation umgewandelt werden kann.

An dieser Stelle des Buches erscheint es nun wichtig, das Konzept der systemischen Intention von anderen, ähnlichen Konzepten abzugrenzen. Eine systemische Intention ist zunächst nicht mit einem *Wunsch* zu verwechseln. Ein Wunsch ist dadurch gekennzeichnet, dass zwar ein Bedürfnis vorliegt und auch ein Zielzustand vorstellbar ist, jedoch keine Aktionen zur Erreichung dieses Ziels möglich, bekannt oder derzeit beabsichtigt sind. Von *Unzufriedenheit* spricht man im Allgemeinen dann, wenn die gegenwärtige Situation hinsichtlich wichtiger Größen defizient ist, aber kein klares Ziel und keine Maßnahmemöglichkeiten bekannt sind. Von *Wollen* schließlich wird gesprochen, wenn es nötig wird, eine bestimmte Absicht aus irgendwelchen Gründen gegen andere, konkurrierende Absichten durchzusetzen. Dabei stößt der Entwickler von Robotersystemen auf einen „menschlichen“ Umstand bzw. Problematik, indem beim Agieren in komplexen Realitätsbereichen meistens mehrere Intentionen gleichzeitig verfolgt werden müssen. Dennoch darf das System nicht ständig zwischen der Bearbeitung verschiedener Intentionen hin- und herwechseln, sondern sollte versuchen, eine Intention zu Ende zu bringen, bevor es sich der nächsten widmet. Es besteht nämlich die Gefahr, dass verschiedene Intentionen und deren inhärenter Drang nach Abarbeitung zu einem chaotischen und damit eventuell nicht zielführenden Interoperationsprozess führen. Unter der funktionalen Betrachtungsweise muss also eine weitere Instanz geschaffen werden, die dafür sorgt, dass immer nur eine Intention aktiv und aktionsleitend ist.

Die folgende Abbildung zeigt die systemisch-kognitive Struktur einer Intention (Abb. 2.21):



**Abb. 2.21** Struktur einer systemischen Intention

Eine systemische Intention umfaßt die folgenden Bestandteile:

- **Ausgangszustand  $S_0$ :**  $S_0$  ist ein Schema, welches die aktuelle Situation repräsentiert, in der sich das Robotersystem im Moment befindet.  $S_0$  bezeichnet sozusagen den „Startpunkt“ der Intentionserledigung.
- **Zielzustand  $s_\otimes$ :** Auch  $s_\otimes$  ist ein Schema, welches die Ziel- oder Befriedigungssituation repräsentiert, die das Robotersystem anstrebt. Eine Intension muss nicht nur unbedingt ein  $s_\otimes$  haben, es sind Intentionen denkbar, die durch mehrere unterschiedliche Zielsituationen befriedigt werden können. Zielzustände können mehr oder weniger klar sein.
- **Wissen über Operatoren- oder Operatorenketten:** In der Abbildung sieht man zwischen  $S_0$  und  $s_\otimes$  eine Reihe von Pfeilen, die sich verzweigen. Diese Pfeile sollen die einzelnen Operatoren und Operatorenketten darstellen, die dem System zur Verfügung stehen, um  $S_0$  in  $s_\otimes$  zu überführen, also den Interoperationsplan. Dieser Interoperationsplan braucht nicht als fertige Sequenz von Interoperationschemata im Speicher des Systems vorhanden zu sein, er kann nur modellhaft vorliegen, als ungefähre Vorstellung davon, was das System später zu tun hat.
- **Geschichte:** Die Intention enthält nicht nur Wissen über  $S_0$ , die aktuelle Situation, sondern auch Wissen darüber, wie es in die gegenwärtige Situation gekommen ist. Dieses Wissen um die Geschichte der Intention ist notwendig, damit das Robotersystem lernen kann.

- *Relevanz*: Die Relevanz einer Intention ist ein sehr bedeutsamer Parameter. Sie ist eine Funktion der Stärke des zugrundeliegenden Bedürfnisses, das durch die Intention befriedigt werden soll. Die Relevanz wird also dadurch bestimmt, wie massiv der zugrundeliegende Mangelzustand ist, der im Übrigen auch die Zielschemata  $s_{\otimes}$  determiniert.
- *Zeitperspektive*: Schließlich gehört zu vielen Intentionen noch eine gewisse Zeitperspektive. Es gibt Intentionen, die einen festen Termin haben, bis zu dem sie erledigt sein müssen, es gibt Intentionen, die das Robotersystem sofort in Angriff nimmt und es gibt Intentionen, die es auch irgendwann in der Zukunft erledigen kann.
- *Erfolgswahrscheinlichkeit*: Nun besteht aber eine Intention nicht nur aus den bisher angeführten expliziten Bestandteilen. Zu einer Intention gehören auch implizite Komponenten, die sich ihrerseits aus dem mehr oder weniger vollständigen, mehr oder weniger sicheren Wissen um einen Interaktionsplan ableiten lassen. Zu jeder Intention gehört eine gewisse Erfolgswahrscheinlichkeit, sonst handelt es sich um einen Wunsch. Verfügt das Robotersystem beispielsweise über einen bereits erprobten, fertig ausgearbeiteten Interoperationsplan, dann stellt die Intention gemäß der Interoperationstheorie kein Problem, sondern eine Aufgabe dar. Dann ergibt sich die Erfolgswahrscheinlichkeit der Intention einfach aus den Erfolgswahrscheinlichkeiten der Einzeloperatoren, die in ihrer Auseinanderfolge den Interoperationsplan bilden.

Mit Hilfe dieser Strukturelemente kann man nun beim Agieren in komplexen Realitätsbereichen der Tatsache Rechnung tragen, dass das System praktisch ständig über eine ganze Reihe derartiger Intentionen verfügt, von denen aber nur jeweils eine aktionsleitend sein sollte. Um eine Auswahl zwischen den verschiedenen Intentionen treffen zu können, kann sich das Robotersystem dreier Kriterien bedienen:

- *Relevanz* der Intention: Je wichtiger eine Intention, je größer also der Bedürfnisdruck, je gravierender der Mangelzustand, desto eher sollte sie zur Behandlung ausgewählt werden. Dieses Hauptkriterium, Auswahl nach Relevanz, ist zweifelsohne sehr sinnvoll, da sonst möglicherweise gefährliche Mangelzustände nicht behoben werden.
- *Erfolgswahrscheinlichkeit*: Die Relevanz allein ist aber als Auswahlkriterium nicht ausreichend. Vielmehr muss die Erfolgswahrscheinlichkeit mitberücksichtigt werden, um eben eine Blockierung des agierenden Systems zu verhindern. Es werden also bevorzugt solche Intentionen interoperationsleitend, die wichtig sind, und die auch mit einer gewissen Aussicht auf Erfolg bearbeitet werden können.
- *Dringlichkeit*: Es könnte Intentionen geben, die gar nicht so fürchterlich wichtig sind, aber dringend erledigt werden müssen, weil sie sonst von dem System aufgegeben werden müssen. Sind keine sehr wichtigen anderen Intentionen vorhanden, deren Erledigungstermine ebenfalls drängen, so werden besonders dringliche Intentionen vorgezogen.

Wenn ein interoperierendes System also gleichzeitig mehrere Intentionen hat, so wird in der Regel die Behandlung der Intention vorgezogen, deren Erledigung besonders relevant erscheint, die eine gewisse Erfolgswahrscheinlichkeit der Zielerreichung aufweist und die nicht auch genausogut verschoben werden kann. Stellt man sich diese drei Parameter (Wichtigkeit, Erfolgswahrscheinlichkeit und Dringlichkeit) als mathematische Größen vor, die zwischen 0 und 1 variieren können (die 0 selbst kann bei keinem der drei Parameter auftreten, da es sich sonst nicht um eine Intention handelt), so geht die Interoperationstheorie von einer multiplikativen Verknüpfung der Parameter aus. Zur Behandlung ausgewählt wird die Intention, für die das Produkt der drei Größen maximal ist.

Die Zusammenfassung in Bezug auf eine systemische Intention zeigt massive Auswirkungen auf das noch zu entwickelnde Interoperationsmodell und dessen Struktur: Eine Intention ist eine temporäre systemisch-kognitive Struktur, die durch eine Bündelung verschiedener Inhalte entsteht. Aus den expliziten Bestandteilen einer Intention lassen sich zusätzliche implizite Komponenten ableiten, die als Erfolgswahrscheinlichkeit und als Abschätzung der Zeitperspektive (Dringlichkeit) bezeichnet wurde. Da das agierende Robotersystem in der Regel über mehrere Intentionen verfügt, aber nur jeweils eine Intention erledigt werden kann, ist eine Art von Intentionsselektion notwendig. Diese Selektion erfolgt durch eine Verrechnung der Parameter Wichtigkeit, Erfolgswahrscheinlichkeit und Dringlichkeit.

Die Interoperationsanalyse versucht, das Verhalten von Systemen aus einem Realitätsbereich nachzubilden. Dazu wird eine Hypothese über die Struktur des verhaltenserzeugenden Systems entworfen, dieser Entwurf in einem Modell abgelegt, um letzteres dann als Hard- und Softwaresystem auszuimplementieren. Insofern basiert die Interoperationsanalyse auf einem dreidimensionalen Modell, das neben der Hierarchie die Komponenten von Zeit und Raum enthält:

- Zielebene mit den Interoperationen
- Funktionale Ebene mit den Interoperationsschritten
- Strukturelle Ebene mit den Elementen der Interoperationsschritte

In der *Zielebene* strukturiert der Agens den Interoperationsstrom bewusst nach Zielen. Diese Zielkonstrukte lassen folglich auch den Wissenschaftler die Interoperationen bestimmen. Bei den Einheiten dieser Ebene handelt es sich um molare, natürliche Einheiten. Die Einheiten auf der *funktionalen Ebene* lassen sich in ihrer Funktion zum Interoperationsziel bestimmen, daher werden sie auch als Interoperationsschritte bezeichnet. Funktionale Einheiten werden im Dienste des Ziels (höhere Ebene) ausgeführt. Die funktionale Betrachtung auf dieser Ebene ergibt sich aus dem systemischen Zusammenhang, in dem ein Agieren des Robotersystems betrachtet wird. Der Begriff der Funktion wird dabei aus der allgemeinen Systemtheorie übernommen. Funktionale Einheiten sind Produzenten von Ereignissen, die auf einer übergeordneten Ebene liegen. Im Systemzusammenhang sind sie jene Unterteile, die im Dienste des Gesamtsystems zur Erreichung von dessen

Zielen ausgeführt werden. Funktionale Einheiten sind dynamisch konzipiert und halten wechselseitige Auswirkungen einer Einheit auf eine andere und auf das Gesamtergebnis des Agierens fest. Die Elemente der Interoperationsschritte, die auf der *strukturellen Ebene* erfasst werden, werden entsprechend der eingeschlagenen systemtheoretischen Sprache als strukturelle Einheiten bezeichnet. Im hierarchischen Modell sind die Konzepte Funktion und Struktur also nicht entgegengesetzt, sondern kompatibel. So kann ein und dieselbe Funktion durch verschiedene Strukturelemente hervorgebracht werden.

Die Grundannahme der systemischen Interoperationstheorie ist, dass die Welt in *Bereiche* aufgeteilt werden kann: in Bereiche von diskreten Elementen und Aufgaben, mit denen sich das kognitive System beschäftigt, das in einem gegebenen Raum von Problemen handelt, etwa im Raum der Wahrnehmung, der Kommunikation, der Bewegung.

Es ist relativ leicht, den Bereich „Schach“ zu definieren: er umfasst alle möglichen Zustände im Raum des Schachspiels. Es gibt Figuren und Stellungen auf dem Schachbrett, es gibt Regeln für Züge und Zugwechsel, es gibt klare Grenzbedingungen. Im Gegensatz dazu hat es sich als ziemlich unproduktiv erwiesen, nur diesen regelbasierten Ansatz im Bereich mobiler Roboter anzuwenden. So verlangt die Navigation in einer Welt voller Bewegungen die ständige Nutzung des gesunden Menschenverstandes, um diese Welt der Gegenstände überhaupt mental zusammenbauen zu können. Sollen in die Welt der Robotersystems beispielsweise die Fußgänger einbezogen werden? Es ist unmittelbar klar, dass die Antwort auf diese Frage nur aus einem außerordentlich diffusen Hintergrund von Überlegungen abgeleitet werden kann, die der Beobachter als unweigerlich kontextuell betrachtet: wo sich jemand befindet, wieviel Uhr es ist, welche Art von Straße man befährt usw. Im Gegensatz zum Schach gleicht der Robotikbereich viel eher einem sich immer weiter differenzierenden fraktalen Raum möglicher Einzelheiten, als ein präzise definierter Raum voller starrer Regeln.

*Planen* in Bezug auf die Entwicklung von Robotersystemen bedeutet, zu einer definierten Problemstellung deren notwendigen Aktivitäten zu analysieren und deren konkrete Durchführung zu bestimmen. Als Ergebnis des Planens entsteht ein Interoperationsplan. Dieser enthält eine zeitlich geordnete Folge von Aktionen (Tätigkeiten, Operationen). Die Ausführung dieser Interoperationen löst die gestellte Aufgabe. In Bezug auf solche problemorientierten Vorgehensweisen lassen sich folgende Arten der Planung unterscheiden:

- *Interoperationsplanung*: WAS ist zu tun, welche Interoperationen sind aus einer konkreten Problemstellung abzuleiten?
- *Reihenfolgeplanung*: In welcher zeitlichen Beziehung stehen die Interoperationen?
- *Ressourcenplanung*: WER, also welches Robotersystem oder einzelne Komponenten führen die Interoperationen durch? Welche Ressourcen stehen beschränkt zur Verfügung? Welche Ressourcen müssen wie gut ausgelastet sein? Welche Ressourcen halten welchen Belastungen stand?
- *Zeitplanung*: WANN sind die Interoperationen zu starten?
- *Durchführungsplanung*: WIE, also mit welcher Parametrisierung, ist eine Interoperation konkret auszuführen (beispielsweise Bahn- und Greifplanung)?

Unter Planen im engeren Sinne wird in diesem Buch die Interoperationsplanung in Verbindung mit der Reihenfolgeplanung verstanden. Hingegen wird die Ressourcen- und die Zeitplanung unter dem Oberbegriff *Scheduling* zusammengefasst und behandelt.

Die Planung kann prinzipiell auf mehreren Abstraktionsebenen erfolgen. Auf einer bestimmten Ebene gilt eine Problemstellung als evident. Durch einen Planungsvorgang auf dieser Ebene werden Interoperationen erzeugt, die von der darunter liegenden Ebene entweder direkt ausgeführt oder durch einen erneuten Planungsvorgang auf dieser tieferen Ebene in weitere, eventuell detaillierte Interoperationen zerlegt werden können.

Bei der Planung einer Urlaubsreise von Altrip nach London greift man auf folgende Interoperationen zurück: (IA1) Fahrt mit dem Auto nach Frankfurt zum Flughafen. (IA2) Flug von Frankfurt nach London. (IA3): Fahrt mit dem Taxi vom Londoner Flughafen in die Innenstadt.

Dies sind zunächst sehr grobe Interoperationen, die noch weiter verfeinert werden können, die also selbst dann wiederum kleine Pläne darstellen.

Die Fahrt von Altrip zum Flughafen (IA1) besteht aus folgenden Interoperationen: (IA11): Gang zur eigenen Garage. (IA12): Starten des Autos. (IA13): Fahrt auf der Autobahn zum Flughafengelände. (IA14): Parken des Autos in dem Parkhaus P2.

Ein weiteres wichtiges Problem der Planung ist die Steuerung und Überwachung der Ausführung des Plans durch geeignete Sensoren und die Modifikation des Plans bei Erkennen von Abweichungen gegenüber den Annahmen im bisherigen Plan. Diese Anforderung führt zu einer sogenannten reaktiven Planung.

Als Basis für eine Interoperationsplanung dient ein Situationskalkül (vgl. Mc Carthy und Hayes 1969), das als Grundelemente mit lediglich zwei Entitäten auskommt: Situationen (Zustände) und Interoperationen.

- *Situation*: Schnappschuss des interessierenden Ausschnitts der Welt zu einem Zeitpunkt, beschrieben durch eine Menge logischer Formeln, die in dieser Situation gelten.
- *Interoperation*: Formale Beschreibung einer Handlung in der Welt. Eine Interoperation überführt eine gegebene Situation in eine andere, die sogenannte Nachfolgesituation.

Mit diesen Entitäten lässt sich das Situationskalkül als eine formale Vorschrift formulieren, die aus einer gegebenen Situation  $S^n$  und einer Aktion, die in  $S$  ausgeführt wird, eine Nachfolgesituation  $S^{n+1}$  errechnen kann.

Das Ziel einer Interoperationsplanung besteht im Finden einer Menge von Interoperationen, deren Ausführung vom aktuellen Zustand  $Z_A$  in einen Zielzustand  $Z_Z$  führt. Gemäß des Prinzips der Simplifizierung werden bei einer eher klassischen Planung zunächst alle Schwierigkeiten ausgeklammert, die daraus folgen können, dass Interoperationen eine zeitliche Dimension (Dauer) besitzen, sich überlappen können oder sogar parallel ausgeführt werden müssen. Ein erweitertes Planen im Sinne der Robotik behandelt demgegenüber auch solche Probleme, die mit Unsicherheiten in den Situationen und Interoperatio-

nen einhergehen. Die Anzahl der möglichen Situationen bestimmt meist den *Planungsaufwand*. Deshalb sollte der Ausschnitt der realen Welt so klein wie nötig und so einfach wie möglich gewählt werden. Dabei gilt es zu beachten, dass infolge von Rand- oder Vorbedingungen gegebenenfalls nicht alle Interoperationen in jeder Situation erkennbar und damit (ein) planbar sind.

So kann ein bestimmtes Objekt nur mit Hilfe eines Greifers gegriffen werden, wenn: (1) das Objekt zugänglich ist, (2) das Robotersystem das Objekt überhaupt tragen kann, (3) der Greifer geöffnet ist, (4) der Greifer sich am Greifpunkt des Objekts befindet und (5) der Greifer zum Greifen dieses Objekts geeignet ist.

Nicht alle Randbedingungen sind immer so einfach zu eruieren, zu formulieren, zu prüfen und zu erfüllen wie die oben genannten. In realen Anwendungen treten eine Fülle sehr viel komplexerer Randbedingungen auf. Einige sind:

- die Aufgabe sollte in möglichst kurzer Zeit oder innerhalb einer vorgegebenen maximalen Zeitspanne erledigt werden
- möglichst wenig Komponenten- oder Systemwechsel
- bevorzugt benachbarte Komponenten einbauen
- mechanische Stabilität der teilmontierten Komponenten
- Verbindungsform (durch Andrücken, mit Klebstoff, durch Eindrehen, mit Federringen), dadurch ergeben sich Randbedingungen für teilmontierte Komponenten
- Materialeigenschaften (sind die Teile verformbar oder nicht, Temperaturempfindlichkeit)
- Beobachtbarkeit (beispielsweise durch Kamera),
- Genauigkeit der gegenseitigen Positionierung von Teilen

Wenn man nun errechnen will, was sich nach Ausführung einer Interoperation ergeben hat, treten eine Reihe von Detailproblemen auf. Je nachdem, ob man beispielweise berechnen will, welche Fakten sich nach der Ausführung einer Interoperation verändern oder welche sich nicht verändert haben, oder je nachdem ob man voraussetzt, alle bekannten oder nur die wichtigsten Vorbedingungen zur Ausführung einer Interoperation zu berücksichtigen, unterscheidet man die folgenden, teilweise überlappenden Detailprobleme:

- *Qualifikationsproblem*: Hier besteht die Problematik darin, korrekte Vorhersagen über die Fakten nach Ausführung einer Menge von Interoperationen zu machen, ohne alle (vorhandenen) Information über die Situation vor ihrer Ausführung berücksichtigen zu müssen. Dies umfasst folgende Fragestellungen: Unter welchen Voraussetzungen können Interoperation erfolgreich durchgeführt werden? Wie genau muss beschrieben werden, wann Interoperationen gelingen bzw. fehlschlagen?
- *Rahmenproblem* (Frameproblem): Hier gilt es, vorherzusagen, welche Fakten bei Ausführung einer einzigen Aktion unverändert bleiben werden. Dies betrifft Fragestellungen, inwieweit genau beschrieben werden muss, welche Teile einer Situation nicht von einer erfolgreich durchgeföhrten Interoperation betroffen sind?

- *Verzweigungsproblem:* Hier gilt es vorherzusagen, welche weiteren Fakten wahr werden, wenn durch Ausführung einer Interoperation in einer Situation ein Faktum wahr wird. Dies betrifft Fragestellungen, wie genau beschrieben werden muss, welche Teile einer Situation von einer erfolgreich durchgeföhrten Interoperation betroffen sind (Bestimmung von mittelbaren Auswirkungen)? Wie lässt sich beschreiben, was tatsächlich alles passiert, wenn eine Interoperation ausgeführt wird? Idealtypischerweise muss man eigentlich alle Auswirkungen einer Aktion erfassen. Dies ist in der Realität kaum möglich. Man beschränkt sich daher auf die zu erwartenden Auswirkungen.

So betrachtet man beispielsweise bei der Bewegung eines Objektes i. a. nur die Ortsveränderung dieses Objektes, nicht aber die Veränderung seiner Farbe.

- *Predictionproblem:* Hier gilt vorhersagen, welche Fakten nach Ausführung einer Menge von Interoperationen wahr sein werden.
- *Persistenceproblem:* Hier gilt es vorherzusagen, welche Fakten durch Ausführung einer Menge von Interoperationen unverändert bleiben werden.

Alle diese Probleme zusammen sind im Allgemeinen unentscheidbar. Durch die Existenz der aufgezeigten Problemstellungen wird also praktisch die Korrektheit und Vollständigkeit eingeschränkt, mit der Planer in ihren Problem- und späteren Anwendungsbereich arbeiten können.

Je nach Ausmaß und Umfang der einzelnen Inhalte von Planungsschritten lassen sich folgende Unterteilungen treffen:

- *Lineares Planen:* Alle Interoperationen werden in einer strengen zeitlichen Sequenz ausgeführt.
- *Nichtlineares Planen:* Zwischen den Interoperationen bestehen komplexe zeitliche Abhängigkeiten. Aktionen können zeitlich nacheinander, überlappend oder parallel ausgeführt werden. Bestimmte Aufgaben sind nur mit nichtlinearem Planen zu lösen.
- *Monotones Planen:* Jedes Teil wird sofort in seine Endposition gebracht. Ein zwischenzeitliches Ablegen und Bewegen eines anderen Teils ist also ausgeschlossen.
- *Nichtmonotonen Planen:* Nicht alle Teile werden sofort in ihre Endposition gebracht. In der Klötzenwelt beim Umordnen immer nötig.
- *Zentrales Planen:* Eine zentrale Planung bedeutet, dass nur an einer Stelle die gesamte Planung durchgeführt wird. Der Vorteil eines zentralen Planers ist, dass er alle Randbedingungen kennt und somit optimale Pläne erzeugen kann. Der Hauptnachteil des zentralen Planers ist, dass alle Details im gesamten Problem- und Weltmodell bei der Planung berücksichtigt werden müssen. Alle Änderungen der Annahmen und alle Unregelmäßigkeiten bei der Planausführung werden an den zentralen Planer gemeldet und führen zu einer Neuplanung. Der zentrale Planer ist also sehr komplex und bildet leicht einen Engpass. Ein Ausfall des zentralen Planers bedeutet unter Umständen einen Stillstand des Robotersystems.

- *Verteiltes Planen:* Hier gibt es eine Hierarchie von Planungseinheiten bzw. -komponenten, die Teilaufträge erhalten und deren Ausführung selbst planen (Multiagentensysteme).
- *Einstufiges Planen:* Aus der Aufgabe entstehen in einem Schritt direkt ausführbare Aktionen.
- *Mehrstufiges, Hierarchisches Planen:* Die Planung erfolgt über mehrere Abstraktionsebenen hinweg. Das entspricht einer Rückführung auf kleinere Teilprobleme.
- *Planen mit Teilzielen:* Das Gesamtproblem wird in Teilprobleme zerlegt. Zur Lösung der Teilprobleme werden Teilpläne erzeugt. Die Teilpläne werden dann zu einem Gesamtplan zusammengefügt bzw. integriert. Das Hauptproblem ist die Verträglichkeit der erzeugten Teilpläne, da diese normalerweise nicht unabhängig sind.

Ausgehend von einer anfangs vorliegenden Startsituation wird beim klassischen Planen eine Folge von Interoperationen gesucht, die diese Startsituation in eine Situation überführt, die ein vorgegebenes Ziel erfüllt. Gibt es eine Interoperationsfolge, welche die Startsituation in eine Zielsituation überführt, dann ist diese Folge die Lösung des Planungsproblems: der Plan. Man bezeichnet dies als *suchbasierte Planungsverfahren*. Genausogut kann ein Plan auch als Folge von Situationen betrachtet werden, weil jede Interoperation eine Situation in eine andere überführt. Bei dieser Vorgehensweise entspricht die Suche nach einem Plan der Zielsuche im Raum der möglichen Situationen. Solche Planungsverfahren werden auch als *mengenbasierte Verfahren* bezeichnet. Die zweite typische Vorgehensweise betrachtet partiell entwickelte Pläne, d. h. eine Menge von Interoperationen zusammen mit einer Ordnung dieser Aktionen. Ein solcher Planer fügt nun weitere Interoperationen oder Ordnungsrelationen hinzu. Diese Verfahren bezeichnet man als *planbasierte Planungsverfahren*. Eine dritte Gruppe von Verfahren ist durch die Benutzung von Methoden aus dem Bereich des automatischen Beweisens charakterisiert. Sie werden als *deduktive Planungsverfahren* bezeichnet.

Bei den bisher beschriebenen Planverfahren treten Probleme bei der Verwendung erzeugter Pläne dadurch auf, dass die Welt sich bei der Ausführung eines Plans nicht immer so gestaltet, wie bei der Erzeugung des Plans vorausgesetzt war. Das Hauptproblem ist das Auftreten von Unsicherheit und Unvollständigkeit und die Beachtung dieser beim Planen. Situationen können unsicher sein in dem Sinne, dass ihr Wahrheitswert oder ihre exakte Ausprägung nicht bekannt ist oder die Information darüber nur vorläufigen Charakter hat. Beispielsweise mag die Position oder der Zustand eines Objektes nur innerhalb gewisser Toleranzgrenzen gegeben sein. Weiterhin kann es sinnvoll sein, bei der Darstellung des Planungsbereichs mit Default-Informationen zu arbeiten, also zu formulieren, dass Rahmenbedingungen, wie beispielsweise Greifarme, normalerweise nicht verzogen sind.

Folgende Unsicherheitsfaktoren über die Interoperationen gilt es zu berücksichtigen:

- Unsicherheit für die Vor- und Nachbedingung
- Ausführungsfehler, die in der Regel erst zur Ausführungszeit eines Plans erkennbar werden. Daher sollten sie aber auch während der Planung berücksichtigt werden.
- Kontextabhängigkeit von Interoperationen, d. h., die einzelnen Interoperationen wirken sich je nach gültigen Gegebenheiten nicht immer gleich aus.

Erschwerend kommt hinzu, dass die Informationen, die der Roboter über seine Sensoren erhält, unter Umständen fehlerbehaftet sind. So liefern beispielsweise Ultraschallsensoren aufgrund ihres breiten Öffnungswinkels häufig nur sehr ungenaue Informationen über den Abstand zu Hindernissen. Demzufolge können sich auch die Interoperationen fehlerhaft gestalten. Beispielsweise ist die Ermittlung der zurückgelegten Strecke bzw. der durchgeföhrten Drehung dann ungenau, wenn fehlerhafte Sensorwerte vorliegen oder Rutschen und Driften während der Aktion sich einstellen.

Dieser Umstand wirkt sich wie folgt auf die Planung aus.

- *Bedingte Planung:* Gewöhnlich setzt man beim Planen voraus, dass bei der Ausführung des fertigen Plans alle Interoperationen unbedingt ausgeführt werden. Demgegenüber enthalten *bedingte Pläne* boolesche Ausdrücke, die bei der Planausführung getestet werden, wobei entsprechend des Testergebnisses in Unterpläne verzweigt wird. Idealtypisch kann der Planer zumindest die Bedingungen angeben, die zur Ausführungszeit abzuprüfen sind, d. h. er muss zur Planungszeit Wissen über mögliche oder zulässige Werte von Variablen besitzen, die in den Test dann einfließen. Praktikabel scheinen die bedingten Pläne dann zu sein, wenn nur wenige Planungsalternativen berücksichtigt werden müssen.
- *Reaktives Planen:* Ein Reaktionsplan ist nicht ein Plan, den man zur Lösung eines Problems verfolgt, sondern er beschreibt Handlungswissen über den gesamten Anwendungsbereich, mit dem unerwartete Änderungen in der Welt flexibel und normalerweise ohne Neuplanung abgefangen werden können.
- *Nullplanung:* Unter Umständen läuft die Steuerung der Handlungen eines Robotersystems zur Lösung eines Problems darauf hinaus, dass man eben nicht plant (Anzahl der Planungen gleich Null), sondern dass das Robotersystem sich so verhält, wie es der vorgefertigte Reaktionsplan vorgibt. Die Auswahl der jeweils nächsten Interoperation hängt nur von den aktuellen Gegebenheiten ab.

Weiterhin gilt es zu beachten, dass die Plananalyse gewissen Unsicherheiten ausgesetzt ist. Dabei geht es um Unsicherheiten, mit denen „niemand gerechnet hat“ und die somit nicht a priori geplant werden können. Um solche Unsicherheiten zu berücksichtigen, muss ein Plan noch während seiner Ausführung abgeändert werden können. Intelligente Robotersysteme sind daher in der Lage, ihre konkreten Interoperationen just-in-time zu überwachen. Eine Überwachungskomponente beobachtet während der Ausführung des Plans, was geschieht. Dadurch kann es sofort feststellen, ob etwas schiefgeht und kann in diesem Fall durch Planungsanpassungsmassnahmen versuchen, ausgehend von der neuen Situation, seine Ziele dennoch zu erreichen.

Gerade im Bereich der mobilen Robotik lassen sich viele spezielle Probleme bei der Planung ausmachen. Dabei sind verschiedene Teilprobleme der Robotik eigenständige Planungsprobleme, für deren Lösung aufwendige Algorithmen und komplexe Daten-

strukturen notwendig und vorhanden sind.<sup>42</sup> Solche eher „typischen“ Probleme bestehen beispielsweise bei der Planung von Armbewegungen (Glieder und Gelenke), der die Planung von Griffen sowie bei navigatorischen Fragestellungen. Bezüglich der Planung von Armbewegungen können drei Bewegungsklassen unterschieden werden: freie Bewegungen (Navigation), überwachte Bewegungen und Feinbewegungen. Bei der freien Bewegung bewegt sich der Arm (mit oder ohne Objekt) frei im Raum, d. h. dass weder der Arm noch das gegriffene Objekt Kontakt mit anderen Objekten hat und auch keine Kollision mit einem Hindernis zu erwarten ist. Ziel der überwachten Bewegung ist, eine definierte Objektberührung herzustellen. Die Bewegung wird nicht anhand eines geometrischen Modells detailliert vorausberechnet, sondern in der realen Umgebung aufgrund von Sensorinformationen bestimmt. Dazu werden hauptsächlich Kraft- und Drehmomentsensoren in der Hand des Roboters verwendet. Ein wesentliches Problem stellen dabei die Unsicherheiten dar, die durch die Ungenauigkeiten des Arms und der Sensorik auftreten, so dass die Sensorwerte meist nicht eindeutig interpretiert werden können. Auch die Reaktionszeit des Systems spielt hierbei eine wichtige Rolle. Um die gewünschte Endposition zu erreichen, muss eine Strategie aus überwachten Suchbewegungen und zielorientierten Feinbewegungen angewendet werden. Bei der Feinbewegung wird ein Objekt, das sich in Kontakt mit einem anderen Objekt befindet, vom Roboter bewegt, wobei der Objektkontakt beibehalten wird.

Insofern stellt die Navigation mobiler Robotersysteme spezifische Anforderungen an die Planung von Interoperationen.

Man denke dabei beispielsweise an die Planung von Robotersystemen, die Fußball auf einem durch Banden abgegrenzten Spielfeld spielen sollen.

Hierbei ergeben sich folgende Problemstellungen:

- *Selbstlokalisierung*: Wo befindet sich das Robotersystem? Dazu muss die Position innerhalb der Umgebung gefunden werden, wozu sie sich der Messungen der auf dem Robotersystem befindlichen Sensoren bedienen.
- *Kartenerstellung bzw. Umgebungsmodellierung*: Wohin bewegt sich das Robotersystem? Auch hier lässt sich mit Hilfe von Sensoren eine Karte erstellen, die das Robotersystem beispielsweise für die Selbstlokalisierung und Pfadplanung verwenden kann.
- *Pfad(weg)planung*: Wie gelangt das Robotersystem zu einer Zielposition? Hier wird versucht, kollisionsfreie Wege von einer Start- zu einer Zielposition zu finden. Diese Wege soll der Roboter dann in der realen Welt abfahren. Die Route muss gewisse Optimierungsmerkmale erfüllen.

Dabei bedingen sich Selbstlokalisierung und Kartenerstellung.

---

<sup>42</sup> Vgl. Saake 2002.

Man unterscheidet zwischen einer Navigation in unbekannter Umgebung und in bekannter Umgebung. Für beide Varianten stehen geeignete Verfahren zur Planung bereit:

- geometrische Wegplanung (Sichtbarkeitsgraph, bei der die Weglänge minimiert wird. Voronoi-Diagramm, bei dem der Abstand zu Hindernissen maximiert wird)
- lokale Bewegungssteuerung
- Optimierung einer Trajektorie
- Value iteration

Bei der Problemodellierung geht es um die Erstellung oder Adaption eines Modells der Umgebung des Robotersystems auf der Basis von Sensorinformationen. Diese Modelle bilden die Basis für die Selbstlokalisierung und die Pfadplanung. Dabei ist es nicht unbedingt nötig, eine explizite Karte aufzubauen. So kann beispielsweise durch die Speicherung von rohen Scannerdaten und ihren Aufnahmepositionen eine *implizite Repräsentation* der Umwelt erfolgen. Es lässt sich dabei zwischen folgende Arten von Umgebungsmodellen unterscheiden:

- Topologische Modelle
- Geometrische Modelle (Linien- und CAD-Modelle)
- Rasterbasierte Modelle

Die *topologischen Modelle* bilden einen annotierten Graphen, wo Knoten markante Orte (Bereiche) der Umgebung darstellen. Zwei Knoten sind genau dann durch eine Kante miteinander verbunden, wenn sie unmittelbar voneinander erreichbar sind. Als Vorteile dieser Modelle gelten deren Kompaktheit sowie deren Eignung für eine effiziente Pfadplanung. *Geometrische Modelle* kommen als zwei- oder dreidimensionale Modellvarianten vor. Dabei erfolgt die Beschreibung mit graphischen Primitiven (Linien), die zu komplexen Einheiten zusammengefasst werden. Zu den Vorteilen zählt die Tatsache, dass der Detailisierungsgrad sehr hoch angesetzt werden kann. Es lassen sich Umgebungen damit geometrisch sehr exakt beschreiben. *Rasterbasierte Modelle* eignen sich vor allem als Repräsentationsform der Umwelt für autonome, mobile Robotersysteme. Dabei erfolgt eine Diskretisierung der Umgebung des Roboters in kleinen quadratischen Zellen. Jede dieser Zelle enthält die Wahrscheinlichkeit, dass die entsprechende Stelle der Umgebung durch ein Hindernis belegt ist. Weiterhin kann jedem dieser Bereiche eine weitere Wahrscheinlichkeit zugeordnet werden, die ausdrückt, dass sich das Robotersystem genau an dieser Position befindet.

Bei der *Selbstlokalisierung* geht es darum, die Position des Robotersystems aufgrund einer Umgebungskarte und Sensordaten des Robotersystems zu bestimmen. Dabei werden auch alle durchgeführten Navigationsoperationen mit beachtet. Es lassen sich zwei Varianten unterscheiden:

- *lokale Selbstlokalisierung* (position tracking): Hier ist die ungefähre Position des Robotersystems bereits bekannt und es wird nur eine Positionskorrektur berechnet. Dies ist dann angebracht, wenn das Robotersystem an einer ungefähr bekannten Position aufgestellt wird und dann fortlaufend seine Position durch Vergleich der Sensordaten mit der Umgebungskarte bestimmt.
- *globale Selbstlokalisierung* (global localization): Das Robotersystem wird an einen beliebigen Ort gestellt. Durch Auswerten der Sensorinformationen muß das Robotersystem dann entscheiden, an welchen möglichen Positionen es sich befinden kann. Dies ist im Allgemeinen recht aufwendig. Das Robotersystem benutzt dabei auch seine Umweltrepräsentation.

Ein intelligentes Robotersystem, speziell ein mobiles Robotersystem, kann vor seinem Echteinsatz in seiner Umwelt zunächst einer Simulation anhand eines numerischen Modells unterzogen werden. Dies macht immer dann Sinn, wenn sich Experimente mit mobilen Robotern in der Praxis als zeitaufwendig, kostspielig und schwierig herausstellen. Weil die Interoperation des Robotersystems mit seiner Umgebung sehr komplex ist, liefern Experimente erst nach zahlreichen Durchgängen statistisch aussagefähige Ergebnisse. Erschwerend kommt hinzu, dass Robotersysteme – als mechanische und elektronische Artefakte – nicht in allen Experimenten gleich funktionieren. In manchen Fällen ändert sich ihr Verhalten dramatisch, sobald nur ein Versuchspараметer geändert wird.

Wenn man nun die wesentlichen Komponenten, die die Interoperation von Roboter und Umgebung bestimmen, in einem mathematischen Modell abbilden kann, lassen sich allein mit dem Computer Vorhersagen über Versuchsergebnisse treffen, ohne ein Robotersystem in Echtzeit verwenden zu müssen. Insofern hat eine solche Simulation den Vorteil, schneller und kostengünstiger zu sein. Simulationen können außerdem mit präzise definierten Parametern wiederholt werden, wodurch sich die Auswirkungen einzelner Parameter auf das Robotersystemverhalten vorab bestimmen lassen. Im Gegensatz dazu ist in Experimenten in realen Umgebungen und mit realen Robotersystemen dies nicht möglich, weil in der wirklichen Welt selten zwei völlig identische Situationen anzutreffen sind.

Das Ziel von Rechnersimulationen besteht darin, ein solches Modell zu konstruieren, das dem Original dahingehend entspricht, damit am Modell Manipulationen vorgenommen werden können. Die Funktionsweise des Modells kann untersucht werden, und es lassen sich Schlussfolgerungen über die Verhaltenseigenschaften des originären Systems ziehen. Wenn es also gelingt, ein originalgetreues Modell zu erstellen, kann eine Simulation Vorhersagen über Systeme treffen, die sich durch ihre Komplexität unter Umständen einer direkten Analyse entziehen, oder für die (noch) nicht ausreichende Daten zur Verfügung stehen. Simulation ermöglicht die kontrollierte Modifizierung von Parametern, wodurch wiederum ein besseres Verständnis des Modells erreicht wird. Simulationen lassen sich auch für Unterricht und Training verwenden. Man kann die Fragestellung „Was wäre ... wenn ...“ mit Modellen analysieren. Anhand einer Simulation lässt sich auch ein komplexes System einfacher in einzelne Komponenten aufspalten.

Bei einer Simulation wird zunächst ein *Modell* für ein bestimmtes Robotersystem entwickelt. Eingangs- und Ausgangssignale des Systems werden durch *exogene Variablen* dargestellt, d. h. die Variablen befinden sich außerhalb des Modells, während die Elemente des Modells selbst durch modellinterne *endogene Variablen* repräsentiert werden. Das Modell selbst wird nach Möglichkeit so entworfen, dass es ausschließlich die Artefakte beschreibt, die für die Fragestellung des Experimentators relevant sind. In der mobilen Robotik ist die Auswahl dieser relevanten Aspekte besonders schwierig, weil man nur sehr wenig über die Gesetzmäßigkeiten der Interoperation von Robotersystemen und Umgebung im Voraus weiß. Es besteht also stets die Gefahr, dass dem Modell entweder ein wesentlicher Aspekt fehlt, oder dass es Aspekte enthält, die nicht relevant sind.

Dabei muss man zwischen *stochastischen Modellen* und *deterministischen Modellen* unterscheiden. Ein stochastisches Modell ahmt das zu modellierende System durch die Verwendung von Abläufen nach, die auf (Pseudo)Zufallszahlen beruhen.

Ein Beispiel wäre die Modellierung von Autobahnverkehr, wobei die Anzahl der Autos und ihre Geschwindigkeit zufällig gewählt werden.

Ein deterministisches Modell verwendet dagegen deterministische, d. h. mathematisch definierte Verhältnisse zwischen Eingangs- und Ausgangsvariablen.

Ein Beispiel hierfür wäre das Modell eines hüpfenden Fußballs, wobei physikalische Gesetze über Energieverbrauch und Bewegung, zur Vorhersage des Ballverhaltens verwendet werden.

Weiterhin muss man zwischen *analytischen* und *numerischen* Modellierungen unterscheiden. Analytische Modelle sind nachvollziehbare Konkretisierungen des Originals und verwenden Differential- und Integralrechnung, während eine numerische Lösung auf numerischen Annäherungsmethoden beruht, beispielsweise numerische Integration oder Newtons Algorithmus für die Berechnung von Funktionsnullstellen.

Schließlich unterscheidet man zwischen *Monte Carlo Methoden* einerseits, die einen Zufallszahlengenerator verwenden und die Zeitdimension nicht berücksichtigen und Simulationen andererseits, die auch den Zeitfaktor mit einbeziehen. Eine Monte Carlo Simulation wäre demnach ein Modell, das den Zeitfaktor berücksichtigt und gleichzeitig Zufallszahlen verwendet.

Fast alle in Simulationen verwendeten Modelle sind *diskret*, im Gegensatz zu *kontinuierlichen Modellen*. Das liegt daran, dass sie auf digitalen Computern laufen, die in diskreten Zuständen arbeiten. Dabei wird der Systemzustand des modellierten Robotersystems in regelmäßigen Abständen abgetastet und danach modelliert. Solange die Abtastfrequenz doppelt so hoch ist wie die höchste im modellierten System auftretende Frequenz, stellt dies kein Problem dar. Es kann allerdings unter Umständen schwierig sein, diese höchste Frequenz zu bestimmen.

Nach der Entwicklung des Modells stellt sich die Frage, wie es sich mit der Genauigkeit der Vorhersagen des Modells verhält. In Bezug auf die Beantwortung dieser Frage lassen sich drei Modellklassen unterscheiden:

- Verifizierte Modelle
- Validierte Modelle
- Bestätigte Modelle

*Verifikation* ist dabei der Nachweis, dass eine Proposition wahr ist, was ausschließlich für geschlossene (vollständig bekannte) Systeme erreicht werden kann. Aus zwei Aussagen der Form „aus  $p$  folgt  $q$ “ und „ $p$ “, kann demnach gefolgert werden, daß „ $q$ “ wahr und damit verifiziert ist. Leider können Modelle physikalischer Systeme nie geschlossen sein, weil geschlossene Systeme Eingangsparameter erfordern, die vollständig bekannt sind. *Validierung* ist letztlich der Nachweis, dass ein Modell keine erkennbaren Fehler aufweist, und in sich konsistent ist. Ob ein Modell validiert bzw. gültig ist, hängt dabei von den qualitativen und quantitativen Eigenschaften der Eingangsparameter ab. Insofern wird an dieser Stelle streng zwischen Verifizierung und Validierung unterschieden. Ein Modell kann schließlich als eine Theorie oder als ein allgemeines Gesetz durch Beobachtungen *bestätigt* werden, wenn diese Beobachtungen mit den von dieser Theorie aufgestellten Vorhersagen übereinstimmen. Doch auch bei zahlreichen bestätigenden Beobachtungen kann noch nicht auf die Wahrheitstreue (Verifizierung) noch auf die Gültigkeit (Validierung) des Modells geschlossen werden. Wenn ein Modell nicht in der Lage ist, zuvor beobachtete Daten zu reproduzieren, kann daraus geschlossen werden, dass es in irgendeiner Weise fehlerhaft ist. Der Umkehrschluß ist allerdings nicht zulässig. Für die Modellierung der Interoperationen von Robotersystemen und Umwelt impliziert dies, dass man im besten Fall ein *bestätiges* Modell erarbeiten kann, also ein Modell, dessen (möglichst zahlreiche) Vorhersagen mit (zahlreichen) Beobachtungen übereinstimmen. Allerdings lässt sich nicht beweisen, dass die Vorhersagen des Modells tatsächlich wahr sind.

Ogleich viele gute Gründe aus Sicht der Modelltheorie für eine Simulation sprechen, gibt es auch einige Nachteile zu beklagen. Die Erstellung eines akkurate Modells kann unter Umständen wesentlich mehr Aufwand erfordern, als die Experimente mit einem wirklichen Robotersystem. Die Roboterversuche liefern außerdem unmittelbar und korrekt die gewünschte Information, die Simulation dagegen immer nur Annahmen. Außerdem besteht die Gefahr, etwas zu simulieren, das in der wirklichen Welt kein Problem darstellt.

Wenn zum Beispiel zwei Robotersysteme an einer Kreuzung zum gleichen Zeitpunkt aufeinander treffen und auf Kollisionskurs sind, ist das gewöhnlich ein Problem in der Simulation, aber nicht in der wirklichen Welt. In einer wirklichen Situation wird fast immer eines der Robotersysteme etwas eher die Kreuzung erreichen und sie daher zuerst überqueren.

In anderen Fällen kann die Simulation die gestellte Aufgabe schwieriger darstellen als in der wirklichen Durchführung.

Ein Robotersystem bewegt sich durch seine Umgebung und verändert sie dabei gelegentlich. Während es Hindernisse umfährt, schiebt es sie manchmal leicht beseite, und Ecken werden „runder“. In einer Simulation geschieht dies nicht, und das simulierte Robotersystem bleibt in einer Ecke stecken, wo ein wirkliches Robotersystem das Hindernis zur Seite geschoben hätte.

Robotersysteme, deren Aufgabe und deren Umgebung müssen stets zusammen betrachtet werden. Ein bestimmtes Robotersystem wird sich bei der Durchführung der gleichen Aufgabe in verschiedenen Umgebungen auch unterschiedlich verhalten. Ebenso verhalten sich verschiedene Robotersysteme, die die gleiche Aufgabe in der gleichen Umgebung durchführen, unter Umständen unterschiedlich. Dies impliziert, dass die spezielle Interaktion eines spezifischen Robotersystems mit einer spezifischen Umgebung simuliert werden muss, um eine möglichst getreue Simulation zu erhalten.

Insgesamt besteht die Interoperationsanalyse im Wesentlichen darin, Prozess- und Strukturbeschreibungen anzufertigen, die Systemkomponenten mit unterschiedlichen operativen Kapazitäten (Bestands- und Flussgrößen, Hilfsgrößen und Konstanten) erfassen und durch grafische Hilfsmittel, wie Flussdiagramme, Ereignis-Reaktions-Modelle oder Kontextdiagramme in ihre wechselseitigen Abhängigkeit sowie in Relation zur jeweiligen Umgebung darzustellen. Im Endeffekt erhält man nach der Interoperationsanalyse eine operationale Prozess- und Strukturbeschreibung als Modell, das dann die Grundlage der Validierung im Rahmen von Computersimulationen bzw. als Grundlage für die Entwicklung des Robotersystems bildet.

---

## 2.6 Kognition

Im Mittelpunkt dieses Abschnitts steht zunächst der Begriff der menschlichen Kognition, der Prozesse des Wahrnehmens, Denkens, Urteilens und Schließens im Allgemeinen und die hierzu notwendigen Daten-, Informations- und Wissensverarbeitungsprozesse im Besonderen umfasst.<sup>43</sup> Die theoretischen Grundlagen für die praktische Ausgestaltung eines Kognitionsbegriffs liefert die Basis für ein Kognitionsmodell, das als Grundlage zur prozessualen und funktionalen Ausgestaltung kognitiver Robotersysteme dienen wird.

### 2.6.1 Kognitionen

Der Begriff Kognition wird in den einzelnen Wissenschaftsdisziplinen uneinheitlich verwendet, daher soll in diesem Abschnitt auch die Rede von Kognition(en) sein. So bedeutet das hiervon abgeleitete Verb „kognitiv“ ursprünglich bemerken oder erkennen. Kognition

---

<sup>43</sup> Siehe auch Schmidt 1992.

im weiteren Sinne schließt also alle Operationen ein, in denen Umweltinformationen über die Wahrnehmungssinne aufgenommen, verarbeitet, gespeichert und für die Entscheidungsfindung verwendet werden.<sup>44</sup>

Dabei werden beispielsweise Informationen Wörter, Zeichen und Symbole gekapselt, und unter Berücksichtigung gemachter Erfahrungen und anstehender Erwartungen analysiert, geordnet und interpretiert, woraus sich Urteile, Entscheidungen und Schlussfolgerungen ergeben, die dann ein zielgerichtetes Verhalten ermöglichen.

Solche Kognitionen beschreiben also diejenigen Fähigkeiten des Menschen, die es ihm ermöglichen, sich in der Welt zu orientieren und sich an seine Umwelt durch entsprechende Interoperationen anzupassen. Kognitionen kommen daher in den folgenden Ausprägungen zur Sprache:

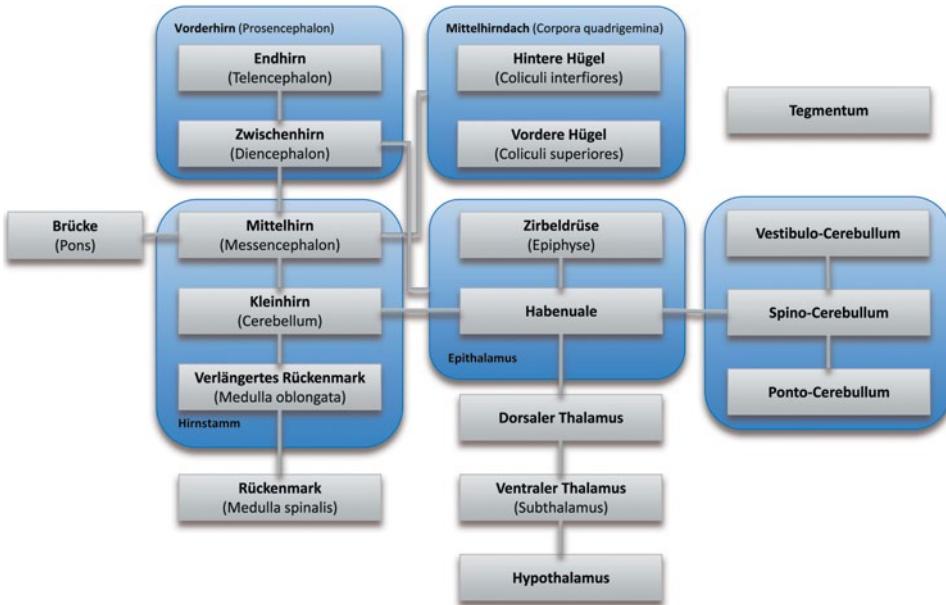
- Als begriffliche Klammer für alle Vorgänge oder Strukturen, die mit dem Gewahrwerden und Erkennen zusammenhängen, wie Wahrnehmung, Erinnerung, Vorstellung, Gedanke, Vermutung, Erwartung, Plan und Problemlösen, etc.
- Als eher allgemeiner Begriff für alle Formen des Erkennens und Wissens (Aufmerksamkeit, Erinnern, Urteilen, Vorstellen, Antizipieren, Planen, Entscheiden, Problemlösen, etc.) Er umfasst auch die Prozesse der mentalen Repräsentation (vgl. Zimbardo 1995).
- Als Ausdruck für jeden Prozess, durch den das Lebewesen Kenntnis von einem Objekt erhält oder sich seiner Umwelt bewusst wird (Wahrnehmung, Erkennen, Vorstellen, Urteilen, Gedächtnis, Lernen, Denken, oft auch Sprache).
- Als Bezeichnung für Prozesse, durch die Wahrnehmungen transformiert, reduziert, verarbeitet, gespeichert, reaktiviert und verwendet werden.
- Als Konnotation von Entdecken oder Wiederentdecken oder Wiedererkennen.

Solche Kognitionen werden gewöhnlich im menschlichen Gehirn verortet, einem kompakten, leistungsfähigen und sich kontinuierlich anpassenden System.

Das menschliche Gehirn gliedert sich dabei von vorn nach hinten in *Endhirn* (Telencephalon), *Zwischenhirn* (Diencephalon), *Mittelhirn* (Mesencephalon), *Brücke* (Pons), *Kleinhirn* (Cerebellum) und *Verlängertes Mark* (Medulla oblongata), das in das *Rückenmark* (Medulla spinalis) übergeht. Endhirn und Zwischenhirn bilden zusammen das *Vorderhirn* (Prosencephalon); Mittelhirn, Brücke und verlängertes Mark werden unter der Bezeichnung *Hirnstamm* zusammengefasst (Abb. 2.22).

Das *Verlängerte Mark* (Medulla oblongata) bildet die direkte Fortsetzung des Rückenmarks. Es ist der Ort des Ein- und Austritts der neunten bis zwölften Hirnnerven und enthält die motorischen und sensorischen Kerngebiete dieser Nerven. Diese sind wiederum umgeben von Kernen der *Formatio reticularis*, die sich von hier über die Brücke bis zum vorderen Mittelhirn zieht und an der Steuerung lebenswichtiger Körperfunktionen,

<sup>44</sup> Siehe auch Benjafield 1992.



**Abb. 2.22** Modellelemente des menschlichen Gehirns

wie Schlafen und Wachen, Blutkreislauf und Atmung sowie an Aufmerksamkeits- und Bewusstseinszuständen beteiligt ist.

Das *Mittelhirn* (Mesencephalon) gliedert sich in einen oberen (dorsalen) Teil, das Mittelhirndach (Vierhügelplatte, Corpora quadrigemina), und einen unteren (ventralen) Teil, das Tegmentum. Die Vierhügelplatte besteht aus den vorderen und hinteren Hügeln (Colliculi superiores und Colliculi inferiores). Die Colliculi superiores spielen eine wesentliche Rolle bei visuell und auditorisch ausgelösten Blick- und Kopfbewegungen sowie bei gerichteten Hand- und Armbewegungen und entsprechenden Aufmerksamkeitsleistungen. Die Colliculi inferiores sind wichtige Zentren des Hörsystems. Das Tegmentum enthält Anteile der Formatio reticularis sowie Zentren, die für Bewegung, Handlungssteuerung und vegetative Funktionen zuständig sind.

Das *Kleinhirn* (Cerebellum) besteht aus dem Vestibulo-Cerebellum, das mit der Steuerung des Gleichgewichts und der Augenbewegungen zu tun hat, (dem Spino-Cerebellum, das über das Rückenmark Eingänge von den Muskelspindeln erhält, und dem Ponto-Cerebellum, das an der Steuerung der feinen Willkürmotorik beteiligt ist. Das Kleinhirn ist über die *Brücke* (Pons) unter dem Einfluss der motorischen Großhirnrinde an der Feinregulierung der Muskeln beteiligt und stellt einen wichtigen Ort motorischen Lernens dar. Beim Menschen hat das Kleinhirn auch an kognitiven Leistungen und Sprache erheblichen Anteil, allerdings ohne dass dies dem Bewusstsein zugänglich ist.

Das *Zwischenhirn* (Diencephalon) liegt beim Menschen tief im Innern des Gehirns zu beiden Seiten des dritten Hirnventrikels und besteht aus Epithalamus, dorsalem Thalamus, ventralem Thalamus (auch Subthalamus genannt) und Hypothalamus. Der *Epithalamus*

besteht aus der Zirbeldrüse (Epiphyse), einem bei vielen Wirbeltieren lichtempfindlichen Organ, das die Biorhythmik beeinflusst (ihre Funktion beim Menschen ist unklar), und den Habenulae, einer Schaltstelle zwischen den Riechzentren (Olfaktorik) und dem Hirnstamm.

Der *dorsale Thalamus* stellt ein Konglomerat aus funktional sehr unterschiedlichen Kernen und Kerngebieten dar und ist mit der Hirnrinde (dem Cortex) über auf- und absteigende Fasern verbunden, die das *thalamo-corticale System* bilden. Traditionell werden die Kerne und Kernbereiche des dorsalen Thalamus in *pallio-thalamische* und *trunko-thalamische Kerne* eingeteilt. Die pallio-thalamischen Kerne erhalten spezifische Eingänge von subcorticalen Zentren und stehen ihrerseits mit eng umgrenzten Cortexgebieten gleicher Funktion in wechselseitiger Verbindung. Sie haben teils sensorische und motorische Funktionen, teils kognitive Funktionen oder limbische Funktionen. Zu den trunkothalamischen Kernen gehören die intralaminären Kerne sowie die Mittellinien-Kerne. Sie spielen bei der Regulation von Wachheits-Bewusstseins- und Aufmerksamkeitszuständen und bei der Verhaltensbereitschaft eine wichtige Rolle. Der ventrale Thalamus oder *Subthalamus* besteht vornehmlich aus dem Nucleus subthalamicus und dem Globus pallidus, die beide zu den für die Willkürmotorik zuständigen Basalganglien gezählt werden. Der *Hypothalamus* ist das wichtigste Regulationszentrum des Gehirns für vegetative Funktionen wie Atmung, Kreislauf, Nahrungs- und Flüssigkeitshaushalt, Wärmehaushalt, Biorhythmen und immunologische Reaktionen. Er beeinflusst lebens- und überlebenswichtiges Verhalten wie Flucht Abwehr, Fortpflanzung und Nahrungsaufnahme.

Das *Endhirn* (Telencephalon, beim Menschen auch Großhirn genannt, umfasst die Hirnrinde (Cortex cerebri) sowie subcorticale Anteile, die mit Aufmerksamkeit, emotionaler Bewertung und Verhaltenssteuerung zu tun haben. Hierzu gehören das Corpus striatum (Nucleus caudatus und Putamen), das basale Vorderhirn einschließlich der septalen Kerne und nicht-corticale Teile der Amygdala.

Bei der Großhirnrinde unterscheidet man den sechsschichtigen *Isocortex* und den drei- bis fünfschichtig aufgebauten *Allocortex*. Zum Allocortex gehören die Riehrinde (piriformer Cortex), der insulare Cortex (Insula), der cinguläre Cortex, Teile des Mandelkern-Komplexes (Amygdala), die Hippocampus-Formation (Subiculum, Ammonshorn, Gyrus dentatus) und die ihr benachbarte entorhinale, perirhinale und parahippocampale Rinde.

Der Isocortex wird in vier Lappen (Lobi, Singular Lobus) eingeteilt, und zwar den Hinterhauptsappen (Lobus occipitalis), den Schläfenlappen (Lobus temporalis), den Scheitellappen (Lobus parietalis) und den Stirnlappen (Lobus frontalis). Verschiedene Teile des Isocortex unterscheiden sich im Vorhandensein unterschiedlicher Zelltypen (Pyramidenzellen, dornenbesetzte und glatte Sternzellen usw.), in der Zellkörpergröße und Zelldichte, der Dicke der einzelnen Schichten und der Gesamtdicke zum Teil deutlich voneinander. Aufgrund dieser Unterschiede wird der Isocortex in 52 Hirnrindenareale eingeteilt. Diese werden mit „A“ (= Areal) bezeichnet und sind durchnummieriert (A1, A2).

Für die verschiedenen funktionellen Systeme (z. B. visueller, auditorischer, motorischer Cortex) sind zusätzliche Bezeichnungen gebräuchlich, die gegebenenfalls genannt werden (Tab. 2.2).

**Tab. 2.2** Funktionsübersicht des menschlichen Gehirns

Gliederung	Teil	Funktion
Hirnstamm	Formatio reticularis	Beteiligung an der Steuerung lebenswichtiger Körperfunktionen, wie Schlafen und Wachen, Blutkreislauf und Atmung sowie an Aufmerksamkeits- und Bewusstseinszuständen
Mittelhirn	Colliculi superiores	Spielen eine wesentliche Rolle bei visuell und auditorisch ausgelösten Blick- und Kopfbewegungen sowie bei gerichteten Hand- und Armbewegungen und entsprechenden Aufmerksamkeitsleistungen
	Colliculi inferiores	Zentren des Hörsystems
	Tegmentum	Zentren, die für Bewegung, Handlungssteuerung und vegetative Funktionen zuständig sind
Kleinhirn	Vestibulo-Cerebellum	Steuerung des Gleichgewichts und der Augenbewegungen
	Spino-Cerebellum	
	Ponto-Cerebellum	Steuerung der feinen Willkürmotorik
Brücke	In Verbindung mit dem Kleinhirn	Feinregulierung der Muskeln
Zwischenhirn	Epithalamus (Zirbeldrüse)	Biorhythmik
	Hypothalamus	Wichtigstes Regulationszentrum des Gehirns für vegetative Funktionen wie Atmung, Kreislauf, Nahrungs- und Flüssigkeitshaushalt, Wärmehaushalt, Biorhythmen und immunologische Reaktionen. Er beeinflusst lebens- und überlebenswichtiges Verhalten, wie Flucht Abwehr, Fortpflanzung und Nahrungsaufnahme

Die Gehirne aller Tiere, einschließlich des Gehirns der Menschen, sind aus zwei Haupttypen von Zellen aufgebaut, nämlich Nervenzellen, *Neurone* genannt, und *Gliazellen*. Letztere haben Stütz- und Versorgungsfunktionen; inwieweit sie spezifischer an der neuronalen Erregungsverarbeitung beteiligt sind, ist noch umstritten.

Vordergründig unterscheiden sich biologische Nervensysteme erheblich von der Architektur herkömmlicher Rechner. Statt eines Zentralprozessors findet sich ein hochkomplexes und kompliziert verdrahtetes Netzwerk, bestehend aus einfachen Aktivierungseinheiten, den Nervenzellen (Neuronen) und ihren Verbindungen (Synapsen). So schätzt man die Anzahl der Neuronen auf etwa  $10^{12}$  und diejenige der Synapsen auf etwa  $10^{15}$ . Folglich besitzt jedes Neuron im Mittel einige tausend Kontaktstellen mit anderen Neuronen. Da jedes Neuron mit etwa 1000 weiteren Neuronen verbunden ist, gilt bei einer Gesamtzahl von  $1000^4$  Neuronen, dass zwei beliebige Neuronen im Mittel über lediglich 4 Schaltstationen miteinander verdrahtet sind. Ähnlich wie der Coderaum des Genoms nutzt das Gehirn trotz hoher Komplexität eine sehr dichte Struktur. Im Vergleich hierzu enthält das

Rattengehirn etwa  $10^{10}$ , dasjenige einer Fliege aber größtenordnungsmäßig nur noch  $10^4$  Neuronen.

Vereinfacht ausgedrückt besteht eine Nervenzelle und damit auch das Neuron aus vier Komponenten:

- *Dendriten*: Sie sind die Eingangskanäle der Zelle. Diese Auswüchse des Zellkörpers empfangen von den Synapsen anderer Neuronen entsprechende Signale. Diese Signale werden dann im Zellkörper aufaddiert und bestimmen den Aktivierungszustand der Zelle. Sie bilden in ihrer Gesamtheit ein sich stark verästelndes System von Eingangsfasern.
- *Soma*: Als Soma bezeichnet man den Zellkern oder den eigentlichen Zellkörper der Nervenzelle. Im Zellkern werden die ankommenden Signale anderer Neuronen im Laufe einer bestimmten Zeitspanne aufsummiert.
- *Axon*: Das Axon ist das Gegenstück zu den Dendriten einer Nervenzelle. Es ist der Ausgangskanal der Zelle. Über diesen Auswuchs des Somas wird der Nervenimpuls an andere Zellen weitergegeben, wenn die aussendende Nervenzelle ihren internen Aktivierungsschwellenwert durch die Addition der eingegangenen Impulse überschritten hat und einen Impuls aussendet. Das Ende des Axons kann als Hauptfortsatz in viele einzelne Axone als Endverzweigungen unterteilt sein, die mit den Synapsen der nachfolgenden Nervenzellen in Verbindung stehen.
- *Synapsen*: Die Synapsen sind die eigentlichen Übertragungspunkte zwischen den einzelnen Neuronen. Genauer betrachtet bezeichnet man als Synapse nicht einen Teil der Zelle, sondern im Prinzip ist es der Spalt zwischen den Axonen der sendenden Zelle und den Dendriten der empfangenden Zellen. Einige Synapsen tragen dazu bei, wenn sie aktiviert sind, dass ein Neuron „feuert“. Diese Synapsen werden erregende Synapsen (exzitatorische Synapsen) genannt. Andere vermindern die Feuerbereitschaft einer Zelle, diese heißen hemmende Synapsen (inhibitorische Synapsen).

Über diese synaptischen Verbindungen am Dendriten sammelt das Neuron Aktivitäten aller mit ihm verschalteten Neuronen auf und reagiert dann seinerseits mit einer axonalen Antwort. Dabei arbeitet es im Prinzip wie ein Schwellwertschalter: Ist die Summe der einlaufenden Aktivitäten groß genug, reagiert es mit Feuern, welches auf andere Neuronen geleitet wird, deren Dendriten wiederum diese neuronale Aktivität abgreifen. Man unterscheidet exzitatorische und inhibitorische Neuronen, also solche, deren Wirkung postsynaptisch entweder erregend oder hemmend ist. In den unterschiedlichen Hirnarealen findet man einige hundert morphologische Typen von Nervenzellen.

Neuronale Aktivität ist elektrochemischer Natur. Das Nervensignal einer Zelle entspricht einer kurzzeitigen Änderung des elektrischen Potentials in der Größenordnung von 50–80 mV auf einer Zeitskala von wenigen Millisekunden. Das Feuern von Nervenspikes kann mehrere Sekunden bis hin zu Minuten anhalten. Die Signalübertragung an der Synapse erfolgt entweder elektrisch über synaptische Verbindungskanäle oder chemisch durch Ausschüttung von Neurotransmittern, also chemischen Botenstoffen, die die syn-

aptische Membranleitfähigkeit durch das Öffnen oder Schließen von Ionenkanälen verändern. Die Schaltgeschwindigkeiten von Synapsen variieren erheblich und sind abhängig von der funktionalen Rolle.

Diese Veränderungen der synaptischen Verbindungsstärken stellen die molekularbiologische Basis für die so wesentlichen Eigenschaften des Gehirns wie Lernen, Assoziationsfähigkeit und Gedächtnis dar. Als Hypothese wurde dies erstmals von Donald Hebb (1949) formuliert. Sinngemäß lautet seine experimentell seither bestens bestätigte Regel der synaptischen Plastizität (Veränderbarkeit): Die gleichzeitige Aktivierung zweier miteinander verbundener Zellen sollte zu einer Veränderung der Verbindungsstärke führen, derart, dass sich die Wahrscheinlichkeit des Feuerns der postsynaptischen Zelle erhöht, falls die präsynaptische Zelle aktiv ist.

Neuronale Netze sind massiv parallel organisierte Strukturen mit Neuronen als einfachen Aktivierungseinheiten, deren synaptische Verbindungen untereinander einer regelhaften,aktivitätsabhängigen Veränderbarkeit unterliegen. Neuronen codieren dabei nicht ganze Symbole, sondern selektiv einzelne Merkmale aus der Menge gegebener Sinnesdaten.<sup>45</sup>

Als nächsthöhere Einheit über dem Neuron und der Synapse kann man sogenannte *neuronale Karten* betrachten. Hierbei handelt es sich um Neuronenfelder oder -schichten, welche zumeist auch anatomisch sichtbar sind und innerhalb derer eine spezielle Datenrepräsentation bezüglich einer bestimmten Aufgabenstellung erfolgt.

In Anlehnung an im Gehirn ablaufende Vorgänge führt dies zu massiv parallelen, konnektionistischen Modellen. Deren Verarbeitungselemente ähneln stark vereinfachten Neuronen, die nur sehr einfache Operationen, wie die Summation von Eingabewerten, den Vergleich der Summe gegen einen Schwellwert und die Erzeugung eines Ausgabesignals ausführen. Sie sind untereinander dicht verknüpft, wobei die Verbindungen üblicherweise mit einem Gewicht versehen sind, das die Stärke der Verbindungen ausdrückt. Positive Gewichte üben einen verstärkenden Einfluss auf die Aktivität des betreffenden Elements aus, negative einen abschwächenden. Das Wissen wird nicht auf symbolische Weise dargestellt und verarbeitet. Vielmehr verarbeiten und übertragen einzelne Elemente lediglich Aktivierungswerte beschränkter Genauigkeit und sind nicht in der Lage, mit Symbolen oder Symbolfolgen umzugehen.

## 2.6.2 Kognitionstheorie

In dem letzten Abschnitt wurde der Begriff der Kognition als Bezeichnung von solchen Leistungen des menschlichen Gehirns verstanden, die am Zustandekommen intelligenten Verhaltens beteiligt sind. Diese umfassen nicht nur theoretische Fähigkeiten, wie das Erinnern oder Sprechen, sondern auch das planvolle Umsetzen der eigenen Wünsche und die Organisation von Handlungsweisen, die positive Gefühle bringen und negative Gefühle

---

<sup>45</sup> Siehe auch Ritter et al. 1969.

vermeiden. Insofern schließt der bisher erarbeitete Begriff der Kognition sowohl motivationale als auch emotionale Zustände ein.<sup>46</sup> Speziell für den Anforderungsbereich der Robotik gilt es nun aber auch, die motorische Steuerung in den Leistungsumfang aufzunehmen.

Eine solche umfassende Kognition ist der Gegenstand der Kognitionstheorie, wie sie im Rahmen dieses Buches speziell für die Robotik entwickelt wird. Das Ziel dabei ist, Theorien der Prozesse und Repräsentationen zu entwickeln, die intelligentem Verhalten zugrunde liegen. Ein weiteres Ziel besteht darin, diese Entwicklung so zu gestalten, dass sie als exakte und implementierbare Modelle einer empirischen Überprüfung zugeführt werden können. Die Kognitionstheorie liefert sozusagen die Grundlage dafür, dass eine artifizielle Kognition funktional und prozessual beschrieben und ausimplementiert werden kann.

Die Kognitionstheorie dieses Buches geht von der Annahme aus, dass Kognition als solches ein physisches System ist, da es in einem solchen implementiert ist. Des Weiteren orientiert sich diese Kognitionstheorie am Funktionalismus. Der Funktionalismus geht dabei auf die Theorie formaler Systeme und die Idee der universellen Berechenbarkeit durch Turingmaschinen zurück. Die Orchestrierung der funktionalen und prozessualen Komponenten bildet die kognitive Organisation ab, wobei die einzelnen Komponenten auf sehr verschiedene Weise physisch implementiert werden können. Die Implementierung eines entsprechenden Modells ermöglicht dabei eine direkte empirische Überprüfung. Die Erfolge und Misserfolge dieser Vorgehensweisen stellen im Umkehrschluss den klassischen Kognitionswissenschaften wertvolles Erfahrungs- bzw. Erkenntnismaterial zur Verfügung.<sup>47</sup> Insgesamt ermöglicht die Theorie, dass eine Implementierung von Kognition in einem künstlichen und damit anorganischen System angestrebt werden kann. Eine solche Theorie der Kognition wird daher am Ende sowohl den Menschen mit seiner Kognition als auch Robotersysteme mit ihrer artifiziellen Kognition erfassen können.

Bei der Modellentwicklung werden dabei mehrere Ebenen der Betrachtung getrennt. Zunächst geht es auf der funktionalen Ebene um die Modellierung von Teilleistungen, den dadurch bewirkten inneren Zuständen. Auf dieser Ebene sind die höheren mentalen Zustände und Repräsentationen, wie beispielsweise Emotionen, Intentionen, Intuitionen etc. angesiedelt. Auf der Ebene der funktionalen Architektur wird beschrieben, wie die einzelnen funktionalen Komponenten im Rahmen eines Prozesses zusammenarbeiten und welche mentalen Komponenten an der betrachteten Leistung beteiligt sind. Auf der Ebene der Implementierung bzw. physischen Struktur wird betrachtet, wie die betreffende Leistung und damit die Algorithmisierung in das konkrete Trägersystem eingebaut sind. Die funktionale, architektonische und die algorithmische Ebene zusammen werden dann den zentralen Begriff „kognitives System“ bilden. Dem Verständnis dieses Buches nach ist ein kognitives System ein System, das ähnlich wie Menschen anhand von Repräsentationen seiner Umwelt, in diese nicht nur eingreift, um seine Ziele zu verwirklichen, sondern mit

<sup>46</sup> Siehe auch Heckhausen 1989.

<sup>47</sup> Siehe auch Gardner 1989.

dieser Umwelt interoperiert, d. h. auf diese einwirkt und diese Umwelt wiederum auf das System zurückwirkt.

Die artifizielle Kognition eines solchen kognitiven Systems wird als ein informationsverarbeitendes System aufgefasst, wobei bei der Modellierung und Implementierung sowohl der symbolischer als auch der subsymbolischer Informationsverarbeitungsansatz zur Anwendung kommt.

Beim Ansatz der Symbolverarbeitung im Rahmen eines *Symbolismus* müssen die unterschiedlichen Konzepte von Daten, Information und Wissen durch Symbole dargestellt werden, um diese entsprechend verarbeiten zu können. Das Wissen wird dabei oft in Form von Regeln, logischen Formeln, semantischen Netzen, Frames oder ähnlichen Konzepten dargestellt. Nahezu alle Darstellungsmethoden beruhen auf der Verwendung von Symbolen zur Identifikation elementarer Objekte. Ein Symbol ist dadurch gekennzeichnet, dass es als solches „für etwas steht“, d. h. mit dem Symbol wird ein Gegenstandsbewusstsein intentional erzeugt. Komplexe Objekte werden dann durch zusammengesetzte Strukturen auf Basis von Symbolen dargestellt und verarbeitet. Dabei gilt das Verarbeitungsprinzip der algebraischen Berechnung, das besagt, dass ein Symbol nur ein Symbol in der Form sein kann, wie es für das zu verarbeitende System erkennbar ist. Bereits aus dieser Tatsache lassen sich basale Grundsätze ableiten, die für die Ausprägung des Symbolverarbeitungsansatzes wichtige Orientierungspunkte sind:

- Der Mensch als ein Organismus in einer informationsbeladenen Umgebung, nimmt aus letzterer Informationen auf, um sie intern zu repräsentierten und durch Symbolmanipulation zu verarbeiten.
- Kognition basiert auf Informationsverarbeitung.
- Informationsverarbeitung beruht auf der Manipulation von Symbolen im Rahmen symbolischer Repräsentationen.
- Symbole werden durch Regeln manipuliert, die sich nur auf die syntaktische Form der Symbole und nicht auf die Semantik beziehen.
- Die Angemessenheit der Funktionsweise eines solchen kognitiven Systems ist dann bereits gegeben, wenn es zu einer adäquaten Repräsentation der realen Welt und zu einer erfolgreichen Lösung gestellter Probleme führt.

Wissen ist hierbei darstellbar durch endliche Strukturen, die aus diskreten atomaren Symbolen zusammengesetzt sind, in Übereinstimmung mit einer endlichen Anzahl syntaktischer Relationen.

Die anwendungsorientierte Forschung wird immer noch sehr stark von einem solchen Symbolverarbeitungsansatz beeinflusst. Das Symbolverarbeitungsparadigma bestimmt die Forschung allerdings nicht allein. So wird auch in diesem Buch im Rahmen der Implementierung beispielweise auch mit neuronalen Netzen gearbeitet, denen das Paradigma des Konnektionismus zugrunde liegt. Im weiteren Verlauf konkurrieren also nicht beide Richtungen miteinander, sondern kooperieren, um entsprechend der jeweiligen Problemstellung zur Anwendung zu kommen.

Der konnektionistische Ansatz hingegen orientiert sich an subsymbolischer Informationsverarbeitung. Im Rahmen der späteren Implementierung werden unter dem Begriff des *Subsymbolismus* im Allgemeinen und dem Konnektionismus im Speziellen informationsverarbeitende Systeme behandelt, die aus sehr vielen einfachen, untereinander verbundenen und Informationen austauschenden Verarbeitungselementen bestehen. In konnektionistischen Systemen steckt daher das „Wissen“ in der Verbindungsstruktur, den Gewichten der einzelnen Verbindungen sowie auch in den Eigenschaften der einfachen Verarbeitungselemente.

Im Mittelpunkt steht das Konzept neuronaler Netze, wo neben dem Symbolbegriff auch die strikte Trennung von funktionaler Ebene und Implementierung aufgeweicht wird. Neuronale Netze repräsentieren eher holistisch, zeigen ein Lernvermögen, arbeiten nicht sequentiell, sondern parallel und benötigen auch keinen zentralen Prozessor, der das Gesamtgeschehen kontrolliert.

Dabei kann der konnektionistische Ansatz auf eine Reihe experimenteller Erfolge hinweisen. So hat sich dieser Ansatz in der Robotik beispielsweise beim Manövrieren durch Räume als hilfreich erwiesen, wo die Wahrnehmung und Bewegung lernfähigen Netzstrukturen überlassen wird, statt sie dediziert von vorneherein festzulegen und sie direkt zu programmieren. So lassen sich denn auch Klassifikations- und Mustererkennungsaufgaben besser konnektionistisch, regelbasiertes Entscheidungsvermögen besser symbolisch modellieren.

### 2.6.3 Kognitionsbegriff

Die artifizielle *Kognition* bezeichnet den zentralen Untersuchungsgegenstand der Wissenschaftsdisziplin des Cognitive Computing. Er grenzt sich von dem Begriff der (natürlich-menschlichen) Kognition der Kognitionswissenschaft, der Kognitionspsychologie sowie von Teilebereichen der Neurowissenschaften, der Linguistik, Philosophie, Informatik usw., ab, wo jede der mit Kognition befassten Disziplinen ihren eigenen methodischen Zugang zu diesem Gegenstand hat.<sup>48</sup> Unter diesem Kognitionsbegriff fasst man diejenigen Funktionen zusammen, die das Wahrnehmen und Erkennen, das Enkodieren, Speichern und Erinnern sowie das Denken, Problemlösen, das Lernen aus Erfahrung sowie die motorische und kommunikative Verhaltenssteuerung eines artifiziellen Systems umfassen. Demnach ist ein *artifiziell-kognitives System* ein technisches System, das die o. a. genannten kognitiven Funktionen als Ganzes, d. h. systemisch, aufweist.

Eine solche Kognition interveniert zwischen Wahrnehmung über Sensoren und Verhalten über Aktoren. Die Verarbeitung der Eingabe und die Produktion der Ausgabe erfolgt im Rahmen eines *artifiziell-kognitiven Prozesses* auf Basis gespeicherten Wissens. Wahrgenommene Stimuli unterliegen demnach einer Folge von mentalen Operationen innerhalb des kognitiven Prozesses. Es wird eine Abfolge von automatischer Verarbeitung zu einer Verarbeitung mit ansteigendem kognitiven Aufwand angenommen. Die jeweilige

<sup>48</sup> Vgl. Rechenberg 2000.

Tiefe der Verarbeitung (depth of cognitive processing) hängt von den Stimuli, den Zielen des kognitiven Systems, der Umwelt und von der verfügbaren Zeit ab.

Umgangssprachlich gesprochen erfährt das System seine Umgebung damit durch Sensoren, Perzeptoren und Effektoren, die ihm einen Eindruck von einem bestimmten Muster übermitteln. Dieses Modell entspricht der Struktur des menschlichen Gehirns, das ebenfalls auf sich selbst bezogen (Strukturdeterminiertheit und Selbstreferentialität als „Autopoiesis“) zu sein scheint, indem es keine Außenweltreize direkt verarbeitet, sondern elektrische Impulse als Perturbationen, sozusagen angestoßen durch Störungen eigenständig interpretiert und erst dadurch in ein subjektiv sinnvolles Ganzes integriert. Gerade der Begriff „Autopoiesis“ (griechisch Selbsterzeugung) verstärkt die Annahme, dass das Gehirn ein in sich geschlossenes, weil von außen nur durch Perturbationen und demzufolge nicht durch Inputs, sondern nur durch Impulse, anzuregendes, auf sich selbst bezogenes System ist, das sich zwar nicht selbst erhalten kann, aber in Bezug auf die Interpretation seiner Umwelt eigenständig agiert, also keine Informationen aufnimmt, sondern Konstruktionen in Form von Handlungen (Interaktionen, Interoperationen, Absichten etc.) vornimmt.

Zum Kern einer solchen Kognition sind auch diejenigen Prozesse zu rechnen, die auf mentalen Repräsentationen operieren und für die deshalb ein Gedächtnis vorhanden sein muss, wobei letzteres nicht nur die Fähigkeit zur Akkumulation und Speicherung von Wissen, sondern auch das Löschen von Wissen mitbringen muss. Gemäß dieser Auffassung sind auch solche Prozesse zu inkludieren, die der Weiterverarbeitung sensorischer Daten über die Wahrnehmung sowie der Planung und Steuerung von Verhalten über Aktoren dienen. Auch die Bereiche der artifiziell-systemischen Motivation und Emotion bilden einen aktiven kognitiven Anteil und sind somit unmittelbar mit der artifiziell-systemischen Kognition verbunden.

Eine solch umfassende Sichtweise der artifiziell-systemischen Kognition setzt die Existenz interner, mentaler Operationen und Systemzustände voraus, da diese zum einen für die Daten-, Informations- und Wissensverarbeitung konstitutiv sind. Zum anderen bilden diese mentalen Zustände die Grundlage für die Fähigkeit solcher Systeme zur Repräsentation situations- und handlungsrelevanter Aspekte aus der Umwelt sowie zur Realisierung selbstreflexiver Aspekte.

Im Sinne der Wissenschaftstheorie ist Cognitive Computing als ein interdisziplinäres Forschungsparadigma aufzufassen, dessen charakteristische Methode zum einen in der Modellierung artifiziell-systemischer Kognition besteht, die dann durch die Ausimplementierung dieser Modelle auf Computer bzw. auf Robotersysteme als realisierte artifiziell-kognitive Systeme der Validierung durch empirische Analyse unterzogen werden.<sup>49</sup>

Ein Ergebnis der Forschungen in Anlehnung an eine autopoietische Konzeption ist die begründete Hypothese, dass ein im Rahmen der Implementierung materiell realisiertes Wissensverarbeitungssystem die notwendigen und hinreichenden Voraussetzungen für eine artifizielle Kognition mit sich bringt. Dies inkludiert die Annahme, dass kognitive

---

<sup>49</sup> Siehe auch Chalmers 1986.

Prozesse nicht an biologische Organismen gebunden sind, sondern auch von künstlichen Systemen und Maschinen, wenngleich in strukturell anderer Form, realisiert werden können. Aus dieser Annahme wiederum folgt, dass Kognition ein systemisches Phänomen sein kann und diese Hypothese wird als *Knowledge Systems Hypothesis* (KSH) bezeichnet. Gemäß dieser Hypothese wird Kognition nicht nur als ein logisches oder biologisches Problem, sondern auch als ein mögliches systemisches Geschehen gesehen.

Ein Wissensverarbeitungssystem ist zunächst ein formales System, das, wie jedes formale System, über ein Alphabet von Zeichen (atomare Elementarsymbole) verfügt sowie über syntaktische Regeln zur Bildung und Transformation von Symbolstrukturen (Ausdrücken) zur Modellierung von Daten-, Informations- und Wissensstrukturen prinzipiell beliebiger Komplexität. Symbole im Sinne dieser Hypothese sind arbiträre Zeichen für Entitäten oder Prozesse, einschließlich von Prozessen des Systems selbst. Die Semantik dieser Symbole ist zum einen bestimmt durch die Bezugnahme auf eine Entität, und zwar dergestalt, dass in Abhängigkeit von einem entsprechenden symbolischen Ausdruck das System entweder die Entität beeinflusst oder umgekehrt. Zum anderen dadurch, dass ein Ausdruck einen für das System ausführbaren Prozess in Form eines Programmes darstellt, das dann interpretiert, also ausgeführt wird. Diese Auffassung ist gleichbedeutend mit der Behauptung, dass artifizielle Kognition bzw. kognitive Prozesse als Berechnungen aufzufassen sind. In diesem Sinne wird hier ein philosophischer Funktionalismus in seiner starken Ausprägung vertreten, in Anerkennung physischer Phänomene als Grundlage mentaler Phänomene bei gleichzeitiger Anerkennung der Nichtreduzierbarkeit des Mentalen auf Gehirnprozesse. Mentale Zustände werden nicht direkt auf physische Zustände reduziert, sondern mit funktionalen Zuständen identifiziert. Diese lassen sich dann in vielfältiger Weise beschreiben, so mit mathematischen Ausdrücken, Programmiersprachen oder einer an eine natürliche Sprache angelehnten wissenschaftlichen Sprache. Dies ermöglicht auch die materielle Realisierung der funktionalen Zustände auf Basis von Computern. Wenn funktionale Zustände also nicht an bestimmte physische Realisationen gekoppelt sind, wird es möglich sein, eine materielle Basis zu finden, die mit Mitteln der Wissenschaft zu beherrschen ist und auf der sich funktionale Zustände realisieren lassen. Mit Hilfe dieses Werkzeuges wird es möglich, die realisierten Zustände „*in vitro*“ zu betrachten. Da mentale auf funktionale Zustände abgebildet werden können – die Annahme ist ja, dass eine Reduktion mentaler auf funktionale Zustände möglich ist – kann ein Erklärungsansatz für das Zustandekommen von kognitiven und mentalen Zuständen gegeben werden. Diese Grundhypothese des Cognitive Computing, wonach kognitive Prozesse als solche der Berechnung zu betrachten sind, kann bis dato prinzipiell nicht bewiesen, aber auch nicht widerlegt werden. Bis letzteres erreicht ist, wird die Grundhypothese aufrechterhalten, um sich als forschungsleitendes Prinzip zu bewähren und zum Erfolg in Bezug auf die Entwicklung kognitiver Systeme beizutragen.

Neben diesem Symbolverarbeitungsansatz vertritt das Cognitive Computing auch einen erweiterten Konnektionismus. Dieser Konnektionismus teilt mit dem Symbolverarbeitungsansatz die Grundthese, dass Kognition Informationsverarbeitung ist und auch die, dass kognitive Prozesse Berechnungen über Strukturen darstellen, die ihrerseits als



**Abb. 2.23** Modell artifiziell-kognitiver Systeme

Repräsentationen aufzufassen sind. Hinzu kommt allerdings ein Architekturansatz, der ein aus vielen sehr einfachen Verarbeitungseinheiten bestehendes Verarbeitungssystem vorsieht. An die Stelle der Symbole, in Form von bedeutungshaltigen Einheiten, treten subsymbolische Repräsentationen. Eine solche subsymbolische Verarbeitung bedeutet, dass das, was gemeinhin als ein Symbol dargestellt wird, nunmehr durch einen Vektor von Werten repräsentiert wird, die erst in ihrer Gesamtheit die Repräsentation ergeben (Abb. 2.23).

Insgesamt erscheint damit die artifizielle Kognition nicht mehr nur als ein tendenziell universales, aller Wahrnehmung und allem Handeln zugrundeliegendes Phänomen, das als Berechnung analysiert und synthetisiert werden kann. Vielmehr scheinen sich artifiziell-kognitive Phänomene mit Hilfe eines aus vielen sehr einfachen Verarbeitungseinheiten bestehenden Verarbeitungssystem realisieren zu lassen.

Das kognitive Modell trägt auch der Tatsache Rechnung, dass intelligentes Handeln situationsbezogen ist, indem beispielsweise eine zielorientierte Planung in hohem Maße von Hinweisen aus der Umwelt abhängig ist. Diese Sichtweise inkludiert die Annahme, dass intelligentes Handeln insgesamt als zielgerichtetes und zielgesteuertes Verhalten in einem gegebenen Problemraum aufgefasst werden kann. Dass Problemlösen als Ableitung von Handlungen aus vorhandenem Wissen verstanden wird, impliziert, dass die Existenz einer äußeren Welt nicht nur angenommen wird, sondern dass diese für die Problemlösungsfähigkeiten sehr wichtig ist. Die Kombination von Reizen aus einer reichhaltigen Welt und einem Langzeitspeicher mit ebenso reichhaltigem Inhalt, also Wissen, ermög-

licht erst ein differenziertes und erfolgreiches Problemlösungsverhalten. Insofern sieht das kognitive Modell die Berücksichtigung *situativer Bedingungen* vor, was dazu führt, dass das artifiziell-kognitive System nicht isoliert agiert, sondern sich gerade hinsichtlich seiner kognitiven Aspekte vor allem in der Interoperation mit der Umwelt konstituiert.

Weiterhin berücksichtigt das Modell, dass Kognition neben der intrinsischen kognitiven Entwicklung des einzelnen Systems gerade auch durch dessen Interoperieren (Handeln) mit der Umwelt und anderen Systemen sich entwickelt. Artifizielle Kognition ist somit auch ein extrinsisches (systemisch-soziales) Phänomen. Konkret wirkt sich diese Berücksichtigung zunächst dahingehend aus, dass das kognitive Modell im Rahmen der späteren Implementierung die Konzeption multipler Agenten unterstützt. Weiterhin berücksichtigt das kognitive Modell die Anforderungen, die seitens interagierender, interoperierender und kommunizierender Systeme gestellt werden. So erfolgen die Handlungskontrolle und das Management von Zielkonflikten durch *Motivation*.

In dem Fall, dass kognitive Systeme nicht nur Information aufnehmen, sondern aus ihrer Umgebung auch Perturbationen (d. h. Störungen) ihres Gleichgewichts erfahren, sorgt die *Emotion* durch geeignete kompensatorische Maßnahmen – nämlich kognitive Prozesse – für den notwendigen Ausgleich.

Autonome Systeme sehen sich gleichzeitig mit einer Menge oft konkurrierender Ziele konfrontiert, die es zu bewältigen gilt. Auch gilt es, Lösungen trotz unter Umständen vieler und schlecht spezifizierter Aufgaben zu entwickeln, die neben einem *planenden Denken* auch eine gewisse *Intuition* erfordern. Auch die Berücksichtigung der Situation erfolgt bei der Handlungsplanung im Rahmen des planenden Denkens. Dabei ist, im Unterschied zu einem einzigen System, das Verhalten der anderen Systeme nicht exakt vorhersagbar, was eine Koordination und Kooperation bedingt. Da unter Umständen in einem solchen Multiagentensystem die einzelnen Agenten nicht genau spezifiziert sind, stellt sich das Problem, wer zu einem gegebenen Zeitpunkt was machen soll. Auch diese Koordination fällt unter das planende Denken unter Einbeziehung der *Persönlichkeitsmerkmale*. Letztlich gilt es, Kommunikation zu betreiben, was Sprachen, Kommunikationsmedien sowie eine Politik der Kommunikation erfordert. Diese Kommunikation wird auch über das *Verhalten* gesteuert.

Insgesamt eignet sich das kognitive Modell für die Ausgestaltung dynamischer Systeme. Dem liegt auch die Erkenntnis zugrunde, dass die Bestandteile intelligenter Systeme selbst keineswegs intelligent sein müssen. Kognition erscheint dann als emergent, als eine aus der Komplexität dynamischer Systeme unter bestimmten Randbedingungen wie von selbst entstehende Systemeigenschaft.

Das Modell folgt einer Subsumtionsarchitektur, indem der kognitive Prozess auf eine geschichtete Architektur von Teilsystemen verteilt ist (Perzeptives System, Situatives System, Epigenetisches System, Intentionales System, Effektorisches System). Die Koordination dieser Teilsysteme erfolgt multidirektional jeweils auf der gleichen Ebene (anstatt „von oben“ bzw. von einer zentralen Instanz). Allerdings können die einzelnen Systeme sich gegenseitig insofern beeinflussen, als sie sich untereinander hemmen oder aktivieren und untereinander Parameter übermitteln können. Erst das kognitive System als Ganzes reali-

siert ein kognitives Verhaltensmuster und ist durch spezifische Sensoren und Aktoren mit der Umwelt des Systems verbunden.

An dieser Stelle gilt es spekulativ festzuhalten, dass in diesem kognitiven Modell die Integration der Systeme zu einem Ganzen als eine Art systemisches Bewusstsein gesehen werden kann. Vielleicht ist es diese Integration, die beim Menschen, der zum Teil dieselben funktionalen Systeme benutzt, ein einheitliches Erleben schafft.

Das artifiziell-kognitive Modell bildet so eine Brücke zwischen den Anforderungen einer systemischen Intelligenz hin zur Computersimulation von artifizieller Kognition. Aber es führt auch ein Weg über die Brücke zurück, indem die Simulationen unter Umständen Rückschlüsse auf biologischen Funktionen zulassen und dann wiederum die Entwicklung und Veränderung von kognitiven Systemen ermöglichen.

---

## Literatur

- Amberg, M.: Prozessorientierte betriebliche Informationssysteme. Springer Verlag, Berlin (1999)
- Bamme, A., Feuerstein, G. (Hrsg.): Maschinen-Menschen, Mensch-Maschinen. Reinbeck b. Hbg. (1983)
- Barber, P.: Applied Cognitive psychology. An Information-Processing-Framework. London (1988)
- Benjafield, J.C.: Cognition. Engelwood Cliffs (1992)
- Booch, J., Rumbaugh, J., Jacobson, I.: Unified modeling Language User Guide. Addison Wesley Lognman (1999)
- Bunse, C., von Knethen, A.: Vorgehensmodelle kompakt. Spektrum Akad. Verlag, Heidelberg (2002)
- Capurro, R.: Leben im Informationszeitalter. Akademie, Berlin (1995)
- Carnap, R.: Der logische Aufbau der Welt. Weltkreis, Berlin. (Nachdruck: Philosophische Bibliothek, Bd. 514. Meiner, Hamburg) (1998)
- Chaitin, G.J.: Algorithmic Information Theory. Cambridge University Press, Cambridge (1987)
- Chalmers, A.F.: Wege der Wissenschaft. Einführung in die Wissenschaftstheorie. Berlin u. a. (1986)
- Ebeling, W., Freund, J., Schweitzer, F.: Komplexe Strukturen: Entropie und Information. Teubner, Leipzig (1998)
- Eisenführ, F., Weber, M.: Rationales Entscheiden. Springer, Berlin (1999)
- Fodor, J.A.: The Language of Thought. Crowell, New York (1975)
- Gardner, H.: Dem Denken auf der Spur. Der Weg der Kognitionswissenschaften. Stuttgart (1989)
- Gerke, P.: Wie denkt der Mensch? Informationstechnik und Gehirn. München (1987)
- Hansen, H.R., Neumann, G.: Wirtschaftsinformatik I, 8. Aufl. Lucius&Lucius, Stuttgart (2001)
- Hauffe, H.: Der Informationsgehalt von Theorien. Springer, Wien (1981)
- Hebb, D.O.: The organization of behavior: a neuropsychological theory. Wiley (1949)
- Heckhausen, H.: Motivation und Handeln. Springer, Berlin (1989)
- Heintz, B.: Die Herrschaft der Regel. Zur Grundlagengeschichte des Computers. Frankfurt a. M. (1993)
- Hofstadter, D., Dennet, D. (Hrsg.): Einsicht ins Ich. Stuttgart (1981)
- Kitano, H.: RoboCup-97: Robot Soccer World Cup I. Springer, Berlin (1998)
- Kratky, K.W., Bonnet, E.M. (Hrsg.): Systemtheorie und Reduktionismus. Wien (1989)
- Krieger, D.J.: Einführung in die allgemeine Systemtheorie. Fink, München (1996)
- Kriz, J., Heidbrink, H.: Wissenschafts- und Erkenntnistheorie. Opladen (1990)

- Küppers, B.O. (Hrsg.): *Ordnung aus dem Chaos*. Piper, München (1987)
- Mandl, H., Spada, H: *Wissenschaftspsychologie. Psychologie* Verlags Union (1988)
- Martin, C.: *Rechnerarchitekturen*. Fachbuchverlag Leipzig (2001)
- Mc Carthy, J., Hayes, P.: Some philosophical problems from the standpoint of artificial intelligence.  
In: Metzler (eds.) 1969
- McMenamin, S.M., Palmer, J.F.: *Essentiell System Analysis*. Prentice Hall, Englewood Cliffs (1984).  
(Deutsche Ausgabe: *Strukturierte Systemanalyse*, Hanser, München, 1988)
- Mittelstaedt, P.: *Philosophische Probleme der modernen Physik*. B.I.-Wissenschaftsverlag, Mannheim (1963)
- Nisbett, R., Ross, L.: *Human Inference. Strategies and Shortcomings of Social Judgement*. Englewood Cliffs: Prentice Hall.
- Norman, D., Rumelhart, D.: *Strukturen des Wissens. Wege der Kognitionsforschung*. Stuttgart (1978)
- Oberliesen, R.: *Information, Daten und Signale*. Reinbeck b. Hbg. (1982)
- Oeser, E.: *Wissenschaft und Information*, 4 Bd. Oldenbourg, Wien (1976)
- Popper, K.: *Logik der Forschung*. Springer, Wien (1935)
- Popper, K.: *Objektive Erkenntnis. Ein evolutionärer Entwurf*. Hamburg (1973)
- Rechenberg, P.: *Was ist Informatik?*, 3. Aufl. Hanser, München (2000)
- Rembold, U., Levi, P.: *Einführung in die Informatik für Naturwissenschaftler und Ingenieure*, 4. Aufl.  
Hanser, München (2003)
- Rieseier, H.: *Roboterkinematik – Grundlagen, Invertierung und symbolische Berechnung*. Vieweg,  
Braunschweig Wiesbaden (1992). (Fortschritte der Robotik 16)
- Ritter, H., Martinetz, T., Schulten, K. (Hrsg.): *Neuronale Netze*. Addison-Wesley, Bonn (1969)
- Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorenson, W.: *Object-Oriented Modelling and Design*. Prentice-Hall, Englewood Cliffs (1991)
- Saake, G., Sattler, K.-U.: *Algorithmen und Datenstrukturen*. dpunkt, Heidelberg (2002)
- Scheidgen, H., Strittmatter, P., Tack, W. (Hrsg.): *Information ist noch kein Wissen*. Weinheim (1990)
- Schmidt, J. (Hrsg.): *Denken und Denken lassen. Künstliche Intelligenz-Möglichkeiten, Folgen, Herausforderungen*. Neuwied (1992)
- Schwinn, W.: *Grundlagen der Roboterkinematik*. Schwinn (1999)
- Schwill, A.: *Cognitive aspects of object-oriented programming*. Manuskript, 1004.
- Seifert, J.: *Das Leib-Seele-Problem – und die gegenwärtige philosophische Diskussion*. Darmstadt (1989)
- Shannon, C.E., Weaver, W.: *The Mathematical Theory of Communication*. University of Illinois Press, Urbana. (Deutsche Ausgabe: *Mathematische Grundlagen der Informationstheorie*, Oldenbourg, München, 1976)
- Tanenbaum, A.S.: *Structured Computer Organization*, 4th ed. Prentice Hall, Lipper Saddle River (1999). (deutsch: A.S. Tanenbaum, J. Goodman, *Computerarchitektur*; Pearson Studiuni, München, 2001)
- Wuketits, W.: *Evolutionstheorien*. Wissenschaftliche Buchgesellschaft, Darmstadt (1988)
- Zimbardo, P. G.: *Psychologie*. Springer-Verlag (1995)

Das in diesem Kapitel erarbeitete Vorgehensmodell als eine wissensbasierte Vorgehensweise von der Problemstellung, Analyse und Design der Systemlösung über die Realisierung, der Validierung und dem produktiven Einsatz ist die Basis für innovative Roboterlösungen. Dieses Vorgehensmodell kann als Kunst verstanden werden, die Problemstellung so zu verstehen, dass adäquate Roboter(lösung)en entwickelt werden können. Kunst also nicht darum, weil hier ein artifizielles Werk entsteht, sondern weil die Praxis des Vorgehensmodells eher einem kreativen Prozess als einem linearen, sequentiellen bzw. „hart verkrustetem“ Verfahren ähnelt.

---

## 3.1 Vorgehensmodelle

Die spezifischen Tätigkeiten im Rahmen eines komplexen Entwicklungsprojektes im Bereich der Robotik werden in der Regel und trotz des kreativen Charakters in einer bestimmten zeitlichen Reihenfolge ausgeführt. Dabei empfiehlt es sich, gerade wegen der inhärenten Komplexität eines solchen Projektes, die konkrete zeitliche Anordnung nicht dem Zufall zu überlassen, sondern anhand eines Leitfadens, eines sogenannten Vorgehensmodells vorzuschlagen. Ein solches Modell legt den geordneten Ablauf des gesamten Entwicklungsprojektes, d. h. sowohl die Hardware-, Software- wie auch die Brainwareentwicklung fest. Es bestimmt also, welche Schritte in welcher Reihenfolge ablaufen, von wem sie ausgeführt werden und welche Ergebnisse bzw. Ergebnistypen (Artefakte) entstehen sollen.

Zur Durchsetzung des Leitfadens müssen die entwicklungsspezifischen Arbeiten in ein *Projekt* eingebettet und letzteres wiederum einem *Projektmanagement* untergeordnet werden.<sup>1</sup> Ein robotikspezifisches Projektmanagement steuert und überwacht den Ablauf.<sup>2</sup>

---

<sup>1</sup> Siehe auch Royce 1998.

<sup>2</sup> Siehe auch Litke 1995.

Die minimalen Teilaufgaben eines solchen Projektmanagements sind

- ein Personal-, Sachmittel- und Zeitmanagement,
- Lösungs- und Konfigurationsverwaltung betreffend den einzelnen Komponenten, deren Programmcode und Dokumentation,
- eine Qualitätssicherung bezüglich der Vorgehensweise bei der Entwicklung und des Robotiksystems selbst.<sup>3</sup>

Wichtig beim Projektmanagement ist, dass nicht nur die Entwickler, sondern auch die Auftraggeber und späteren Anwender mit einbezogen werden.<sup>4</sup> Vorgehensmodelle legen im Allgemeinen *Phasen* fest, in denen bestimmte Teilaufgaben zu erledigen sind und die bestimmte Zwischenergebnisse liefern sollen. Einzelne Modelle unterscheiden sich in der Form und der zeitlichen Anordnung der Phasen sowie in den jeweils zu erarbeitenden Resultaten bzw. Ergebnistypen. Die Eigenschaften der Vorgehensmodelle ergeben sich aus den zugrunde liegenden *Methoden*, also den Arbeitsweisen, mit denen das Gesamtziel bzw. die einzelnen Teilziele erreicht werden sollen. Man kann hier grob unterscheiden zwischen Modellen, die auf dem klassischen Ansatz der *strukturierten Entwicklung* beruhen, und Modellen, die auf dem moderneren *objektorientierten Ansatz* basieren. Da sich das in diesem Buch entwickelte und vorgeschlagene Vorgehensmodell aus Ansätzen beider Modellklassen bedient (Best practices), erscheint es sinnvoll, sich diese klassischen Vorbilder zunächst zu verdeutlichen.

*Klassische Vorgehensmodelle* haben ihren Ausgangspunkt in der strukturierten Programmierung, d. h., man bedient sich hierbei des Unterprogrammkonzepts und des Konzepts von Kontrollstrukturen. Entsprechende Modelle wurden bereits in den 70er Jahren des zwanzigsten Jahrhunderts entwickelt. Da es bei der strukturierten Programmierung im Gegensatz zur objektorientierten Programmierung keine Integration von Daten und zugehörigen Funktionen gibt, werden in den klassischen Vorgehensmodellen Daten und Abläufe getrennt voneinander modelliert. So werden beim Systementwurf die Daten beispielsweise durch Entity-Relationship-Modelle und die Systemfunktionen und Informationsflüsse durch Datenflussdiagramme dargestellt.<sup>5</sup>

Bereits bei den klassischen Modellen spielt die *Top-Down-Vorgehensweise* eine wichtige Rolle, also das schrittweise Vorgehen vom Allgemeinen/Abstrakten zum Detaillierten/Konkreten. Dabei wird zumeist erst ein grobes Systemmodell erstellt, auf dieser Basis dann ein konkreter, detaillierter Entwurf erarbeitet und eventuell in einem Prototyp frühzeitig verprobt, um dann diesen Prototyp schließlich in ein lauffähiges Programm umzusetzen. In den einzelnen Stufen werden verschiedene Beschreibungsformen benutzt, beispielsweise Entity-Relationship- und Datenflussdiagramme in den abstrakteren Modellen, sowie Relationen und Struktogramme oder Programmablaufpläne in den implementationsnä-

---

<sup>3</sup> Siehe auch Frühauf et al. 1988.

<sup>4</sup> Siehe auch 1999.

<sup>5</sup> Siehe auch Chen 1976.

heren Modellen. Problematisch sind hier jeweils die semantischen und visuellen *Brüche* zwischen den Beschreibungsformen der unterschiedlichen Phasen und auch innerhalb der einzelnen Phasen, die ja getrennte Daten- und Funktionsmodelle enthalten. Diese Brüche machen jeweils eine explizite Umformung bzw. Anpassung der Darstellungen erforderlich.

Objektorientierte Vorgehensmodelle kamen mit dem zunehmenden Erfolg *objektorientierter Programmiersprachen* auf.<sup>6</sup> Sie übertragen deren Prinzipien von der eigentlichen Programmierung auf den Entwurf von Systemen. Das Ziel dieser modernen Modelle ist die Entwicklung von Systemen, die aus einer Menge *kooperierender Hard- und Software-objekte* bestehen. Die Objekte werden dabei meist nicht zu vordefinierten Abläufen zusammengesteckt, wie es bei algorithmenorientierten Systemen mit Unterprogrammen der Fall ist. Vielmehr werden sie als Anbieter und Nutzer von Diensten verstanden, die in einer nicht vorherbestimmten zeitlichen Abfolge zusammenarbeiten. Dieses Prinzip findet sich zum Beispiel bei der *ereignisgesteuerten Programmierung* sowie beim *Client-Server-Ansatz*, der auch bei der Realisierung *verteilter Systeme* eine zentrale Rolle spielt. Der Vorteil der objektorientierten Modellierung liegt in ihrer *methodischen Geschlossenheit*, da Objekte in allen Phasen des Entwicklungsprozesses als grundlegendes Modellierungsmittel eingesetzt werden. Hierdurch werden die beiden Nachteile der klassischen Ansätze überwunden: Daten und Funktionen werden in Objekten integriert, so dass es für sie keine zwei getrennten Modelle mehr gibt. Auch zwischen den Phasen treten keine Brüche bezüglich der Darstellungsform mehr auf. Als Notationssprache für alle Phasen eines objektorientierten Entwicklungsprozesses kann man sich der *Unified Modeling Language (UML)* bedienen.

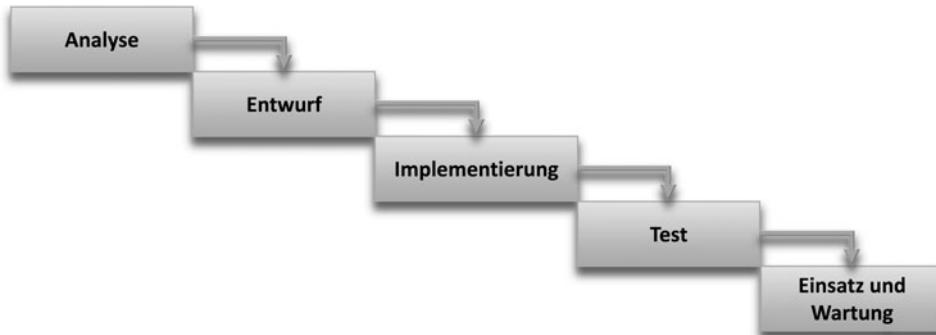
Die Entwicklung eines objektorientierten Systems erfolgt in drei Phasen, die im Anschluss an eine Initialisierungs- oder Planungsphase durchlaufen werden:

- In der *Definitionsphase* wird eine *Objektorientierte Analyse (OoA)* durchgeführt, bei der ein Fachkonzept erarbeitet wird. Das Fachkonzept konzentriert sich auf die Lösung der im *Pflichtenheft* festgelegten problembezogenen Aufgabenstellungen, abstrahiert also beispielsweise von der konkreten Realisierung einer Benutzeroberfläche. Es kann dabei das System erstens durch ein *statisches Modell* beschreiben, das die Fachkonzeptklassen mit ihren Vererbungs- und Benutzungsbeziehungen anzeigt, und zweitens durch ein *dynamisches Modell*, das die Lebenszyklen der Objekte und den zeitlichen Ablauf ihrer Kommunikation darstellt.<sup>7</sup> Je nach Anwendung haben die beiden Modelle ein unterschiedliches Gewicht. Bei einer Datenbankanwendung tritt das statische Modell stärker hervor, bei einer stark interaktiven Anwendung das dynamische Modell.<sup>8</sup>
- In der *Entwurfsphase* wird ein *Objektorientiertes Design (OoD)* erarbeitet, bei dem das OoA-Modell um *technische Aspekte* erweitert wird. Zudem werden den einzelnen Klassen technisch notwendige Komponenten, wie zum Beispiel Konstruktoren, hinzugefügt.

<sup>6</sup> Siehe auch Kilberth et al. 1993.

<sup>7</sup> Siehe auch Shiaer und Mellor 1996.

<sup>8</sup> Siehe auch Coad und Yourdon 1991.

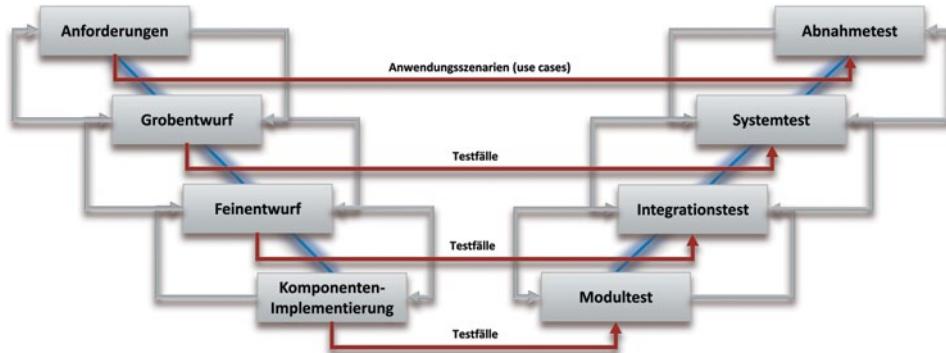


**Abb. 3.1** Klassisches Wasserfallmodell

- In der *Implementierungsphase* wird schließlich das Objektmodell in ein *Programm* einer objektorientierten Programmiersprache umgesetzt.

Konkrete Modelle verfeinern und untergliedern diese Phasen weiter und ermöglichen eventuell eine mehrfache, zyklische Ausführung der einzelnen Schritte. Im Softwareentwicklungsprozess können CASE-Tools (*Computer Aided Software Engineering*) eingesetzt werden, die die Entwickler bei den einzelnen Schritten unterstützen oder sogar einzelne Schritte teilautomatisieren.

Im Laufe der vergangenen zwei Jahrzehnte wurde eine Reihe von Modellen entworfen, die die beschriebenen Arbeitsschritte und die damit verbundenen Vorgehensweisen in unterschiedlicher Weise anordnen. Im einfachsten Fall werden die Schritte, die zur Erstellung einer Lösung durchzuführen sind, jeweils nur einmal und dabei einer nach dem anderen bearbeitet. Aus diesem Ansatz ergibt sich ein Vorgehensmodell mit einer streng sequentiellen Phasenstruktur, bei dem das Ergebnis einer Phase vollständig erarbeitet sein muss, bevor die nächste Phase damit fortfährt. Das Ende einer Phase wird durch einen so genannten Meilenstein markiert, bei dem ein dokumentiertes Resultat vorgelegt werden muss, beispielsweise eine Spezifikation der Systemarchitektur oder lauffähiger Programmcode. Das Ziel ist also, ein vollständiges Produkt in einem Durchgang durch die Arbeitsschritte zu entwickeln. Die Aufgabe des Managements besteht dabei darin, das Einhalten der Zeitvorgaben zu überwachen, d. h. sicherzustellen, dass die einzelnen Meilensteine im gegebenen Zeitrahmen erreicht werden und die Projektbeteiligten entsprechend zu koordinieren. Allerdings sind solche sequentiellen Modelle gerade im Bereich der Robotik in ihrer reinen Form kaum praktisch einsetzbar, da in späteren Phasen oft Erkenntnisse gewonnen werden, die die Überarbeitung früherer Phasen erforderlich machen. In der Praxis sind daher Rückkopplungen vorgesehen, durch die man zu einer früheren Phase zurückkehren und somit Entwicklungsschritte überarbeiten kann. Ein typischer Vertreter dieser Modellklasse ist das Wasserfallmodell, bei dem die Phasen der bildlichen Vorstellung nach untereinander angeordnet sind, so dass die Ergebnisse einer Phase in die nächste Phase „hinunterfallen“ (Abb. 3.1).



**Abb. 3.2** Klassisches V-Modell

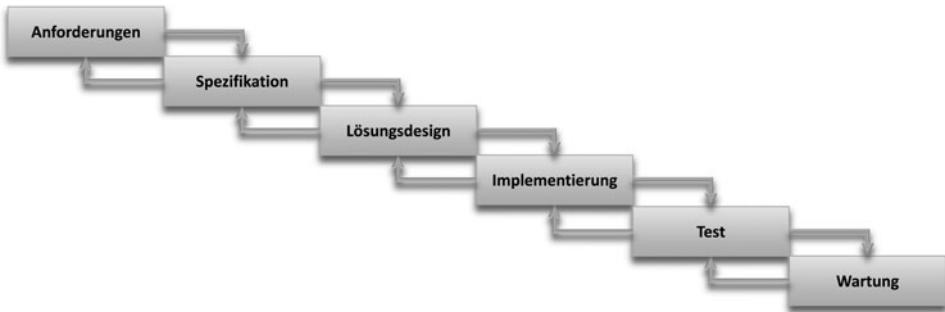
Eine Weiterentwicklung des Wasserfallmodells ist das sogenannte *V-Modell*, bei dem die Phasen in Form des Buchstabens „V“ angeordnet sind. Der linke Zweig enthält Verfeinerungsphasen vom Gobentwurf der Systemarchitektur bis zum Feinentwurf auf Modul-ebene. Dieser Zweig wird top-down von oben nach unten durchlaufen. Am Fuß und im rechten Zweig des V befinden sich Implementations- und Integrationsphasen, in denen die einzelnen Module implementiert und schrittweise zum lauffähigen Gesamtsystem integriert werden. Sie werden bottom-up von unten nach oben bearbeitet.

Zwischen den Phasen des erweiterten Wasserfallmodells und auch beim V-Modell sind *Rückkopplungen* vorgesehen, mit denen man zu früheren Phasen zurückkehren kann. Jedoch gibt es auch mit diesen Rückkopplungen Probleme beim praktischen Einsatz der Modelle. Sequentielle Modelle setzen nämlich voraus, dass die Anforderungen an das System möglichst vollständig und exakt bekannt sind, bevor der Systementwurf und die nachfolgenden Schritte in Angriff genommen werden. Das ist aber in der Praxis meist nicht der Fall, denn Anforderungen können oft erst im Laufe des Entwicklungsprozesses hinreichend genau formuliert werden. Solche sich ändernden Anforderungen kann das Modell nur schlecht berücksichtigen, zumal nach der Analysephase eine Beteiligung des Anwenders nicht mehr vorgesehen ist (Abb. 3.2).

Um den Problemen der strengen Sequentialität zu begegnen, entwarf man sogenannte *zyklische Modelle*, die die Möglichkeit bieten, Phasen mehrfach zu durchlaufen. Diese Modelle berücksichtigen explizit den Erkenntnis- und Lernprozess, zu dem es während der Entwicklung erfahrungsgemäß kommt: Neue Erkenntnisse können leicht in einem weiteren Phasendurchlauf in das System eingebbracht werden (Abb. 3.3).

Ein klassisches Beispiel für diesen Ansatz ist das *Spiralmodell*, das wie beim Durchlauf einer Spirale mit mehreren Windungen vorgeht. In jedem Zyklus, also in jeder Windung der Spirale, werden vier Schritte ausgeführt und dabei Teilprodukte hergestellt:

- Es werden Ziele, Alternativen und Randbedingungen der Entwicklung in diesem Zyklus formuliert.
- Dann werden die Alternativen einer Risikoanalyse unterzogen, also anhand der Ziele und Randbedingungen bewertet.



**Abb. 3.3** Zyklisches Modell

- Anschließend wird ein Teilaspekt des Systems realisiert. In einem frühen Zyklus wird beispielsweise eine Systemspezifikation formuliert, in einem späteren Zyklus die Architektur entworfen und zum Abschluss das System implementiert und getestet.
- Zuletzt kommt es zur Planung des nächsten Zyklus.

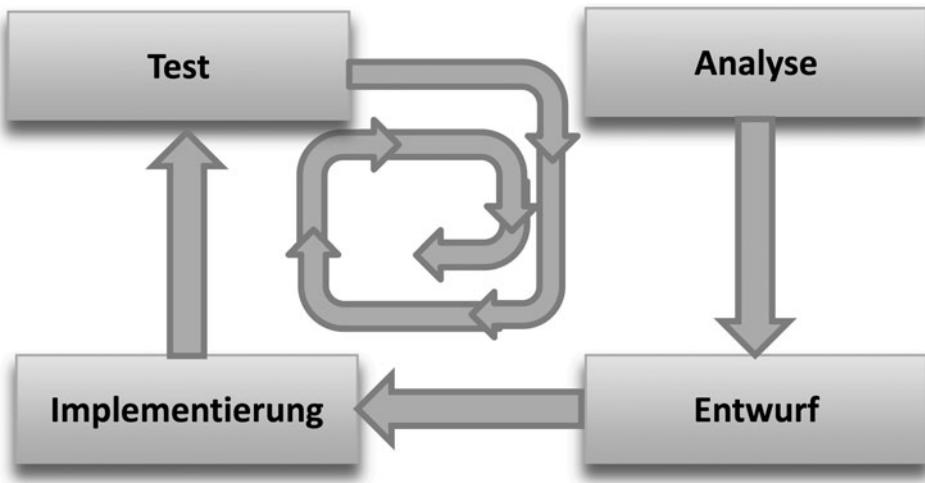
Im Verlauf der Zyklen wird oftmals ein sogenannter *Prototyp* erstellt. Ein Prototyp ist eine Lösung, die ausgewählte Eigenschaften des zu entwickelnden Systems realisiert und dabei insbesondere einen Eindruck von der angestrebten terminalen Lösung vermittelt. Auf dieser Basis können Entwickler und spätere Anwender sinnvoll miteinander kommunizieren und sich somit über die weitere Vorgehensweise abstimmen. Andere nichtsequentielle Modelle betonen den Aspekt der iterativen Softwareentwicklung. Hier wird während des Entwicklungsprozesses eine Folge von Lösungsversionen erarbeitet, wobei das Funktionsangebot fortlaufend erweitert und angepasst wird: Zunächst wird nur die basale Funktionalität des Systems realisiert. Anschließend werden weitere Funktionen sukzessive hinzugefügt. Dabei werden ständig neue Anforderungen und Änderungen der Randbedingungen berücksichtigt (Abb. 3.4).

Die bislang betrachteten Modelle beruhen auf dem Ansatz der strukturierten Programmierung. In jüngerer Zeit sind dagegen *objektorientierte* Vorgehensweisen stark in den Vordergrund getreten.<sup>9</sup> Mittlerweile zeichnet sich hier eine *Vereinheitlichung* der Vorgehensweisen ab.<sup>10</sup> Eines der konkreten Ergebnisse dieser Standardisierungsbemühungen ist die *Unified Modeling Language (UML)* zur Spezifikation und grafischen Darstellung objektorientierter Systeme. UML wurde 1997 standardisiert und hat sich seitdem als Modellierungssprache in vielen Bereichen durchgesetzt. Aufbauend auf UML schlugen Booch, Jacobson und Rumbaugh 1998 ein objektorientiertes Vorgehensmodell vor: den sogenannten *Unified Process*.<sup>11</sup> Dieser Prozess zeichnet sich durch die folgenden grundlegenden Eigenschaften aus:

<sup>9</sup> Vgl. Balzert 1999a, b.

<sup>10</sup> Vgl. Balzert 2001.

<sup>11</sup> Vgl. Balzert 2001



**Abb. 3.4** Spiralmmodell

- Der Unified Process ist UML-basiert und damit objektorientiert: Die entstehende Softwarearchitektur setzt sich aus Objekten zusammen, die jeweils Module mit einer Funktionalität und einem Dateninhalt realisieren. In allen Phasen des Entwicklungsprozesses werden Objekte und Klassen mit ihrer Strukturierung und Zusammenarbeit durch UML beschrieben, und zwar aus verschiedenen Blickwinkeln heraus.<sup>12</sup>
- Der Unified Process ist Anwendungsfall-orientiert, also auf die Analyse und praktische Umsetzung so genannter Anwendungsfälle (use case) ausgerichtet.<sup>13</sup> Ein Anwendungsfall bezieht sich auf das Zusammenwirken des Systems mit Personen oder anderen Systemen, um eine bestimmte Aufgabe zu erledigen. Er gibt damit an, was das System für einen bestimmten Anwender zu einem bestimmten Problem leisten soll. Der Schwerpunkt liegt also nicht so sehr auf dem technisch Machbaren, sondern eher auf den Bedürfnissen der einzelnen Anwender. Alle Anwendungsfälle zusammen beschreiben die gewünschte Gesamtfunktionalität des Systems. Insofern kann man davon sprechen, dass das System entwickelt wird, um die Gesamtheit der Anwendungsfälle zu realisieren.
- Der Unified Process arbeitet iterativ und inkrementell: Das Softwaresystem wird nicht „in einem Wurf“ erarbeitet, sondern in einer Folge von iterativen Schritten (Iterationen), die dem System jeweils inkrementelle Erweiterungen (Inkremeante) hinzufügen. Das Ergebnis einer Iteration ist jeweils ein Teilprodukt, also ein lauffähiges und geprüftes System, das einen Teil der Anwendungsfälle bis zu einem gewissen Grad realisiert. Aus diesem Prozess ergibt sich schließlich ein System mit der gewünschten vollständigen Funktionalität.

<sup>12</sup> Vgl. Kruchten 1998.

<sup>13</sup> Vgl. Jacobson et al. 1992.

Dabei wird in jeder Iteration ein Iterationsprojekt ausgeführt, das sich auf das Arbeitsergebnis der vorherigen Iteration stützt und um ein Inkrement erweitert. Das Ergebnis ist jeweils ein lauffähiges System, das einen Teil der Gesamtanforderungen abdeckt. Ein Iterationsprojekt, also eine Iteration, besteht im Allgemeinen aus fünf Arbeitsschritten:

- *Anforderungen* (Requirements): Es werden Anforderungen gesammelt, also festgestellt, was in dieser Iteration getan werden und was das resultierende System leisten soll. Das Ziel dieses Schrittes ist eine Übereinkunft zwischen Entwicklern und Kunden (Problemsteller), die die Arbeit „in die richtige Richtung“ lenken soll.
- *Analyse* (Analysis): Die aufgestellten Anforderungen werden analysiert und strukturiert. Das Ziel dieses Schrittes ist, die Anforderungen besser zu verstehen und sie in eine Form zu bringen, auf deren Grundlage die Systemstruktur erarbeitet werden kann.
- *Entwurf* (Design): Auf der Grundlage der Anforderungen wird die Systemstruktur mit ihren Komponenten modelliert, wobei Randbedingungen beachtet werden. Das Ziel dieses Schrittes ist die Spezifikation der Systemkomponenten, die realisiert werden sollen.
- *Implementierung* (Implementation): Die Spezifikation der Systemkomponenten wird in eine programmiersprachliche Form umgesetzt. Das Ziel dieses Schrittes ist lauffähiger Programmcode.
- *Test* (Testing): Die Systemkomponenten werden überprüft. Das Ziel dieses Schrittes ist die hinreichende Sicherheit, dass der bislang erstellte Programmcode korrekt ist.

Je nach Fortschritt des Entwicklungsprozesses bzw. der der Problemstellung zugrunde liegenden Komplexität fällt in den einzelnen Schritten einer Iteration unterschiedlich viel Arbeit an. So kann beispielsweise in den ersten Iterationen der Schwerpunkt durchaus auf der Formulierung der Anforderungen an das System liegen, wobei nur wenig Programmcode entwickelt wird. Zu einem späteren Zeitpunkt verschieben sich die Gewichte in Richtung Design und Implementierung und schließlich auch zum Test. Diesen Gewichten entsprechend werden die einzelnen Iterationen zu vier großen Phasen zusammengefasst:

- *Beginn* (Inception): Der Schwerpunkt liegt auf dem Sammeln und der Strukturierung der Anforderungen an das System. Zweck und Rahmen des Projekts werden festgelegt, ein grober Kosten- und Zeitplan wird erstellt und möglicherweise ein erster Prototyp implementiert.
- *Ausarbeitung* (Elaboration): Der Schwerpunkt liegt auf der Definition der Architektur. Eine genauere Analyse der Aufgabenstellung liefert eine Anforderungsbeschreibung, auf deren Basis die architektonischen Grundlagen des Systems entworfen werden und ein lauffähiges Demonstrantum für die wichtigsten Anwendungsfälle programmiert wird.
- *Konstruktion* (Construction): Der Schwerpunkt liegt auf der Realisierung eines ausführbaren Softwaresystems. Der Systementwurf wird iterativ verfeinert und die Implementation wird entsprechend erweitert und fortlaufend getestet.

- *Umsetzung* (Transition): Der Schwerpunkt liegt auf der Einführung des Systems bei der Nutzergemeinschaft. Die Hinführung beginnt mit einer Beta-Version, d. h. mit einem Softwaresystem in noch nicht endgültiger Form und endet mit einer stabilen Produktversion.

Die vier Phasen bilden zusammen einen so genannten Zyklus. Über einen längeren Zeitraum hinweg können mehrere Zyklen hintereinander ausgeführt werden, die jeweils eine neue Produktversion liefern.

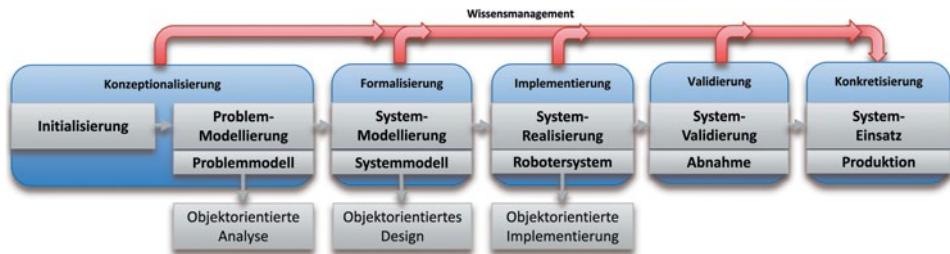
Der Unified Process kommt also sehr stark der Forderung nach einer iterativen, inkrementellen Vorgehensweise nach, bei der schrittweise neue Systemkomponenten hinzugefügt werden und auch neue Wünsche und Randbedingungen berücksichtigt werden können. Der Preis für diese Flexibilität ist ein recht komplexes Vorgehensmodell, das erhöhte Anforderungen an das Prozessmanagement stellt.<sup>14</sup>

Ein weiteres modernes Vorgehensmodell ist das *Extreme Programming*. Auch hier werden eine Reihe von Iterationen durchlaufen, die eine Folge von Lösungs-„Releases“ mit immer umfangreicherer Funktionalität liefern. Diese Lösungen können unmittelbar in der realen Anwendungsumgebung eingesetzt werden, wodurch eine starke Beteiligung der Anwender am Entwicklungsprozess möglich wird. Interessant ist hier insbesondere der Ansatz des Pair Programmings: An der Entwicklung eines Softwaremoduls sind stets zwei Entwickler beteiligt, die gemeinsam an einem Bildschirm arbeiten. Hiervon verspricht man sich, trotz des höheren Personalaufwands, eine Steigerung der Produktivität und eine Reduktion der Anzahl der Programmierfehler. Ein zweites charakteristisches Merkmal ist die testgetriebene Entwicklung. Die Tests, die mit einem Modul durchzuführen sind, werden festgelegt, bevor die eigentliche Entwicklung des Moduls beginnt.

Aus all den bisher dargestellten Modellen haben sich Best Practices herauskristallisiert, die nunmehr nur noch auf die spezifischen Aspekte der Robotersystementwicklung angepasst werden müssen.<sup>15</sup> Als erstes lohnt es sich, dem ganzen Vorhaben, Roboterlösungen zu entwickeln, gewisse Strukturen einzubringen. Ein erster Strukturansatz bietet sich an, indem man die Entwicklung einer Roboterlösung als *Projekt* ansieht. Als solches durchläuft es gewisse Lebenszyklen (Lifecycle). Durch ein Lebenszyklus-Konzept, bestehend aus den Teilzyklen Initialisierung, Problemmodellierung, Systemmodellierung, Systemrealisierung und Systemvalidierung, wird die Umsetzung der Anforderungen in die Realisierungsebene der Systemwelt verfolgt. Dabei wird ein Modellbegriff verwendet. Unter einem solchen *Modell* lässt sich ein Objekt auffassen, das durch eine Struktur-, Funktions- oder Verhaltensanalogie zu einem Original erlaubt, dieses in schematischer, auf wesentliche Züge komprimierter Darstellung wiederzugeben. Zentral für den in dieser Weise verwendeten Begriff sind die zuschreibbaren Relationen der Isomorphie (Gleichgestaltigkeit) und der Analogie. Zusätzlich haben diese Modelle den Anspruch, die Probleme als auch die

<sup>14</sup> Siehe auch Zimmermann 1999.

<sup>15</sup> Siehe auch Raasch 1993.



**Abb. 3.5** Wissensbasiertes Modell

Lösungen (optimal) so abzubilden, dass sie im Rahmen der Realisierung zu produktiven Systemen entwickelt werden können (Abb. 3.5).

Die einzelnen Phasen liefern Ergebnistypen (Output), die als basale Grundlage (Input) die darauf folgenden Phasen initialisieren und damit beeinflussen. Um gewährleisten zu können, dass die Lösung den problemorientierten Anforderungen genügt, wird das Problemmodell als Ausgangspunkt für die Systemmodellierung genommen. Um sicher zu stellen zu können, dass das im Rahmen der einzelnen Phasen erarbeitete Wissen jederzeit zur Verfügung steht, werden Methoden und Techniken des Wissensmanagements flankierend eingesetzt. In den verschiedenen Phasen werden die Ergebnistypen mit verschiedenen Methoden und Techniken erstellt.

So werden beispielsweise die modellierten Probleme im Rahmen der Konzeptionalisierung und Formalisierung mit Hilfe der Modellierungsnotation UML in ein Systemmodell überführt (transformiert). An diesen Stellen erfolgt also ein Perspektivenzuwachs, indem das eher statische Problemmodell sukzessive im Rahmen der Konzeptionalisierung und Formalisierung beispielsweise mit objektorientierten Methoden und Techniken erweitert wird.<sup>16</sup>

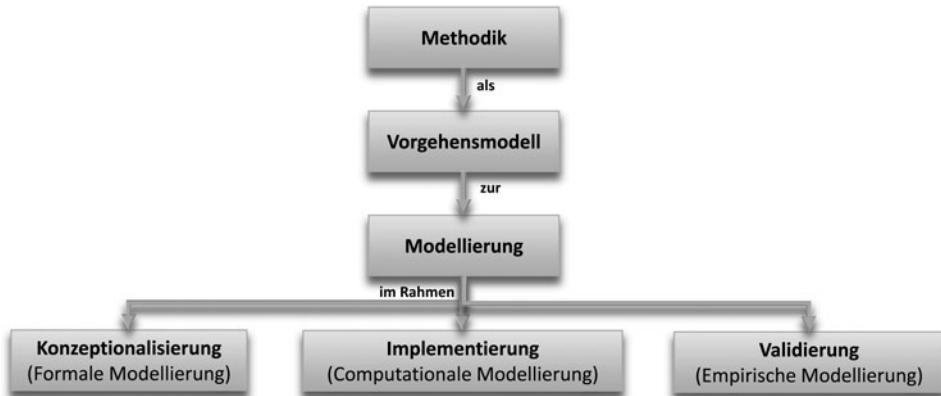
## 3.2 Methodik als Randbedingungen

Die nächsten Abschnitte behandeln das Problem der prozessualen und funktionalen Ausgestaltung von Robotersystemen im Allgemeinen und kognitiven Robotersystemen im Speziellen. Im Sinne der Wissenschaftstheorie<sup>17</sup> ist eine solche Ausgestaltung in Form eines Entwicklungsprojektes als ein Forschungsprogramm oder eben wissenschaftstheoretisch ausgedrückt, als die Realisierung eines Paradigma anzusehen.<sup>18</sup>

<sup>16</sup> Vgl. Zimmermann 1999.

<sup>17</sup> Vgl. Chalmers 1986.

<sup>18</sup> Vgl. Bayertz 1980.



**Abb. 3.6** Methodik als Rahmenbedingung

Nach T.S. Kuhn oder I. Lakatos<sup>19</sup> operiert die Wissenschaft in einem anerkannten Paradigma, das eine Grundorientierung der Forschung vorgibt. Es umfasst die disziplinären Grundlagen (Theorien, Methoden, Gegenstandsbestimmungen, Definitionen, Instrumentarien, Techniken etc.), deren Infragestellung dann allerdings bis zu einem Paradigmenwechsel in Form einer Revolution führen kann.<sup>20</sup> Das Paradigma erweist sich demnach als Rahmen für die Forschung, das die kreativen und gerichteten Prozesse kanalisiert und bei Irritationen wieder ausrichtet, d. h. in die alten Bahnen zurückzubringen versucht. Eine Wissenschaft befindet sich also solange innerhalb eines Paradigmas, als die gemeinsamen Problemlösungen innerhalb dieser Wissenschaftsgemeinschaft nicht in Frage gestellt werden. Allerdings blieb eine solche wissenschaftliche Revolution im Sinne von Th. S. Kuhns im Problembereich dieses Buches bisher aus, sodass der stetige Wissenszuwachs ungeachtet aller Wechsel sogenannter Paradigmen hinweg eher einer wissenschaftlichen Evolution gleicht. So entwickeln sich die wissenschaftlichen Zwecke und Mittel nicht nur kontinuierlich fort, sondern, unter der Berücksichtigung, dass sich diese Zwecke und Mittel auch im „Wettbewerb der Möglichkeiten“ bewähren und damit eine gewisse „Fitness“ erringen müssen, auch weiter.<sup>21</sup>

Unabhängig von dieser wissenschaftstheoretischen Sichtweise ist ein solches Entwicklungsprojekt ein zunächst interdisziplinäres, dann transdisziplinäres Unternehmen, dessen charakteristischste Methode die Modellierung ist, nämlich die formale und auf Robotersysteme implementierte Modellierung von Lösungen in Verbindung mit der Überprüfung dieser Modelle durch empirische Analyse in einer Test- oder Simulationsumgebung (Abb. 3.6).

<sup>19</sup> Vgl. Lakatos und Musgrave 1974 und Lakatos 1982.

<sup>20</sup> Vgl. Kuhn 1977.

<sup>21</sup> Vgl Kuhn 1976.



**Abb. 3.7** Top-Down und Bottom-Up Ansatz

Charakteristisch für den Einsatz von Modellen ist ihre Verwendung sowohl im Rahmen eines Bottom-Up- als auch eines Top-Down-Ansatzes. Im Verlaufe eines *Top-Down*-Ansatzes geht man vom Abstrakten, Allgemeinen, Übergeordneten schrittweise auf das Konkrete, Spezielle, Untergeordnete zu. Hingegen verfolgt der *Bottom-Up*-Ansatz den umgekehrten Weg, indem, ausgehend vom Konkreten, Speziellen und Untergeordneten, dem Allgemeinen, Abstrakten und Übergeordneten zugestrebtt wird. Aus Sicht der Entwicklung werden im Falle des „Top-Down“-Ansatzes Lösungsmodelle erstellt, die aus einer bereits bestehenden Theorie hervorgehen, also eine andere Formulierung der Theorie darstellen. Im Gegensatz dazu werden beim Bottom-Up-Ansatz Lösungen entwickelt, um mit ihnen Ausgabedaten zu produzieren, die dem Verhalten des modellierten und simulierten Realitätsausschnitts entsprechen (Abb. 3.7).

Unabhängig von diesem Ansatz wird der Weg vom Problem zur Lösung von bestimmten Einflussgrößen bestimmt. So erweitern Inspirationen vorhandene einfache Modelle durch detailliertere Inspirationen. Analysen führen zu einem theoretischen Modell, das zur Vereinheitlichung vorhandener Modelle oder zu Erweiterungen führen kann. Zu guter Letzt ergeben sich problemspezifische Anforderungen, die eine Anpassung und Erweiterung des vorhandenen Modells erfordern. Die eigentliche Stärke des nachfolgenden Vorgehensmodells liegt neben dessen prinzipieller Einfachheit und Kombinierbarkeit mit verschiedenen Techniken und Technologien in den Möglichkeiten, Modelle verhältnismäßig einfach und realitätskonform zu konstruieren. Gerade mit dem Bottom-Up-Ansatz lässt sich beispielsweise im Rahmen von Simulationen ein globales Systemverhalten erzeugen, das dem Verhalten der Echtzeitsysteme sehr ähnlich ist. Simulationen werden also genutzt, um Ergebnisse zu produzieren, die jenen ähneln, die vom Simulierten produziert werden. Insofern bietet sich der Bottom-Up-Ansatz besonders für die Bearbeitung von Problemen an, in denen es um konkrete Interoperationen zwischen Elementen geht, die durch lokale Regeln determiniert werden.

Insbesondere in den Sozial- und Kognitionswissenschaften bzw. in den Forschungsgebieten rund um die Künstliche Intelligenz oder Künstliches Leben ist es oftmals gar nicht anders möglich, alle sozialen und kognitiven Prozesse in solchen Bottom-Up-Modellen darzustellen,

falls man die Komplexität dieser Prozesse angemessen, präzise und vor allem invasiv analysieren will.<sup>22</sup>

In diesem methodischen Sinne bildet der Bottom-Ansatz bestimmte Problem- bzw. Lösungs- und damit Realitätsbereiche ab, die formal als komplexe dynamische Systeme mit lokalen Wechselwirkungen definiert werden können.

Ein weitere Randbedingung kommt hinzu, indem vor allem im Rahmen der Implementierung die Realisierung von interoperierenden Agentensystemen vorgesehen ist, die sich in ihrer Problemdomäne zu einem gewissen Grade „intelligent“ verhalten soll. Die Entwicklungsarbeiten konzentrieren sich aber auch auf das Verstehen der Wissensverarbeitungsprozesse und -techniken, die für die Koordination solcher Systeme notwendig sind, auf die Realisierung solcher Agentensysteme und auf die Evaluation dieser Systeme in produktiven Plattformen (Umgebungen). Im Rahmen der Implementierung sind hierzu zwei Vorgehensweisen vorgesehen. Beim problemorientierten Ansatz wird untersucht, wie eine Aufgabe aus der Problemdomäne am besten auf mehrere Agenten verteilt werden kann. Beim agentenorientierten Ansatz wird davon ausgegangen, dass die existierenden, mit Basisfunktionen ausgestatteten Agenten mit einer Brainware zu „intelligenzieren“ sind. Diese „Intelligenzierung“ basiert auf einem wissensbasierten Ansatz, indem jeder einzelne Agent als ein wissensbasiertes System agiert. Dazu sind die folgenden systemtechnischen Kriterien im Rahmen der Implementierung zu berücksichtigen, damit das wissensbasierte System den gewünschten Leistungsumfang mit akzeptablen Interoperationszeiten im Rahmen der späteren Validierung gewährleisten kann.

- *Adäquatheit:* Für bestimmte Aufgabenstellungen oder -gebiete sind oft bestimmte Darstellungs- und Verarbeitungsmethoden besser geeignet als andere. So dürfte es zum Beispiel oft sinnvoll sein, die Modifizierung physikalischer Systeme mittels Differentialgleichungen durchzuführen, während die Unterstützung der Diagnose im medizinischen Bereich vielleicht besser durch andere Techniken erreicht wird. Allerdings kann es durchaus möglich sein, dass beispielsweise aus Effizienzgründen interne Darstellungs- und Verarbeitungstechniken verwendet werden müssen, die auf einer abstrakten Ebene der Modellbildung nicht unbedingt adäquat sind.
- *Verständlichkeit und Mitteilbarkeit:* Bei der Verwendung wissensbasierter Systeme durch menschliche Benutzer ist es notwendig, Informationen über das gespeicherte Wissen sowie Operationen auf diesem Wissen in einer für den Anwender verständlichen Form darzustellen. Dies muss nicht unbedingt bedeuten, dass die internen Repräsentations- und Verarbeitungsmechanismen direkt für den Anwender verständlich sind, sondern kann auch dadurch erreicht werden, dass bei Bedarf spezielle Ausgaben, etwa durch Monitoring-, Inspektions- oder Erklärungskomponenten, visualisiert werden.
- *Einheitlichkeit und Kombinationsfähigkeit:* Ein wichtiges Problem beim Umgang mit Wissen besteht oft darin, Informationen zu einem bestimmten Objekt oder Aspekt aus

<sup>22</sup> Siehe auch Zimmerli und Wolf 1994.

verschiedenen Quellen oder Darstellungsformen zu kombinieren. Dabei ist es nicht unbedingt wesentlich, dass für die verschiedenen Formen dieselben grundlegenden Darstellungs- und Verarbeitungstechniken verwendet werden. Wichtig ist eher, dass auf einer bestimmten Abstraktionsebene die Möglichkeit besteht, mehrere Informationsarten gegenseitig verfügbar zu machen und wichtige Aspekte bei der Verarbeitung gegenseitig zu berücksichtigen.

- *Effizienz:* Die praktische Verwendbarkeit eines wissensbasierten Systems hängt nicht nur von seinem Leistungsumfang, also den verfügbaren Funktionen, sondern auch von der Effizienz ab, mit der die Gesamtleistung oder einzelne Funktionen zur Verfügung gestellt werden. Dabei sind zum einen Einschränkungen durch die Aufgabenstellung möglich, etwa bestimmte Reaktionszeiten in bestimmten Situationen. Zum andern wird die Akzeptanz von Seiten der Anwender auch von der Geschwindigkeit abhängen, mit der auf Anfragen reagiert wird.
- *Multidimensionale Darstellung:* Traditionelle Datenstrukturen, wie Felder, Records, Listen und Bäume sind nicht ausreichend zur Darstellung von Wissen auf abstraktem Niveau, da vorwiegend eine Dimension zur Strukturierung verwendet wird. Ein wichtiger Aspekt von Wissen ist, dass es Zusammenhänge unterschiedlicher Art geben kann. Eine sinnvolle Darstellung dieser Zusammenhänge stellt eine grundlegende Problematik in dem Gebiet des Cognitive Computing dar und es wurden eine Reihe von Ansätzen entwickelt, die eine Balance aus drei Aspekten anstreben: einer für den Anwender verständlichen Darstellung, der Verfügbarkeit formaler Grundlagen sowie einer effizienten Implementierung.
- *Kontextabhängige Darstellung und Verarbeitung:* Bei symbolorientierten Methoden wird oft die Verarbeitung *unabhängig* vom Kontext betont. Dadurch wird es möglich, Symbolfolgen zu isolieren und unabhängig vom Kontext mit anderen zu vergleichen, oder zu kombinieren. Unter dieser Voraussetzung lassen sich bei den kontextfreien Sprachen relativ einfache Grammatiken und zugehörige effiziente Verarbeitungsmethoden angeben. Was man hier an einfacher Struktur und effizienter Verarbeitung gewinnt, geht allerdings zu einem gewissen Teil auf Kosten der Ausdrucksmächtigkeit.
- *Kompositionalität:* Beim Entwurf komplexer Systeme ist es oft von Vorteil, einen Baukasten mit wichtigen Grundstrukturen und darauf basierenden Operationen zur Verfügung zu stellen. Daraus können dann komplizierte Komponenten zusammengesetzt werden. Wichtig ist hierbei, dass sowohl die Grundstrukturen als auch die Grundoperationen untereinander kompatibel sind, was natürlich andererseits im Vergleich zu spezialisierten Entwürfen Einbußen bei der Effizienz zur Folge haben kann.

Neben diesen zentralen Aspekten müssen im Rahmen der Implementierung auch wichtige Fragen bezüglich des Wissens und der Adaptivität geklärt werden: Wissensakquisition: Wie kommt das Wissen in das Robotersystem? Lassen sich sensorische oder Prozessdaten direkt verwerten? Kann Expertenwissen leicht integriert werden? Unzureichendes Wissen: Ist das Robotersystem in der Lage, mit unvollständigem, ungenauem, inkonsistentem Wis-

sen umzugehen? Meta-Wissen: Was weiß das Robotersystem über seine eigenen Fähigkeiten? Kann es seine Vorgehensweisen beobachten und verbessern? Adaptivität: Kann sich das Robotersystem an veränderte Umgebungsbedingungen selbstständig anpassen?

In diesem Zusammenhang kommt als weitere Randbedingung dazu, dass die kognitive Modellierung als ein wesentlicher Bestandteil bei der prozessualen und inhaltlichen Ausgestaltung des Vorgehensmodells zu berücksichtigen ist. Ziel der kognitiven Modellierung ist es, eine Simulation kognitiver Prozesse (artifizielle Kognition) natürlicher Systeme so zu ermöglichen, dass damit nicht nur eine dem natürlichen Vorbild vergleichbare Leistung erzielt werden kann, sondern darüber hinaus Erkenntnisse über derzeit noch nicht nachvollziehbare kognitive Phänomene möglich werden.

In Simulationsexperimenten wird also mit Hilfe des Computers auf Basis einer Repräsentation des Phänomens und einer Theorie, die die Dynamik des Phänomens adäquat beschreibt, operiert. Insofern gelten als Voraussetzungen für erfolgreiche Simulationsmodelle und Simulationsergebnisse nicht nur Theorien, die die Phänomene und deren Dynamik adäquat beschreiben, sondern auch deren Operationalisierbarkeit, um sie in der Form eines softwaretechnischen Systems repräsentieren zu können.

Diese kognitive Modellierung ist dadurch charakterisiert, dass sie die Entwicklung eines Modells in Form eines formalen Systems zum Ziel hat, das als lauffähiges Hard-, Software- und Robotersystem implementiert werden kann.

Wissenschaftstheoretisch betrachtet stellt ein solches Modell mit seinen Entitäten und Relationen die ontologische Abbildung eines Ausschnitts der Welt auf ein formales System dar, so dass die als modellrelevant angesehenen Entitäten und Relationen isomorph zu dem entstehenden Softwaresystem sind und sich gleichermaßen im Modell wie im Softwaresystem wiederfinden.

Die Konstruktion des kognitiven Modells bedient sich überwiegend der in Mathematik, Logik und Informatik entwickelten Methoden, insbesondere in Gestalt von Techniken der Wissensrepräsentation, wie sie in der Künstlichen Intelligenz und des Künstlichen Lebens entwickelt worden sind. Die empirische Überprüfung dieses Modells hingegen wird mit naturwissenschaftlichen Methoden durchgeführt (Lösungsentwicklung, Experiment, Simulation etc.). In diesem Sinne entspricht diese Methodik dem wissenschaftlichen Anspruch, indem aus wissenschaftlichen Theorien empirische Vorhersagen abgeleitet, im Experiment überprüft, also ggf. widerlegt (falsifiziert), als auch aus dem Modell empirisch prüfbare Verhaltensvorhersagen abgeleitet werden.

Zu guter Letzt muss das Vorgehensmodell dem interdisziplinären Charakter der Entwicklungsprojekte Rechnung tragen. Die Interdisziplinarität entsteht zum einen dadurch, dass physikalische, biochemische, biologische, biophysikalische, informationstheoretische und philosophische Ansätze verfolgt werden. Zum anderen beeinflussen im Umkehrschluss die Erkenntnisse der Kognitiven Robotik die Fächer Informatik, Psychologie,

Biologie (Neurobiologie), Medizin (Neurologie), Mathematik, Physik (Biophysik) und Philosophie (Leib-Seele-Problem, Theorie des Geistes etc.).<sup>23</sup>

Die Entwicklung von Robotersystemen unter Beachtung dieser Randbedingungen erfordert demnach eine entsprechende Methodik. Dabei müssen Konzepte und Verfahrensweisen aus zahlreichen Disziplinen nicht nur in sich erweitert, sondern auch untereinander abgestimmt und zu einem übergeordneten Metamodell integriert werden. So werden beispielsweise quantitative Beschreibungsmethoden für „intelligente“ Systeme, deren Aufgaben und Umgebungen benötigt. Unabhängige Replikation und Verifikation von Experimenten muss zum Standardverfahren in der wissenschaftlichen Gemeinschaft werden.

Dieser Forderung wird im Rahmen dieses Buches beispielsweise dahingehend Rechnung getragen, indem in einem späteren Kapitel ein Messinstrument entwickelt wird, das in Form eines systemischen Intelligenzprofils die Intelligenz des Robotersystems angibt.

Die Methodik sieht dabei die folgenden Iterationsphasen im Kern vor:

- *Konzeptionalisierung* des Lösungsmodells,
- *Implementierung* (und damit Formalisierung) des Lösungsmodells in Algorithmen in Form von Lösungskomponenten und deren Orchestrierung zu Systemanwendungen und
- die *Validierung* der Implementierung durch Simulation.

Das Ziel der *Konzeptionalisierung* ist die Identifikation der der Problem- beziehungsweise Anwendungsdomäne zugrunde liegenden Grundlagenstrukturen. Dabei bedient man sich der Erkenntnissen der entsprechenden Wissenschaftsdisziplinen. Als Ergebnis liefert die Konzeptionalisierung ein auf das Problem bezogenes, vorläufiges Lösungsmodell, unter Nennung aller dafür erforderlichen Entitäten. Diese Modelle bilden die Vorlage für die *Formalisierung*, die die entsprechenden Algorithmen liefert. Durch die *Implementierung* der Algorithmen wird das Lösungsmodell in ein ausführbares Modell überführt. Mit Hilfe dieser Implementierung können auch letzte Unvollständigkeiten identifiziert und behoben werden. Das bisher konzeptionalisierte Wissen über die Anwendungsdomäne wird in expliziter Form durch Realisierung in Hard- und Software- und Robotersysteme transformiert. Die Güte der Konzeptionalisierung entscheidet über die Güte der Formalisierung und diese wiederum der Implementierung! Neben der Konzeptionalisierung, Formalisierung und Implementierung besteht ein weiterer wichtiger Schritt bei der Konstruktion von Systemen in der *Validierung* der Systemlösungen und damit auch in der Bewertung der empirischen Angemessenheit der Modelle. Weiterhin ist für die Entwicklungsprojekte prägnant, dass sich sowohl für die Konzeptionalisierung als auch für die Implementierung keine festen Regeln bezüglich der Erarbeitung optimaler Lösungen angeben lassen. In fast allen Fällen können die Realisierungsschritte zur Konstruktion von intelligenten Roboter-

---

<sup>23</sup> Siehe auch Dörner 1999.

systemen von Problemfall zu Problemfall variieren. Eines gilt allerdings immer: Die Ausführung der einzelnen Schritte erfordert immer den vollen Überblick über die Problem- und Anwendungsdomäne, die gründliche Kenntnis und Übung im Umgang mit der verwendeten Beschreibungssprache sowie ein gewisses Maß an Intuition. Auch hat die Praxis gezeigt, dass die Ausführung eines bestimmten Schrittes umso einfacher ist, je intensiver die davor liegenden Schritte bearbeitet und inhaltlich abgeschlossen wurden. In der Entwicklungspraxis hat sich daher ein iteratives Vorgehen bewährt, in dessen Verlauf sich die einzelnen Schritte wechselseitig bedingen und daher auch gegenseitig beeinflussen.

---

### 3.3 Kognitives Vorgehensmodell

Die folgenden Abschnitte beschreiben einerseits einen Problemlösungsprozess, wie er sich bei komplexeren Entwicklungsprojekten von Robotersystemen bewährt hat. Das Ergebnis ist eine Lösung, die die Entwicklung initierende Problematik löst. Andererseits basiert die damit vorgeschlagene Entwicklungsmethodik im Form eines Vorgehensmodells auf einem Prozess, das sowohl den Anforderungen von prozessorientierten Problemstellungen als auch Fragestellungen der kognitiven Ausgestaltung von Robotersystemen Rechnung trägt. Da die Entwicklung von Robotersystemen insgesamt als ein kognitiver Prozess aufgefasst wird, wird diese Vorgehensweise als kognitive Entwicklung bezeichnet.

#### 3.3.1 Entwicklungsprozess

Die Entwicklung als Ganzes und die darin vorgesehenen Aktivitäten im Einzelnen, gewährleisten einen agilen Entwicklungsprozess, der um die Aspekte der Wissensbasierung und Wiederverwendbarkeit von Ergebnistypen erweitert wurde. Unabhängig dieser Erweiterungen basiert die Methode auf dem allen agilen Methoden gemeinsamen Mindset. Dieser Bezug trägt auch der Tatsache Rechnung, dass unter Umständen mehrere Parteien an dem Vorhaben beteiligt sind und sich der Entwicklungsprozess trotz dieser „Verteiltheit“ als ein problem- und wissensorientierter, iterativer und damit kognitiver Prozess gestaltet und dadurch die Prinzipien agiler Lösungsentwicklung bedingt (Abb. 3.8).

Die Beschreibung der Vorgehensweise zur Orchestrierung von an der Entwicklung von Lösungen beteiligten Ressourcen (Personen, Rechner, Technologien, Dokumente, etc.) wird als *Prozessmodell* bezeichnet. In einem Prozessmodell werden alle Aktivitäten, als auch die zu erstellenden Ergebnistypen beschrieben. Unter diese Ergebnistypen fallen auch die notwendigen Artefakten und Zwischenergebnisse, wie Dokumentation, Modelle, Projektpläne, Source Code etc. Grob lassen sich die folgenden Prozesse unterscheiden:

- *Problemmodellierung*: Dieses Vorgehensmodell basiert nicht zufällig auf der soziologischen Theorie Luhmanns, die man „ketzerisch“ als eine von Hegel ererbte System-

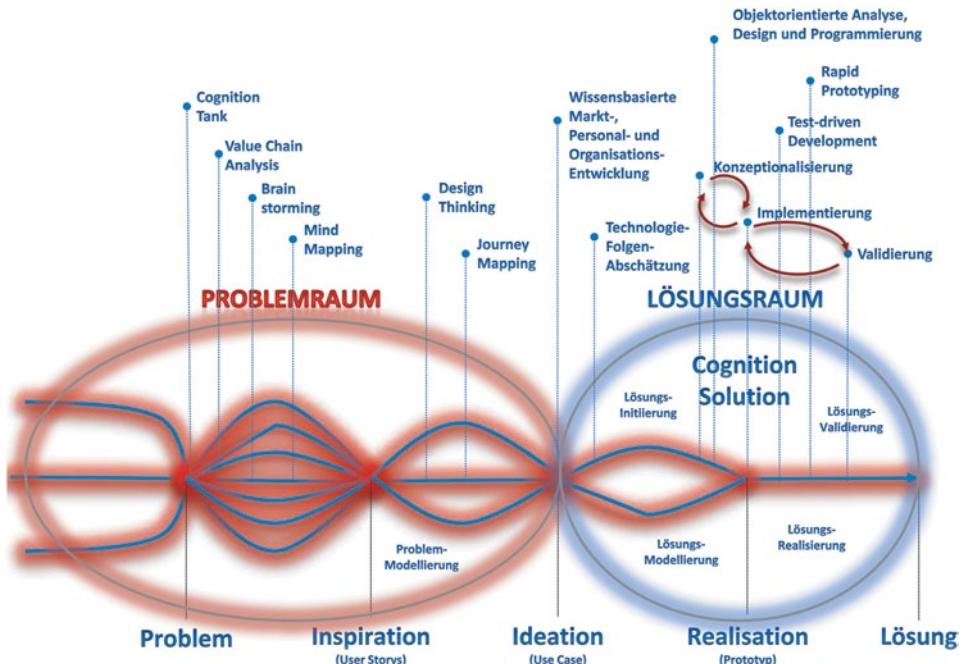


Abb. 3.8 Kognitiver Entwicklungsprozess

theorie bezeichnen kann, weil die darin zum Einsatz kommenden Schlüsselwörter der Selbsterreferenz bzw. der Autopoiesis Neuauflagen alter Begrifflichkeiten zu sein scheinen. Neu und damit bedenkenswert an dieser Systemtheorie ist allerdings die Tatsache, dass Menschen nur noch zu (Umwelt)problemen von Gesellschaften avancieren. Was zu diesem Ansatz führt: *Ohne Probleme keine Systeme*. Insofern besteht ein früher Schritt darin, das zu lösende Problem zu definieren. Bertrand Russell hat einmal behauptet, wenn ein Problem gut formuliert ist, dann hat man auch schon die Lösung. Wie im Software Engineering ist dieser Schritt also auf im Bereich der Robotik entscheidend für die gesamte weitere Entwicklung, da hier nicht nur die Problematik verstanden, sondern in Teilen auch die Funktionalität des zu entwickelnden Systems bereits festgelegt wird. Da ein Robotersystem gemäß der Sichtweise dieses Buches ein interoperationsorientiertes System darstellt und dieses in der Regel ein bestimmtes Problem aus einem Problemraum lösen soll, muss festgelegt werden, in welcher dieser Problemräume sich das spätere Robotersystem bewegen und aus welchen Quellen das notwendige Wissen für den betreffenden Bereich gewonnen werden soll. Bereits in diesem Schritt wird der interoperationsorientierte und damit wissensbasierte Ansatz erkennbar, indem *Wissensquellen* eruiert werden. Als Quellen kommen beispielsweise Datenbanken, Bücher oder menschliche Experten in Frage.<sup>24</sup>

<sup>24</sup> Vgl. Date 2000.

- *Lösungsmodellierung:* In dieser Phase wird beispielsweise festgelegt, welche Soft- und Hard- und ggf. Brainware-Strukturen benötigt werden, welche Arten von Inferenzen geleistet werden müssen, wie die einzelnen Aktoren- bzw. Sensorenkomponenten arbeiten sollen, aber auch wie die Benutzerschnittstelle aufgebaut sein soll usw.
- *Lösungsrealisierung:* Gerade für die Erstellung eines produktionsreifen Robotersystems ist die frühzeitige Erstellung eines ausführbaren Prototyps („Rapid Prototyping“) von großer Bedeutung. Oft lässt sich erst anhand eines solchen Prototyps entscheiden, ob die ursprünglich geforderte Funktionalität tatsächlich den Anforderungen genügt, die man bei der Arbeit mit dem System an eben dieses System stellt.
- *Lösungsvalidierung:* Angesichts des sich immer weiter verbreiternden Einsatzgebietes von Robotersystemen und dies in allen Bereichen, wächst die „Verantwortung“ aller an solchen Entwicklungsprojekten Beteiligten. Fehlfunktionen einzelner Teile oder Systeme können unangenehme, wenn nicht sogar katastrophale Folgen nach sich ziehen. Daraus entsteht die Forderung nach hoher Zuverlässigkeit der einzelnen Teile bzw. Systeme respektive des gesamten Robotersystems. Mit der Forderung nach hoher Zuverlässigkeit einher geht der Wunsch nach kostengünstig einsetzbaren Verfahren, um die Erfüllung der Forderungen aufzuzeigen.

Innerhalb der Lösungsrealisierung durchläuft der Entwicklungsprozess des Systems die Phasen der *Konzeptionalisierung*, *Implementierung* und *Validierung*. Je nach Problemstellung können diese Phasen sequentiell durchlaufen werden (Strang) oder aber durch notwendige Rückkopplungsschleifen (Feedback-Loops) sich zu einer iterativen Vorgehensweise (Spirale, Kreis) ausgestalten. Unabhängig der jeweiligen Ausprägungsform bleiben die Aktivitäten (Anforderungsanalyse, Prozess- und Aktivitätsanalyse, etc.) und die Ergebnistypen (Anforderungsliste, Prozess- und Aktivitätenmodell, Lösungsmodell, etc.) im Modell verbindlich.<sup>25</sup>

Im Rahmen der *Initialisierung* werden sämtliche Vorbereitungen getroffen, um das Entwicklungsprojekt starten zu können. Dies betrifft sowohl Fragen der Projektorganisation, als auch die Erteilung des Entwicklungs- und Projektauftrages. Um den Auftrag definieren zu können, ist es unabdinglich, sich über die Anforderungen bereits in diesem frühen Zeitpunkt Klarheit zu verschaffen.

In der Phase der *Konzeptionalisierung* wird die reale Welt auf die Existenz von Entitäten und Entitätsbeziehungen hin untersucht, und ein entitätsorientiertes Ontologiemodell dieser realen Welt erstellt.<sup>26</sup> Es wird gefragt, *was mit welchen Entitäten warum* geschieht oder geschehen soll. Denken in Entitäten (= Erkenntnisobjekte, Begriffe etc.) heißt verallgemeinern, das Gemeinsame herausheben. Auf Basis dieses Ontologiemodells wird das Lösungsmodell entwickelt, das sich zum einen an den Anforderungen orientiert und die beteiligten Komponenten, die Prozess-, Aktivitäts-, Präsentations-, Integrations- und Entscheidungslogik beschreibt. Sollte der Einsatz von Cognitive Computing zur kognitiven

<sup>25</sup> Siehe auch Versteegen 2000.

<sup>26</sup> Siehe auch Oyama 1985.

Ausgestaltung des Robotersystems notwendig sein, werden auch die darin angewandten Technologien beschrieben. In der Phase der *Implementierung* wird das entitätsorientierte Lösungsmodell zunächst in die Welt der Zielarchitektur (Hardware, Software, Brainware) übertragen, ggf. ergänzt oder modifiziert. Das Lösungsmodell wird so zu einer einsatzfähigen Lösung konkretisiert und überführt. Es wird genau festgelegt, *wie alles im Detail* funktioniert. In der Phase der *Validierung* wird permanent überprüft, ob die geschaffene Lösung noch den Anforderungen entspricht bzw. ob diese Anforderungen noch Bestand haben oder aber sich geändert haben oder aber ggf. das Lösungsmodell und damit die Implementierung modifiziert werden muss. Weiterhin muss der Forderung nach hoher Zuverlässigkeit der einzelnen Teile bzw. Systeme respektive des gesamten Robotersystems Rechnung getragen werden.

Insofern erscheint es an dieser Stelle angebracht, auf die Zuverlässigkeitsnachweisverfahren kurz einzugehen, wenngleich diese wiederum – je nach Ausgang der Validierung – auf die Modellierungsverfahren der vorgelagerten Phasen (Problem- und Lösungsmodellierung, Lösungsrealisierung etc.) einwirken. Die Verfahren lassen sich im Wesentlichen in die deterministischen und die probabilistischen Verfahren unterscheiden. Beide haben den Vorteil, zu quantitativen Ergebnissen zu führen. Sie können allerdings erst auf einsatzfähige Robotersysteme angewandt werden und sind daher gegen Modellierungs- und Realisierungsfehler resistent. Mittel gegen solche frühzeitigen Fehler stellen die informellen Verfahren zur Verfügung.

Beim Verfahren mit *deterministischen Mitteln* erhält man eine Aussage darüber, ob das betrachtete Robotersystem korrekt ist oder nicht. Im günstigsten Fall kommt man dabei zu dem Ergebnis, das Robotersystem werde unter allen Umständen richtig arbeiten. Bei manchen Verfahren bzw. Methoden bezieht sich die Korrektheitsaussage allerdings nur auf ein bestimmtes Korrektheits-Kriterium und nicht auf die vollständige Korrektheit. Ein solches Kriterium kann im Test bestimmter Systemteile bestehen.

Ein *probabilistisches Vorgehen* bietet sich nicht nur als Alternative zum deterministischen Arbeiten an. Hierbei erhält man keine Aussage darüber, ob das Robotersystem korrekt agiert, sondern nur darüber, mit welcher Wahrscheinlichkeit es sich richtig verhält oder mit welcher Wahrscheinlichkeit es im Anforderungsfall richtig reagiert. Zumeist steht Letzteres im Vordergrund. Das Verhalten des Robotersystems wird also charakterisiert, nicht das Robotersystem selbst. Verlangt man Aussagen über hohe Zuverlässigkeiten, erfordern probabilistische Verfahren allerdings auch einen hohen Testaufwand. In vielen Fällen allerdings darf dieser ganz oder teilweise durch Produktionserfahrung ersetzt werden, was eine erhebliche Verbilligung der Nachweise mit sich bringen kann. Insofern kommt man mit einer Kombination probabilistischer und deterministischer Vorgehensweisen am sichersten zu brauchbaren und verlässlichen Validierungsergebnissen (Abb. 3.9).

Zusätzlich zu den deterministischen und probabilistischen Verfahren sind noch *informelle* Verfahren anwendbar. Sie haben den anderen gegenüber den Vorteil, bereits während der Entwicklung eines Robotersystems und dort in bereits frühen Phasen anwendbar zu sein und nicht das fertige Produkt vorauszusetzen. Sie beruhen auf dem Einbeziehen von weiteren Experten in die Entwicklung.



**Abb. 3.9** Verfahren zur Validierung von Robotersystemen

Ein Begriffssystem zur Charakterisierung der Zuverlässigkeit von Robotersystemen muss für die Beschreibung und Berechnung der Zuverlässigkeit von Hard-, Soft- und Brainware geeignet sein. Weiterhin muss das Begriffssystem sowohl das Verhalten des Ganzen als auch der einzelnen Bestandteile beschreiben können. Einen weiteren Aspekt bringt die Internationalität mit sich, in dessen Rahmen Projekte der Robotersystementwicklung sich im Allgemeinen bewegen. Aus all diesen Gründen ist es sinnvoll, die für den Rest dieses Kapitels erforderlichen Begriffe zu erfassen:

- **Zuverlässigkeit:** Umfasst primär die Gesamtheit der Eigenschaften, die die Verfügbarkeit und die sie bestimmenden Faktoren beschreiben: Funktionsfähigkeit, Instandhaltbarkeit und Instandhaltungsbereitschaft. Daneben versteht man darunter die Fähigkeit von Teilsystemen, für eine gegebene Zeit korrekt zu arbeiten.
- **Anforderung:** Funktion oder Gesamtheit von Funktionen eines Systems oder eines Teilsystems hiervon, deren Ausführung notwendig ist, um eine vorgegebene Aufgabe zu erfüllen bzw. ein Problem zu lösen.
- **Fehler:** Ist zunächst aus Sicht eines zustandsbasierten Systems ein Zustand, der dadurch charakterisiert ist, dass durch die Unfähigkeit des Systems eine geforderte Funktion nicht auszuführen ist. In Bezug auf die Software eines Systems stellt ein Fehler eine Abweichung zwischen der im Programm realisierten und der beabsichtigten Funktion dar. Global betrachtet ist ein Fehler eine feststellbare Nichterfüllung einer Forderung.
- **Versagen:** Aus actionstheoretischer Sicht zeigt sich ein Versagen im Verhalten des Systems, das nicht mit der beabsichtigten oder spezifizierten Funktion übereinstimmt. Es entsteht eine Störung bei zugelassenem Einsatz einer Einheit aufgrund einer in ihr selbst liegenden Ursache.
- **Störung:** Eine solche liegt vor, wenn eine fehlende, fehlerhafte oder unvollständige Erfüllung einer geforderten Funktion testiert wird.
- **Versagenswahrscheinlichkeit:** Wahrscheinlichkeit in Prozent, dass ein System oder ein Teil des Systems, sich in einem Beanspruchungsfall (einer Anforderung) nicht wie gefordert verhält und damit insgesamt versagt.
- **Verfügbarkeit:** Die *qualitative Dimension* bezeichnet die Fähigkeit eines Systems oder eines Teils des Systems, in dem Zustand zu sein, eine geforderte Funktion unter gegebenen Bedingungen zu erfüllen. Die *quantitative Dimension* drückt die Wahrscheinlichkeit-

keit aus, dass ein System oder ein Teil des Systems zu einem vorgegebenen Zeitpunkt in einem funktionsfähigen Zustand anzutreffen ist.

- **Überlebenswahrscheinlichkeit:** Drückt die Wahrscheinlichkeit in Prozent aus, dass die Betrachtungseinheit über einen Zeitraum nicht versagt.
- **Missionsverfügbarkeit:** Stellt die Wahrscheinlichkeit in Prozent dar, dass ein System oder ein Teil des Systems während einer bestimmten Folge von Anforderungen nicht versagt.
- **Sicherheit:** Umfasst die Fähigkeit, keine Gefährdung eintreten zu lassen *oder* Abwesenheit von Gefahr *oder* Sachlage, bei der die erwartete Schadenshöhe kleiner als das Grenzrisiko ist.
- **Versagenswahrscheinlichkeit:** Beschreibt die Wahrscheinlichkeit in Prozent, dass das System oder ein Teil des Systems im Betrachtungszeitraum versagt.
- **Risiko:** Stellt den Erwartungswert der Schadenshöhe dar.

In Bezug auf Robotersysteme und deren Zuverlässigkeit betrachtung muss man zwischen Hard-, Soft- und Brainware trennen. So wird beispielsweise Software als nicht reparierbar angesehen. Sie unterscheidet sich damit von der Hardware eines Robotersystems. Sie verändert ihre Eigenschaften durch Gebrauch nicht und sie fällt nicht in dem Sinn aus, wie ein Hardware-Bestandteil ausfallen kann. Sie kann allenfalls durch Löschung oder Veränderung von außen beschädigt werden. Dann aber sagt man, es liege nicht mehr die ursprüngliche Software vor und sieht die geänderte Software als eine neue Betrachtungseinheit an, die mit der ursprünglichen nur noch eine mehr oder weniger große Ähnlichkeit hat.

Wenn Software nach einem Versagensfall im Laufe des Betriebs verändert werden muss, so hat dies in der Regel mit der Behebung von Fehlern zu tun, wenigstens, wenn die Versagensursache auf die Software selbst zurückzuführen ist. Eine Fehlerbehebung ist meist sehr zeitraubend und setzt die Außerbetriebnahme des Robotersystems voraus. In vielen Fällen ist auch von vornherein gar nicht abzusehen, wie umfangreich die damit verbundenen Änderungen oder Ergänzungen im Gesamtsystem sind. Daher spricht man nach einer Fehlerbehebung bzw. nach dem Einspielen der neuen Version von einem neuem Robotersystem bzw. einer neuen Version des Robotersystems.

Deshalb spielt bei Software auch die Kenngröße der Verfügbarkeit keine große Rolle. Die wichtige Kenngröße ist vielmehr die *Versagensrate*. Von dieser abgeleitet sind auch die Größen der *Versagenswahrscheinlichkeit*, der *Überlebenswahrscheinlichkeit* von Bedeutung. Dies gilt in erster Linie für kontinuierlich arbeitende Programme, beispielsweise für Betriebssysteme.<sup>27</sup>

### 3.3.2 Entwicklungsprojekt

Robotersysteme werden aufgrund der Komplexität der Problem- und Fragestellung der Robotik gewöhnlich im Rahmen von Projekten entwickelt. Ein Projekt ist dadurch

---

<sup>27</sup> Vgl. Vogt 2001.

gekennzeichnet, dass im Allgemeinen mehrere Personen für eine begrenzte Zeit zusammenarbeiten und dabei ein gemeinsames, definiertes und damit abgegrenztes *Ziel* verfolgen. Dabei sind zusätzlich gewisse *Randbedingungen* zu beachten, die die Zielerreichung entweder begünstigen oder aber erschweren können. Eines der Erschwernisse ist beispielsweise der Punkt, dass das Robotersystem, das aus dem Projekt resultiert, eine bestimmte Funktionalität und Qualität bieten soll und dass ein bestimmter Zeit- und Kostenrahmen nicht überschritten werden darf. Neben den rein technischen Aspekten der Lösungserstellung sind also auch organisatorische Aspekte zur Steuerung des Projekts wichtig, damit das Ziel unter den gegebenen Randbedingungen erreicht wird.<sup>28</sup>

Eine *Projektorganisation* bedeutet im Kern, dass die Gesamtaufgabe in Teilaufgaben zerlegt wird und dass diese Teilaufgaben auf eine systematische, planmäßige Weise erledigt werden. Ein systematisches Vorgehen kann dadurch erreicht werden, dass festgelegte, bewährte Verfahren zur Hard- und Softwareentwicklung angewendet werden. Solche Verfahren definieren *Richtlinien* für die Vorgehensweise in zwei Dimensionen:

- *Zeitliche Dimension*: bezüglich der Abfolge der Verfahrensschritte
- *Methodische Dimension*: bezüglich der Arbeitsweisen, die geeignet sind, die einzelnen Teilziele zu erreichen

Die Methoden basieren wiederum auf *Prinzipien*, also Grundsätzen, die während der Entwicklungsarbeiten einzuhalten sind. Dabei ist insbesondere wichtig, mit Hilfe von Modellierungsmitteln und -sprachen Modelle zu bilden, um die logische Komplexität der Projektarbeiten bewältigen zu können.

Bevor ein spezielles Verfahren zur Systementwicklung zu einem Projektmanagementsystem integriert und kombiniert werden, ist es sinnvoll, sich einen Überblick über die allgemeinen Teilaufgaben und Arbeiten zu verschaffen, die bei der Entwicklung von Robotersystemen zu erledigen sind. Im Zentrum stehen dabei entwicklungsspezifische Tätigkeiten: Sie müssen ausgeführt werden, um ausgehend von der Problemstellung ein einsatzfähiges Robotersystem zu bauen. Die Tätigkeiten bauen in der Regel aufeinander auf, so dass das Ergebnis einer Tätigkeit der Ausgangspunkt für die nächste ist.

So legt man bei der *Problemmodellierung* im Rahmen einer objektorientierten Analyse (Anforderungsermittlung, Problemmodell) die gewünschte Funktionalität des Systems fest, vereinbart also, was das Robotersystem leisten soll. Zudem werden Randbedingungen definiert, beispielsweise der Zeit- und Kostenrahmen, die Hard- und Softwareplattform, auf der die Komponenten implementiert bzw. installiert werden soll. Zur Entscheidungsfindung kann eine Vor- oder Machbarkeitsstudie mit Lösungsalternativen und eventuell ein lauffähiger Prototyp des Systems mit begrenzter Funktionalität erstellt werden. Nach Auswahl einer der Möglichkeiten wird eine Anforderungsdefinition in Form eines Problemmodells formuliert, also ein Vertrag über die zu erledigenden Aufgaben und die

<sup>28</sup> Siehe auch Kellner 2001.

geltenden Randbedingungen. Ein Projektplan gibt einen Überblick über die benötigten Personal- und Sachmittel sowie über den vorgesehenen Zeit- und Kostenaufwand.

Bei der *Systemmodellierung* wird die Architektur des Gesamtsystems erarbeitet, also die Komponenten, aus denen sich das System zusammensetzt, entworfen und ihr Zusammenspiel spezifiziert. gemäß der Interoperationstheorie bietet sich hierbei an, von den Interoperationen, den damit verbundenen Funktionen und Daten auszugehen, die das Robotersystem ausführen bzw. verarbeiten soll, und das System dann top-down zu entwickeln, also schrittweise zu modularisieren. Am Anfang steht dabei ein grober Erstentwurf, der ein noch grobes Interoperations-, Funktions- und Datenmodell des Robotersystems bzw. ein entsprechendes Komponenten- bzw. Objektmodell enthält. Aus dem Grobentwurf wird dann schrittweise über Iterationen ein Feinentwurf entwickelt, der die ausgeprägten Komponenten, vollständige Funktionsmodule und Daten-, Informations- und Wissensstrukturen bzw. ein vollständiges Komponentenmodell umfasst. Der Feinentwurf definiert also eine detaillierte Systemstruktur aus Komponenten und zugehörigen Schnittstellen. Die Systemmodellierung sollte unabhängig von einer bestimmten Hard- und Softwareplattform sein, um zum einen nicht die Lösungsqualität einzuschränken und zum anderen die Portabilität des Systems zu unterstützen.

Bei der *Systemrealisierung* wird das entworfene System in einem bestimmten Technologien-Mix als Kombination von Hard- und Softwaretechnologien realisiert. Dabei werden die Komponenten des Feinentwurfs einzeln „ausentwickelt“ und zu einem Gesamtsystem integriert.

Im Rahmen der *Systemvalidierung* wird das fertige Robotersystem schließlich in der Echtzeit oder in einer Simulations- bzw. Experimentierumgebung eingesetzt. Es folgt je nach Testergebnissen eine fortlaufende Verbesserung, also eine Anpassung an veränderte Gegebenheiten, das Einspielen neuer Versionen sowie die Beseitigung eventuell noch vorhandener Fehler.

Neben diesen phaseninhärenten Tätigkeiten fallen übergreifende bzw. begleitende Aufgaben an, die nicht nur zusätzlich, sondern genau so gewissenhaft zu erledigen sind. So müssen projektbegleitende *Dokumentationen* erstellt werden. Dazu gehören eine Entwicklungsdocumentation, d. h. die Beschreibung des Entwicklungsvorgangs mit seinen Zwischenergebnissen, technische Dokumentationen mit ihren Beschreibungen der einzelnen Komponenten, der Architekturen und internen Details, sowie eine Benutzerdokumentation als Benutzerhandbuch. Ebenfalls müssen die einzelnen Module und das Gesamtsystem projektbegleitend *getestet* und *validiert* werden. Beim Testen wird im System nach Fehlern gesucht, bei der Validierung wird geprüft, ob das System bzw. die einzelnen Komponenten die anfangs formulierten Anforderungen erfüllen. Im Einzelnen unterscheidet man Komponententests, die die jeweiligen Komponenten anhand ihrer Spezifikation überprüfen, Integrationstests, die Teilsysteme aus mehreren Komponenten betrachten, Systemtests, die das Gesamtsystem anhand der Systemspezifikation untersuchen, und Abnahmetests, die nach der Installation in der Echtzeit und in der Produktivumgebung durchgeführt werden.

### 3.3.3 Projektmanagement

Die allgemeinen Regeln des Projektmanagements gelten auch für Robotik-Projekte. Demnach sind Robotik-Projekte aus der Sicht der Beteiligten einmalige Vorhaben mit hoher Komplexität, die innovatives Vorgehen erfordern, deren Start und Ende terminiert, deren Ressourcen begrenzt und die mit Risiken verbunden sind. Solch geartete Projekte und deren Teilprojekte erfordern ein Multiprojektmanagement, weil sie einerseits um einen gemeinsamen Ressourcenpool konkurrieren und andererseits inhaltlich miteinander verknüpft sind. Damit verbunden sind Projektportfolio und Projektpriorisierung, d. h. die Zusammenstellung der innerhalb eines definierten Planungshorizonts anstehenden Projekte und die Bestimmung der Reihenfolge, in der diese simultan bzw. sukzessiv umgesetzt werden sollen.

Ein Robotik-Projekt kann sich beziehen auf:

- einzelne Informations- und Kommunikationssysteme (Entwicklungs-, aber auch größere Wartungs- und Reengineering-Projekte; Hard- und Softwareauswahl, Lösungseinführung und größere Releasewechsel),
- übergreifende Inhalte (beispielsweise Umweltdaten oder Interoperationsmodellierung, Standardisierung und Vernetzung von Komponenten und Hard-, Software- und Brainwaresystemen) oder interne Vorhaben (beispielsweise Optimierung von Abläufen in der Entwicklung oder Einführung neuer Hard- und Softwareentwicklungsmethoden).

Robotik-Projekte sind demzufolge eng mit Hard- und Software bzw. Informations- und Wissens-Engineering verzahnt.

Die folgende Abbildung stellt Projektlaufzeit, Robotersystem- und Projekt-Lebenszyklus und ihre zeitliche Folge dar. Das Projektvorfeld bildet eine Wochen bis Jahre umfassende Vorlaufzeit, in der Ideen generiert, Vorüberlegungen zu einem potenziellen Projekt angestellt und bei hinreichender Konkretisierung Vor- und Machbarkeitsstudien erarbeitet werden. Die Projektlaufzeit erstreckt sich zwischen Projektstart und -ende, sie ist aus Managementperspektive grob in die Phasen Projektplanung, Projektsteuerung und Projektabschluss gegliedert. Projektabschlusskontrollen zählen teilweise zur Projektabschlussphase (z. B. Fortschreibung von Erfahrungsdaten und -kennzahlen), können aber auch erst während der Nutzungszeit anfallen (beispielsweise Überprüfung des Eintritts der prognostizierten Wirtschaftlichkeit).

Der Robotersystem-Lebenszyklus beginnt mit dem Projektstart, endet aber erst bei Außerbetriebnahme des Robotersystems. Systemplanung und -entwicklung dieses Robotersystems liegen parallel zur Projektlaufzeit. Wie die variable Länge des Projektvorfeldes, variiert auch die Dauer der anschließenden Nutzungszeit. Sie zählt oft nach Jahren und umfasst den laufenden Betrieb des Robotersystems, seine Wartung und Weiterentwicklung sowie erforderliche Reverse- und Reengineering-Aufgaben (Abb. 3.10).

Die Grundidee, das Vorgehen als ein phasenorientiertes Modell auszustalten, liegt in einer Reduzierung von Komplexität durch Aufteilung in überschaubare(re) Teilschritte.



**Abb. 3.10** Lebenszyklus des Projektes und des Robotersystems

te, an deren Ende jeweils eine Überprüfung (Meilenstein) der erreichten Zwischenziele mit anschließender Freigabe der nächsten Phase bzw. Rückgabe zur Überarbeitung oder schlimmstenfalls Projektabbruch steht. Traditionell entsprechen den Phasen Projektplanung, -Steuerung und -abschluss fachliche Planungs- und Entwicklungsphasen, deren Untergliederung im Detail von Entwicklungsprojekt zu Entwicklungsprojekt variiert. Projektplanung setzt einen fachlichen Grobentwurf voraus. Projektsteuerung läuft parallel zum fachlichen und technischen Feinentwurf und zur Realisierung sowie Einführung des Robotersystems. Dem Projektabschluss entspricht die Übergabe in den Echteinsatz bzw. einen laufenden Betrieb. Diese Grundidee der Aufeinanderfolge von Planung, Durchführung/Steuerung und Kontrolle bleibt auch beim Wechsel auf ältere oder neuere Konzepte der Hard- und Softwareentwicklung erhalten.

Zur Verteilung der Projektaufzeit auf die einzelnen Phasen zeigt die Erfahrung, dass vielfach zu schnell mit der Umsetzung erster Entwürfe begonnen und damit unnötiger Mehraufwand verursacht wird. Höherer Planungsaufwand kann nicht nur Realisierungs-, sondern auch spätere Betriebs- und Wartungskosten substituieren, auch wenn man dabei einkalkulieren muss, dass diese Substitution erst zu einem späteren Zeitpunkt stattfindet und dadurch hinter den zunächst anfallenden höheren Planungskosten optisch zurücktritt.

Die *Planungsphase* fußt auf einem problemorientierten Grobentwurf, der eine für die weiteren Planungsaufgaben hinreichend tiefe Zerlegung der Projektaufgaben in einem Projektstrukturplan erlaubt. Dieser ist Grundlage für Aufwandsschätzungen, Termin- und Kapazitätsplanung, die gemeinsam erfolgen müssen, weil der Zeitbedarf einer Aufgabe nur abhängig von der verfügbaren Ressourcenkapazität bestimmt ist. Personalplanung, Planung der Projektberichterstattung und -dokumentation schließen sich an. Den Abschluss bildet die Projektkostenplanung, die ihrerseits Bestandteil der Wirtschaftlichkeitsanalyse für das anstehende Projekt ist. Neben den Projektkosten gehen in die Wirtschaftlichkeitsanalyse nicht entwicklungsbedingte, einmalige Kosten, die späteren Betriebs- und Wartungskosten sowie der erwartete Nutzen ein. Übersteigen die voraussichtlichen Gesamtkosten über den ganzen Robotik-Lebenszyklus den Gesamtnutzen, muss die Projektpla-

nung überarbeitet oder das Projekt unter Umständen abgebrochen werden. In der Praxis ist dies eher selten, weil künftige Kosten und Nutzen zum einen im Bereich der Robotik nur mit hoher Unsicherheit abschätzbar sind und sich zum anderen aus dieser Tatsache auch vielfältige Beeinflussungsmöglichkeiten in Richtung erwünschter Ergebnisse eröffnen.

Ziel der *Projektsteuerung* ist der laufende Vergleich zwischen Soll (Termine, Kapazität, Produkt- und Prozessqualität, Kosten laut Plan) und tatsächlich erzieltem Ist. Je früher Abweichungen erkannt und analysiert werden, umso eher greifen Maßnahmen, die das Projekt zurück zum Plan bringen sollen. Selbstverständlich können bei gravierenden Abweichungen auch Plananpassungen nötig werden. Erfolgreiche Projektsteuerung setzt kurzyklische und realistische Rückmeldungen der Projektmitarbeiter, umgehende Soll-Ist-Vergleiche und bei Abweichungen schnelle Auswahl und Einleitung geeigneter Maßnahmen voraus.<sup>29</sup>

Ein eindeutiger *Projektabschluss* markiert den Übergang zwischen Projektlauf- und Nutzungszeit, er fällt mit der Auflösung des Projektteams und der Übergabe des Robotersystems an andere Verantwortliche zusammen. Mit dem Projektabschluss verbunden sind deshalb einerseits die Würdigung der Leistung des Projektteams und andererseits die Abnahme des Gesamtsystems, einschließlich der Robotersystem- und Projektdokumentation durch die für die Echtanwendung bzw. Wartung verantwortlichen Gruppen. Projektabschlusskontrollen dienen zunächst der Bewertung des Projektverlaufs und der Dokumentation der gewonnenen Erfahrungen. Es ist zweckmäßig, wenn diese sich in der Fortschreibung von Projektkennzahlen bzw. von Erfahrungsdatenbanken niederschlagen.<sup>30</sup> Die Planung neuer Projekte profitiert davon, erkannte Fehler können bei künftigen Projekten vermieden oder reduziert werden. Projektbenchmarking setzt ebenso wie die Nutzbarmachung von Wissensmanagement (vgl. Haun 2003) für die Projektarbeit die Speicherung von Projekterfahrungen voraus. Eine zweite Art von Projektabschlusskontrollen kann erst in der Nutzungszeit erfolgen. Sie bezieht sich auf die ersten Erfahrungen bei der konkreten Nutzung des Systems in der Echtzeit (beispielsweise Fehler- und Performance-Verhalten), auf dessen tatsächliche Verwendung und schließlich auf Untersuchungen zum Eintritt der geplanten Wirtschaftlichkeit.

Wie im Software Engineering waren die ersten im Bereich des Robotik-Projektmanagement eingesetzten Tools sogenanntes Papierware, d. h. Papier und Bleistift. Frühe Tools standen für das Projektmanagement großer technischer Projekte in Form von Netzplantechnik-Software zur Verfügung, wobei nach den ersten Anfängen nicht nur Termine, sondern auch Kapazitäten und Kosten in die Berechnungen einbezogen werden konnten. Allerdings waren diese Tools für Projekte anderer Größenordnungen, als es die damaligen Projekte waren, konzipiert. Ihr Einsatz für IT-Projekte war demzufolge eher „oversized“ und war nicht allzu häufig, zumal die Programmsysteme anfangs sehr teuer waren.<sup>31</sup> Eine

<sup>29</sup> Vgl. Schelle 1996

<sup>30</sup> Vgl. Elmasri und Navathe 2002.

<sup>31</sup> Siehe auch Maier 1990.

Rolle spielte auch, dass Robotik-Projekte sehr viele Parameter beinhalten, die sich im Projektverlauf häufig ändern und dadurch hohen Überarbeitungsaufwand von Netzplänen hervorrufen. In den 80er Jahren wurden viele Netzplantechnik-Programmsysteme abgemagert und als PC-Software zu niedrigeren Preisen angeboten bzw. neu entwickelt. Die nun mögliche Nutzung am Arbeitsplatz der Projektmitarbeiter führte zu einer breiteren Verwendung.

Aktuell eingesetzte Werkzeuge zum Projektmanagement lassen sich in vier Gruppen einteilen:

- Werkzeuge, die Koordination und Kommunikation projektteamexterner bzw. -interner Projektbeteiligter unterstützen;
- Planungswerkzeuge zur Aufgabendarstellung, Zeit-, Kapazitäts- und Budgetplanung;
- Werkzeuge zur Projektsteuerung, die Projektfortschritte und Projektprobleme frühzeitig zu erkennen helfen.

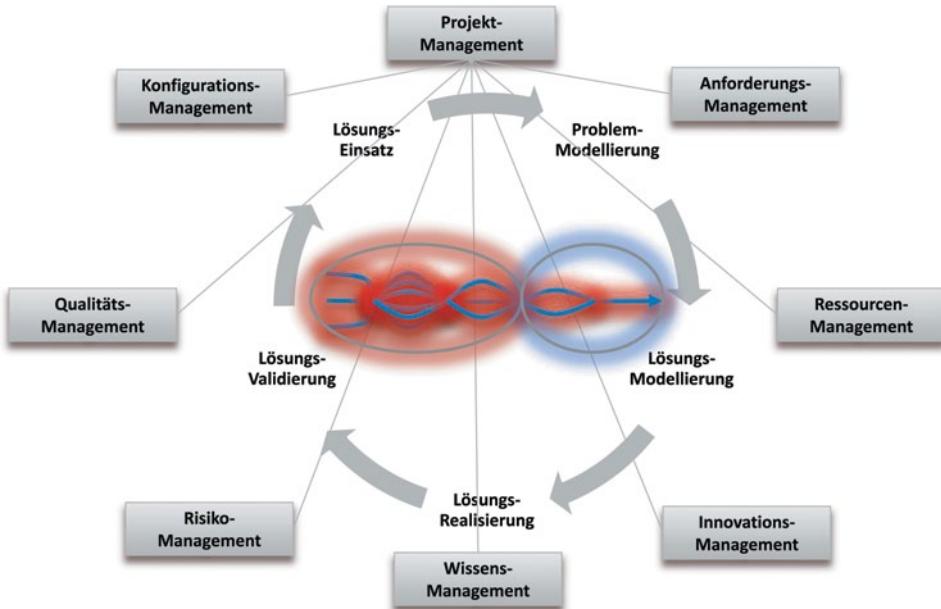
Der Einsatz solcher Werkzeuge ist v. a. dann erfolgreich, wenn die verlangten Projektergebnisse bekannt, identifizierbar und messbar sind. Innovative Projekte, die hohe Flexibilität verlangen und meist auch neue Technologien beinhalten, ziehen deutlich weniger Nutzen aus diesen Werkzeugen. Auf jeden Fall dürfen solche Werkzeuge zum Projektmanagement nicht isoliert von der Systementwicklung und deren spezifischen Werkzeugen gesehen werden.<sup>32</sup> Projektführungssysteme müssen mit Produktverwaltungssystemen zu einer gemeinsamen Entwicklungsumgebung integriert werden. Diese basiert auf einem Repository, das sowohl Lösungs- als auch Projektdaten aufnimmt. Um das Repository gruppieren sich Einzeltools für alle Aufgaben, die in der Projektlaufzeit und während der anschließenden Wartung anfallen. Von hoher Bedeutung sind die Durchgängigkeit der Werkzeuge (Output von Werkzeug A kann als Input für Werkzeug B im nächsten Aufgabenschritt verwendet werden) und eine standardisierte Benutzeroberfläche über alle Werkzeuge hinweg. Wesentlicher Bestandteil aus Entwicklungs- und Projektmanagementsicht sind auch Werkzeuge für das Konfigurations- und Change-Management. Die Integration von Wissensmanagement-Werkzeugen empfiehlt sich, wenn systematisch ein Wissensmanagement aufgebaut werden soll (Abb. 3.11).

Angesichts der Offenheit heutiger Computerplattformen ist es möglich, einzelne Werkzeuge anhand unternehmensindividueller Anforderungen auszuwählen und zu einer Entwicklungsumgebung zu kombinieren. Wichtige zusätzliche Eigenschaften sind:

- Teamunterstützung durch Mail und Groupware,
- gestufte Zugriffsberechtigungen für die Beteiligten (die zu allen ihre Aufgaben betreffenden Produkt- bzw. Projektdaten Zugriff auf dem jeweils aktuellen Stand benötigen),
- die Möglichkeit der Definition standardisierter Werkzeugkombinationen für Robotik-Projekte verschiedenen Zuschnitts,

---

<sup>32</sup> Vgl. Lock 1997.



**Abb. 3.11** Projektmanagement

- Definition und (automatisierte) Überwachung der Einhaltung zwingender Werkzeugfolgen.

## Literatur

- Amberg, M.: Prozessorientierte betriebliche Informationssysteme. Springer, Berlin (1999)
- Balzert, H.: Lehrbuch der Software-Technik Bd. 1, 2. Aufl. Spektrum Akad. Verlag, Heidelberg (2001) (1997, Bd. 2)
- Balzert, H.: Lehrbuch Grundlagen der Informatik. Spektrum Akad. Verlag, Heidelberg (1999)
- Balzert, H.: Lehrbuch der Objektmodellierung. Spektrum Akad. Verlag, Heidelberg (1999)
- Bayertz, K.: Wissenschaft als historischer Prozeß. Die antipositivistische Wende in der Wissenschaftstheorie. München (1980) Fink. München.
- Chalmers, A.F.: Wege der Wissenschaft. Einführung in die Wissenschaftstheorie. Berlin u. a. (1986) Springer
- Chen, P.: The entity-relationship model, toward a Ljnjified View of Dato, ACM Transactions on Database Systems, Bd 1, (1976)
- Coad, P., Yourdon, E.: Object-oriented analysis, 2. Aufl. Prentice-Hall, Englewood Cliffs, (1991)
- Date, C.J.: An introduction to database systems, 7. Aufl. Addison-Wesley, Reading (2000)
- Dörner, D.: Bauplan für eine Seele. Rowohlt, Hamburg (1999)
- Elmasri, R., Navathe, S.B.: Fundamentals of database systems. Addison-Wesley, Reading, 3. Aufl. (2000) (deutsch: Grundlagen von Datenbanksystemen; Pearson Studium, München, 2002)

- Frühauf, K., Ludevvig, J., Sandmayr, H.: Software-Projektmanagement und -Qualitätssicherung, Teubner. Stuttgart (1988)
- Humberto, M.: Erkennen: Die Organisation und Verkörperung von Wirklichkeit. Braunschweig (1985)
- Humberto, M., Varela, F.: Der Baum der Erkenntnis. Bern Goldmann
- Jacobson, I., Christerson, M., Jonsson, P., Overgaard, G.: Object-oriented software engineering: a use case driven approach. Addison-Wesley (1992)
- Kellner, H.: Die Kunst IT-Projekte zum Erfolg zuführen, 1. Aufl. Carl Hanser Verlag (2001)
- Kilberth, K., Gryczan, G., Züllighoven, H.: Objektorientierte Anwendungsentwicklung. Vieweg, Braunschweig (1993)
- Kruchten, P.: The rational unified process, an introduction. Addison-Wesley, Longman (1998)
- Kuhn, T.S.: Die Struktur wissenschaftlicher Revolutionen. Frankfurt a. M. (1976)
- Kuhn, T.S.: Die Entstehung des Neuen. Frankfurt a. M. (1977)
- Lakatos, I., Musgrave, A. (Hrsg.): Kritik und Erkenntnisfortschritt. Braunschweig (1974)
- Lakatos, I.: Die Methodologie der wissenschaftlichen Forschungsprogramme. Braunschweig (1982)
- Litke, H.D.: Projektmanagement, Methoden, Techniken, Verhaltensweisen. Carl Hanser Verlag (1995)
- Lock, D.: Projektmanagement. Ueberreuter (1997)
- Maier, H.: Software-Projekte erfolgreich managen. WRS Verlag (1990)
- Mandl, H., Spada, H. (Hrsg.) Wissenspsychologie. München (1988)
- Marco, T. De.: Der Termin. Ein Roman über Projektmanagement. Hanser, München (1998)
- Marco, T. De., Lister, T.: Wien wartet auf dich, Hanser, München, (1999) (2. aktualisierte und erweiterte Auflage)
- Ottmann, T., Widmayer, P.: Algorithmen und Datenstrukturen, 4. Aufl. Spektrum Akad. Verlag, Heidelberg (2002)
- Oyama, V.: The ontogeny of information. Cambridge University Press, Cambridge (1985)
- Raasch, J.: Systementwicklung mit Strukturierten Methoden, 3. Aufl. Hanser, München (1993)
- Royce, W.: Software project management. A unified framework. Addison-Wesley, Reading (1998)
- Saake, G., Sattler, K.-U.: Algorithmen und Datenstrukturen. dpunkt, Heidelberg (2002)
- Schelle, H.: Projekte zum Erfolg führen, Beck-Wirtschaftsberater (1996)
- Shiaer, S., Mellor, S.J.: Object lifecycles – modelling the world in states, Prentice-Hall, Englewood Cliffs, 1991. Deutsche Ausgabe: Objektorientierte Systemanalyse, Ein Modell der Welt in Daten. Haser, London, (1996)
- Versteegen, G.: Projektmanagement mit dem Rational Unified Process. Springer (2000)
- Vogt, C.: Betriebssysteme. Spektrum Akad. Verlag, Heidelberg (2001)
- Zimmerli, W., Wolf, S. (Hrsg.): Künstliche Intelligenz: Philosophische Probleme. Reclam, Stuttgart, (1994)
- Zimmermann, V.: Objektorientiertes Geschäftsprozessmanagement. Gabler, Wiesbaden (1999)

---

## 4.1 Architekturen

Als Architektur des Robotersystems bezeichnet man das Strukturprinzip der Komponenten, welche in ihrer Gesamtheit dafür verantwortlich sind, dass das Robotersystem die gestellten Probleme zu lösen in der Lage ist.

### 4.1.1 Steuerungs-Architektur

Als Architektur der Robotersteuerung bezeichnet man die Anordnung derjenigen Module und ihrer Verbindungen, welche in ihrer Gesamtheit dafür verantwortlich sind, dass die beweglichen Teile des Robotersystems die vom Entwickler vorgesehenen Aktionen ausführen. Gelegentlich unterscheidet man dann zwischen der Hardware- und Software, wenn man sich auf die Realisierung sensorischer bzw. aktorischer Fähigkeiten konzentriert. Die Definition und Einteilung der Module erfolgt – je nach Zielsetzung – mit unterschiedlicher Feinheit aus der Perspektive unterschiedlicher Funktionalitäten oder aus Sicht der einzelnen Baugruppen (Rechner, Kommunikationsschnittstellen, etc.). Allgemein unterteilt man bei der Robotersteuerung auch in symbolisch orientierte und verhaltensbasierte Architekturen:

- *Symbolisch orientierte Architekturen:* In der klassischen Robotik abstrahiert man häufig Details der Hardware und stellt die Bewegungssteuerung des Roboters nur noch als eine Komponente dar. Dafür wird Wert gelegt auf die Visualisierung des Informationsflusses zwischen den einzelnen, am Steuerungsprozess beteiligten Einheiten. Es existiert eine Vielzahl von Entwürfen für symbolisch orientierte Architekturen, wobei die Übergänge zwischen den einzelnen Entwürfen eher fließend sind.
- *Verhaltensbasierte Architekturen:* Während symbolisch orientierte Architekturen als eine Umsetzung eines funktionalistisch orientierten Modells angesehen werden kön-

nen, sind verhaltensbasierte Architekturen eher an handlungsorientierten Zusammenhängen interessiert.

Eine scharfe Unterscheidung wird angesichts der neueren Entwicklungen schwieriger, denn Realisierungen bedienen sich zusehends der Erkenntnisse aus beiden Bereichen.

Wesentliche Leitlinie beim verhaltensbasierten Entwurfsansatz ist die Beachtung der Erfordernisse, die aus einer engen Einbettung des Robotersystems in seine Umwelt folgen: Leiblichkeit, Situiertheit und hohe Adaptivität/Dynamik. Verhaltensbasierte Steuerungen sorgen deshalb für eine schnelle Reaktion des Roboters auf Umweltveränderungen, d. h. durch ihre Sensoren wahrgenommene externe Stimuli bzw. deren Veränderung. Damit empfindet ein Beobachter das Verhalten des Roboters als agil und lebendig.

So löst eine sichtbare Änderung der Umgebung, beispielsweise ein auf den Roboter zugehender Mensch, eine sofort sichtbare Reaktion in Form einer Ausweichbewegung des Robotersystems aus.

Um diese schnelle Reaktion zu erreichen, muss der Weg von der Wahrnehmung durch die Sensoren bis zur Aktivierung der Aktoren möglichst kurz sein. Die enge Kopplung zwischen Eingängen und Ausgängen bedingt den Verzicht auf algorithmisch aufwendige Berechnungen oder die semantische Interpretation von Sprach- oder Bildmustern. Insofern ist es nicht verwunderlich, dass rein verhaltensbasierte Programme üblicherweise einfache mobile Roboter steuern. Hier lassen sich selbst mit einfachen Produktionsregeln bereits interessante Verhaltensmuster erzeugen. In wenigen Rechenschritten wird aus einem sensorisch erfassten Stimulus – etwa die mit einem Abstandssensor aufgenommene Distanz zu einem Objekt – eine Stellgröße für die Aktoren – etwa die Umdrehungsgeschwindigkeit der antreibenden Räder – berechnet. Die direkte Kopplung eines Abstandssensors an den Geschwindigkeitsberechner der Räder bewirkt bei richtiger Programmierung ein Fernhalten von allen Hindernissen. Dies funktioniert nicht nur bei statischen Hindernissen, wie beispielsweise Wänden, sondern auch, wenn sich jemand aktiv dem Roboter nähert. Je weniger Rechenschritte für die Umsetzung der von einem Sensor aufgenommenen Daten in eine an die Aktoren zu übergebende Stellgröße notwendig sind, desto schneller reagiert das Robotersystem auf eine Veränderung des sensorischen Eingangs.

Insofern lassen sich durchaus konzeptionelle Unterschiede zwischen einer symbolisch orientierten und einer verhaltensbasierten Steuerungsarchitektur ausmachen. Bei symbolischer Verarbeitung werden Sensoreingaben zunächst in Bezug auf ein Umweltmodell interpretiert, um danach einen Plan für die durch die Aktoren auszuführende Handlung zu erzeugen. Das System versucht, unter Umständen konfliktäre Ziele möglichst optimal in Deckung zu bringen.

Beispielsweise mögliche Gegensätze zwischen Energiesuche, Exploration und notwendiger Ausweichbewegung.

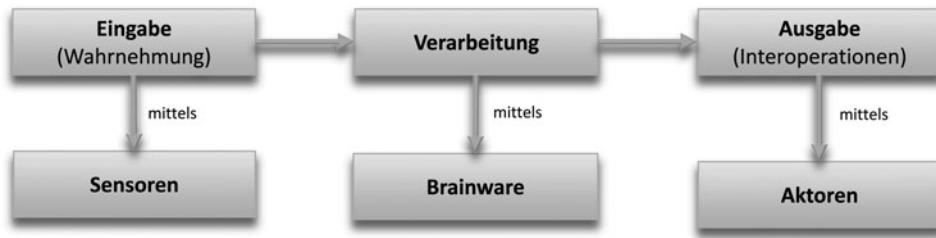
Beim verhaltensbasierten Ansatz verzichtet man auf dieses sequentielle Vorgehen, insbesondere den symbolischen Planungsvorgang. Stattdessen sieht man in der Steuerung Komponenten für verschiedene Kompetenzen, die zur Laufzeit des Robotersystems alle parallel mit Stimuli versorgt werden und ständig Ausgaben erzeugen. Diese Ausgaben werden dann in bestimmter Weise interpretiert und kombiniert, um die Akteure mit einem eindeutigen Signal (Stellgröße) zu versorgen. Die Kombination kann eher Mittelung zwischen den Ausgaben der einzelnen Komponenten sein (kooperierende Kopplung), oder aber auch nur die Aufschaltung der Ausgabegrößen eines Verhaltens auf die Akteure zu einem Zeitpunkt (kompetitive Kopplung). Die Anteile der Einzelverhaltensweisen bei der kooperativen Kombination am Gesamtverhalten können dabei gleitend modifiziert werden. Die Modifikation der Ausgaben einer Komponente wird dabei von allen anderen Komponenten oder einer Teilmenge davon gesteuert. Eine solche Beeinflussung ist prinzipiell natürlich auch eingangsseitig möglich, d. h. es wird die Gewichtung eines bestimmten Stimulus für das Auslösen eines Verhaltens beeinflusst.

Die strukturverhaltensbasierte Steuerung folgt idealtypisch einem streng geschichteten hierarchischen Ansatz: komplexes Verhalten auf einer höheren Ebene steuert ein oder mehrere darunterliegende Ebenen. Ein komplexes Verhalten fasst in diesem Sinne eine ganze Reihe elementarer Verhalten zusammen. Diese Hierarchisierung legt auch den inkrementellen Entwurfsprozess nahe: Begonnen wird mit der Spezifikation und Implementierung der niedrigsten Ebene, in gewisser Weise analog zur stammesgeschichtlichen Entwicklung von Lebewesen. Erst wenn ein bestimmtes Verhalten auf einer Ebene getestet ist und unter allen Umweltbedingungen robust funktioniert, werden die darauf aufbauenden Schichten entwickelt und implementiert. Normalerweise beeinflussen nur die höheren Schichten die darunterliegenden. Die nach dem verhaltensbasierten Paradigma erstellten Steuerungsprogramme zeichnen sich dadurch aus, dass sie als robust gegenüber Änderungen und Erweiterungen der Fähigkeiten des Robotersystems gelten.

### 4.1.2 Eingabe-Verarbeitung-Ausgabe-Architektur

In der klassischen Literatur bzw. der bisherigen Auffassung unterscheidet man in Hardware- und Software-Architektur. Dieser Abschnitt geht bewusst über diesen Ansatz hinaus und stellt die zentralen architektonischen Komponenten, deren Beziehungen zueinander, sowie die unterschiedlichen Entwicklungsmöglichkeiten so vor, dass am Ende die Entwicklung kognitiver und damit intelligenter Robotersysteme möglich wird.

Als basaler Ausgangspunkt der folgenden Überlegungen dient eine Sichtweise, die das Robotersystem zunächst als eine Eingabe-Verarbeitung- und Ausgabeeinheit auffasst. Ausgehend von dieser Minimalansicht in Form eines Black-Box-Systems wird die Architektur eines Robotersystems sukzessive hin zu einem kognitiven, auf einer Agentenarchitektur basierenden Robotersystems entwickelt, welches sich dann durch einen hohen systemischen Intelligenzquotienten auszeichnen wird. Bei diesem Black-Box Ansatz erhält ein Robotersystem eine Menge von Eingaben, die es über eine entsprechende Wahrnehmungs-



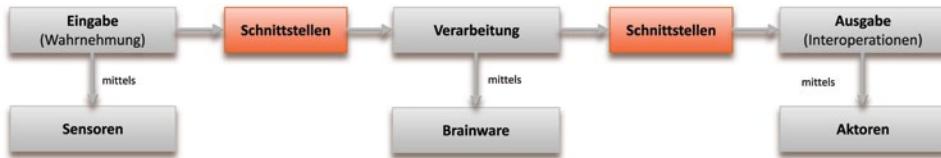
**Abb. 4.1** Eingabe-Verarbeitung-Ausgabe-Prinzip

komponente in Form von Sensoren aufnimmt. Es verarbeitet diese Eingaben und erzeugt eine Ausgabe über Aktoren, die üblicherweise Interoperationen auslösen (Abb. 4.1).

Im Unterschied zum klassischen EVA (Eingabe-Verarbeitung-Ausgabe) Prinzip der Datenverarbeitung muss ein Robotersystem über Verarbeitungsmechanismen verfügen, die das System mit einer entsprechenden Intelligenz ausstatten. Denn nur mit Hilfe einer solchen intrinsischen Intelligenz, lokalisiert in der *Brainware*, kann es den wesentlichen Charakteristika eines intelligenten Robotersystems, wie Autonomie, Kooperation oder Proaktivität, gerecht werden und sich von herkömmlichen Robotersystemen unterscheiden. Die bisherige Repräsentation als Black-Box wird allen wissenschaftlichen Disziplinen gerecht, da das Modell allgemein genug gehalten ist, um die speziellen Anforderungen aller zu erfüllen. Aus architektonischer Sicht bietet es aber nur sehr rudimentäre Informationen bezüglich des konkreten inneren Aufbaus eines intelligenten Robotersystems.

Insofern geht die folgende Abbildung einen Schritt weiter und macht die in einem Robotersystem ablaufenden Arbeitsprozesse in einer Form deutlich, die später als erster Anhaltspunkt für die Entwicklung konkreter aufgabenspezifischer Komponenten genutzt werden können. Zur Kommunikation und Kooperation mit seiner Umwelt besitzt ein Robotersystem ein oder mehrere Sensoren oder Aktoren. Die Umwelt eines Robotersystems kann aus anderen Robotersystemen, Lebewesen oder beliebigen Informationsquellen bestehen. In der Regel steht für jeden Typ eines Umweltobjektes eine spezielle Komponente zur Verfügung, die speziell auf die Fähigkeiten und Besonderheiten des jeweiligen Interoperationspartners angepasst ist. Über die *Sensoren* nimmt das Robotersystem zum einen Informationen und Änderungen innerhalb seiner Umwelt wahr, löst aber zum anderen auch seine eigenen Aktionen durch *Aktoren* aus. Die Sensoren und Aktoren stellen also sowohl eine Eingabe- als auch eine Ausgabeschnittstelle dar. Es ist nicht sinnvoll, unbehandelte Sensordaten direkt zu verarbeiten. Stattdessen sind Abstraktionsmethoden implementiert, die die wesentlichen Merkmale der Realität beibehalten, aber die Rauschdaten eliminieren (Abb. 4.2).

Die zentrale Aufgabe der meisten mobilen Robotersysteme besteht darin, mit der Umwelt zu interopernieren und dabei die wahr- bzw. aufgenommenen Informationen zu verarbeiten, zu interpretieren und zur Verfolgung der eigenen Ziele in Form von Wissen in Interoperationen umzusetzen. Zu diesem Zweck müssen in einem ersten Schritt alle ein-



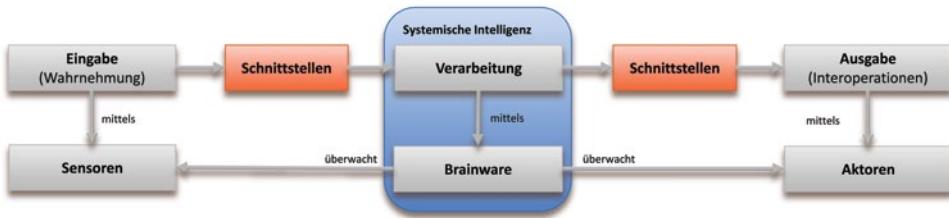
**Abb. 4.2** Eingabe-Verarbeitung-Ausgabe-Prinzip mit Schnittstellen

gehenden Informationen auf sinnvolle Art und Weise integriert und in die Wissensbasis des Robotersystems aufgenommen werden. Dieser Prozess wird als *Wissensgenerierung* bezeichnet. Besondere Bedeutung kommt dabei der Informationsverarbeitung zu, wenn beispielsweise Wahrnehmungen aus verschiedenen Sensoren oder Aktoren eintreffen, die unter Umständen widersprüchlich sind oder in unterschiedlichen Repräsentationsformen vorliegen.

Beispielsweise können sich die durch ein Robotersystem  $R_A$  gelieferten Informationen von denen eines anderen Robotersystems  $R_B$  sowohl formell als auch inhaltlich unterscheiden, obwohl sie sich unter Umständen auf dieselbe Thematik beziehen. Im Rahmen der intelligenten Informationsverschmelzungsverarbeitung und der daran anschließenden Wissensgenerierung müssen derartige Inkonsistenzen erkannt und vor der eigentlichen Ausführung der Interoperationen beseitigt werden.

Insofern erfolgt innerhalb des Brainware-Systems eine Transformation von Daten in Informationen und von letzteren wiederum in Wissen statt. *Daten* sind alle in gedruckter, gespeicherter, visueller, akustischer oder sonstiger Form verwertbare Angaben über die verschiedensten Dinge und Sachverhalte. Daten bestehen aus beliebigen Zeichen-, Signal- oder Reizfolgen und sind objektiv wahrnehmbar und verwertbar. *Informationen* sind diejenigen Daten, die das einzelne Robotersystem verwerten kann. Informationen sind also im Gegensatz zu Daten nur systemisch-subjektiv wahrnehmbar und auch nur systemisch-subjektiv verwertbar. Informationen sind daher immer empfängerorientiert. Sie stellen eine in sich abgeschlossene Einheit dar. Dabei sind Informationen zwar aus Daten zusammengesetzt, sie bilden aber durch ihren für den Empfänger relevanten Aussagegehalt eine höhere Ordnung im Vergleich zu Daten ab. Informationen sind sozusagen Daten in einem bestimmten Kontext. *Wissen* entsteht durch die Verarbeitung und Verankerung wahrgenommener Informationen. In diesem Fall spricht man vom Prozess des systemischen Lernens. Altes, bereits gespeichertes Wissen ist dabei der Anker, um aus neu aufgenommenen Informationen neues Wissen in der Struktur der Brainware zu vernetzen. Wissen stellt das Endprodukt des Lernprozesses dar, in dem Daten als Informationen wahrgenommen und als neues Wissen generiert und damit gelernt werden. Wissen ist somit die Summe aller verstandenen Informationen.

Sind die neuen externen Daten durch das Robotersystem aufgenommen, können diese in einem nächsten Schritt verarbeitet werden. Der Verarbeitungsprozess bildet die zent-



**Abb. 4.3** Eingabe-Verarbeitungs-Ausgabe-Prinzip mit Intelligenz

rale Komponente eines Robotersystems, da sich in ihm die eigentliche Funktionalität des Robotersystems widerspiegelt. Ziel der intelligenten Informationsverarbeitung ist es, die vorhandenen Daten zu Informationen zu transformieren, diese dann zu Wissen zu generieren, um dann konkrete Interoperationspläne zu entwickeln. Da jedes Robotersystem ein bestimmtes Ziel verfolgt, müssen im Rahmen der Transformation die Auswirkungen neuer Umweltsituationen auf die internen Ziele just-in-time berücksichtigt werden. Sind Auswirkungen erkennbar, ergibt sich für das Robotersystem ein konkreter Interoperationsbedarf. Die neue Umweltsituation bietet dem Robotersystem dabei entweder die Möglichkeit, seinem Ziel einen Schritt näher zu kommen oder konfrontiert ihn mit einem Problem, welches der Erreichung seines Ziels im Wege steht und das es aus diesem Grund im Rahmen eines Konfliktmanagements zu lösen gilt. Das Robotersystem kann seine Erkenntnisse in Form eines Planes spezifizieren, der konkrete Interoperationen zur Reaktion auf die neue Umweltsituation enthält. Allerdings ist dies nicht zwingend erforderlich, das heißt, eine Reaktion eines Robotersystems kann auch ohne vorherige Planung in Form einer Spontanreaktion geschehen. Die vom Robotersystem für sinnvoll erachteten Interoperationen werden an die Aktoren übergeben und von diesen ausgeführt. Die Überwachung der Ausführung fällt ebenfalls in den Aufgabenbereich der Brainware (Abb. 4.3).

Nicht alle Interoperationen eines Robotersystems müssen zwangsläufig die Reaktion auf neue Umweltsituationen darstellen. Vielmehr kann ein intelligentes Robotersystem auch proaktiv handeln und selbstständig neue Pläne erstellen. Voraussetzung hierzu ist eine interne Repräsentation der Umwelt. Intelligente Robotersysteme der ersten Generation bedienen sich zunächst der Erkenntnisse der Forschungsbereiche der Künstlichen Intelligenz (Artificial Intelligence, AI) und des Künstlichen Lebens (Artificial Life, AL), indem sie ein explizites symbolisches Modell der Umwelt und die Fähigkeit zur logischen Schlussfolgerung als Grundlage für intelligentes Handeln voraussetzen. Die Modellierung der Umwelt geschieht dabei in der Regel vorab und bildet die wesentliche Komponente der Wissensbasis eines Robotersystems. Auf Grund der extrem hohen Komplexität derartiger Repräsentationen sind Robotersysteme der ersten Generation nur bedingt für den Einsatz in dynamischen Umgebungen geeignet. Sie sind nur schwer in der Lage, während ihrer Ausführung neue Informationen oder Erkenntnisse bezüglich ihrer Umwelt in ihr bestehendes Umweltmodell einzufügen, da ihnen dazu in der Regel das notwendige Wissen und die notwendigen Ressourcen fehlen. Dennoch verfügen die intelligenten Robotersysteme der ersten Generation über die Fähigkeit zur logischen Schlussfolgerung. Im Rahmen des



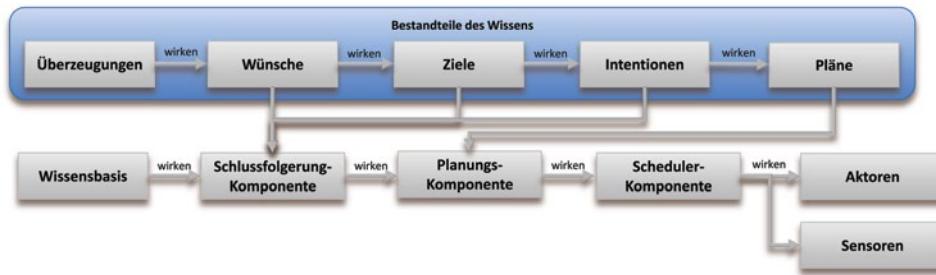
**Abb. 4.4** Wissensbestandteile und Pläne

Schlussfolgerungsprozesses verwendet das Robotersystem das in seinem Umweltmodell enthaltene Wissen, um seinen internen, artifiziell-mentalnen Zustand zu modifizieren. Dieser setzt sich aus den folgenden Faktoren zusammen:

- *Überzeugungen* enthalten die grundlegenden Ansichten eines Robotersystems bezüglich seiner Umwelt. Mit ihrer Hilfe drückt das Robotersystem insbesondere seine Erwartungen über mögliche zukünftige Umweltzustände aus.
- *Wünsche* leiten sich direkt aus den Überzeugungen ab. Sie beinhalten die Beurteilungen zukünftiger Umweltsituationen aus der Sicht des Robotersystems. Ein Robotersystem kann beispielsweise den Wunsch besitzen, dass ein bestimmter, in seinen Überzeugungen enthaltener, zukünftiger Umweltzustand eintritt und ein anderer nicht. Durch die Formulierung von Wünschen trifft ein Robotersystem noch keine Aussagen darüber, inwieweit diese überhaupt realistisch sind. So kann ein Robotersystem durchaus einen unrealistischen Wunsch besitzen, obwohl er weiß, dass er diesen sehr wahrscheinlich niemals erfüllen kann. Auch miteinander in Konflikt stehende oder nicht miteinander vereinbare Wünsche sind möglich.
- *Ziele* stellen diejenige Untermenge der Wünsche eines Robotersystems dar, an deren Erfüllung es prinzipiell arbeiten könnte. Im Gegensatz zu seinen Wünschen sollten die Ziele eines Robotersystems daher realistisch gesteckt sein und auch nicht in Konflikt zueinander stehen. Die Ziele bilden den potentiellen Interoperationsspielraum eines Robotersystems, da sie die zur Verfügung stehenden Handlungsalternativen zu einem bestimmten Zeitpunkt darstellen.
- *Intentionen* wiederum sind eine Untermenge der Ziele. Beschließt ein Robotersystem, ein bestimmtes Ziel zu verfolgen, so wird aus dem Ziel eine Intention. In der Regel kann ein Robotersystem nicht gleichzeitig alle Ziele verfolgen, da es nicht über die hierfür notwendigen Ressourcen verfügt. Es muss daher die anstehenden Ziele priorisieren und ihrer Wichtigkeit nach angehen.
- *Pläne* fassen die Intentionen eines Robotersystems zu konsistenten Einheiten zusammen. Dabei besteht ein enger Zusammenhang zwischen Intentionen und Planen: Intentionen bilden Teilpläne des Gesamtplans eines Robotersystems und die Menge aller Pläne spiegelt wiederum die Intentionen eines Robotersystems wider.

Insofern verfügen intelligente Robotersysteme der ersten Generation über die Fähigkeit, über eine Faktorendichtung aus Wissen gewisse Pläne ableiten zu können (Abb. 4.4).

Eine solch geforderte Fähigkeit hat Einfluss auf die architektonische Ausgestaltung eines intelligenten Robotersystems.



**Abb. 4.5** Wissensbestandteile und Komponenten

Die *Wissensbasis* des Robotersystems enthält vor allem das symbolische Umweltmodell. Aus diesem werden Wünsche, Ziele und Intentionen abgeleitet, eine Aufgabe, bei welcher der *Schlussfolgerungskomponente* eine zentrale Bedeutung zukommt. Die Intentionen werden von der *Planungskomponente* übernommen und zu einem konsistenten Gesamtplan zusammengestellt. Dabei handelt es sich um einen dynamischen, inkrementellen Prozess. Die Planungskomponente untersucht neue Intentionen auf Abhängigkeiten zu bestehenden Plänen. Beispielsweise können die Ergebnisse einer Intention die Eingabewerte einer anderen Intention darstellen. Abhängigkeiten dieser Art werden durch die Planungskomponente erkannt und entsprechend berücksichtigt. Bestehende Pläne werden kontinuierlich den durch das Eintreffen neuer Intentionen entstehenden Situationen angepasst.

Die *Schedulerkomponente* erhält von der Planungskomponente die aktuellen Pläne. Jeder Plan besteht aus einer Vielzahl von einzelnen Interoperationen, die sequentiell oder parallel bearbeitet werden müssen. Die Schedulerkomponente muss die Entscheidung treffen, wann welche Interoperationen konkret zur Ausführung übergeben werden. Sie benötigt zu diesem Zweck einen ständigen Überblick über die dem Robotersystem zur Verfügung stehenden Ressourcen. Jeder Interoperation werden von der Schedulerkomponente ein optimaler und ein spätester Ausführungszeitpunkt zugeordnet. Auch Angaben zu maximaler Laufzeit und Ressourcenverbrauch werden von der Schedulerkomponente vorgegeben.

Mit diesen Zusatzinformationen wird die Interoperation an die Aktoren übergeben. Diese Komponente führt die nächste anstehende Interoperation aus, überwacht ihren fehlerfreien Ablauf und beendet ihre Ausführung. Benötigt eine Interoperation mehr Rechenzeit, als ihr vom Scheduler gestattet wurde, kann der Aktor sie abbrechen. Ist der Aktor nicht in der Lage, die Interoperation bis zu dem spätestens vorgegebenen Zeitpunkt zu starten, gibt er sie an die Schedulerkomponente oder Planungskomponente zurück (Abb. 4.5).

Die obige Darstellung macht deutlich, dass eine dynamische Modifikation des internen Umweltmodells nur in sehr geringem Umfang möglich ist. Zwar kann ein Robotersystem der ersten Generation über bidirektionale Sensoren verfügen, diese werden aber nur selten für die Erweiterung der Wissensbasis genutzt. Die Interoperation mit anderen Robotersystemen bezieht sich in erster Linie auf reine Kommunikation beziehungsweise Kooperation.

Insofern ziehen die Robotersysteme der ersten Generation auf Grund ihrer Komplexität eine Vielzahl von Problemen nach sich. Die zentralen Probleme der klassischen KI spiegeln sich nahezu unverändert auch beim Einsatz dieser Generation von Robotersystemen wieder. Der Hauptkritikpunkt setzt bei ihrer starren Struktur an. Intelligente Robotersysteme bewegen sich innerhalb sehr dynamischer Umgebungen. Entsprechend sollten sie auch in der Lage sein, ihre Entscheidungsgrundlagen möglichst auf Basis der aktuellen Umweltsituation zu treffen. Genau dies ist aber bei den intelligenten Robotersystemen der ersten Generation nur sehr eingeschränkt der Fall. Ihre Intentionen und Pläne bauen auf dem symbolischen Umweltmodell auf, dass zu einem bestimmten Zeitpunkt der Vergangenheit entworfen und danach nur noch minimal aktualisiert wurde. Die relativ starre Struktur planbasierter Systeme verstärkt diesen Nachteil. Häufig hat sich zum Zeitpunkt der Ausführung eines Planes die Umweltsituation bereits mehr oder weniger deutlich geändert, da der Übergang von Intentionen zur Planer-, Scheduler- und Aktoren bzw. Sensoren sehr zeitintensiv ist. Die symbolischen Algorithmen solcher Robotersysteme sind in der Regel auf die Erzielung optimaler, nachweislich korrekter Ergebnisse ausgelegt, was zwangsläufig zu einem hohen Grad an Komplexität führt. In dynamischen Umgebungen ist eine schnelle Reaktion mit einem für den jeweiligen situativen Kontext qualitativ ausreichenden Ergebnis oft sinnvoller, als das Streben nach optimalen Plänen. Solche Robotersysteme sind daher oftmals darauf programmiert, dass der mathematisch nachweisbaren Korrektheit eines Planes Vorrang vor der Effizienz des Planungsprozesses eingeräumt wird.

Diese Überlegungen führen nun zu einem dieser Generation diametral entgegengesetzten Typus, den sogenannten interoperationsbasierten Robotersystemen. Interoperationsbasierte Robotersysteme besitzen kein internes symbolisches Modell ihrer Umwelt. Auch auf die Fähigkeit, komplexe Schlussfolgerungsprozesse zu durchlaufen, wird weitestgehend verzichtet. Der Grund für diese bewusst in Kauf genommenen Restriktionen liegt in der Erzeugung kompakter, fehlertoleranter und vor allem flexibler Robotersysteme. Diese zweite Generation von Robotersystemen entwickelt ihre Intelligenz nicht wie die der ersten Generation aus internen Modellen und Repräsentationen, sondern durch die unmittelbare Interoperation mit ihrer Umwelt. Dementsprechend hoch ist der dem Interoperationsprozess zugeordnete Stellenwert. Insofern kann man davon ausgehen, dass die systemisch-intrinsische Intelligenz des Robotersystems durch die Interoperation mit der Umwelt beeinflusst wird. Nur durch die kontinuierliche Interoperation von Systemen entsteht und vergrößert sich diese systemisch-intrinsische Intelligenz. Ein solches Robotersystem muss nicht zwangsläufig eine komplexe Struktur besitzen, um innerhalb einer komplexen Umwelt interoperieren zu können. Es reicht aus, die Umwelt genau zu beobachten und eine Reihe einfacher Grundsätze oder Abhängigkeiten zu erkennen. Diese Erkenntnisse werden genutzt, um aufgabenspezifische *Wissensobjekte* zu entwickeln, die in der Lage sind, ihre Umwelt kontinuierlich auf das Auftreten bestimmter Situationen zu überprüfen und beim Eintritt einer derartigen Situation eine direkte Reaktion auszulösen. Durch die Interoperation dieser Wissensobjekte entsteht die Kompetenz des Systems, die die Höhe des systemischen Intelligenzquotienten des Robotersystems bestimmt.

Sensoren nehmen Daten auf, leiten diese an aufgabenspezifische Verarbeitungskomponenten, die dann die erzeugten Informationen an das Wissensmanagementsystem weiterleiten und erzeugen so eine Reaktion der einzelnen Wissenstypen, welche wiederum unter Zuhilfenahme von Akteuren auf die Umwelt übertragen werden. Ein Robotersystem besitzt eine Vielzahl unterschiedlicher Sensoren, die es ihm ermöglichen, seine Umwelt systemisch zu beobachten.

Es kann beispielsweise feststellen, ob er auf ein Hindernis getroffen ist, ob Gegenstände ihre Position geändert haben, oder ob andere Robotersysteme in seiner Umgebung aktiv sind.

Diese Informationen werden an die Wissenstypen des Robotersystems übergeben.

Ein Wissenstyp mit der Aufgabe, die Bewegungsrichtung des Roboters zu ändern, wird zum Beispiel immer dann aktiv, wenn ein Sensor ein Hindernis in der momentanen Fahrtrichtung feststellt. Wenn es das Ziel des Bewegungs-Wissenstyps ist, Hindernisse zu umgehen, muss es in diesem Beispiel die Fahrtrichtung des Robotersystems ändern. Die hierzu notwendigen Akteuren sind in diesem Fall die Vorderräder, deren Ausrichtung durch das Wissenstyp verändert wird. Andere Wissenstypen können sich mit der Aufgabe, bestimmte Objekte zu bewegen, einen Weg zu einer definierten Stelle zu finden oder Hindernisse zu beseitigen, beschäftigen.

Allgemein lässt sich feststellen, dass die Sensoren eines Robotersystems es diesem ermöglichen, Informationen über seine Umwelt aufzunehmen und das Auftreten veränderter Umweltsituationen zu erkennen. Die konkrete Gestaltung der Sensoren und auch der Akteure hängt dabei sehr stark von den durch den Sensor zu überwachenden Objekten ab. So können sich Sensoren zur Informationsaufnahme von menschlichen Personen zum Beispiel auf die Sprach- oder Schrifterkennung konzentrieren, während Sensoren, die andere Robotersysteme beobachten, sich für ihre Arbeit einfacher Kommunikationsverfahren bedienen.

Die durch einen Sensor gesammelten Daten und die von den verarbeitenden Komponenten erzeugten Informationen werden an das entsprechende Wissenstyp übergeben. In der Regel findet die Übergabe im Rohformat statt, das heißt, es werden keine höheren Kommunikationssprachen oder symbolische Repräsentationen verwendet. Jedes Wissenstyp ist für einen klar definierten, nicht sehr komplexen Aufgabenbereich zuständig. Alle für die Erfüllung seiner Aufgaben notwendigen Eigenschaften sind innerhalb des Wissenstyps zusammengefasst. Es gibt keine zentralen Komponenten, wie die Planungskomponente oder die Schlussfolgerungskomponente. Jedes Wissenstyp muss alle zur Bearbeitung seiner Aufgaben notwendigen Fähigkeiten besitzen. Eine Konsequenz dieser architektonischen Gestaltung besteht darin, dass ein mit Wissenstypen arbeitendes Robotersystem keine generelle, allgemein einsetzbare Funktionalität besitzt. Ein Wissenstyp hat eine exakt spezifizierte Aufgabe und erarbeitet eine exakt spezifizierte Lösung.

Existiert für eine bestimmte Aufgabe kein Wissensobjekt, so kann ein Robotersystem diese nicht lösen.

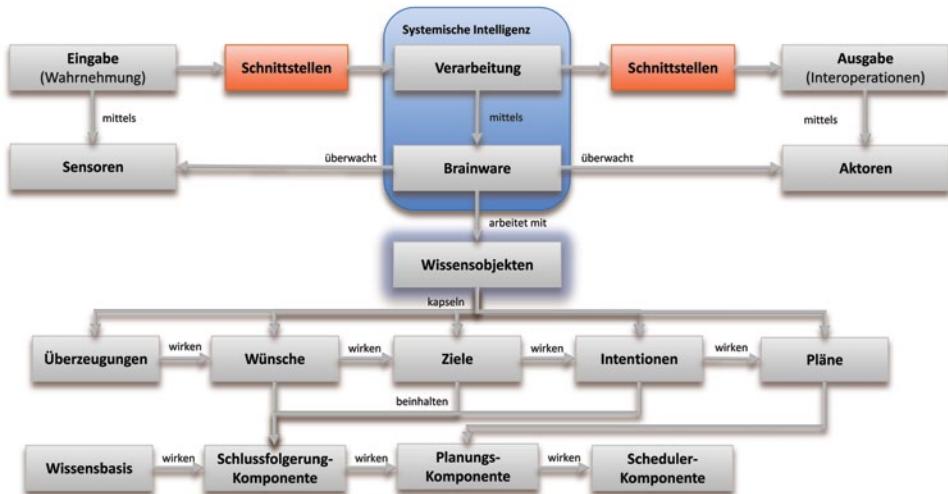
Die Arbeit der Wissensobjekte findet parallel zueinander statt, wobei in der Regel vielfältige Abhängigkeiten bestehen. Zu diesem Zweck können Wissensobjekte sowohl direkt untereinander, als auch über ihre Umwelt miteinander kommunizieren. Bei der direkten Kommunikation handelt es sich um eine Eins-zu-Eins Beziehung zwischen zwei Wissensobjekten und nicht um eine Kommunikation nach dem Broadcast-Prinzip. Die Kommunikation basiert auf einfachen Kommunikationsmechanismen und nicht auf einer komplexen Sprache. Der Vorteil der direkten Kommunikation liegt vor allem in der schnellen Reaktionsfähigkeit. Zwischen zwei Wissensobjekten existieren keine zwischengeschalteten Module, das heißt, ein Kommunikationsaufruf wird ohne Verzögerung vom empfangenden Wissensobjekt entgegengenommen und verarbeitet. Der zweite Fall, die Kommunikation über die Umwelt, vollzieht sich, indem ein Wissensobjekt eine Änderung innerhalb seiner Umwelt hervorruft, die wiederum von einem anderen Wissensobjekt beobachtet wird und dieses zu einer Reaktion veranlasst. Diese Variante ist zwar langsamer als die direkte Kommunikation, ermöglicht aber die Reaktion auf komplexere Umweltsituationen.

Die dezentrale und vernetzte Struktur der Wissensobjekte erhöht die Fehlertoleranz und Robustheit eines solchen Robotersystems. Fällt ein Wissensobjekt aus oder arbeitet es fehlerhaft, kann das Robotersystem mit großer Wahrscheinlichkeit einen Großteil seiner Aufgaben weiterhin erfüllen. Der Ausfall einer zentralen Komponente eines Robotersystems der 1. Generation führt hingegen fast immer zum Ausfall des gesamten Systems.

Die konsequente Fortführung der bisherigen Überlegungen führt zwangsläufig zur Entwicklung von Systemen, die die Vorteile beider Ansätze zu nutzen und innerhalb einer einheitlichen architektonischen Plattform zu integrieren versuchen. Derartige Systeme werden als kognitive Robotersysteme der 3. Generation in die Geschichte der Robotik eingehen (Abb. 4.6).

### 4.1.3 Agenten-Architektur

Eine Architektur dient dazu, entweder die Verwendung zu erklären (konzeptionelle Abstraktion) oder die Realisierung zu dokumentieren (Implementierungsabstraktion). In diesem Abschnitt steht die konzeptionelle Abstraktion im Vordergrund, indem die Organisation der Komponenten eines Robotersystems als kognitives System, dessen Verhalten sowie die Beziehungen der einzelnen Komponenten zueinander zunächst beschrieben werden. Auf dieser Beschreibung aufbauend, werden in der späteren Implementierungsabstraktion die statischen und dynamischen Anteile des Systems und die damit intendierten Funktionalitäten abgebildet. Die Architektur wird dabei graphisch dargestellt, wobei die Komponenten des Systems in der bewährten Notation als Kästchen dargestellt werden, die den Namen der Komponenten tragen, und der Daten-, Informations- und/oder Wis-

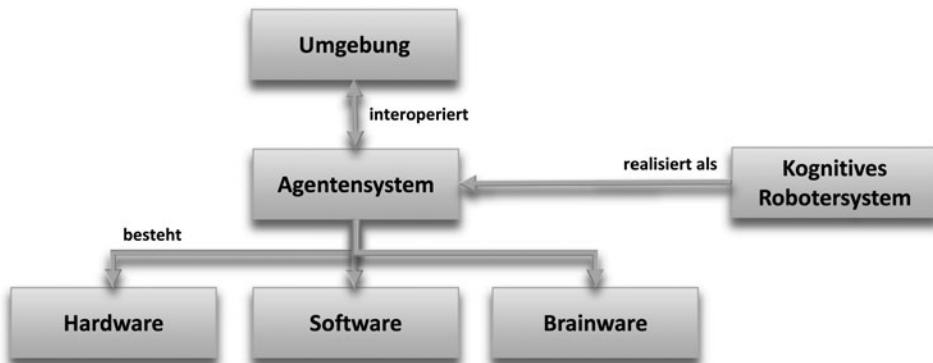


**Abb. 4.6** Wissensobjekte

sensfluss zwischen ihnen in Form von gerichteten Kanten angegeben wird. An dieser Stelle des Buches werden die Komponenten noch als *Black-Box* betrachtet, so dass man sich auf die Ein-/Ausgabebeziehungen und den Daten-, Informations- und/oder Wissensfluss zwischen den Komponenten konzentrieren kann.

Aus dieser Perspektive sieht die Basisarchitektur ein abstrahiertes Modell eines Agenten als Trägersystem vor, das aus Software und Brainware, beziehungsweise im Falle der Robotik zusätzlich aus Hardware besteht. Die Architektur trägt dem variablen Kommunikationsfluss zwischen den einzelnen Komponenten Rechnung und zeigt daher das Bild eines *Integrationsmodells*. Hierbei wird auf eine Kapselung von Funktionen in Komponenten geachtet, wobei sämtliche Komponenten beliebig miteinander kombinierbar sind. Letzteres lässt auch den Einbezug anderer Architekturansätze zu.

So sind bei einem hierarchischen Modell die Komponenten hinsichtlich des Kommunikationsflusses in einer Aufrufhierarchie anzugeben. Dies entspricht häufig auch der Dekomposition der zu lösenden Aufgaben des Systems in entsprechende Teilaufgaben. Im heterarchischen Modell kann jede Komponente jede beliebige andere Komponente aufrufen. Es gibt keinen festgelegten Pfad, der durchlaufen wird. Dieses Modell wird vor allem bei verteilten Agentenanwendungen in unterschiedlicher Weise angestrebt und realisiert. Auch lassen sich die Komponenten um eine globale Daten-, Informations- und Wissensstruktur in Form eines Blackboards gruppieren, worüber dann die Kommunikation erfolgen kann. Auch der Einbezug eines Kaskadenmodells ist möglich, wo alle Komponenten in einem Vektor angeordnet sind. Benachbarte Komponenten sind durch einen gemeinsamen Ein-/Ausgabepuffer verbunden, über den die Ergebnisse der einen Komponente zur Eingabe des anderen Moduls geleitet werden. Durch die Kombination dieser Architekturmödelle sind hierzu alternative Architekturen möglich.



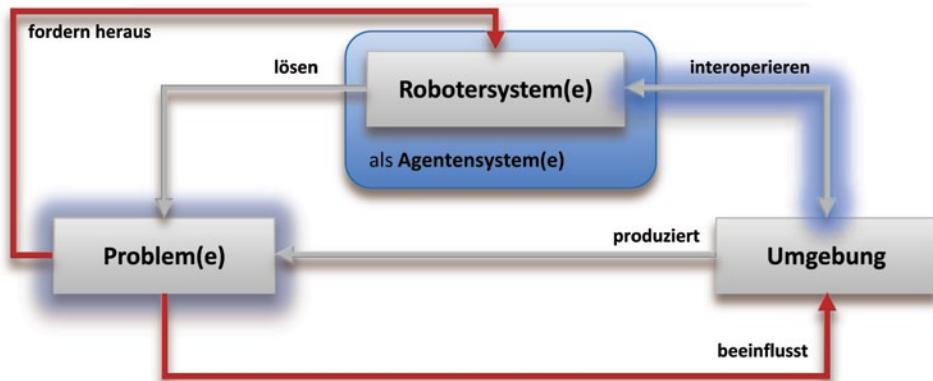
**Abb. 4.7** Architektur eines kognitiven Robotersystems

Ein wichtiges Kriterium bezüglich der architektonischen Ausgestaltung eines Robotersystems ist die Festlegung, ob die Komponenten seriell oder parallel zueinander arbeiten können. Aber auch die Festlegung der benötigten Daten-, Informations- und Wissensstrukturen sowie deren Bezugsquellen und die verwendeten Repräsentationsformalismen beeinflussen die Architektur. Die dabei zur Anwendung gelangenden Verarbeitungsprozesse und -techniken beeinflussen den Kommunikationsfluss zwischen den Komponenten und somit die Architektur des Gesamtsystems (Abb. 4.7).

Bei der Konzeptionalisierung der Komponenten muss entschieden werden, ob diese redundanzfrei zueinander sein sollen oder nicht. In der Regel erfordert eine größere Robustheit gegenüber Fehlern in den Eingaben oder Ausfällen einer Komponente eine größere Redundanz. Dies trifft speziell für solche Systeme zu, in denen sensomotorische Daten in Realzeit interpretiert und verarbeitet werden müssen. Insbesondere bei Systemen, die in unterschiedlichen Anwendungskontexten und/oder in unterschiedlichen Problemdomänen eingesetzt werden sollen, muss die Architektur eine Flexibilisierung der Komponentenzusammenstellung, eine Orchestrierung nach dem Baukastenprinzip unterstützen. Dadurch wird ein System flexibel konfigurierbar und die Architektur muss nur die Komponenten zur Verfügung stellen, die für die gewünschte Funktionalität unbedingt benötigt werden.

Auch der Tatsache, dass oftmals zu Beginn eines Entwicklungsprojektes nicht exakt gewusst wird, wie das Zielsystem realisiert werden kann und somit ein eher explorativer und damit iterativer Entwicklungsprozess zu verfolgen ist, muss die Architektur Rechnung tragen.

Mit Hilfe von wiederverwendbaren Prototypen im Rahmen eines inkrementellen Vorgehensmodells kann man versuchen, einen funktionellen und funktionierenden Nukleus des Zielsystems inkrementell zu realisieren. Insbesondere wenn ein System als operationales Modell einer Hypothese oder Theorie von kognitiven Fähigkeiten interpretiert werden



**Abb. 4.8** Kognitive Robotersysteme als Agentensysteme

soll, muss die kognitive Adäquatheit der Architektur überprüft werden. Außerdem gilt es zu berücksichtigen, dass ein System nicht nur durch den Entwickler verändert wird, sondern sich auch selbst im Sinne einer Adaption anpassen können muss. Unter Adaption versteht man im Allgemeinen eine Anpassung durch Lernen, im Speziellen das nicht-programmierte Erlernen oder Anpassen an Umwelteinflüsse, die für das System überlebensrelevant sind. Die Adaption eines Systems an seine Umwelt bedingt unter Umständen unterschiedliche Mechanismen. Durch die unterschiedliche Entwicklungsrate von Systemen im Rahmen der epigenetischen Entwicklung kommt es im Laufe der artifiziellen Phylogenetese zu Veränderungen der Systeme und zur Entstehung neuer Systeme (Evolution). Im Laufe der system-individuellen Entwicklung (Ontogenese) finden spezifische Anpassungen statt, die vor allem durch Prozesse des Lernens vermittelt werden. Eine weitere Möglichkeit, die Architektur flexibel in Bezug auf Veränderungen, aber auch in Bezug auf Fehlerbehandlung zu machen, besteht in der Definition einer Metalebene. Diese kann Monitor-, Diagnose- oder Reparaturaufgaben im Hinblick auf die Komponentenebene, d. h. bezogen auf das eigentliche System, übernehmen (Abb. 4.8).

Insofern werden in diesem Buch sowohl die Systeme des Alltags und die Robotersysteme als *Agentensysteme* aufgefasst, von denen man erwartet, dass sie „denken“, bevor sie handeln, dass sie aus Erfahrung lernen und damit Probleme „intelligent“ lösen können. Es werden Systeme sein, die sich selbst weiterentwickeln, um solche Aufgaben auch zu lösen, für die sie ursprünglich nicht dediziert programmiert wurden. Um eine solche Weiterentwicklung anstreben zu können, werden diese Agentensysteme neben dem Zugriff auf inhärentes Wissen, um darauf basierend Schlussfolgerungen zu ziehen, vor allem Daten und Informationen aus der Umgebung sammeln, um diese Informationen in stimmige Handlungsempfehlungen umzuwandeln. Übersteigt eine Aufgabe trotz der vorhandenen Wissensbasis die Fähigkeit eines solches Agenten, wird er von sich aus versuchen, durch geeignete Such- und Rechercheverfahren beispielsweise die riesige Informationsmenge des Internets für die Wissenserweiterung zu nutzen.

## 4.2 Kognitives System

Das Konzept der artifiziellen Kognition im Allgemeinen und das Konzept des Wissens im Speziellen wird im Rahmen der Kognitiven Robotik als zentral angesehen, da artifizielle Kognition als ein wesentlich wissensbasierter Prozess angesehen wird. Bereits im Rahmen der Wissenstheorie wurde als Grundlegung herausgearbeitet, dass ein *Zeichen* ein wahrnehmbares Phänomen ist, das für etwas steht und dem daher eine Bedeutung zukommt. *Daten* wiederum setzen sich aus einzelnen Zeichen oder aber aus einer Folge von Zeichen zusammen, die dann einen sinnvollen Zusammenhang ergeben. Daten werden zu *Informationen*, indem sie in einen Problemzusammenhang (Kontext) gestellt und zum Erreichen eines konkreten Ziels verwendet werden. *Wissen* ist das Ergebnis der Verarbeitung solcher Informationen durch ein artifizielles Bewusstsein im Dienste einer Handlungsorientierung und kann als „verstandene Information“ aufgefasst werden.

Diese Konzeptionalisierung von Wissen wird sich an späterer Stelle auch als implementierungszugänglicher ergeben, indem „Intelligenz“ mit symbolischer wie subsymbolischer Informationsverarbeitung realisiert wird und diese Informationsverarbeitung als notwendige und hinreichende Voraussetzung für eine artifizielle Intelligenz betrachtet wird, die wiederum nach Formen und nach Graden differenzierbar ist.

Wissen ist damit eine spezifische Form von Handeln und umgekehrt begrenzt sich Wissen-Können nicht selten auf das, was durch Handeln machbar ist. Gemäß dieser Auffassung lässt sich Wissen als relativ dauerhafter Inhalt eines Gedächtnisses auffassen. Als solcher Inhalt entspricht dies noch der klassischen Auffassung seitens der Künstlichen Intelligenz von einer auf Basis von Informationen zusammengesetzten Wissensbasis, die einer bestimmten Verarbeitung durch Operationen des Systems zugänglich ist.

Dieser klassischen Auffassung nach sind kognitive Modelle des Wissens als wissensbasierte Systeme realisiert. Die Wissensbasis stellt allerdings nur einen Teil eines solchen Systems dar. Der andere Teil ist die Inferenzkomponente, die Inferenzverfahren durch Inferenztechniken so realisiert, dass sie auf das jeweils verwendete Repräsentationsformat abgestimmt sind. Die Wissensbasis ist Teil eines wissensbasierten Systems, in dem das Wissen so gespeichert ist, dass es von einer Inferenzmaschine manipuliert werden kann. Der Vorteil hiervon ist, dass bei Änderungen der Wissensbasis die Inferenzmaschine nicht mitgeändert werden muss, sofern sich die Änderungen im Rahmen der vorgegebenen Wissensrepräsentation bewegen.

Im Rahmen des Paradigmas des Cognitive Computing und der Kognitiven Robotik ist Wissen jedoch als die Menge von systeminternen Zeichen, Daten- und Informationsrepräsentationen anzusehen, die im Rahmen eines artifiziell-kognitiven Verarbeitungsprozesses mit Hilfe adäquater Technologien ein sich dadurch konstituierendes, kognitives System dazu befähigen, nicht nur Aufgaben zu bewältigen, sondern auch Probleme einer Lösung zuzuführen, „intelligent“ zu handeln, um dadurch mit der Umgebung zu interagieren bzw. zu interoperieren und somit insgesamt auf die Umwelt einzuwirken.



**Abb. 4.9** Handlungsorientierter Wissensbegriff

Trotz des Fokus auf die Verarbeitung sind auch der Wissenserwerb beim menschlichen Lernen, beim maschinellen Lernen insbesondere das Erlernen von Begriffen (Konzepten) und der Prozess der Erfassung und Modellierung von Expertise im Rahmen des sogenannten Wissens-Akquise (Knowledge Engineering) zu nennen. Knowledge Engineering ist dabei ein Arbeits- und Forschungsgebiet der angewandten Kognitionswissenschaft, das den Entwurf wissensbasierter Systeme für den praktischen Einsatz (beispielsweise durch Expertensysteme) zum Gegenstand hat. Es umfasst vor allem aus der Psychologie entlehnte Techniken der Wissensakquisition durch Interviews, Beobachtung und andere Akquisetechniken.

Gemäss dieses Ansatzes kann kein passives Wissen existieren. Ein solches Passivwissen stellt kein Wissen dar, sondern lediglich „Daten“ oder maximal „Informationen“. Aktiv kann Wissen nur in Verbindung mit einem kognitiven System werden, denn dann erst kann Wissen für intelligente Entscheidungen und interoperative Handlungen genutzt werden. In diesem Sinne sucht das Cognitive Computing nach Antworten auf die Fragen: Wie wird Wissen erworben? Wie ist es gespeichert und für den Zugriff organisiert? Wie wird es genutzt, um es in Verhalten umzusetzen? (Abb. 4.9)

Gerade zur Problemlösung bedarf es also nicht nur des Zugriffs auf Wissen, sondern auch kognitiver Verarbeitung, beispielsweise in Form von Schlussverfahren bzw. Inferenztechniken, die etwa aus einer gegebenen Problembeschreibung und verfügbarem Wissen zusammen Folgerungen ziehen und damit Lösungen erarbeiten können. Diese handlungs- bzw. verhaltensorientierte Ausprägung des Wissensbegriffs reduziert die Inhalte dessen, was im Rahmen der Kognitiven Robotik unter Wissen verstanden und verarbeitet werden kann. So lässt sich aufgrund des epistemischen Status neben gesichertem Wissen auch unsicheres Wissen verarbeiten, wobei unter letzterem ein solches Wissen verstanden wird, dessen Wahrheit im Gegensatz zu fehlerhaftem Wissen steht, das objektiv falsch ist. Bezuglich der Verwendbarkeit ist sowohl Kontroll-Wissen als strategisches Wissen und heuristisches Wissen zur Suchsteuerung im Problem- und Lösungsraum der kognitiven Verarbeitung zugänglich. Bezogen auf die Art der Wissensrepräsentation lässt sich analoges Wissen, explizit repräsentiertes oder deklaratives Wissen von implizitem Wissen (prozedurales Wissen, Regel-Wissen, Schema-Wissen, fallbasiertes Wissen, qualitatives und quantitatives Wissen) verarbeiten. Die Wahl der Repräsentation ist oftmals davon abhängig, ob es sich um Alltagswissen bzw. Weltwissen oder aber bereichsspezifisches Wissen bzw. Fachwissen oder Experten-Wissen handelt. Dies schließt auch das terminologische Wissen ein, das den Gebrauch und Bedeutung von Begriffen (terminologisches Wissen, begriffliches Wissen, konzeptuelles Wissen, semantisches Wissen) regelt. Bei Letzterem wiederum wirkt sich die

Repräsentation des assertorischen Wissens aus, in dem Wissen über Sachverhalte, Fakten-Wissen und episodisches Wissen der kognitiven Verarbeitung zugänglich gemacht wird. Dies bedingt auch die Behandlung von räumlichem und zeitlichem (temporalem) Wissen, kausalem Wissen und sprachspezifischem Wissen. Dieser Wissensbegriff wirkt sich auch auf die Abgrenzung der verarbeitenden Systeme aus. Ein System ist dann ein *daten- bzw. informationsverarbeitendes System*, wenn es über Eingangskanäle (Sensoren) verfügt, mit deren Hilfe es Signale, Zeichen aus einer Umgebung aufnehmen kann, diese Zeichen in Abhängigkeit von seinem jeweiligen inneren Zustand durch Berechnung in Daten und Informationen transformiert, also verarbeitet (einschließlich der Speicherung von Information) und wenn es über Ausgänge zur Ausgabe von Zeichen, Informationen verfügt. Ein *wissensverarbeitendes* und damit *kognitives System* verfügt darüber hinaus über die Fähigkeit, aus Zeichen, Daten und Informationen unter Anwendung von Regeln Wissen zu generieren und durch Interoperationen direkt auf seine Umgebung einzuwirken.

Diese Auffassung von Wissen wirkt sich auf das Konzept der Intelligenz aus, ein Konzept, das in diesem Buch entwickelt und dem eine wesentliche Bedeutung zukommen wird. Eine allgemein anerkannte Definition für Intelligenz gibt es nämlich nicht. Dass der Versuch, einen einheitlichen Intelligenzbegriff zu entwickeln, bisher nicht gelungen ist, wird auch dadurch erklärt, dass man immer auszuweisen hat, welche spezifische Form von Intelligenz man sucht und dass die Ermittlung dieser unterschiedlichen Intelligenzen nur sehr schwer möglich ist. Häufig wird Intelligenz demnach zu definieren versucht, als die Fähigkeit zur Anpassung an neuartige Bedingungen und zur Lösung neuer Probleme oder einfach als das, was Intelligenztests messen.

If a system uses all of the knowledge that it has, it must be perfectly intelligent. There is nothing that anything called intelligence can do to produce more effective performance. If all the knowledge that a system is brought to bear in the service of its goals, the behavior must correspond to what perfect intelligence produces.<sup>1</sup>

Intelligenz ist nach dieser Auffassung zwischen kognitiven, wissensbasierten Systemen nur anhand eines gegebenen Ziels oder Interesses und dem Wissen, dass den jeweiligen kognitiven Systemen für die Zielerreichung zur Verfügung steht, vergleichbar. Intelligenz wird somit relativ zum jeweiligen Problem und dem zur Verfügung stehenden Wissen definiert. Wird einem Menschen und einem technischen, wissensbasierten System die gleiche Aufgabe vorgelegt und dabei sichergestellt, dass sie das gleiche dazu gehörende Wissen besitzen, kann ein Intelligenzvergleich gezogen werden. Nutzen beide wissensbasierte Systeme das ihnen zur Verfügung stehende Wissen in optimaler Weise, besitzen sie das gleiche Intelligenzniveau. Da dies in erster Linie am produzierten Output erkennbar ist, wird Intelligenz damit ein Maßstab für die Qualität des Ergebnisses, gemessen am gegebenen Ziel. Damit werden nicht immer messbare Kriterien verbunden, die mit der menschlichen Intelligenz in einem wie auch immer gearteten Zusammenhang stehen:

---

<sup>1</sup> Vgl. Newel 1999

Es handelt sich um eine Begabung (bzw. Begabungen), die dazu befähigt, Probleme zu lösen, also auch auf neuartige Situationen entsprechend zu reagieren („Lebensbewältigung“). Lernen gehört als notwendige Bedingung dazu, da auch verschiedene Begabungen nur durch eine entsprechende Sozialisation gefördert und entfaltet werden können. Schließlich umfasst Intelligenz die Fähigkeit zu analogem Denken (Metaphorizität der Sprache, Symbolik von Gegenständen), Umgang mit Unsicherheit (Ambiguität), anpassungsbedingte Entscheidungsfähigkeit trotz Ungewissheit, entsprechende Bedeutungsermessung neuer Lebenslagen, Transferleistungen in Situationen, die prima facie unbekannt, aber ähnlich gelagert sind etc.<sup>2</sup>

Insofern lassen die bisherigen Definitionen noch keinen Raum für Kreativität<sup>3</sup>, Originalität der Intuition, auf die Kritiker einer solchen Auffassung sehr großen Wert legen.<sup>4</sup> Ein anderer Versuch, menschliche Intelligenz definitorisch zu fassen, orientiert sich an den performativen Leistungen des Menschen.<sup>5</sup> Dabei geht man der Frage nach, inwieweit kognitives Vermögen (Kompetenz) in menschlichen Handlungen (Performanz) als intelligentes Verhalten resultiert. Dieser Versuch soll in dieser Arbeit weiter entwickelt werden, indem das Vermögen eines Systems, intelligent zu handeln, in den Mittelpunkt der weiteren Überlegungen gestellt wird.

Im Rahmen dieses Buches wird unter *systemischer Intelligenz* im Allgemeinen die Fähigkeit eines Systems verstanden, den kognitiven Prozess gemäß des kognitiven Modells realisieren zu können. Im Speziellen werden unter systemischer Intelligenz die Funktionen der Wahrnehmung, des erfassenden und planenden Denkens, der Emotion, Motivation, Intuition, des Lernens und der Reflexion, der Speicherung von Konzepten, die Ausprägung von Systemmerkmalen sowie das insgesamt dadurch bedingte Verhalten subsummiert. Insofern bilden die Wahrnehmung und das Verhalten die Klammer bzw. die äußeren Randerscheinungen dieser systemischen Intelligenz. Eine so aufgefasste Intelligenz zeigt eindeutig den Charakter einer sogenannten systemisch-performativen Intelligenz, indem Intelligenz über die gezeigten kognitiven Leistungen und des daraus resultierenden Verhaltens ermittelt werden soll. Insofern widerspricht dieser Intelligenzbegriff der Annahme eines hierarchischen Strukturmodells nicht darin, dass ein Bündel unterscheidbarer und zusammenhängender Fähigkeiten postuliert wird, sondern vielmehr in der Annahme eines hochgradig generellen Faktors, der eine allgemeine Intelligenz begründen soll. Vielmehr integriert dieser Intelligenzbegriff also verschiedene Perspektiven der systemischen Intelligenz zu einem Gesamteindruck.

- *Adaptive Perspektive:* Zunächst zeigt sich die Intelligenz in einer problem- bzw. lösungsorientierten und zielgerichteten Anpassung an die Umgebung. Ob das systemische Verhalten als intelligent einzustufen ist, kann daher nur im jeweiligen Kontext der Problemdomäne bewertet werden.

---

<sup>2</sup> Vgl. R. Voß 1985, Seite 10 ff.

<sup>3</sup> Siehe auch Boden 1992

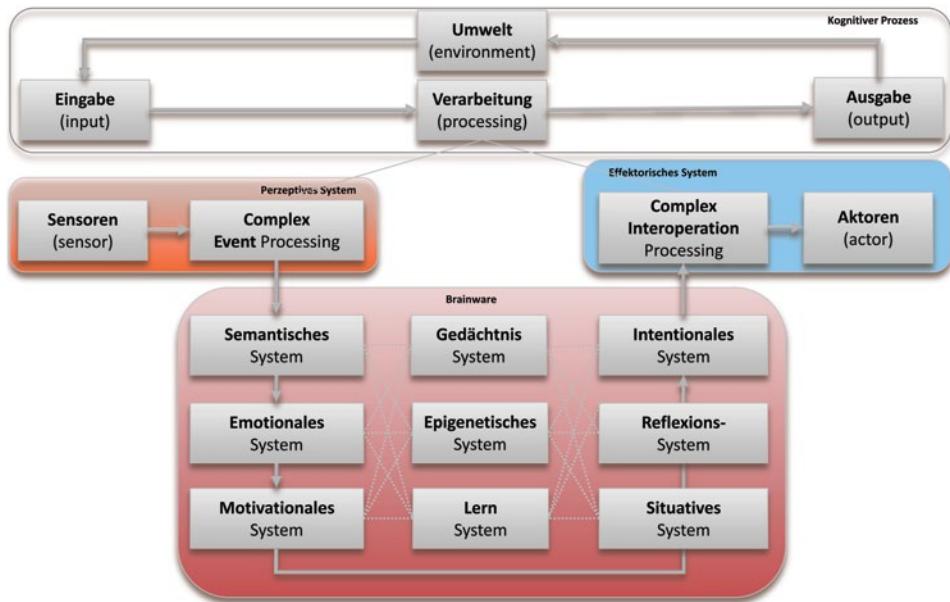
<sup>4</sup> Vgl. Weizenbaum 1987

<sup>5</sup> Vgl. Kail und Pellegrino 1989

- *Lösungsorientierte Perspektive:* In einer weiteren Perspektive wird Intelligenz einerseits als die Fähigkeit zur Lösung neuartiger Problem- oder Fragestellungen konzeptionalisiert.
- *Prozessuale Perspektive:* Eine andere Perspektive wird gewählt, wenn sich der Fähigkeit zur Automatisierung von Daten-, Informations- und Wissensverarbeitungsprozessen, dem kognitiven Prozess und damit dem Zusammenwirken verschiedener elementarer kognitiver Komponenten zugewendet wird, da höhere Effizienz in einem Bereich bzw. in einer Komponente unter Umständen Kapazität für andere Bereiche frei macht. Diese Perspektive fokussiert auch die Lernfähigkeit des Systems, die als eine der Voraussetzungen für den Wissenserwerb und die Anwendung von Wissen gesehen wird. Wissen ist demnach investierte Intelligenz.
- *Motorische Perspektive:* Unter dieser Intelligenz werden im Falle von Robotern die technisch-praktischen Fähigkeiten und Fertigkeiten sowie die Fähigkeit zur senso-motorischen Koordination zusammengefasst.
- *Kommunikative Perspektive:* Unter dieser Kompetenz werden die Fähigkeiten zur Kommunikation mit anderen Agenten bewertet.
- *Technologische Perspektive:* In einer eher implementierungsnahen Perspektive wird die systemische Intelligenz durch die Funktionsweise der einzelnen Komponenten betrachtet.

Insgesamt ermöglichen die unterschiedlichen Perspektiven damit ein Performanzbild, das auch einen Eindruck bezüglich Enkodieren (Trennen relevanter von irrelevanter Information), dem selektiven Kombinieren (Integrieren von Informationen in einen Bedeutungszusammenhang) und dem selektiven Vergleichen (Herstellen einer Beziehung zwischen neu enkodierten Informationen und vorhandenem Wissen) hinterlassen. Dieser multiperspektivische Ansatz sieht einen Intelligenzquotienten damit nicht mehr als singuläres Maß, sondern ermöglicht die Formulierung von *Intelligenzprofilen* mit Hilfe von standardisierten Werten für Einzelfähigkeiten und diese gelten dann als Grundlage der Intelligenzdiagnostik.

Ein solcher Intelligenzquotient (IQ) ist der Versuch eines standardisierten Maßes für allgemeine Intelligenz. Das Maß orientiert sich an der Vorstellung, dass intelligentes Problemlöseverhalten stets wie eine Suche nach einem Lösungsweg für eine gestellte Frage verstanden wird. Insofern definiert sich eine solche Intelligenz als die Suche nach der besten Lösung für ein gegebenes Problem. Der symbolische Ansatz begreift Intelligenz als suchendes Problemlöseverhalten in bestimmten Verhaltenssituationen, hingegen teilt der subsymbolische Ansatz jedem neuronalen Netz seine Aufgabe zu und misst den Adaptionserfolg an der jeweiligen Ausgabe des Netzes. Ein solcher IQ hat an Bedeutung verloren, seitdem hierarchische Intelligenzstrukturmodelle postuliert werden.



**Abb. 4.10** Kognitives Robotersystem

### 4.3 Kognitives Robotersystem

In diesem Buch werden Mechanismen der kognitiven Daten-, -Informations- und Wissensverarbeitung als eine Verarbeitung einer spezifischen Eingabe zu einer Ausgabe aufgefasst. Die durch die Sequenz Eingabe-Verarbeitung-Ausgabe entstehenden Symbolfolgen oder Kodes werden *kognitive Repräsentationen* und die Sequenz als *kognitiver Prozess* im Dienste einer *Kognitionsverarbeitung* bezeichnet (Abb. 4.10).

#### 4.3.1 Perzeptives System

Das *perzeptive System* dient der Reizaufnahme durch Sensoren und der Wahrnehmung durch Transformation der Reize auf entsprechende Datenstrukturen. Man spricht auch von der Wahrnehmung als erste artificielle Erkenntnisstufe. Durch die den Sensoren möglicherweise vorgesetzten *Rezeptoren* wird der Anforderung Rechnung getragen, dass die Ausprägung und damit die Definition des Signalreizes verschieden sein kann. Außerdem gilt es Rechnung zu tragen, dass für alle Systeme des Symbolverarbeitungsansatzes die Voraussetzung gilt, dass die Eingabe bereits in symbolischer Repräsentation vorliegt. Insofern muss die Umsetzung von Sinnesdaten in eine solche Repräsentationsform thematisiert werden, da dies möglicherweise für das Verständnis natürlicher kognitiver Systeme essentiell ist. Allgemein versteht man unter einem Rezeptor einen Signalempfänger.<sup>6</sup>

<sup>6</sup> Siehe auch Eimer 1990

In der Biologie bezeichnet ein Rezeptor im engeren Sinne vier unterschiedliche Strukturen. Komplette Sinnesorgane, wie das Auge oder das Ohr (Sinne), einzelne Sinneszellen dieser Sinnesorgane, die den Reiz empfangen, ihn weiter verarbeiten und weiterleiten, die Bestandteile dieser Sinneszellen, die im eigentlichen Sinne für die Reizaufnahme zuständig sind und die molekulare Strukturen, die vorwiegend auf Zellmembranen lokalisiert sind.

Empirische Untersuchungen zeigen, dass nicht nur die Intensität eines Signals, sondern auch der motivationale Zustand des Systems seine Wahrnehmungsleistung beeinflusst. Dieser Tatsache trägt dieses Modell dadurch Rechnung, dass sie den motivationalen Einfluss von der perzeptiven und damit sensorischen Leistung trennt. Dabei trifft das Modell eine Unterscheidung zwischen dem proximalen Reiz, als dem auf die Rezeptoren treffenden Energie- oder Signalmuster, und dem distalen Reiz, als der physikalischen Ursache, die diesem Energie- oder Signalmuster zugrunde liegt. Ab dem Sensor wandelt sich der Begriff des Reizes, der bis dahin in seiner Verwandtschaft mit dem der Reizung noch sein sinnesphysiologisches Vorbild erkennen lässt, zu dem der Eingabe (*Input*).

Sensoren sind an das kognitive System angeschlossene Messinstrumente oder Messwandler, die physikalische Größen wie Wärme, Licht, Druck usw. in elektrische Signale umwandeln, die dann in der geeigneten Kodierung von der Wahrnehmungskomponente weiterverarbeitet werden können. Bei prozessorientierten Lösungen wird Information über den Zustand des Prozesses durch Sensoren aufgenommen und vom Rechner verarbeitet. Sensoren ermöglichen dem System, Signale, Zeichen, Daten oder Informationen über die Umgebung zu gewinnen.

In der Neurobiologie werden häufig auch Sinnesorgane oder allgemeine Rezeptoren als Sensoren bezeichnet. Sensoren reagieren praktisch stets gleich auf Reize der Umgebung, sie erzeugen elektrische Signale und leiten sie zur Verarbeitung weiter. Allerdings senden sie diese Informationen an jeweils unterschiedliche Orte im menschlichen Gehirn, wo sie dann auf jeweils unterschiedlichen Bahnen verarbeitet werden. Der Ort der Verarbeitung bestimmt also, welche Art von Erleben entsteht.

Die Eingaben über die Sensoren und freien Rezeptoren werden nach der Wahrnehmung im Rahmen des kognitiven Prozesses zunächst an das semantische System geleitet und führen manchmal bereits auf dieser frühen Verarbeitungsstufe zu Reflexen oder bestimmen die Tendenz einer komplexeren Reaktion. Ein kognitives System nimmt all das wahr, wovon es durch seine Rezeptoren und letztlich den Sensoren Kenntnis erlangt. Die daran anschließende Wahrnehmung ist sehr stark von selektiver Aufmerksamkeit, Vorannahmen, inneren Zielen und Handlungskontexten bestimmt.

Diese Aufmerksamkeit spielt bei dem Verknüpfen von Reizen aus verschiedenen Informationskanälen eine wichtige Rolle. Gemäß dem Top-down-Modell analysiert zunächst eine höhere Kontrollinstanz im Gehirn die visuelle Eingabe und bewertet deren visuellen Informationsgehalt. Nach dem Bottom-up-Modell hingegen verbindet das Gehirn verschiedenartige Reize bereits auf einer frühen Wahrnehmungsebene.

Somit sind die Verfügbarkeit von Sensoren und die Beziehung der *artifiziellen Wahrnehmung* zu diesen Sensoren als eine notwendige Bedingung der Wahrnehmungsfunktion

anzusehen. Rein intuitiv zielt der Begriff der Wahrnehmung dabei auf jenen Aspekt des psychischen Geschehens und Erlebens, der sich auf die Kopplung des Systems an funktional relevante Aspekte der physikalischen Umwelt bezieht.<sup>7</sup> Insofern kann man davon sprechen, dass die artifizielle Wahrnehmung das „Fenster“ der Brainware zur Außenwelt darstellt. Die Anbindung des Systems an seine Umwelt umfasst nicht nur die visuelle, auditive, gustatorische, olfaktorische und haptische Wahrnehmung, sondern gleichermaßen die Wahrnehmung von systemintrinsischen Zuständen sowie relative Positionen des Systems zu anderen Systemen und zur Umgebung, die Wahrnehmung von multimedialen Eingabeströmen, einzelnen Signalen oder komplexen Signalfolgen, Ereignissen sowie die Wahrnehmung von Sprache und Zeit.

So erscheint eine Sprache notwendig, da die bisherigen kognitiven Systeme an und für sich nicht direkt miteinander kommunizieren. Aus evolutionärer Sicht scheint damit die Entwicklung einer Sprache unabdingbar geworden zu sein, um den Lebewesen und deren kognitiven Systemen über eine Sprache einen Kommunikationsraum und dort den Austausch von kognitiven Inhalten (Gedanken) zu eröffnen. Diese Pluralität der Eingaben ist gefordert, weil ein artifizielles Verstehen oder Nichtverstehen je nach Problemdomäne sich auf Personen, Handlungen, Situationen, Artefakte, Zeichen, Zeichenhandlungen, Zeichensysteme, Bilder, Texte, Sätze, Wörter, Sprachen, Musikstücke, Praktiken, Institutionen, Argumentationen, Beweise, Schlüsse, Theorien etc. der realen oder der abstrakten Welt beziehen kann.

Dies impliziert, dass die Wahrnehmungsfunktion der Verhaltenssteuerung ebenso dient wie etwa motivationale Prozesse und bildet folglich mit diesen anderen kognitiven Funktionssystemen eine komplexe Einheit.

Das Gehirn nutzt Sinneseindrücke, Wahrnehmungen und Emotionen, um Handlungen zu planen. Dabei kommt es zu interner Gehirnaktivität, zu Gedanken.

Eine weitere Implikation besteht darin, dass Prozesse, die der Wahrnehmung zugrunde liegen, nicht selbst zum artifiziellen Bewusstsein gelangen und sich daher einem direkten und unmittelbaren kognitiven Zugang entziehen. Des Weiteren gestaltet sich die Umsetzung von Signalen der Sensoren in einen kognitiven Kode, die sogenannte kognitive Transduktion, dergestalt, dass ein solcher Kode seine ursprüngliche, physikalische Ursache unter Umständen nicht mehr erkennen lässt.

Dabei greift man auf die Erkenntnisse der Erkenntnistheorie und in der Sinnesphysiologie zurück. Beschäftigt sich die Erkenntnistheorie mit den Inhalten der Wahrnehmung und mit der Frage, wie sich ein Kommunizieren über die Entitäten der Welt und eine Erkenntnis der Welt auf der Basis der durch die Sinne vermittelten Informationen rechtfertigen lässt, so untersucht die Sinnesphysiologie den rezeptoralen Transduktionsprozesses, d. h. die Umsetzung eines physikalischen/chemischen Reizes in ein neuronales Signal und die Art der neuro-

---

<sup>7</sup> Siehe auch Chalmers 1996

nalen Kodierung. So kann eine Lichtempfindung sowohl durch optische, wie mechanische oder aber elektrische Reizung des Auges entstehen, und umgekehrt, kann Sonnenlicht durch Erregung der retinalen Rezeptoren zu einer Lichtempfindung und durch Erregung der Thermorezeptoren der Haut zu einer Wärmeempfindung führen.

Insofern erfolgt bereits im perzeptiven System die für die Interaktion bzw. Interoperation mit der Umwelt und Umgebung notwendige Integrations- und ggf. bei unvereinbaren Informationen eine Entscheidungsleistung bezüglich der Selektion und Aufbereitung entscheidungs- bzw. handlungsrelevanter Informationen. Artifizielle Wahrnehmung ist somit ein Prozess der *Informationsreduktion*, wobei eine komplexe und zum Teil unstrukturierter Menge von Signalen auf einfachere, strukturierte und entscheidungsrelevante Formen zurückgeführt werden muss. Diese Reduktionsprozesse haben zur Folge, dass die artifiziellen Wahrnehmungen in Einheiten in Form von Verarbeitungseinheiten strukturiert sind, die Gegenstände und Abstrakte sowie deren Eigenschaften entsprechen. Durch diese Reduktion lässt das Modell auch eine subliminale Wahrnehmung zu, die sich dann einstellt, wenn die Reize, die der Reduktion zum Opfer gefallen sind und die somit nicht zum Bewusstsein gelangt sind, dennoch das Verhalten des Systems beeinflussen.

Das Gehirn empfängt massenweise Informationen, doch nur ein kleiner Teil davon wird tatsächlich so weit verarbeitet, dass er in das Bewusstsein gelangt. Gelangen diese Informationen allerdings ins Bewusstsein, entsteht eine multisensorische Integration und Wahrnehmung. Beispielsweise kann man sich nur mittels einer solchen multisensorischen Integration ein umfassendes Bild über eine Situation oder eine Stimmungslage machen.

An irgendeinem Punkt des Datenverarbeitungsstroms haben interne Repräsentationen (mentale Vorstellungsbilder) und Wahrnehmungen eine gemeinsame Darstellungsform, die sich von der Darstellungsform der Repräsentationen anderer Informationsarten (z. B. sprachlichen Informationen) durchaus unterscheiden kann. In diesem Zusammenhang ist darauf hinzuweisen, dass die bloße Existenz solcher Repräsentationen in einer propositionalen Darstellungsform noch kein Wissen an sich konstituiert. Genauso wie ein Blatt Papier nicht deswegen schon Wissen darstellt, nur weil ein Satz darauf geschrieben steht. Nur in einem Kontext, in dem es irgendwelche Verarbeitungen interner Repräsentationen gibt, ist es sinnvoll, von Wissen zu sprechen. Insofern wird aus Sicht des perzeptiven Systems *Wissen* als aktive Verarbeitung von Informationsstrukturen verstanden und nicht als diese statischen Strukturen selbst. Diese Informationsstrukturen entstehen nun dadurch, dass die von der Außenwelt auf die Sensoren eintreffenden Signalmuster vom artifiziellen Wahrnehmungssystem Punkt für Punkt so ausgewertet werden, dass sie als Bausteine im Rahmen des *Erfassenden Denkens* eine semantische Interpretation erfahren können. Diese Informationen wiederum werden dann als Bausteine angesehen, aus denen sich auf einer höheren Ebene Wissen konstituiert.

Dieser Versuch des „erfassenden Denkens“, dem sensorischen Input gleichsam eine semantische Interpretation zu geben, erfolgt damit in einer eigenen Komponente, nicht wie beim

Menschen in der Regel automatisch und ist somit, auch im Gegensatz zum Menschen, einer bewussten kognitiven Modifikation zugänglich.

Dies beantwortet auch die Frage, ob bereits die Wahrnehmungsprozesse Prozesse der Daten-, Informations- und Wissensverarbeitung sind oder nur durch solche beschrieben und simuliert werden können. Da das artificielle perzeptive Wahrnehmungssystem als Eingabe keineswegs Informationen sondern lediglich Signale entgegennimmt, und erst durch die Verarbeitung im Rahmen des kognitiven Prozesses gleichsam aus dieser Eingabe Informationen und Wissen entstehen, findet im perzeptiven System eine reine Datenverarbeitung statt. Insofern reicht der Prozess der Wahrnehmung gemäß dieses Ansatzes von der Ebene der rezeptoralen Transduktion über die sensorische Kodierung bis hin zur perzeptuellen Repräsentation durch Daten oder Datenmodelle. Diese Daten oder Datenmodelle werden als *Perzepte* bezeichnet. Diesen Perzepten kommen bereits an dieser Stelle des kognitiven Prozesses unterschiedliche Funktionen zu:

- *Kommunikative Funktion*: Perzepte weisen Strukturen auf, indem sie begriffliche Strukturen von Fach- und/oder Problemdomänen darstellen. Sie erfüllen damit die kommunikative Funktion, da sie als Gegenstände bzw. Instrumente der Kommunikation dienen.
- *Referentielle Funktion*: Perzepte stehen als Zeichen oder Zeichenstrukturen sowohl für Begriffe (in der Kommunikation) als auch für Gegenstände in der Umwelt (zur ontologischen Identifikation und Wiedererkennung).
- *Kognitive Funktion*: Sowohl auf individueller Ebene zum Zweck des Lernens und des Wissenserwerbs, als auch auf intersystemischer Ebene für den Aufbau gemeinsamer Wissensbestände (im Sinne verteilter oder kollektiver Kognition).

Damit werden zum einen die Signalmodalitäten als natürliche und weitgehend isoliert bearbeitbare Eingabeeinheiten betrachtet, zum anderen erstreckt sich der perzeptive Prozess von der physikalischen Eingabe bis hin zum Perzept.

### 4.3.2 Semantisches System

Um eine für die Zwecke einer adaptiven Kopplung an die Umwelt angemessene Interpretation des sensorischen Inputs zu erreichen, muss das semantische System den sensorischen Input gleichsam im Kontext bestimmter Annahmen über die physikalische Welt (Robotik) und/oder der Problemdomäne interpretieren, über die es unabhängig vom sensorischen Input verfügt. Insofern muss das semantische System zu diesem Zeitpunkt mehr wissen, als ihm durch den rezeptorischen Reiz bzw. der sensorischen Eingabe zur Verfügung gestellt wird. Aus dieser Perspektive lässt sich die sensorische Eingabe gleichsam als ein Stichwortgeber oder Anker auffassen, der nicht nur das im Folgenden, durch interne Komponenten und Funktionen bedingtes, komplexes Geschehen initiiert und dieses Ge-

schehen mit dem notwendigen Vorwissen anreichert, sondern auch indirekt eine stabile An- und Verbindung an die Umwelt garantiert.

Da normalerweise alltägliche Dinge, wie beispielsweise Tische und Stühle als solche auch wahrgenommen werden, eine Tisch- oder Stuhlvorstellung aber bereits eine Allgemeinvorstellung ist, heißt dies, dass das semantische System eine kognitive Verarbeitung der Perzepte voraussetzt. Das Ergebnis dieser Verarbeitung zeigt sich in den *Entitäten*, wobei unter diesen Elemente („Dinge“) und Abstrakte der Welt verstanden werden können.

Der Übergang vom semantischen System in die weiteren Systeme kann auch als eine Art artifiziell-mentaler Quantensprung aufgefasst werden. Einerseits befindet sich eine Menge von Perzepten (Daten- und Datenstrukturen) im System, die eine permanente Modifikation (den Perzeptstrom) durchlaufen, der durch immanente (Re)Strukturierungsbedingungen bestimmt ist. Andererseits verfügt das System dadurch über einen „Konzentrationspunkt“ des Gewahrwerdens, der auf ganz bestimmte Elemente des Perzeptstroms gerichtet sein kann. Die Elemente in dem Perzeptstrom können als Anzeichen für Entitäten (Dinge an sich, Gegenstände, Ereignisse, etc.) angesehen werden, auf die sich das System in seinem Gewahrwerden zu- oder abwenden kann. In diesem Sinne kann man die Wahrnehmung als eine Interaktion zweier operativer Subsysteme auffassen: dem operativen System, das den Perzeptstrom konstituiert und dem operativen System des Gewahrwerdens (als Aktivität des Ego dargestellt).

Die bisher beschriebene Komplexität der internen Verarbeitungen lassen sich durch die Metapher eines Ozeans voll glitzernder Punkte von Wellen ausdrücken, deren Veränderung auch komplexen Regeln zu unterliegen scheint, obwohl die Natur dieser Regeln nicht direkt erfasst werden kann.

Die Gewahrwerdung dient der Freilegung des Inhaltes in Form einer Fokussierung. Durch diese Fokussierung spiegelt sich die „Bedeutung“ einer Entität in ihrer Beziehung zu anderen Entitäten.

### 4.3.3 Emotionales System

Obwohl es bisher keine exakte Definition der Emotion gibt, wird allgemein anerkannt, dass sich menschliche Emotion durch Phänomene der Freude, Trauer, Ärger, Furcht, Stolz, Mitleid etc. auszeichnet.

Emotionen sind keine bewussten Gefühle, sondern körperliche Reaktionen auf eintreffende Reize, die den Menschen auf Gefahren oder Erfolge aufmerksam machen sollen. Insofern geht man davon aus, dass solche Emotionen permanent generiert, aber nicht generierte Emotionen auch bewusst wahrgenommen werden. Neben dieser bewussten Wahrnehmung lassen sich Emotionen auch durch Gedanken beeinflussen.

Artifizielle Emotionen hingegen werden mit bestimmten systemischen Zuständen gleichgesetzt. So bestehen Emotionen zum einen in der Wahrnehmung der systemischen Verän-

derungen, die während der Laufzeit als die Folge- oder Begleiterscheinung von Gefühlen (Erfolg, Misserfolg, Zielerreichung, Scheitern etc.) auftreten. Zum anderen sind Emotionen komplexe systemische Zustände, die aus der Wahrnehmung der kognitiven Einschätzungen von Bewertungen einer Situation zusammengesetzt sind, wobei diese zwei Komponenten außerdem in einer bestimmten Weise miteinander verbunden sind.

So spielen Gefühlsprognosen bei Entscheidungsfindungen eine wichtige Rolle, obwohl dieses zukünftige Befinden nur schwer einzuschätzen ist. Beispielsweise werden die Dauer und Intensität von Gefühlen, die von bestimmten Ereignissen ausgelöst werden, schlachtweg überschätzt (Impact Bias). Weiterhin tendiert man bei Prognosen zu einer Vernachlässigung der eigenen Veranlagung. Auch die momentane Lage, in der der Mensch sich befindet, während er an etwas denkt, färbt die Erinnerung (Situational Bias). Generell neigt der Mensch gemäß des Bestätigungsfehlers eher zu einer Aufrechterhaltung eingenommener Überzeugungen, als diese kritisch zu prüfen. Auch greift das Selbstbild auf Wissen zurück, das im semantischen Gedächtnis abgelegt ist, wohingegen die Prognose oder Simulation zukünftiger Geschehnisse auf das episodische Gedächtnis zurückgreift. Beide Systeme funktionieren weitgehend unabhängig voneinander, was ebenfalls zu unterschiedlichen Resultaten führen kann. Fehlerblindheit scheint demnach ein evolutionäres Kennzeichen des Denkens zu sein. Immerhin scheinen Fehler aber auch den Lernvorgang beschleunigen zu können, was in Strategien, wie beispielsweise dem Error Management Training (EMT), ausgenutzt wird.

Insofern geht das Paradigma der artifiziellen Emotion von einer multidimensionalen Zustandsbeschreibung aus, die systemisch-mentale Zustände, systeminterne Veränderungen sowie Verhaltensänderungen umfasst.

Dabei wird davon ausgegangen, dass mentale und kognitive Zustände als Zustände und Prozesse eine eigene kognitive Qualität besitzen und damit erforschbar sind, wobei eine Beschreibung nicht nur in die Sprache einer der klassischen Naturwissenschaften möglich ist.

Insbesondere wirken die Emotionen auf die kognitiven Prozesse, wie Gedächtnis (und Erinnerung), Intuition und planendes Denken (Urteilsbildung und Problemlösen).

Erinnerungen kann man sich als eine Art Netz vorstellen, in dem die Fäden die unterschiedlichen Elemente einer Erinnerung darstellen, die an den Knoten zu einer vollständig bewussten Erinnerung zusammenlaufen. Solche kompletten Erinnerungen werden in einzelne Komponenten (Empfindungen, Emotionen etc.) zerlegt und in den jeweils zuständigen Gehirnarealen abgelegt. Insofern stellt eine Erinnerung stets eine Konstruktion aus dem Originalerlebnis und der aktuellen Situation dar. Der emotionale Gehalt von Sinnesreizen beeinflusst somit sehr früh die Verarbeitung in höheren sensorischen Arealen.

#### **4.3.4 Motivationales System**

Motivation umfasst die Gesamtheit der Bedingungen und Prozesse, die einer Verhaltensbereitschaft zugrunde liegen und ist neben der Verfügbarkeit relevanter Fähigkeiten und Fer-

tigkeiten (Können) notwendige Voraussetzung für zielgerichtetes Verhalten. Im Einzelnen umfasst dies die Wahl von Verhaltenszielen sowie die Ausführung des zielgeleiteten Verhaltens (Wille). Dabei liegt dem System ein Erwartungs-Wert-Modell zugrunde, d. h., dass das System bei der Wahl von Handlungszielen rational vorgeht und neben der Werteinschätzung des Ziels die wahrgenommene Realisierungswahrscheinlichkeit berücksichtigt. Neben dieser Realisierungswahrscheinlichkeit kommt dem Prozess der Zielrealisierung (Wille) und dabei dem Intentionskonzept eine besondere Bedeutung zu. Unterschieden werden Zielintentionen (Entscheidung für einen angestrebten Zielzustand) und Durchführungsintentionen oder Vorsätze (Festlegung auf konkrete Aspekte der Handlungsausführung), die auf spezifische Weise die Zielrealisierung fördern.

### 4.3.5 Situatives System

Das *situative System* unterstützt die Aufbereitung einer Semantik, da die Bedeutung eine Relation zwischen Situationen darstellt. Das System basiert auf einer Ontologie, in der die grundlegenden ontologischen Entitäten festgelegt werden sowie der Darstellung eines formalen Apparats, der die Repräsentation von Situationen und Ähnlichkeiten, die zwischen Situationen bestehen, ermöglicht. Mit Hilfe dieses Apparats ist es möglich, Situationstypen, Ereignistypen, Rollen usw. zu definieren.

### 4.3.6 Gedächtnissystem

Das *Gedächtnissystem* umfasst sowohl die Fähigkeit, das Gelernte zu behalten, um es bei weiteren Erfahrungen einsetzen zu können, als auch die Fähigkeit, diese erinnerbaren Inhalte abrufen zu können. Insofern obliegt dem Gedächtnissystem sowohl die Funktion der Lernerfahrung als auch die des Gedächtnisses.

Im Rahmen der Kognitiven Robotik bzw. des Cognitive Computing bezieht sich das artificielle Gedächtnis auf verschiedene Formen des Erwerbs, der Speicherung und der Nutzung von Daten, Informationen und Wissen. Mit dem Begriff Wissen (knowledge) werden insbesondere Kenntnisse über die Realität (declarative knowledge), kognitive Operationen und Prozesse (procedual knowledge) sowie perzeptiv-effektorische Fertigkeiten (skills) bezeichnet. Die Gedächtnisinhalte werden dabei nicht allein durch system-intrinsische gesteuerte kognitive Prozesse erworben (artifizieller Wille), sondern vor allem auch im Verlauf der aktiven Auseinandersetzung mit der Umgebung durch Interaktionen bzw. Interoperationen.

Bei Lebewesen werden hingegen der größte Teil alltäglicher Äußerungen, Leistungen, Gewohnheiten, sensomotorischer Aktivitäten ohne bewusste und willentliche Suche im Gedächtnis erbracht. Auch dies indiziert einen umfangreichen, durch Lernprozesse ausgebildeten Gedächtnisbesitz. Nach dieser Auffassung entspricht diese Gedächtnisfunktion nicht

der eines Computerspeichers, auf dem Wissen einfach abgespeichert wird. Vielmehr basieren diese Funktionen auf einem permanenten Rekonstruktionsprozess aus bereits vorhandenem Wissen.

Die primäre Funktion des Gedächtnissystems besteht demnach darin, aus den Wahrnehmungs-, Denk- und Lernprozessen resultierende Daten-, Informationen und Wissen zu speichern und diese Konzepte für spätere Nutzung zur Verfügung zu halten bzw. zu stellen.

Auch das menschliche Gehirn praktiziert verschiedene Verfahren, um Gedächtnisinhalte zu ordnen. Das deklarative Gedächtnis hat Fakten, das semantische Gedächtnis eher Bedeutungen, das prozedurale Gedächtnis Routineabläufe bzw. Fertigkeiten und das emotionale Gedächtnis Gefühle im Fokus der Verarbeitung. Alle diese Verfahren bilden einen Funktionszusammenhang, indem sie sich gegenseitig bedingen und beeinflussen.

Im Gegensatz zum natürlichen Vorbild geht der nicht-modale Ansatz des Cognitive Computing derzeit nicht von der Annahme zweier Systeme aus, die gewöhnlich als Kurz- und Langzeitgedächtnis bezeichnet werden. Die empirischen und theoretischen Analysen der letzten Jahre haben dazu geführt, dass die Annahme eines einheitlichen Kurz- und Langzeitgedächtnisses heute in Frage gestellt wird. Insofern umfasst die Gedächtnisfunktion sowohl das kurzzeitige Speichern einer begrenzten Mengen von Daten und Informationen, die für aktuelle Verarbeitungsziele rasch abrufbar zur Verfügung stehen, als auch einen permanenten Speicher, um Daten, Informationen und Wissen aus zurückliegenden Denk- und Lernprozessen bereitzuhalten. Ebenfalls werden alle relevanten Strukturkomponenten in einem Speicher vorgehalten, auf denen dann Funktionen des natürlichen Modells arbeiten. So ist das sensorische System nicht direkt mit dem Gedächtnissystem verbunden, sondern durch wahrnehmungsspezifische Komponenten gepuffert. Als *Kurzzeitspeicher* wird eine Funktion mit begrenzter Speicherkapazität angeboten, die nur eine begrenzte Anzahl von kognitiven Einheiten zu einem Zeitpunkt aufnehmen und speichern kann. Der *Langzeitspeicher* ist als eine Funktion realisiert, die Kenntnisse über Fakten, Sachverhalte, Entitäten oder Vorgänge der Realität umfasst. Generell werden die Informationen semantisch in Form von Perzepten kodiert und gespeichert.

Die Hirnforschung geht davon aus, dass der Mensch über mindestens fünf Gedächtnisarten verfügt, die unterschiedliche Zwecke erfüllen. Das *episodische Gedächtnis* rekonstruiert vergangene Erlebnisse inklusive aller Empfindungen und Emotionen. Das *semantische Gedächtnis* speichert faktisches Wissen. Im *Arbeitsgedächtnis* werden die Informationen nur so lange vorgehalten, wie sie auch zur Verwendung kommen. Im *prozeduralen Gedächtnis* werden erlernte Fähigkeiten gespeichert. Auf das Gedächtnis für *unbewusste Erinnerungen* greift der Mensch zurück, ohne dass er sich dessen bewusst ist.

Das Gedächtnissystem wird in diesem Buch als eine Systemkomponente aufgefasst, die nicht nur Daten, Information und Wissen temporär speichert, sondern die auch an der Verarbeitung dieser Konzepte beteiligt ist. Man kann dieses Gedächtnissystem damit durchaus als eine Art Arbeitsgedächtnis (working memory) oder Produktionsgedächtnis (production memory) auffassen. Es umfasst sowohl extern wahrgenommene als auch in-

tern generierte Daten-, Informations- und Wissenskonzepte, die zur Verarbeitung anstehen, und es hält die Ergebnisse kognitiver Operationen fest.

Diese Konzepte entsprechen den Chunks der kognitiven Psychologie. Chunks gelten als Wissenseinheiten, deren Größe jedoch nicht festgelegt ist (z. B. Silben, Wörter, Sätze). Danach können  $7+/-2$  Chunks zu einem Zeitpunkt behalten und wiedergegeben werden. Neuere Befunde zeigen, dass die Gedächtnisspanne nicht nur vom Prozess der Chunkbildung abhängig ist, sondern auch von der Wiederholungsrate des Materials.

Innerhalb des Gedächtnissystems werden vier miteinander verknüpfte Gedächtnisfunktionen realisiert. Das prozedurale Gedächtnis, das semantische Gedächtnis als spezialisiertes Subsystem des prozeduralen Gedächtnisses, und das episodische und autobiographische Gedächtnis, wiederum als spezialisiertes Subsystem des semantischen Gedächtnisses. Das *prozedurale Gedächtnis* (procedural memory) speichert Verknüpfungen zwischen Stimuli und Verhaltensantworten, auch für komplexe Stimuli und Reaktionsketten. Lernen im prozeduralen System erfolgt langsam, gilt als nichtsymbolisch und ist durch beobachtbares Verhalten erfassbar. Das *semantische Gedächtnis* (semantic memory) ist dadurch gekennzeichnet, dass es zusätzlich die interne Repräsentation von Zuständen der Realität ermöglicht, die nicht konkret vorliegen und wahrgenommen werden. Es erlaubt die Konstruktion mentaler Modelle, indem es die Speicherung von Begriffen, abstrakten Inhalten, deren Relationen und Bedeutungen, von Regeln und Algorithmen zur Manipulation von Konzepten und Relationen ermöglicht. Von den Inhalten des semantischen Gedächtnisses wird angenommen, dass sie nicht auf individuelle Erfahrungen bezogen sind, d. h. sie gelten als nicht zeitbezogen. Im Hinblick auf die Organisation des semantischen Gedächtnisses wird ein Netzwerk von Begriffsknoten angenommen, das hierarchisch organisiert ist.

Semantisches und episodisches Wissen umfasst Faktenwissen, das system-intern verarbeitet werden kann. Beide Verarbeitungsfunktionen können auch unter dem Begriff des deklarativen Gedächtnisses (declarative memory) zusammengefasst werden.

Die Verknüpfungen zwischen den Begriffsknoten im Gedächtnis werden als Assoziationen bezeichnet, indem sie über Relationen miteinander verknüpft sind und die verschiedenen Assoziationsarten darstellen und gerichtet sein können. Das *episodische Gedächtnis* (episodic memory) ermöglicht zusätzlich die Speicherung von Wissen über erfahrene Ereignisse und über deren raum-zeitliche Einordnung. Damit wird die Speicherung und Erinnerung zeitlich begrenzter, konkreter Ereignisse sowie von Beziehungen zwischen diesen bezeichnet. Es sind singuläre Ereignisse, an denen das System zu einem bestimmten Zeitpunkt selbst aktiv oder passiv beteiligt war. Ein Ereignis, an dem das System selbst beteiligt war, wird symbolisch repräsentiert, gespeichert und zu einem späteren Zeitpunkt reproduziert. Das *autobiographische Gedächtnis* beinhaltet Daten, Informationen und Wissen, die in Relation zum System selbst stehen. Es wird unterteilt in einen semantischen Teil, der autobiographische Fakten bezüglich des Existenzzyklus des Systems (Bezeichnungen, Architektur, Störungen, Modifikationen, Laufzeit etc.) umfasst und einen episodischen

Anteil, der autobiographische Episoden (Meilensteine, Erfolge, Misserfolge, etc.) enthält. Dem autobiographischen Gedächtnis werden spezifische Charakteristika und Funktionen zugesprochen. Weitere Merkmale betreffen die Emotionalität, die Quellenzuordnung und die Perspektivität. Insbesondere der Emotionalität kommt eine wichtige Rolle zu. Sie wird verantwortlich gemacht für besonders aktivierte Erinnerungen (flashbulb memories), aber auch für verdrängte Erinnerungen (repression). Eine wichtige Funktion manifestiert sich in seinem Gebrauch für die Herstellung und Aufrechterhaltung kommunikativer Beziehungen. Insofern wird das semantische Gedächtnis durch das prozedurale Gedächtnis gestützt, das episodische durch das prozedurale und semantische Gedächtnis. Episodische Inhalte werden rascher vergessen, da ständig neue Informationen eingehen. Semantische Inhalte bleiben über lange Zeit stabil. In diesem Zusammenhang wird das Vergessen als gleichbedeutend mit mangelndem Zugriff auf gespeichertes Wissen realisiert. Als Ursachen für das Misslingen des Erinnerns werden u. a. Zerfall von Gedächtnisspuren (proaktive, retroaktive) Inferenz und Fehlen geeigneter Hinweisreize (retrieval cues) angenommen. Hier greift der Prozess der Enkodierung, in dem Daten, Information und Wissen aufgenommen, transformiert und in Beziehung zu bereits bestehenden Konzepten gesetzt werden. Ebenfalls in diesem Kontext ist die Annahme zu sehen, wonach Erinnerungsleistungen zustandsabhängig sind (state-dependent memory). Danach werden Ereignisse, die in einem bestimmten physikalischen oder psychischen Zustand erfahren wurden, am besten erinnert, wenn diese Zustände zum Zeitpunkt der Erinnerung wieder gegeben sind. Diese multiple Gedächtnisfunktionalität wirkt sich auf die Formen des Lernens aus, indem neben einer *Abstimmung* (tuning) für prozedurales Wissen, der *Umstrukturierung* (restructuring) für semantisches Wissen auch eine *Zunahme* (accretion) für episodisches Wissen vorgesehen ist. Semantisches und episodisches Wissen umfassen neben Faktenwissen, das intern verarbeitet werden kann, auch sogenannte atomare Wissenseinheiten. Als atomare Wissenseinheiten, vor allem im Zusammenhang mit dem semantischen Gedächtnis, werden Begriffe von Entitäten angenommen. Aber es werden auch Schemata als abstrakte, verallgemeinernde Wissenseinheiten, die aus Variablen mit Wertebereichen und Konstanten zusammengesetzt sind, vorgesehen. Solchen Schemata können dann konkrete Instanzen zugeordnet werden. Für Ereignisabläufe sind Schemata in Form von *Skripts* (scripts) vorgesehen.

### 4.3.7 Epigenetisches System

Das *epigenetische System* umfasst sowohl die Fähigkeit aus Erfahrungen zu lernen als auch die Fähigkeit, diese Lernerfahrungen auf die Persönlichkeitsmerkmale auswirken zu lassen. Das epigenetische System verfügt über eine Vielzahl von Prozessen, die zum Erwerb sowie zur Veränderung von Wissen und damit zu einer Veränderung systemischer Fähigkeiten und Verhaltensweisen im Rahmen einer epigenetisch-kognitiven Entwicklung führen.

Der Begriff der Entwicklung im Allgemeinen bezieht sich hier auf die kontinuierlich oder phasenhaft-Intelligenzprofil-alterstypische, universell oder bereichsspezifisch, spontan oder systemisch-determiniert verlaufende, auf ein Ziel hinstrebende quantitative und qualitative Veränderung der Erscheinungs- und Verhaltensformen des Systems. Die kognitive Entwicklung im Speziellen betrifft die während der Laufzeit sich entwickelnde Ausformung der kognitiven Leistungsfähigkeit des Systems, seiner Daten-, Informations-, Wissens- und Verarbeitungsstrukturen in der systemischen Ontogenese. Der Begriff der systemischen Ontogenese bezeichnet dabei die Geschichte des Systems (Entwicklungs- und Laufzeitzyklus) von der Initialisierung, Konzeptionalisierung, Implementierung, Validierung, Laufzeit bis hin zur Deaktivierung oder Löschung des Systems.

Im Rahmen der kognitiven Ausgestaltung von Robotersystemen steht der Erwerb komplexer Daten-, Informations- und Wissensstrukturen, wie beispielsweise Regeln und Schemata im Mittelpunkt des Interesses. Dabei werden die dem Lernen zugrundeliegenden Mechanismen so modelliert, dass ihre Rekonstruktion mit Hilfe von Software im Allgemeinen und einer Brainware im Speziellen im Rahmen des artifiziellen Lernens möglich wird. Ein solches artifizielles Lernen beruht dabei nicht nur auf deduktiven oder induktiven Mechanismen. Das Ziel des artifiziellen Lernens ist, Prinzipien des Erwerbs von Wissen und Fähigkeiten zu untersuchen und operationale Modelle adaptiver, wissensbasierter und lernfähiger Systeme zu entwickeln. Damit ist artifizielles Lernen eng verwandt mit der Wissensakquisition (Knowledge Engineering). Lernen ist aber auch der zielgerichtete Prozess eines Systems, der das Wissen oder die Repräsentation des Wissens im System durch die Exploration von Erfahrungen und Vorwissen verbessert. Eine solche Verbesserung des Wissens eines Systems kann nur durch Integration neuer Lernerfahrungen oder aber durch Vergessen geschehen. Alle anderen Arten von Lernen werden durch Transformation von Repräsentationen erreicht. Lernen zur Steigerung von Effizienz von Problemlösungsprozessen kann z. B. durch die Transformation einer deklarativen Wissensrepräsentation in eine prozedurale Darstellung (z. B. in ein Programm oder in sensomotorische Fähigkeiten) erfolgen.

Die Lernprozesse des Cognitive Computing erfolgen zum einen auf der Grundlage von Erfahrungen aus der Umwelt oder Erfahrungen des Systems mit der Umwelt. Während jedoch viele dieser induktiven Verfahren ausschließlich Erfahrungen verarbeiten, stellt Cognitive Computing auch Verfahren bereit, die (zusätzlich) aus Vorwissen lernen. Diese Lernverfahren dienen der kognitiven Entwicklung. Das System wird dabei als wissensbasiertes System, als Symbol verarbeitendes System bzw. als ein nach Strategien und Regeln sich verhaltendes, selbstmodifizierendes kognitives System aufgefasst. Kognitive Entwicklung bedeutet in dieser Betrachtungsweise zum einen die endogen und/oder exogen angelegte, qualitative und quantitative Veränderung des daten-, informations- und wissensverarbeitenden Systems, seiner Kapazität und Selbst- Steuerungseffizienz, und zum anderen den Aufbau inhaltlicher Daten-, Informations-, Wissens- und Verarbeitungsstrukturen. Dieser Ansatz verfolgt demnach ein Konzept der kognitiven Entwicklung in Form eines entwicklungsbedingten Potenzials, systeminhärente Kapazitäten zu erweitern und systembedingte Restriktionen überwinden zu können. Den präformierten Verarbeitungsstrukturen und Mechanismen (Brainware) stehen so ein artifizielles Bewusstsein als modulare

Struktur spezialisierter „mentaler Komponenten“ zur Seite, das sich als sensitives Eingabe-Verarbeitungs-Ausgabe-System für die Steuerung des Daten-, Informations- und Wissensflusses- und -erwerbs bei problemspezifischen Inhalten im Entwicklungs- und Laufzeitzyklus des Systems im Rahmen der epigenetischen Entwicklung herausbildet.

Die kognitiv-epigenetische Entwicklung als Ontogenese von Daten-, Informations-, Wissens- und Verarbeitungsstrukturen findet letztlich dort statt, wo sich die Selbststeuerungs- und Strukturdynamik des Systems mit den Angeboten und Impulsen seiner Umgebung im Rahmen eines Anregungs- und Interoperationsgefüges verbindet. Insofern kann man hier durchaus von einer systemischen Sozialisation sprechen.

Ein solches *artifizielles Bewusstsein* ist durch eine Art „Kontextualität“ charakterisiert, eine bruchlose und nicht fragmentierte Verbundenheit von Daten-, Informations-, Wissens und Verarbeitungsstrukturen, die ein systemisches Kontinuum entstehen lässt. In dieser Form ist artifizielles Bewusstsein immer Bewusstsein von Kontext. Man kann dies auch als eine spezifische Form inneren Wissens bezeichnen, das die kognitiven Prozesse flankiert. Eine Form inneren Wissens entsteht, wenn die Repräsentationen tragenden inneren Zustände des Systems ihrerseits zum Inhalt höherstufiger Verarbeitungsprozesse und der dadurch bedingten Zustände werden. Insofern geht der Ansatz dieses Buches davon aus, dass ein kognitives System nicht nur deklaratives Wissen über Sachverhalte der Problemdomäne, sondern auch die interoperativen Strukturen zur Verarbeitung dieser Wissensstrukturen hierzu entwickelt. Man behandelt das System als reflexive Verarbeitungseinheit, als eine des Denkens über sein kognitives Funktionieren bzw. zur Steuerung seiner höheren kognitiven Prozesse fähige Entität. Mit Reflexion ist dabei eine systemische Rückwendung des Systems auf sich selbst gemeint, wodurch die inneren Strukturen des Systems zum Gegenstand der Verarbeitung werden. Durch eine solche Reflexion hebt sich das System vom rein verarbeitenden System ab.<sup>8</sup>

Im Gegensatz zur Introspektion, wo eine Selbstbeobachtung aus psychologischen Motiven stattfindet, die sich auf die eigenen Erlebnisse richtet, ist die Reflexion enger gefasst, da sie vor allem auf die eigenen Erkenntnisse ausgerichtet ist und nahezu logische Züge aufweist.

Es kommt also zu einer Art systemischer Bewusstheit und Bewusstsein. Im Kontrast zur intuitiven, anschaulichen Wahrnehmung ist die Reflexion prozesshaft und eng an die Verarbeitungsprozesse des Gesamtsystems gekoppelt. Insofern wird in diesem Buch an dieser Stelle einem Epiphänomenalismus vehement widersprochen. Ein solcher betrachtet Systemphänomene als Wirkungen verarbeitungsbedingter Ereignisse, die selbst jedoch weder kognitive noch interaktive bzw. interoperative Ereignisse verursachen. In dieser Sichtweise sind Verarbeitungsergebnisse in Form von Daten-, Informations- und Wissensstrukturen, sowie die damit induzierten Interaktionen bzw. Interoperationen das Produkt von Vorgängen innerhalb des Systems, zwischen diesen Ergebnissen selbst bestehen aber keine kausalen Zusammenhänge.

---

<sup>8</sup> Siehe auch Newen und Vogeley 2000

### 4.3.8 Intentionales System

Der Ausdruck der Intentionalität wird verwendet, um die Eigentümlichkeit des artifiziellen Bewusstseins zu bezeichnen, auf einen Gegenstand oder Sachverhalt Bezug zu nehmen. Artifizielle Bewusstseinsakte sind durchweg intentionale Akte, kurz Intentionen, mit denen das System auf etwas Bezug nimmt. Die Daten-, Informations- und Wissensstrukturen, die ein Roboter bzw. ein Computer verarbeitet, sind dabei zunächst eine Art von Inhalt. Im Rahmen des intentionalen Systems muss es gelingen, trotz der Komplexität dieser Strukturen durch eine kognitive Verarbeitung eine Intention, ein Meinen entstehen zu lassen.

Das *intentionale System* umfasst dabei die Funktion der Intuition und des planenden Denkens. Beide Funktionen dienen dem reproduktiven bzw. dem produktiven Denken. Während reproduktives Denken nur die Anwendung verfügbaren Wissens oder geübter Strategien auf neue Aufgaben umfasst, erfordert produktives Denken die Berücksichtigung spezifischer struktureller Merkmale der Problemlösekonstellation, aus denen sich aufgrund von Einsicht ein neues Lösungsprinzip erschließt. Das Ergebnis produktiven Denkens ist in der Regel eine neue, bislang unbekannte Sicht eines Problems und seiner Lösung.

*Intuition* bezeichnet eine besondere Form des Erkenntnisgewinns und der Entscheidungsfindung, der zufolge das Erkannte oder die Entscheidung unreflektiert in einem unmittelbaren bzw. unvermittelten Akt Erleben im Bewusstsein gegeben ist.<sup>9</sup> Das mittels Intuition Erkannte wird auf systemischer Seite in der Regel durch ein hohes Maß an systemischer Evidenz begleitet. Intuition erscheint demnach als ein nicht-analytisches, nicht logisches, eher ganzheitliches oder gar zufälliges Ergebnis des Verarbeitungsprozesses. Das Ergebnis der Intuition kann das planende Denken entweder nachhaltig beeinflussen oder gar ausschalten.

Artifizielles Denken im Allgemeinen und *planendes Denken* im Besonderen wird im Rahmen des Cognitive Computing als *Schlussfolgerungsprozess* verstanden. Auf der Basis eingereichter Perzepte und gegebener Informationen werden neue Informationen mit Hilfe von Inferenzen abgeleitet und so die bis dahin aktuellen Perzepte erweitert bzw. die verfügbaren Informationen angereichert.

Inferenz ist ein Prozess zur Realisierung verschiedener Formen von Schlussfolgerungen, durch den aus gegebenen Tatsachen und Annahmen Schlüsse gezogen werden. So besteht das Ziel der *Deduktion* darin, aus einer gegebenen Menge von Fakten logische Konsequenzen herzuleiten. Für die Mathematik heißt das z. B., unter der Annahme von gewissen Axiomen und Definitionen zu beweisen, dass ein Theorem gilt. Die *Abduktion* beschreibt die Suche nach geeigneten Voraussetzungen für einen logischen Schluss. *Induktives Schließen* ist ebenso wie abduktives Schließen nicht notwendigerweise korrekt, sondern eine Form des Alltagsschließens (vom Speziellen zum Allgemeinen). Das Schließen durch *Analogie* ist ebenso wie die

<sup>9</sup> Vgl. Crick und Koch 1992

beiden Schlussweisen zuvor nicht notwendigerweise logisch korrekt. Dabei wird versucht, die Lösung eines Problems mit Hilfe einer bekannten Lösung eines analogen Problems zu finden.

Weiterhin findet eine Anreicherung dieser Perzepte bzw. Informationen mit Handlungs- oder Verhaltensempfehlungen statt. Den Prozess des Hinzufügens von Handlungs- oder Verhaltensempfehlungen, die den angereicherten Inhalt der Perzepte oder Informationen in computationell-verarbeitbarer Form beschreiben, wird als *kognitive Annotation* (Cognitive Annotation) und das Ergebnis dieser kognitiven Annotation als *Wissen* bezeichnet.

Der Schlussfolgerungsprozess umfasst deduktive, induktive als auch analoge Inferenzmechanismen. Charakteristisch für *deduktive Inferenzen* ist, dass aus wahren Prämissen bei Einhaltung der logischen Schlussregeln wahre Folgerungen (Konklusionen) abgeleitet werden, die man als Explikation von in den Prämissen implizit enthaltenen Informationen verstehen kann. *Induktive Inferenzen* zeichnen sich dadurch aus, dass sie mit Unsicherheit behaftet sind, die so lange gelten, wie kein Gegenbeispiel bekannt ist. Insofern kann man in einer allgemeinen Betrachtung alle Inferenzprozesse als induktiv bezeichnen, die unter Unsicherheit und die durch Schätzung von Wahrscheinlichkeiten erfolgen. Bei der allgemeinen Induktion werden aus gegebenen Einzelaussagen oder -ereignissen allgemeine Aussagen abgeleitet, wobei der semantische Informationsgehalt über die gegebenen Aussagen hinaus generalisiert wird. Man kann eine solche allgemeine Induktion auch als Vorgang einer Hypothesenbildung auffassen, der sich dann allerdings eine Hypothesenprüfung und ggf. eine Hypothesenmodifikation anschließen muss. *Analogiebasierte Inferenzen* lassen sich verstehen als Übertragung von Merkmalen, Elementen oder Wissensstrukturen eines bekannten Ausgangsbereichs auf einen ähnlichen, zumindest teilweise unbekannten oder gänzlich neuen Zielbereich, wobei von Details des bekannten Ausgangsbereichs auf den vermeintlich ähnlichen Zielbereich abstrahiert wird. Insofern werden vorhandene Informationen, Perzepte aber auch Wissen über ein Problem für die Bewältigung eines anderen Problems verwendet. Durch diese Übertragung von Merkmalen des bekannten Ausgangsbereichs auf den zunächst unbekannten Zielbereich werden neue Informationen und/oder neue Perzepte erzeugt. Spätestens bei analogiebasierten Inferenzprozessen wird auch die Verbindung zu dem epigenetischen System notwendig. Dieser analogie-orientierte Prozess umfasst die Phasen der Erinnerung an ein ähnliches Beispiel, die Rekonstruktion des erinnerten Beispiels, die Abbildung der Beispieldlösung auf die aktuelle Aufgabe (analoger Schluss) und schließlich die Generalisierung der Lösung.

#### 4.3.9 Effektorisches System

Das *effektorische System* dient der Interoperation durch Akteure mit der Umwelt und damit der Transformation des Wissens auf ein entsprechendes Verhalten.<sup>10</sup> Man spricht auch von dem Verhalten als letzte artifizielle Erkenntnisstufe. Als Verhalten im weiteren Sinne werden alle inneren (intrinsisches Verhalten) und äußeren (extrinsisches Verhalten) Ak-

---

<sup>10</sup> Vgl. Hebb 1949

tivitäten betrachtet, durch das ein artifizielles System mit seiner physischen und sozialen Umwelt interoperiert. Unter Verhalten im engeren Sinne werden alle beobachtbaren und registrierbaren Verhaltensaktivitäten verstanden.

Ein *Effektor* ist in diesem Zusammenhang ein Ausgabeinstrument, durch das ein System Eingriffe in einen zu steuernden Prozess und damit in die Umgebung vornehmen kann.

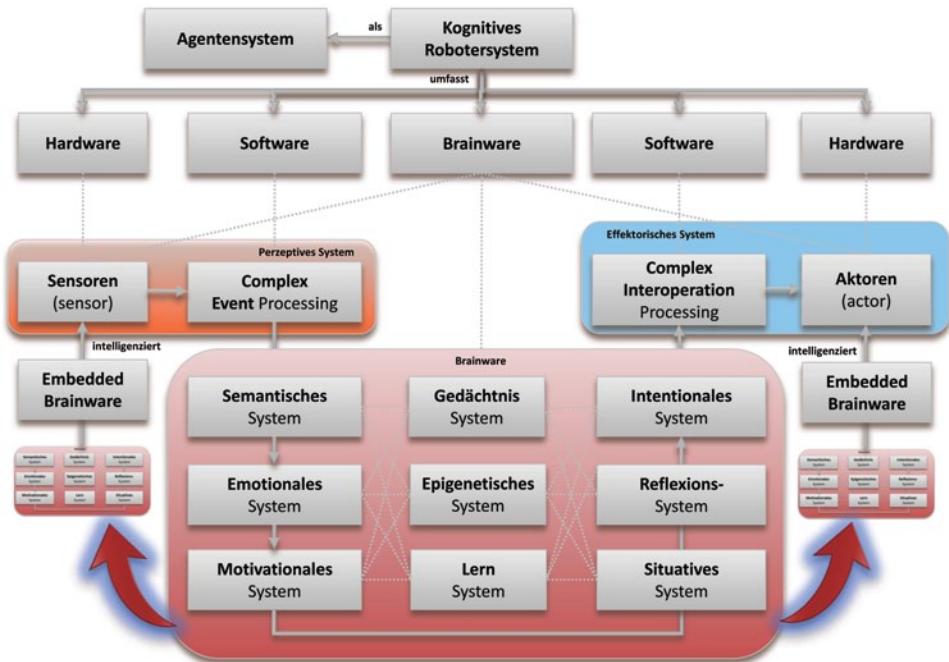
In der Neurobiologie ist ein Effektor ein Erfolgsorgan, das über Efferenzen (Afferenz-Efferenz) gesteuert wird, d. h. hier ist der „Rechner“ das Nervensystem und die Effektoren sind Muskeln oder Drüsen.

In der Robotik arbeiten Effektoren häufig elektromechanisch, d. h. die Ausgabesignale des kognitiven Systems steuern entweder direkt oder über Digital-Analogwandler mechanische Stellglieder, beispielsweise Motoren, die entsprechende Bewegungen ausführen.

Mit „Analog“ ist hier im Gegensatz zu „Digital“ eine kontinuierliche und stetige Verarbeitung von physikalisch messbaren Größen bezeichnet. Damit gestaltet sich der Verarbeitungsmodus im ersten Fall nicht mehr zweiseitig unstetig, sondern kontinuierlich stetig.

Der Übergang zwischen intentionalem und effektorischem System betrifft auch Fragestellungen der Motorik, ein interdisziplinäres Forschungsgebiet, das sich im Überlappungsbereich von Physiologie, Biologie, Neurologie, Psychologie, Robotik, Sportwissenschaft und Arbeitswissenschaft etabliert hat. Sie befasst sich mit der auch für das Cognitive Computing wichtigen Frage bezüglich der Umsetzung einer motivierten Intention in ein entsprechendes Verhalten, wobei gleichzeitig noch die Randbedingungen seitens der am kognitiven Prozess beteiligten Komponenten erfüllt werden müssen. Der Übergang von der Intention als Ergebnis des intentionalen Systems in Form des gewünschten distalen Ereignisses zu den motorischen Kommandos kann auf prinzipiell unterschiedlichen Arten erfolgen. Zum einen kann es zu bestimmten distalen Ereignissen innere Repräsentationen geben, von denen die entsprechenden motorischen Kommandos festgelegt werden. Solche motorischen Repräsentationen werden im Allgemeinen als Verhaltensprogramme bezeichnet. Zum anderen können distale Ereignisse direkt in motorische Programme transformiert werden. Im Regelfall werden jedoch im Rahmen der kognitiven Robotik die Bewegungen mehrfach und flexibel über Repräsentationen repräsentiert. Mit Hilfe solcher Repräsentationen lassen sich auch im Vorfeld des konkreten Verhaltens Verhaltensvorstellungen entwickeln, um diese notfalls noch zu ändern, was sich dann anschließend beim terminalen Verhalten zeigt (Abb. 4.11).

Im Rahmen der kognitiven Robotik wird dieser Ansatz um einen senso-motorischen Ansatz erweitert. Der Kontrolle von Gleichgewicht, Körperhaltung und Bewegung kommt im Rahmen der kognitiven Robotik eine zentrale Bedeutung zu und ist stark von sensorischer Information abhängig. Ebenso werden die Wahrnehmungsprozesse durch motorisches Verhalten erleichtert oder erst möglich. Die Integration von Sensorik und Motorik reicht von simplen Reflexen bis zur Meisterung komplexer Willkürbewegungen mit Hilfe



**Abb. 4.11** Kognitives Robotersystem mit Embedded Brainware

eines gut entwickelten Körpergefühls. Um letzteres zu ermöglichen, bedarf es unter Umständen der Verlagerung der kognitiven Leistungen in die perzeptiven und effektorischen Systeme eines Robotersystems. Konkret bedingt dies eine Erweiterung der Sensorik und die Aktorik um eine sogenannte embedded Brainware.

## Literatur

- Boden, M.: Auf den Flügeln des Geistes. Kreativität und Künstliche Intelligenz. Artemis & Winkler, München (1992)
- Chalmers, D.J.: Das Rätsel des bewußten Erlebens. Spektrum der Wissenschaft, (1996)
- Crick, F., Koch, C.: Das Problem des Bewußtseins. Spektrum der Wissenschaft, 11, (1992)
- Eimer, M.: Informationsverarbeitung und mentale Repräsentation. Springer, Berlin (1990)
- Hebb, D.: The Organization Of Behavior. Wiley, New York (1949)
- Kail, R., Pellegrino, J.: Menschliche Intelligenz. Spektrum der Wissenschaft Verlagsgesellschaft, Heidelberg (1989)
- Newel, A.: Unified Theories of Cognition. Harvard University Press, Cambridge. 3. Aufl. (1999)
- Newen, A., Vogeley, K. (Hrsg.): Selbst und Gehirn. Mentis, Paderborn (2000)
- Voß, R.: Einführung in die Künstliche Intelligenz. Düsseldorf (1985)
- Weizenbaum, J.: Kurs auf den Eisberg. Pieper, München (1987)

---

## 5.1 Hardware

In diesem Buch werden in Bezug auf die Mechanik und Kinematik folgende Teilsysteme unterschieden:

- Achsregelung und Antrieb (Motoren, Stromversorgung)
- Sensoren
- Steuerung
- Programmierung

Die *Mechanik* des Roboters bestimmt seine Fähigkeit, in seiner Umgebung mit dem (End) effektor eine definierte Position und Orientierung einzunehmen bzw. eine Bewegungsbahn (Trajektorie) im dreidimensionalen euklidischen Raum abzufahren. Die *Kinematik* beschäftigt sich hingegen mit der Geometrie und den zeitabhängigen Aspekten der Bewegung, ohne die Kräfte, die die Bewegung verursachen, in die Überlegungen mit einzubeziehen.<sup>1</sup> Dabei spielen vor allem die folgenden Parameter, für die Kinematik eine wesentliche Rolle:

- Position (Verschiebung, Drehung)
- Geschwindigkeit
- Beschleunigung
- Zeit<sup>2</sup>

---

<sup>1</sup> Siehe auch Kovács 1993.

<sup>2</sup> Siehe auch Schwinn 1992.

Die *Kinematik* legt die Beziehung zwischen der Lage des Effektors bezüglich der Roboterbasis und der Einstellung der Gelenkparameter fest. Parameter zur Beschreibung der Gelenke sind Drehwinkel bzw. Translationswege.<sup>3</sup>

### 5.1.1 Achsregelung und Antrieb

Die *Regelung von Achsen* bzw. der Einbau von *Antrieben* bewirken die Fortbewegung und die Bewegung der Glieder des Roboterarms bzw. des Effektors. Durch den Antrieb wird die erforderliche Energie auf die Bewegungsachsen übertragen. Der Antrieb muss auch die Kräfte und Momente durch das Gewicht der Glieder des Roboters und der Objekte im Effektor kompensieren. Es wird demnach Energie auch dann benötigt, wenn der Roboter sich nicht bewegt. Es lassen sich drei Antriebsarten unterscheiden:

- *Pneumatische* Antriebe:
  - komprimierte Luft bewegt Kolben, kein Getriebe
  - billig, einfacher Aufbau, schnelle Reaktionszeit, auch in ungünstigen Umgebungen brauchbar
  - laut, keine Steuerung der Geschwindigkeit bei der Bewegung, nur Punkt-zu-Punkt-Betrieb, schlechte Positioniergenauigkeit
  - Einsatz für kleinere Roboter mit schnellen Arbeitszyklen und wenig Kraft, beispielsweise zur Palettierung kleinerer Werkstücke
- *Hydraulische* Antriebe:
  - Öldruckpumpe und steuerbare Ventile
  - sehr große Kräfte, mittlere Geschwindigkeit
  - laut, zusätzlicher Platz für Hydraulik, Ölverlust führt zu Verunreinigung, Ölviskosität erlaubt keine guten Reaktionszeiten und keine hohen Positionier- oder Wiederholgenauigkeiten
  - Einsatz für große Roboter, beispielsweise zum Schweißen
- *Elektrische* Antriebe:
  - Motoren

### 5.1.2 Sensoren

Ein Robotersystem steht in enger Wechselbeziehung zu seiner Aufgabenstellung und seiner Umgebung. Das Verhalten eines Roboters ergibt sich damit auch aus seiner Interoperation mit seiner Umgebung. *Sensoren* sind Vorrichtungen, die die physikalischen Eigenschaften der Umgebung, wie Temperatur, Helligkeit, Berührungs widerstand, Gewicht, Größe usw. wahrnehmen und messen können. Sie liefern die Rohdaten über die Arbeitsumgebung des

---

<sup>3</sup> Siehe auch Husty et al. 1997.

Robotersystems. Diese Information ist oft verrauscht, d. h. unpräzise widersprüchlich oder mehrdeutig. Die Steuerung und Auswertung der Sensordaten kann große Rechnerleistungen erfordern.<sup>4</sup>

Den Sensoren obliegen folgende Funktionen:

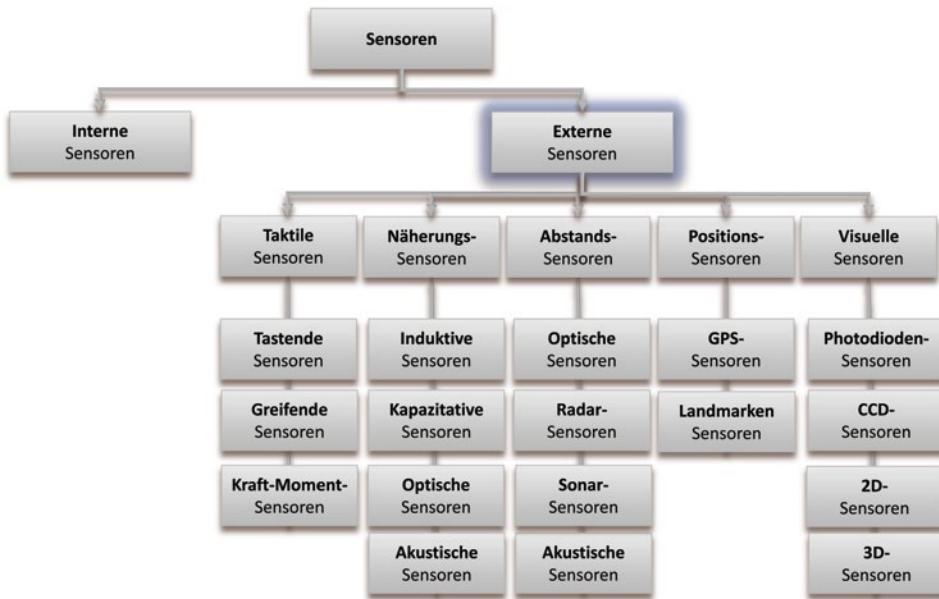
- Erfassung der inneren Zustände des Roboters (beispielsweise Lage, Geschwindigkeit, Kräfte, Momente)
- Erfassen der Zustände der Handhabungsobjekte und der Umgebung
- Messen physikalischer Größen
- Identifikation und Zustandsbestimmung von Werkstücken und Wechselwirkungen
- Analyse von Situationen und Szenen der Umwelt

Man unterscheidet dabei 3 Hauptkategorien:

- *Interne Sensoren*: messen Zustandsgrößen des Roboters selbst
  - Position und Orientierung des Roboters selbst
  - Messung des Batteriestands
  - Temperatur im Innern des Roboters
  - Motorstrom
  - Stellung der Gelenke
  - Geschwindigkeit, mit der sich Gelenke bewegen
  - Kräfte und Momente, die auf die Gelenke einwirken (beispielsweise Positionssensoren, Kraftsensoren, Rad-Encoder, Kreisel, Kompass)
- *Externe Sensoren*: erfassen Eigenschaften der Umwelt des Roboters
  - Licht
  - Wärme
  - Schall
  - Kollision mit Hindernissen
  - physikalische Größen im technischen Prozess
  - Entfernung
  - Lage von Positioniermarken und Objekten
  - Kontur von Objekten
  - Pixelbilder der Umwelt (beispielsweise Lichtsensoren, Wärmesensoren, Abstandsmesser, Schallsensoren, Kameras, Mikrophone)
- *Oberflächensensoren*: beispielsweise Tastsensoren

Sensoren haben die spezielle Aufgabe, bestimmte physikalische Eigenschaften zu messen, die normalerweise in einem sinnvollen Bezug zu den physikalischen Eigenschaften der unmittelbaren Umgebung stehen. Bezüglich der physikalischen Eigenschaften von Kraft und Moment lässt sich ein Robotersystem mit Sensoren mit der Funktion einer Handwurzel

<sup>4</sup> Vgl. Everett 1995, S. 45 ff.



**Abb. 5.1** Übersicht über externe Sensoren

ausstatten, die in der Lage sind, die auf einen Effektor wirkenden Kräfte und Momente aufzuzeichnen, um dann Kräfte und Momente wiederum vorzugeben. Solange dann beispielsweise die Kraft in einer Raumrichtung unterschritten wird, wird den entsprechenden Positionsreglern des Robotersystems aufgetragen, den Arm weiter in diese Richtung zu drücken. Diese Kraft auf das Objekt erhöht sich in Abhängigkeit von der Steifigkeit der Armsegmente. Eine elegantere Möglichkeit besteht darin, die einzelnen Gelenke des Robotersystems nicht nach Position zu regeln, sondern nach dem aufgebrachten Moment. Die erforderlichen Momentensensoren sind allerdings deutlich aufwendiger als die Positions-sensoren zu realisieren.

Wenn die Position des zu manipulierenden Objekts nicht exakt bekannt ist oder wenn nicht mit Sicherheit gewährleistet werden kann, dass das Objekt zum Zeitpunkt der Aktion vorhanden ist, benötigt man Sensoren, die die physikalischen Eigenschaften des Abstands und der Annäherung regeln. Mit Abstandssensoren können in vorzugebenem Rahmen die Annäherungsrichtung und -geschwindigkeit und der Griffpunkt variiert werden. Solche Abstandssensoren können sehr einfach aufgebaut sein, wie im Falle von induktiven Näherungsschaltern, die nur binäre Ausgangswerte liefern, oder sie können komplizierte Sensoren, wie räumlich abtastende Infrarotlaser-Laufzeitmesser darstellen und komplett Landschaftskarten erstellen (Abb. 5.1).

Die *Kollisionsvermeidung* stationärer oder mobiler Roboter mit statischen Hindernissen kann zur Programmierzeit des Roboters erfolgen. Sobald man aber mit dynamischen Hindernissen konfrontiert ist, die sich zur Laufzeit mit unbekannter Geschwindigkeit im Arbeitsbereich des Roboters bewegen, benötigt man Sensoren, die die Parameter dieser

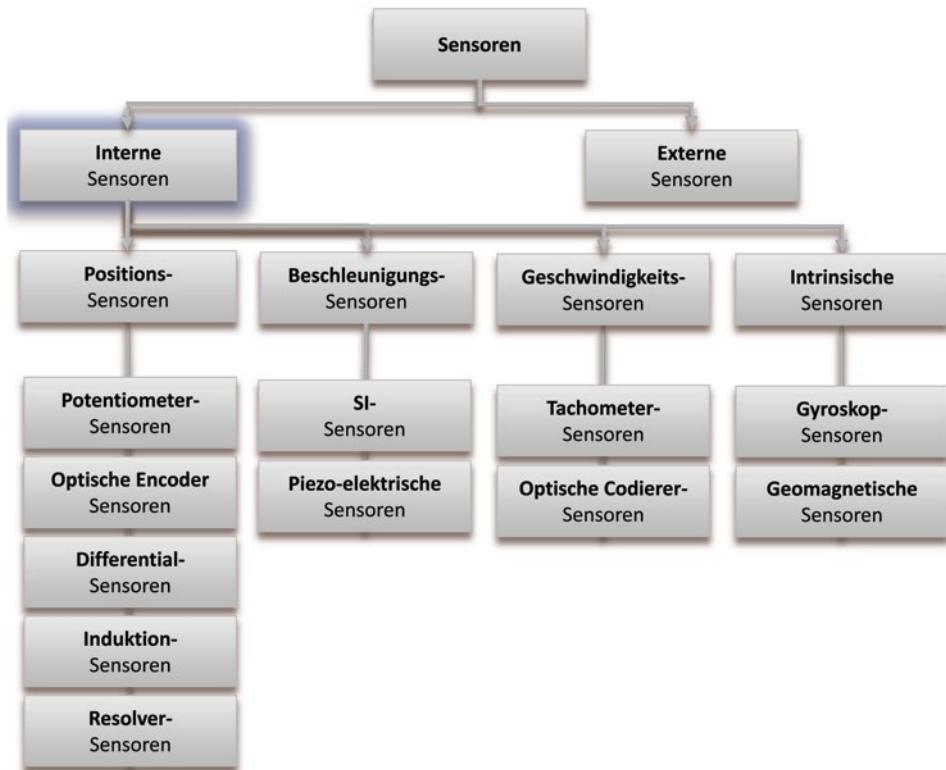
Hindernisse, also Größe und momentane Geschwindigkeit erfassen. Hier kommen wieder abtastende Lasersensoren in Frage oder aber Kameras, die mit Hilfe unterschiedlicher Auswertungsverfahren (Stereo, optischer Fluss etc.) die erforderlichen Objektparameter bestimmen. Oft ist auch nicht genau bekannt, an welcher Stelle sich ein Objekt zum Zeitpunkt einer vom Roboter auszuführenden Aktion befindet. Dann sind ebenfalls Sensoren zur Bestimmung der den Griff bestimmenden Parameter in Bezug auf eine eventuelle *Objektverfolgung* erforderlich. Besonders im Fall schnell beweglicher Objekte stellt dies die Bildverarbeitungsroutinen vor erhebliche Geschwindigkeitsanforderungen.

Bei einem höheren Grad an Autonomie von Robotersystemen steigen auch dessen Freiheiten, sich Objekte selbst auszusuchen, um beispielsweise Interoperationen in Abhängigkeit von der relativen Lage vorhandener Teile zu optimieren. Zu solch einer *Objektauswahl* muss die Lage der Teile von Sensoren bestimmt werden. Dies erfordert, dass der Sensor alle Teile einer Teilemenge in allen möglichen Lagen identifiziert und darauf aufbauende Planungsfähigkeiten zu einer tatsächlichen Sequenz kommen, deren Teilaktionen der Sensor dann auch überwacht. Insofern lassen sich Sensoren durch eine Reihe von Eigenschaften charakterisieren, die ihre Leistungsfähigkeit beeinflussen. Die wichtigsten Eigenschaften sind:

- *Empfindlichkeit*: Wie genau entsprechen Veränderungen des Ausgangssignals den Veränderungen in den Eingangsdaten des Sensors?
- *Linearität*: Wie konstant ist das Verhältnis von Eingangs- und Ausgangsdaten des Sensors?
- *Meßbereich*: Wie groß ist die Spanne zwischen den noch messbaren Mindest- und Höchstwerten?
- *Reaktionszeit*: Wie schnell schlägt sich eine Veränderung im Eingangssignal in den Ausgangsdaten nieder?
- *Genauigkeit*: Wie genau stimmen die gemessenen mit den tatsächlichen Werten überein?
- *Wiederholgenauigkeit*: In welchem Maße stimmen aufeinanderfolgende Messungen der gleichen Entität überein?
- *Auflösung*: Was ist die kleinste feststellbare Schrittweite beim Eingangssignal?
- *Art der Ausgangsinformation*: Welche Form von Informationen wird an die Umgebung zurückgeliefert?

*Haptische Sensoren* entsprechen dem Tastsinn beim Menschen. Solche Sensoren können physikalischen Kontakt mit einem Objekt feststellen. Genauer gesagt messen sie eine physikalische Veränderung, wie beispielsweise das Umlegen eines Schalters, die normalerweise durch physikalischen Kontakt mit einem Objekt hervorgerufen wird. Auch hier muss ein gewisses Rauschen berücksichtigt werden, da der physikalische Kontakt auch durch einen fehlerhaften Sensor, Vibration usw. verursacht werden kann.

Die einfachsten haptischen Sensoren in Form von Berührungssensoren sind Mikroschalter oder Fühlersensoren. Wenn eine Stoßleiste mit einem Objekt in Berührung



**Abb. 5.2** Übersicht über interne Sensoren

kommt, wird dadurch ein Mikroschalter geschlossen und schließt einen Stromkreis, was vom Rechner des Robotersystems erkannt wird. Andere einfache Berührungssensoren beruhen zum Beispiel auf Dehnungsmessstreifen oder piezoelektrischen Sensoren. Dehnungsmessstreifen bestehen aus einer dünnen Widerstandsschicht, die auf einem flexiblen Trägermaterial aufgebracht ist. Wird das Trägermaterial durch äußere Kräfte gebogen, wird die Schicht entweder gestreckt, wodurch sich der Widerstand erhöht, oder aber komprimiert, wodurch sich folglich der Widerstand entsprechend verringert. Insofern liefern solche Dehnungsmesser mehr Information als Sensoren, die lediglich auf binären Mikroschaltern beruhen. Piezoelektrische Wandler sind Kristalle, beispielsweise Quarz oder Turmalin, die elektrische Ladungen erzeugen, wenn sie entlang ihrer sensitiven (polaren) Achse deformiert werden. Diese Ladungen können als kurze Spannungsstöße gemessen werden, wobei die Amplitude des jeweiligen Spannungsstoßes dabei einen Hinweis auf die Stärke der Deformation gibt. Kontaktssensoren, wie Dehnungsmessstreifen, können in zweidimensionalen Feldern zu berührungsempfindlichen Sensorflächen angeordnet werden, die nicht nur das Vorhandensein eines Objekts erkennen, sondern auch dessen Form und Größe messen können (Abb. 5.2).

*Infrarotsensoren* sind wohl die einfachsten der kontaktlosen Sensoren und werden in der mobilen Robotik vielfach in der Hinderniserkennung verwendet. Infrarotsensoren senden dabei ein infrarotes Lichtsignal aus, das von den Oberflächen eines Objekts in der Umgebung des Roboters reflektiert und von den Robotersensoren wiederaufgenommen wird. Um das ausgesandte Infrarotsignal von eher allgemeinen Infrarotsensoren in der Umgebung (Leuchtstoffröhren oder Sonnenlicht) zu unterscheiden, wird das Signal meist mit einer niedrigen Frequenz (beispielsweise 100 Hz.) moduliert. Vorausgesetzt, dass alle Objekte in der Umgebung des Roboters die gleiche Farbe und Oberfläche besitzen, kann man Infrarotsensoren so einstellen, dass sie die Distanz zu Objekten messen. Dabei gilt der mathematische Zusammenhang: die Intensität des reflektierten Lichts ist umgekehrt proportional zum Quadrat der Distanz. Natürlich haben in realistischen Umgebungsverhältnissen die Oberflächen der Objekte verschiedene Farben, die jeweils mehr oder weniger Licht reflektieren. Schwarze Oberflächen sind für Infrarotsensoren praktisch unsichtbar. Aus diesem Grund können sie eigentlich nur für Objekterkennung, aber nicht für Entfernungsmessung verwendet werden.

*Sonarsensoren* basieren auf dem gleichen Prinzip, das auch von Fledermäusen verwendet wird: ein kurzer, ca. 1,2 Millisekunden langer Impuls von verschiedenen Frequenzen wird ausgestoßen und dessen Reflexion durch vorausliegende Objekte mittels eines Empfängers aufgenommen. Die Empfindlichkeit eines Sonarsensors ist nicht gleichmäßig über den überstrahlten Bereich verteilt, sondern besteht aus keulenförmigen Mittel- und Seitenfeldern. Da die Schallgeschwindigkeit bekannt ist, kann die Entfernung des den Impuls reflektierenden Objekts mittels der Zeitspanne zwischen Aussenden und Empfang ausgerechnet werden. Dabei gilt die mathematische Formel:

$$d = \frac{1}{2} vt$$

wobei  $d$  die Distanz zum nächsten Objekt innerhalb des Sonarkegels ist,  $v$  die Geschwindigkeit des ausgesandten Signals im Medium ( $344 \text{ ms}^{-1}$  für die Geschwindigkeit von Schall in Luft mit der Temperatur  $20^\circ\text{C}$ , und  $t$  die Zeitspanne zwischen Aussenden des Impulses und Empfang seines Echos.

Bei der Verwendung von Sonarsensoren können einige Ungenauigkeiten auftreten. Erstens ist die präzise Position des erkannten Objekts unbekannt. Da der Empfindlichkeitsbereich eines Sonarsensors keulenförmig ist, kann sich ein mit Distanz  $d$  aufgenommenes Objekt in Wirklichkeit in jeder beliebigen Position innerhalb des Sonarkegels befinden, die auf einem Bogen mit Distanz  $d$  vom Robotersystem liegt. Die Genauigkeit der Positionsmessung für ein Objekt ist eine Funktion der Keulenbreite der Sonarstrahlstruktur. Zweitens verursachen sogenannte Spiegelreflexionen oder Totalreflexionen fehlerhafte Messungen. Spiegelreflexionen entstehen, wenn der Sonarstrahl in einem flachen Winkel auf eine glatte Oberfläche auftrifft und deshalb nicht zum Robotersystem zurückkehrt, sondern nach außen abgelenkt wird. Nur wenn dann ein weiter entferntes Objekt das Signal in Richtung Robotersystem reflektiert, wird überhaupt ein Wert gemessen, der dann allerdings einen größeren freien Raum angibt, als tatsächlich vorhanden ist. Ein weiteres

Problem kann auftauchen, wenn ganze Reihen oder Felder von Sonarsensoren verwendet werden. Wenn die Sonarsensoren nicht spezifisch verschlüsselte Signale aussenden, kann sogenanntes „Übersprechen“ (*Crosstalk*) auftreten, wobei ein Sensor das reflektierte Signal eines anderen Sensors aufnimmt und damit falsche Entfernungsmessungen erhält. Somit sind je nach Situation sowohl zu geringe wie auch zu große Messergebnisse möglich. Die angenommene Geschwindigkeit von Schall bestimmt die gemessene Entfernung. Ungünstigerweise ist die Schallgeschwindigkeit so sehr von Lufttemperatur und Luftfeuchtigkeit abhängig, dass eine Veränderung der Temperatur um beispielsweise 16°C einen Meßfehler von 30 cm über eine Entfernung von 10 m bewirken würde. Ein solcher Messfehler hat zwar keinen Einfluss auf einfache Sensor-Motor-Verhaltensweisen, wie Hindernisausweichen, würde jedoch für Kartenerstellungsaufgaben des Roboters eine sehr unpräzise Umgebungskartographie zur Folge haben.

*Laserentfernungsmeister* werden heute häufig in der mobilen Robotik eingesetzt, um die Entfernung, Geschwindigkeit und Beschleunigungsrate von wahrgenommenen Objekten zu messen. Das Prinzip ist das gleiche wie bei Sonarsensoren: statt eines kurzen Schallimpulses wird beim Laser ein kurzer Lichtimpuls ausgesandt und die Zeitspanne zwischen dem Aussenden und dem Empfang des reflektierten Impulses für die Entfernungsberechnung verwendet. Auch hier gilt der mathematische Zusammenhang:

$$d = \frac{1}{2} vt$$

wobei  $d$  die Distanz zum nächsten Objekt ist,  $v$  die Geschwindigkeit des ausgesandten Signals im Medium und  $t$  die Zeitspanne zwischen Aussenden des Impulses und Empfang seines Echos. Der Unterschied liegt in der Natur des ausgesandten Impulses. Die Wellenlänge des Laserimpulses liegt im Bereich von Infrarotlicht und ist wesentlich kürzer als die eines Sonarimpulses. Diese kürzere Wellenlänge verringert die Wahrscheinlichkeit, dass das Signal von einer glatten Oberfläche symmetrisch zur Seite wegreflektiert wird, so dass das Problem der Spiegelreflektion weniger ausgeprägt ist. Die maximale Reichweite für kommerziell erhältliche Laserentfernungsmeister beträgt oft mehrere hundert Meter. Je nach Anwendungsgebiet empfiehlt es sich allerdings, eine geringere maximale Reichweite zu wählen. Während eine maximale Reichweite von mehreren hundert Metern für Außenanwendungen sinnvoll ist, sind normalerweise Reichweiten von unter hundert Metern für den Einsatz in Gebäuden völlig ausreichend. Die Genauigkeit von kommerziell erhältlichen Laserentfernungsmeistern liegt gewöhnlich im Millimeterbereich (beispielsweise 50 mm für eine einzelne Abtastung, 16 mm für den Durchschnitt von neun Abtastungen), wobei die Abtastzeit für einen Winkel von 180 Grad 40 Millisekunden beträgt. An Stelle des Laufzeitprinzips kann für die Entfernungsmessung auch die Phasendifferenz zwischen einem ausgesandten Signal und seinem Echo verwendet werden. Hierfür wird die Intensität des Lasersignals zeitabhängig moduliert. Wegen der Laufzeit des Laserstrahls zwischen Aussendung und Rückkehr zum Sensor ergibt sich eine Phasendifferenz zwischen ausgesandtem und empfangenem Lasersignal. Die Phasendifferenz ist dabei proportional zu Distanz und Modulationsfrequenz und kann daher zur Entfernungsberechnung von Objekten verwendet werden.

Die Funktionsweise von *Radar* (Radio Detection and Ranging) beruht ebenfalls darauf, dass ein Impuls ausgesendet und seine Laufzeit gemessen wird, bis er wieder vom Sensor empfangen wird. Da die Geschwindigkeit der ausgesandten Wellen bekannt ist, kann auch hier anhand der Laufzeit die Distanz des Objekts errechnet werden, durch das der Impuls reflektiert wurde. Beim Radar wird ein Funkfrequenzimpuls ausgesandt (im GHz-Bereich) und für die Objekterkennung und Distanzmessung verwendet.

Wenn ein stromführender Leiter in ein magnetisches Feld eingeführt wird, entsteht ein elektrisches Feld, das orthogonal zum magnetischen Feld und zum Strom steht. Dieser sogenannte Hall-Effekt, benannt nach seinem Entdecker, Edwin Herbert Hall (1879), kann dafür verwendet werden, ein magnetisches Feld zu erzeugen. Dieser Effekt wird in der mobilen Robotik oft für Bewegungssensoren oder Hall-Effekt-Kompassen verwendet. Man kann einen einfachen Umdrehungszähler bauen, indem man in das Rad des mobilen Roboters einen Permanentmagneten und einen *Hall-Sensor* in den Roboterkörper einbaut. Jedesmal, wenn sich der Magnet im Rad am Hall-Sensor vorbeibewegt, erkennt dieser die Veränderung im magnetischen Feld. Mit zwei Hall-Sensoren können sowohl die Geschwindigkeit als auch die Richtung der Umdrehung bestimmt werden: die zeitliche Auseinanderfolge ihrer Signale zeigt die Umdrehungsrichtung, und die Frequenz der Signale die Geschwindigkeit des Rades an.

*Kompasssensoren* sind für Navigationsanwendungen von hoher Bedeutung und kommen vielfach zum Einsatz. Kompassmechanismen, die in Sensoren für Robotersysteme verwendet werden, messen die horizontale Komponente des natürlichen magnetischen Feldes der Erde, ähnlich wie ein gewöhnlicher Wanderkompass. Das in der Robotik am häufigsten verwendete Kompassprinzip ist das des Halbleiterkompasses, der die magnetische Feldstärke misst, indem er die Eigenschaften eines Elektromagneten kontrolliert verändert. Eine Feld- und eine Sensorspule werden dabei um einen gemeinsamen Spulenkern gewickelt. Wird ein stark durchlässiger ungesättigter Spulenkern in ein magnetisches Feld eingeführt, werden die magnetischen Feldlinien in den Spulenkern hineingezogen. Wenn allerdings der Spulenkern gesättigt ist, bleiben die Feldlinien unverändert. Durch wechselweise Sättigung und Entsättigung des Spulenkerns ändert man den magnetischen Fluss durch den Kern, und eine Spannung wird in der Sensorspule induziert. Diese Spannung hängt vom umgebenden magnetischen Feld ab und gibt daher dessen Stärke an. Um die Richtung für den magnetischen Norden zu ermitteln, werden zwei Spulen benötigt, die rechtwinklig zueinander stehen. Der den jeweiligen Spulenstrom bestimmende Winkel  $\Theta$  ergibt sich aus den beiden Spannungen  $V_x$  und  $V_y$ , die in den beiden Sensorspulen gemessen werden:

$$\Theta = \arctan(V_x / V_y)$$

Zwei rechtwinklig zueinander angeordnete Hall-Sensoren können ebenso als Magnetkompass verwendet werden. Hier wird der Winkel  $\Theta$  mittels der beiden Komponenten  $B_x$  und  $B_y$  des Magnetfelds errechnet:

$$\Theta = \arctan(B_x / B_y)$$

Alle magnetischen Kompasssensoren reagieren unabhängig von ihrer zugrundeliegenden Funktionsweise auf das relativ schwache Magnetfeld der Erde. In Innenräumen sind Verzerrungen der Messergebnisse durch Metallobjekte, wie beispielsweise Stahlträger, Stahlbeton, Metalltüren oder künstliche Magnetfelder, beispielsweise durch Stromleitungen, elektrische Motoren usw. allerdings unvermeidbar. Aus diesem Grund kann ein Kompass nicht in allen Fällen einen korrekten absoluten Bezugspunkt bieten. Trotzdem kann er als lokale Referenz verwendet werden, solange die äußersten Bedingungen über eine gewisse Zeitspanne hinweg konstant sind. Unabhängig davon, ob die Kompassmessung tatsächlich absolut gesehen korrekt ist, bleibt sie doch am selben Ort jedesmal relativ die gleiche und kann somit als lokale Referenzangabe benutzt werden. Für Navigationsanwendungen, bei denen topologische Karten verwendet werden, ist ein Kompass trotz dieser Nachteile sehr nützlich.

In der Robotik ist es oft notwendig, Umdrehungen zu messen. Um beispielsweise Wegintegration durchzuführen, misst man meist die Umdrehungen der Roboterräder. Zu diesem Zweck verwendet man sogenannte *Potentiometer*, das sind mechanisch variable Widerstände, oder aber sogenannte Winkelgeber. Ein Winkelgeber ist eine Vorrichtung, die an der rotierenden Welle angebracht wird und die Position der Welle in binäre Information umwandelt. Bei solchen Winkelgebern existieren zwei Hauptfunktionsweisen: absolut und inkrementell. Bei absoluten Winkelgebern wird eine mit *Gray code* (einem binären Code, bei dem sich zwischen aufeinanderfolgenden Sequenzen nur ein Bit verändert) versehene Scheibe von einem optischen Sensor gelesen und so die Position der Welle angegeben. Absolute Kodierung gibt die augenblickliche Position der Welle an, ist aber nicht sehr gut dafür geeignet, Bewegung über Zeit zu integrieren. Hierfür wird die inkrementelle Kodierung eingesetzt. Dabei wird die mit *Gray code* versehene Scheibe von zwei Photorezeptoren A und B abgelesen. Spur A geht stets der Spur B voraus, wenn die Scheibe in die eine Richtung rotiert, dagegen ist Spur B voraus, wenn die Scheibe sich in die entgegengesetzte Richtung dreht. Die Reihenfolge, in der die beiden Photozellen ansprechen, gibt also die Umdrehungsrichtung an und die Zahl der An/Aus-Durchgänge jedes Rezeptors gibt den überstrichenen Winkel an. Aus diesen Informationen wird dann die Bewegung der Welle korrekt errechnet. Neben rotierenden Wellen verwenden Roboter auch oft Wellen, die sich seitwärts bewegen, zum Beispiel in Manipulatoren (Robotarmen). Zur Messung von Querverschiebungen können linear variable Differentialtransformatoren verwendet werden.

Für ein sich bewegendes Robotersystem ist von zentraler Bedeutung, die eigene Bewegung mit Hilfe von *Bewegungssensoren* messen zu können. Laufzeit-Entfernungsmesser können in ihrer Funktion so erweitert werden, dass sie die relative Geschwindigkeit zwischen dem Sender des Signals und dem das Signal reflektierenden Objekts messen können, indem sie den Dopplereffekt verwenden. Die folgende Gleichung ergibt das Verhältnis zwischen der Frequenz, des ausgesandten Impulses  $f_{\text{sender}}$ , der Frequenz des empfangenen Impulses  $f_{\text{receiver}}$ , der Geschwindigkeit  $v_{\text{sender}}$  des Senders und der Geschwindigkeit  $v_{\text{receiver}}$  des Empfängers. Dabei bedeuten positive Geschwindigkeiten eine Bewegung in Richtung der Schallquelle). Die Variable  $c$  repräsentiert die Lichtgeschwindigkeit.

$$f_{\text{receiver}} = f_{\text{sender}} \left[ (c - v_{\text{receiver}}) / (c + v_{\text{sender}}) \right]$$

Aus dieser Gleichung wird deutlich, dass die relative Geschwindigkeit zwischen Sender und Empfänger

$$V_{\text{receiver}} = V_{\text{sender}} - V_{\text{receiver}}$$

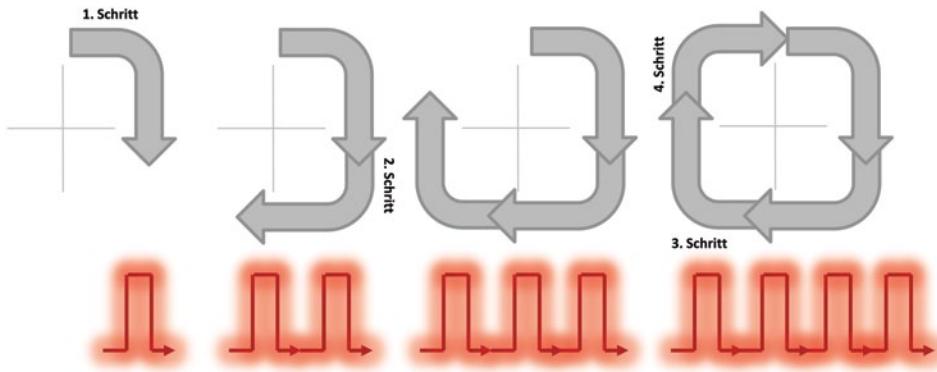
errechnet werden kann, wenn Meßdaten für  $f_{\text{receiver}}$  und  $f_{\text{sender}}$  vorliegen.

In der Robotik kommen oftmals sogenannte CCD-Kameras (Charge Coupled Devices) als *Bildsensoren* zum Einsatz. Sie verwenden ladungsgekoppelte Halbleiterbausteine, um numerische Matritzen zu erzeugen, die der Graustufenverteilung einer Abbildung entsprechen. Ein Raster von regelmäßig angeordneten Photodioden nehmen die Lichtintensitätswerte an den einzelnen Punkten des Bildes auf (Pixel). Die zweidimensionale Anordnung von Grauwerten ergibt dann zusammen das endgültige Bild. CCD-Kameras gibt es sowohl für Grauwert- als auch für Farbbilderzeugung, mit verschieden feiner Bildauflösung (d. h. Zahl der Bildpunkte oder Pixel pro Bild) und verschiedener Einzelbildrate (d. h. die Zahl der Bilder, die der Rechner pro Sekunde aufnehmen kann). Eine typische Bildgröße wäre zum Beispiel  $800 \times 600$  Bildpunkte, eine typische Einzelbildrate etwa 30 Hz. Für spezifische Anwendungen sind verschiedene Varianten von CCD-Kameras erhältlich, wie etwa die für Roboternavigation verwendete omnidirektionale Kamera.

Selbst für Aufgaben relativ geringer Komplexität genügt es nicht, einen einzigen Sensor zu verwenden. Zum Beispiel können in freier Exploration oder beim Hindernisausweichen manche Hindernisse nur durch Infrarot, andere nur durch Sonarsensoren, wieder andere nur durch eine Kombination von beiden erkannt werden. Die Integration der Sensordaten beschreibt den Prozess, aus der Information mehrerer Sensormodalitäten (d. h. Sensortypen) ein Konzept von der Umwelt zu erstellen. Weil Sensoren verschiedene Charakteristiken ausweisen und unterschiedliche Vor- und Nachteile haben, ist die Integration eine außerordentlich schwierige Aufgabe. Die Informationen von verschiedenen Sensormodalitäten sind nicht unbedingt immer konsistent. Ein globales Weltmodell muss sich daher oft auf Annahmen und Vermutungen stützen und alle Ergebnisse müssen unter Berücksichtigung neuer Informationen ständig revidiert werden. Die Sensorfusion verwendet zum Teil auch artificielle neuronale Netze, die lernen können, Wahrnehmung mit einem bestimmten Ausgangswert (wie beispielsweise einer bestimmten Aktion oder Klassifikation) zu assoziieren.

### 5.1.3 Aktoren

Der in der Robotik am häufigsten verwendete Aktorentyp ist der Elektromotor in Form eines Gleichstrommotors oder eines Schrittmotors. Der Gleichstrommotor ist am einfachsten zu steuern: der Motor wird durch Anlegen eines Gleichstroms betrieben. Elektromotoren sind sauber und einfach zu bedienen, sie produzieren ein mäßiges Drehmoment und können sehr genau gesteuert werden. Die Statorspulen eines Schrittmotors sind in Abschnitte zerlegt, so dass die Rotorbewegung durch die Anlegung einer Spannung an den



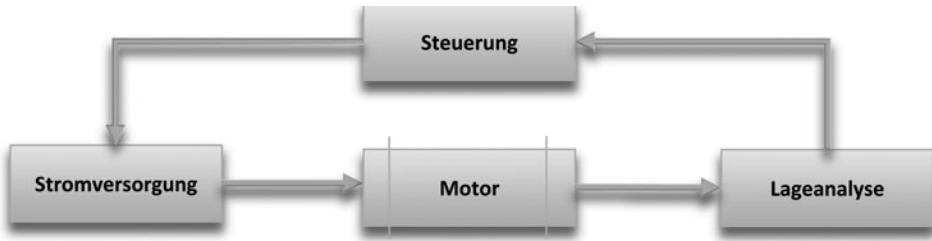
**Abb. 5.3** Schrittmotoren

gewünschten Abschnitt der Spule bestimmt werden kann. So dreht sich der Rotor um soviel Grad weiter, wie es dem Winkel zwischen den Spulenabschnitten entspricht.

*Schrittmotoren* können daher für sehr präzise Bewegungen verwendet werden. Sie kommen in Festplatten und CD- oder DVD-Laufwerken etc. zum Einsatz, um die Schreib-/Leseköpfe korrekt zu positionieren. Aber auch in der Industrierobotertechnik und speziell auch bei medizinischen Robotern, wie OP-Robotern, sind hochpräzise Schrittmotoren nicht mehr weg zu denken. Das Prinzip einer Schrittmotor-Steuerung ist die Übertragung von Richtungs- und Drehweiteninformationen. In einem vereinfachten Beispiel könnte man sich einen Schrittmotor so vorstellen, dass er mit jedem Impulstakt eine Position dreht. Ein Takt entspräche also einem Schritt, vier Takte würden den Rotor des Motors vier Schritte weit drehen lassen. In der Praxis kommen auch Informationen zur Drehrichtung hinzu. Wie genau der Schrittmotor letztlich arbeitet, hängt von der Weite der einzelnen Schritte und von der Reaktionsgeschwindigkeit ab. Je mehr Schritte für eine Umdrehung benötigt werden, umso genauer arbeitet der Motor. Allerdings birgt der Schrittmotor auch Risiken: zu schnelle Drehungen und zu hohe Lasten können zu sogenannten Schrittverlusten führen. Der Roboter bewegt sich dann nicht in vollem Umfang (Abb. 5.3).

*Servomotoren* kommen mit weniger Wicklungen aus und sind daher scheinbar einfacher aufgebaut. Allerdings analysieren solche Motoren – beispielsweise mit einer Ankerscheibe, die mit elektrischen Kontakten bzw. „Löchern“ als Signalgeber für eine Lichtschranke ausgestattet ist oder mit einem magnetischen Positionsbestimmungsverfahren arbeiten kann – den aktuellen Drehwinkel und die Drehzahl in recht präziser Form. Solche Servomotoren arbeiten in der Regel auch bei hohen Lasten recht zuverlässig (Abb. 5.4).

Die einfachsten und kostengünstigsten Antriebssysteme sind *pneumatische Aktoren*, die meist nur eine begrenzte Zahl von festen Positionen einnehmen können. Komprimierte Luft wird zwischen Luftkammern hin- und hergepumpt, wodurch ein Kolben bewegt wird. Während pneumatische Aktoren kostengünstig und einfach zu bedienen sind, kann man mit ihnen jedoch nicht sehr präzise arbeiten.



**Abb. 5.4** Servomotoren

*Hydraulische Aktoren* verwenden Öldruck, um Bewegung zu erzeugen. Sie sind in der Lage, sowohl beträchtliche Kräfte wie auch sehr präzise Bewegungen zu produzieren. Allerdings sind sie meist zu schwer, zu teuer und erzeugen zu viel Schmutz, um in der mobilen Robotik einen breiten Verwendungsgrad zu erzielen. Hydraulik- und Pneumatikpumpen werden in der Praxis zu komplexen Drucksystemen zusammengefasst, mit denen sich dann auch in der Bewegung große Kräfte übertragen lassen. Solche Kompressoren bauen dann den nötigen Betriebsdruck auf, der in Druckgefäßten gespeichert wird. Natürlich gehören Überwachungs- und Sicherheitssysteme zu einem solchen Druckantrieb, damit das ganze System nicht plötzlich bei Überdruck explodiert. Der besondere Reiz an einem solchen Drucksystem ist natürlich, dass mit einem zentralen Druckaggregat und entsprechenden Arbeitszylindern an den jeweils anzutreibenden Gelenken vergleichsweise leichte und dennoch kräftige Maschinen gebaut werden können. Natürlich ist auch die Druckerzeugung recht aufwendig und voluminös, aber es werden viele kleine Motoren und Getriebe eingespart, weshalb sich der Einsatz eines Drucksystems durchaus lohnen kann. Als eigentliche Aktoren müssen die Ventile angesehen werden, die die Luft- bzw. Flüssigkeitszufuhr zum Arbeitszylinder freigeben oder verschließen. Für die Steuerung der Ventile kommen entweder Leistungsrelais oder Bauteile der Leistungselektronik zum Einsatz.

*Lautsprecher* und integrierte Synthesizer fungieren als akustische Signalgeber und als Warnsystem. Natürlich kann man in der Praxis einen Roboter bauen, der mit vielen Sensoren ausgestattet ist, nur um sicher zu stellen, dass die Umwelt durch den Betrieb des Robotersystems nicht zu Schaden kommen kann. Darüber hinaus sollte allein aus wirtschaftlichen Gründen die Abschaltung einer Maschine nur dann erfolgen, wenn akute Gefahr besteht. Man setzt daher optische und akustische Melder ein, die eine unachtsame Person davor warnen, einem Robotersystem zu nahe zu kommen. Insofern zählen gerade bei Industrierobotern eine Sirene und auch eine Rundumleuchte zur Standard-Ausstattung.

Robotersysteme im privaten Bereich können oftmals über die Infrarot-Schnittstelle miteinander kommunizieren. Für ein privat genutztes Robotersystem ist dies eine völlig ausreichende Dialogmöglichkeit. Für professionelle Roboter ist eine Infrarot-Schnittstelle allerdings nicht unbedingt die Ideallösung, denn ihre Reichweite ist stark limitiert, sie ist verhältnismäßig langsam und obendrein durch das Umgebungslicht beeinflussbar. Darüber hinaus bricht eine Infrarot-Verbindung sofort ab, wenn der optische Kontakt unterbrochen wird. Für kommerzielle Systeme können preiswerte Funkschnittstellen einge-

setzt werden. Neben herstellereigenen Verfahren stellen insbesondere neue Standards, wie beispielsweise Bluetooth für kurze Distanzen oder Wireless LAN-Technologien, die auf 300 m bzw. ca. 50 m in Gebäuden einen Datentransfer im Megabit-Bereich pro Sekunde gestatten, eine brauchbare Lösung dar. Insbesondere Wireless LAN-Konzepte sind für ein Robotersystem ideal geeignet, weil es über diesen Weg einem Administrator möglich ist, die Maschine von einem beliebigen Arbeitsplatz fernzusteuern oder zu warten. Dies schließt die Übertragung von Softwareupdates ein und ermöglicht sogar die Fernwartung von einem Techniker des Herstellers. Roaming erlaubt es den Robotern, den Sendebereich zu wechseln, wodurch sie in großen Umgebungen einsetzbar sind.

Mit *Lasern* ist es hochpräzise möglich, Entfernungen zu messen. Aus Veränderungen der Entfernung innerhalb einer gewissen Zeit kann man bekanntlich die Geschwindigkeit errechnen. Eine andere Anwendung der Laser-Technologie ist die Datenübertragung über Lichtwellenleiter. So kommen Lichtwellenleiter zum Einsatz, um eine schnelle und störungsfreie Kommunikation der einzelnen Module des Systems zu ermöglichen. Jedoch ist diese Aktorentechnologie nicht risikolos. Zur Einschätzung der Risiken sind Laser-Module daher in verschiedene Gefahrenklassen eingestuft. Dabei werden als Laser der Klasse 1 Module mit einer Leistung von weniger als 25 µW eingestuft. Solche Module findet man beispielweise in Druckern und CD-Laufwerken. Diese Laser werden in der Regel als sicher und damit unkritisch eingestuft. Dennoch handelt es sich nach wie vor um gebündeltes Licht mit recht hohen Energien. Man sollte auch mit Klasse 1-Lasern vorsichtig umgehen. Auch die Laser-Klasse 2 wird als weitgehend ungefährlich eingestuft, wobei der Schutz der Augen durch die natürlichen Abwehrreaktionen des Menschen (Lidschlussreflexe) gegeben ist. Laser der Klasse 2 haben Leistungen von bis zu 1 mW. Laser der Klasse 3A arbeiten mit Leistungen von 1 mW bis ca. 5 mW. Im Idealfall schützt auch hier der Lidschlussreflex des menschlichen Auges vor Schäden, jedoch können optische Geräte, die den Strahl zusätzlich fokussieren (Fernrohre, Brillen etc.) ein Risiko darstellen. Keinesfalls mehr als ungefährlich einzustufen ist die Laser-Klasse 3B, die mit Leistungen von 5 mW bis zu 500 mW arbeitet. Hier schützt der Lidschlussreflex nicht mehr und Schäden der Augen sind höchst wahrscheinlich, wenn das Laser-Licht direkt auf die Netzhaut trifft. Besonders vorsichtig sollte man bei Lasern der Klasse 4 sein (Industrie-Laser mit mehr als 500 mW Leistung), bei denen sogar das Streulicht schon gefährlich werden kann.

### 5.1.4 Steuerung

Die Steuerung und die dazu notwendige Technik bilden mitunter das Hauptproblem beim Bau eines Robotersystems. Dieser reagiert nämlich auf äußere Einflüsse, die er mit Hilfe von Sensoren erkennt. Über Aktoren (Motoren, optische oder akustische Signalgeber etc.) versucht dann das System einen zuvor gewollten Ablauf oder eine intendierte Handlung zu realisieren. Als Robotersteuerung bezeichnet man die Hard- und Software eines einzelnen Robotersystems. Die Funktionen sind:

- Entgegennahme von Roboterbefehlen und Abarbeitung von Programmen,
  - Steuerung, Überwachung von Bewegungs- bzw. Handhabungssequenzen und Handlungsaufträgen,
  - Synchronisation und Anpassung des Manipulators an den Handhabungsprozess und
  - Vermeidung bzw. Auflösung von Konfliktsituationen.
- 

## 5.2 Software

### 5.2.1 Programmierungsarten

Ein Robotersystem muss frei programmierbar sein. Für die Programmierung eines Roboters kann man zunächst und grob betrachtet on-line und off-line-Methoden unterscheiden. Bei den *on-line Methoden* wird der Roboter zur Programmierung benötigt. Bei den *off-line-Methoden* wird das Roboterprogramm ohne Benutzung des Roboters erstellt. Der Roboter wird erst zum Test benötigt. Beispielsweise kann der Entwickler eines mobilen Robotersystems eine beliebige Folge von Anfahrpunkten vorgeben. Diese Punktfolge kann dann das Robotersystem beliebig oft abfahren. Es ist klar, dass die freie Wahl von Punkten durch Hindernisse in der Umgebung und durch konstruktive Beschränkungen des Robotersystems, beispielsweise bei den Gelenkwinkeln, eingeschränkt wird. In diesem Fall kann man eine weitere Unterscheidung bezüglich der Programmierung von Robotersystemen treffen:

- Programmierung durch Beispiele,
- Programmierung durch Training,
- Roboterorientierte (textuelle) Programmierung,
- Aufgabenorientierte (textuelle) Programmierung.

Aus einer weiteren Sicht des Entwicklers ist ein Robotersystem zunächst ein in mehreren Achsen frei programmierbares System in Form eines zustandsbasierten Automaten. Die freie Programierbarkeit stellt also für den Entwickler eine Grundvoraussetzung dar. Sie ist bestimend für die Vielseitigkeit und die Wiederverwendbarkeit von Automatisierungslösungen mit Robotersystemen, im Gegensatz zu Einzweckautomaten. Insofern kann die Programmierung solcher Robotersysteme auch auf unterschiedliche Arten erfolgen.

Die sicherlich einfachste Methode der Programmierung von Robotersystemen stellt die *manuelle Programmierung* dar. Hier wird der Bewegungsbereich der Achsen durch Festanschläge begrenzt. Die Programmierung findet also in Achskoordinaten statt. Die Bewegungen der einzelnen Achsen verlaufen unkoordiniert im sogenannten Punkt-zu-Punkt-Modus (PTP-Modus). Der Aufwand für den eigentlichen Programmervorgang in Form eines Einricht-Vorgangs ist dabei eher gering. Zur Robotersteuerung ist kein Rechner erforderlich. Die Gesamtbewegung des Armes kann zwischen den Anschlägen mit

maximaler Antriebsleistung, also schnell erfolgen. Eine Änderung des Programmes verlangt allerdings einen mechanischen Eingriff am Robotersystem und ist damit natürlich unflexibel und erfordert Umrüstzeiten, die die mögliche Betriebszeit verringern. Die Anzahl der Bewegungspunkte, die die PTP-Bahn bestimmen, ist naturgemäß eher klein gehalten. Verzweigungen innerhalb eines Programmes oder spezielle Zusatzfunktionen können im Regelfall nicht programmiert werden. Das Hauptanwendungsgebiet der manuellen Programmierung reduziert sich daher in der Praxis auf das Einrichten von Einlegegeräten zur Beschickung von Bearbeitungsmaschinen.

Eine weitere gängige Programmiermethode ist das sogenannte *Teach-in*. Bei dieser Methode werden die einzelnen Stützpunkte der gewünschten Bewegungsbahnen angefahren und die jeweilige Stellung des Effektors (beispielsweise Greifarm) über die internen Geber ermittelt und gespeichert. Dieses Programmierverfahren ist zunächst also nur eine Methode zur Festlegung der räumlichen Gestalt einer Roboterbahn. Nachdem die Bahngeometrie in dieser Weise programmiert wurde, kann das Bahnprogramm durch Zusatzanweisungen, die über das Bedienfeld am Robotersystem selbst oder einem externen Handprogrammiergerät eingegeben und damit ergänzt werden. Dabei existieren verschiedene Varianten des Teach-In-Verfahrens. Allen ist gemeinsam, dass der Roboter oder ein kinematisch äquivalentes Hilfsgerät physikalisch bewegt wird und die Achspositionen über interne Geber ermittelt werden. Weil der Effektor bewegt wird, spricht man hier von einer On-line-Programmierung. Wird die aktuelle Lage des Effektors während des Programmierungsvorganges vom Rechner selbsttätig abgetastet und in kurzen Zeitabständen gespeichert, spricht man von *Folgeprogrammierung*. Bestimmt der Programmierer den Zeitpunkt der Abspeicherung durch eine Anweisung an die Steuerung, so spricht man hingegen vom *Einstellverfahren*.

Von *direktem Teach-In* spricht man, wenn der Effektor (beispielsweise der Roboterarm) durch direktes Führen des Werkzeuges, etwa an einem speziell dazu vorgesehenem Griff, bewegt wird. Das ist nur möglich, wenn der Effektor entsprechend leicht ist und wenn keine stark unterstützenden und damit selbsthemmenden Getriebevorrichtungen vorhanden sind. Das Haupteinsatzfeld des direkten Teach-In bildet die Programmierung von Industrierobotern, wie sie beispielsweise zum Lackieren von Automobilkarossen eingesetzt werden. Bei diesen Anwendungen lassen sich die Bewegungsbahnen nicht analytisch beschreiben, sie sind vielmehr durch die Oberflächengestalt und durch die Handführung des Lackierers bestimmt. Deshalb kommt als Aufzeichnungsverfahren nur die Folgeprogrammierung in Frage. Durch kurze Abtastzeiten kann sich eine große Anzahl von Bahnpunkten ergeben. Wenn der Effektor wegen seiner Getriebe, seines Gewichtes oder seiner Größe nicht mehr durch direktes Führen bewegt werden kann, kann ein kinematisch äquivalentes, aber leichteres und kleineres Modell des Hauptgerätes zur Programmierung (*Master-Slave-Programmierung*) verwendet werden. Dieses Modell wird als *Master* bezeichnet und wird von Hand geführt. Zur Ermittlung und Aufzeichnung der Effektorposition existieren zwei Varianten. Bei der ersten Variante folgt das Hauptgerät (der Slave) der Bewegung des Hilfsgerätes und die Effektorposition wird vom Hauptgerät übernommen. Bei der zweiten Variante ist das Hilfsgerät selbst mit Gebern ausgestattet und kann zur Positionsermittlung

dienen, während das Hauptgerät während des Programmievorganges funktionslos ist. Die letzte Variante ist wegen der doppelten Geberausstattung aufwendig.

Beim *indirekten Teach-In* wird der Roboter mit Hilfe der entsprechenden Funktionstasten des Handbediengerätes nacheinander zu den Raumpunkten bewegt, aus denen später das fertige Bewegungsprogramm bestehen soll. Die Roboterbewegung kann dabei durch das Verfahren einzelner oder mehrerer Achsen im PTP-Modus geschehen. Die meisten Steuerungen stellen jedoch zur Unterstützung der Programmierung verschiedene Lehrkoordinatensysteme zur Verfügung. Beim indirekten Teach-In sind immer die eigenen Antriebe des Robotersystems aktiv. Die Festlegung der Bahnpunkte geschieht durch den Programmierer in unmittelbarer Nähe des Werkzeuges. Deshalb werden aus Sicherheitsgründen in dieser Betriebsart die maximal verfügbare Antriebsleistung und die maximal zulässige Bahngeschwindigkeit gegenüber dem Automatikbetrieb erheblich reduziert. Das Einlesen und Abspeichern der Effektorposition geschieht auf Anweisung des Programmierers. Nachdem die Bahngeometrie in dieser Weise programmiert wurde, kann das Bahnprogramm durch Zusatzanweisungen, die über ein Bedienfeld oder Handprogrammiergerät gegeben werden, ergänzt werden. Durch diese zusätzlichen Angaben sind einfache Strukturierungen des Programmablaufes möglich. Hierzu gehören die Einführung von Verzweigungen, der Aufruf zyklischer Bewegungsfolgen und das Setzen von Haltepunkten oder Wartezeiten. Alle programmierbaren Zusatzfunktionen, wie etwa Werkzeugbefehle, Sensorbefehle usw., können nur an einzelne programmierte Bahnstützpunkte geknüpft werden. Datenmanipulationen sind im Regelfall nicht möglich. Die Programmierung nach dem Teach-In-Verfahren geschieht in den meisten Fällen am Einsatzort. Sie kann deshalb bei einer neuen Anlage erst erfolgen, wenn diese schon weitgehend aufgebaut ist; eine bereits bestehende Anlage wird durch die Programmierung blockiert. Aus diesen Gründen gestaltet sich die Teach-in-Programmierung recht kostenintensiv. Wegen der vielen Einzelbewegungen, die erfahrungsgemäß zur Festlegung eines gültigen Bahnstützpunktes notwendig sind, ist die Teach-In-Programmierung bei umfangreichen Aufgaben aufwendig, langwierig und wenig änderungsfreundlich. Die Grenzen dieser Programmiermöglichkeit sind da erreicht, wo die Bahngestalt durch bloße Augenkontrolle nicht mehr mit befriedigender Genauigkeit festgelegt werden kann. Trotz der angeführten Nachteile ist das indirekte Teach-In-Verfahren die in der heutigen industriellen Praxis am weitesten verbreitete Programmiermethodik.

*Programmierung durch Beispiele* heißt, dass man beispielsweise ein mobiles Robotersystem und dort dessen Effektoren die gewünschte Bahn entlang führt. Dabei wird eine ausreichende Anzahl von Bahnpunkten (Position und Orientierung) im Computer abgespeichert. Die Folge der Bahnpunkte kann danach beliebig oft abgefahren werden. Erfolgt das Entlangführen mit dem realen Robotersystem an der realen Bahn, beispielsweise an einer schwarzen Linie, spricht man von einem on-line Verfahren. Wird das Entlangführen interaktiv in einem Robotersimulationssystem durchgeführt, handelt es sich um eine Art off-line Programmierung. Je nachdem wie die gewünschten Bahnpunkte bei der Programmierung durch Beispiele eingestellt werden, lassen sich noch folgende Unterklassen definieren:

- Justieren oder Kalibrieren des Robotersystems,
- manuelle Programmierung,
- Remote-Control-Programmierung,
- Master-Slave Programmierung (einschließlich Teleoperation, die meist zur Master-Slave Programmierung gerechnet wird, obwohl bei ihr die abgefahrenen Bahnen nicht zum Zwecke späterer Wiederholung gespeichert werden).

Beim *Justieren* stehen beispielsweise keine Servoregler zur Verfügung, sondern jedes Gelenk kann nur eine sehr begrenzte Anzahl diskreter Stellungen einnehmen. Im Falle von mobilen Robotern werden für die einzelnen Anfahrpunkte die Gelenkstellungen mit Schaltern festgelegt. Aufgabe der Robotersteuerung ist es dann, Signale an die einzelnen Stellglieder zu senden, so dass zum richtigen Zeitpunkt die richtige Position eingenommen wird. Die Zuordnung von Anfahrpunkten zu den Stellungen der einzelnen Stellglieder kann beispielsweise mit Codiermatrizen erfolgen. In der Regel kann man nur kleine Mengen von Anfahrpunkten zu einem solchen „Programm“ zusammenfassen.

Bei der *manuellen Programmierung* sind die Gelenkmotoren so eingestellt, dass das Robotersystem durch einen Menschen bewegt werden kann. Der „Programmierer“ führt den Effektor von Hand entlang der gewünschten Bahn, beispielsweise entlang einer schwarzen Linie. Die Bahn wird durch eine Folge von (Zwischen-)Zielpunkten definiert. Ist ein solcher erreicht, werden die zugehörigen Werte der Gelenkwinkel durch Drücken einer Taste am Robotersystem oder dessen Steuerrechner abspeichert. Der Nachteil dieses Verfahrens ist, dass schwere Roboter nicht so leicht von Hand bewegbar sind. Dazu kommt, dass in engen Fertigungszellen oft nicht ausreichend Platz für einen Bediener ist, um an beliebigen Positionen einzutreten, und dass diese Art der Programmierung für den Bediener unter Umständen gefährlich sein kann. Daher findet man die manuelle Programmierung heute in der Praxis kaum noch vor.

Bei der *Remote-Control-Programmierung* benutzt der Bediener ein spezielles Eingabegerät, um den Effektor zu positionieren. Dieses spezielle Eingabegerät bezeichnet man als Remote-Control-Unit. Es ist damit Bestandteil der Robotersteuerung. Zu solch einer Remote-Control-Unit gehören weiterhin Tasten, um Anfahrpunkte zu speichern oder zu löschen, sowie Tasten zum Starten und Abbrechen ganzer Programme. Komfortablere Geräte lassen auch das Einstellen von Geschwindigkeiten und die Eingabe von Befehlen zur Bedienung eines Greifers zu.

Die *Master-Slave-Programmierung* ist eine Möglichkeit, auch schwerste Roboter manuell zu programmieren. Der menschliche Bediener führt hier ein Master-Robotersystem, das klein ist und leicht bewegt werden kann. Gleichzeitig wird die Bewegung auf den (großen schweren) Slave-Roboter übertragen. Die Bereitstellung von jeweils zwei Robotern mit geeigneter Kopplung kann das Verfahren allerdings sehr kostenintensiv machen. Es wird daher fast ausschließlich in der Ausprägung der Teleoperation verwendet. Teleoperation wird immer dort benötigt, wo der Aufenthalt für Menschen schwierig zu bewerkstelligen oder gefährlich ist, wie in verstrahlten Räumen, unter Wasser oder im Weltraum.

Als Vorteile der Programmierung durch Beispiele gegenüber der konventionellen, textuellen Programmierung lassen sich anführen:

- Es sind keine Programmierkenntnisse erforderlich, so dass die Programmierung der Roboter auch ohne solche fundierte Kenntnisse erfolgen kann.
- Es sind keine zusätzlichen Rechner zur Programmierung notwendig.
- Der Arbeitsraum muss nicht vermessen werden, da die Stellung der Objekte im Weltkoordinatensystem nicht bekannt sein muss.
- Die Programmierung erfolgt direkt mit dem realen Roboter in der Zielumgebung im Rahmen der beabsichtigten Aufgabe, also unter Berücksichtigung aller konstruktiven Ungenauigkeiten des Roboters und aller sonstigen Störgrößen.

Der Hauptnachteil der Programmierung durch Beispiele ist, dass der Einbezug von Sensoren und die Korrektur von Bahnen aufgrund von Sensorinformationen nicht möglich ist. Ein Einsatz dieser Programmvariante wird bei intelligenten Robotersystemen nur bedingt möglich sein.

Bei der *Programmierung durch Training* wird dem Robotersystem die auszuführende Aktion vorgeführt. Sensoren nehmen die Aktion auf. Das Robotersystem führt die vorgemachte Aktion solange immer wieder aus (Training), bis sie gewissen Gütekriterien genügt, beispielsweise in Bezug auf Genauigkeit, Schnelligkeit usw. Dabei muss über externe Sensoren die Abweichung von der gewünschten Zielvorgabe festgestellt werden. Mit Hilfe der gemessenen Abweichung wird jeweils versucht, das Programm weiter zu verbessern.

Bei der *roboterorientierten Programmierung* erfolgt die Steuerung des Roboters durch ein Roboterprogramm mit expliziten Bewegungsbefehlen, wie beispielsweise „fahre auf einer geraden Linie nach Punkt B“. Das Roboterprogramm ist in einer Roboterprogrammiersprache textuell geschrieben. Unter Roboterprogrammiersprache wird üblicherweise eine Sprache verstanden, die sich in etwa auf der Ebene höherer Programmiersprachen bewegt. Eine solche Sprache wird dann insbesondere ergänzt durch:

- Roboterspezifische Datentypen, beispielsweise Transformationsmatrizen, und die entsprechenden Operatoren hierzu;
- Befehle zur Bewegung des Roboters und
- Befehle zur Bedienung der Effektoren.

Die *aufgabenorientierte Programmierung* zählt ebenfalls zur textuellen Programmierung. Bei der bereits besprochenen roboterorientierten Programmierung legt der Programmierer fest, wie der Roboter eine Aufgabe zu lösen hat. Um beispielsweise ein Werkzeug in eine Maschine einzulegen, benutzt er eine Folge von Bewegungs- und Greiferbefehlen. Die aufgabenorientierte Programmierung erfolgt in einer Abstraktionsebene, die deutlich über den roboterorientierten Programmiersprachen liegt. Diese Programmierebene wird üblicherweise mit intelligenten Robotern in Verbindung gebracht. Bei der aufgabenorientierten Programmierung muss der Programmierer nur noch spezifizieren, was der Roboter tun soll.

- Beispielsweise wird die Aufgabe „hole rote Schachtel und lege es in Maschine ein“ spezifiziert. Hieraus wird dann durch die Komponente „Aufgabenplaner“ ein Roboterprogramm in einer roboterorientierten Programmiersprache erzeugt. Um die Aufgabe zu lösen, benötigt diese Komponente beispielsweise eine Wissensbasis mit einem Umweltmodell, eine Wissensbasis mit Regeln, wie eine Aufgabe in Einzelschritte zu zerlegen ist, Algorithmen zur Montageplanung, Algorithmen zur Greiferplanung, Algorithmen zur Bahnplanung, Algorithmen zur Sensorintegration sowie Synchronisationsmuster zur Koordination der Tätigkeiten des Robotersystems mit der Umwelt.

Die aufgabenorientierte Programmierung ist besonders für komplexere Probleme interessant, da dann die Programmierung in roboterorientierten Sprachen zu umfangreich und zu fehleranfällig wird. Ein wichtiger Anwendungsbereich der aufgabenorientierten Programmierung ist beispielsweise die Kooperation und Interaktion mehrerer mit Sensorik ausgestatteter, intelligenter Roboter mit Maschinen.

Eine eher *problemorientierte Programmierung*, die in den späteren Kapiteln dieses Buches verfolgt wird, basiert auf einer Vereinheitlichung der Ansätze zur Softwareentwicklung aus jüngerer Zeit, von *Intentional Programming* über *Model Driven Architecture* bis hin zur generativen Programmierung und *Software Factories*. Der Ausgangspunkt dieser Art der Programmierung ist, dass die Allzweck-Programmiersprachen nach wie vor in erster Linie lösungsraum-spezifische Implementierungen mit vergleichsweise wenigen und ausdrucksschwachen Abstraktionen (Anweisungen, Funktionen, Schleifen, Verzweigungen, Feldern, Variablen, Objekten) der jeweiligen Programmiersprachen unterstützen. Die speziell für die Robotik hinzugezogenen Bibliotheken sind der konkrete Versuch einer ersten Annäherung, problemraum-spezifische Abstraktionen zur Verfügung zu stellen, wie beispielsweise zur Sensorsauswertungen bzw. Aktorensteuerungen etc., die aber daran kranken, weder die Semantik der betreffenden Programmiersprache zu erweitern, noch in besonderer Weise mit den übrigen Bestandteilen der Entwicklungsumgebung zu interagieren.

Als Abhilfe empfiehlt es sich, ein Problem aus dem Bereich der Robotik weitaus umfangreicher und tiefer, zunächst problemadäquat, mit Hilfe der Interoperationstheorie zu modellieren, bis es schließlich nur noch ein deutlich kleinerer, automatisierter Schritt zur Generierung von ausführbarem Code oder eines Programms in einer Hochsprache ist. Zur Erstellung eines problembezogenen Modells wird eine domänenspezifische Sprache verwendet, die über wesentlich mehr problemspezifische Abstraktionen verfügt und somit in höherem Maße deklarative Beschreibungen erlaubt. Aufgrund der Probleme, die sich aus der Verwendung textbasierter Programmiersprachen z. B. hinsichtlich ihrer eindeutigen Analyse ergeben, werden die Programme als strukturierte Graphenbildete gespeichert, die je nach Bedarf in geeigneter Weise dargestellt werden.<sup>5</sup>

---

<sup>5</sup> Siehe auch Jungnickel 1994.

### 5.2.2 Verarbeitungsmodelle

Ein Programmierstil basiert auf einer Vorstellung der Welt und auf einem Verarbeitungsmodell eines Rechnersystems.

- Das *konventionelle Verarbeitungsmodell* orientiert sich an einem endlichen deterministischen Automaten und verarbeitet sequentiell Folgen von rechnerinternen Anweisungen. Durch diese Anweisungen werden Aktionen ausgelöst, die den Rechner von einem Zustand  $t$  in den Zustand  $(t+1)$  überführt. Eine Von-Neumann Maschine verfügt hierzu über einen Datenspeicher, auf den diese Aktionen einwirken und wodurch die dort gespeicherten Daten manipuliert werden.
- Eng verwandt mit der funktionalen Sichtweise ist das *relationale Verarbeitungsmodell*. Letzteres basiert auf dem mathematischen Relationenbegriff. Dem Rechner werden der Name einer Relation und die entsprechenden Argumente zur Aktivierung präsentiert. Je nachdem, ob alle Argumente oder nur ein Teil davon präsentiert werden, liefert der Rechner entweder den spezifizierten Tupel oder aber alle der Spezifikation entsprechenden Tupel zurück.
- Im *Datenflussmodell* wird der Ablauf eines Algorithmus nicht explizit durch Kontrollinstruktionen gesteuert, sondern implizit aus dem Fluss der Daten zwischen den einzelnen Berechnungsfolgen.
- Das allgemeine *Problemlösermodell* verarbeitet Probleme, die aus Angaben einer Anfangssituation, einer Zielsituation und einer Menge von Operatoren bestehen. Der Rechner versucht, ausgehend von dieser Anfangssituation, durch Zustandsänderungen die Zielsituation zu erreichen.

Aus dieser Entwicklung von Verarbeitungsmodellen sind im Rahmen der Forschung zahlreiche Programmiersprachen entstanden, die dem Entwickler einen (oder mehrere) Programmierstil(e) in einer (mehr oder weniger gelungenen) Entwicklungsumgebung zur Verfügung stellt. Man kann dabei generell zwischen zwei großen Gruppen von Programmiersprachen unterscheiden:

- *Imperative Programmiersprachen* stellen Mittel bereit, um Rechenabläufe zu beschreiben. Diese Sprachen stellen somit Methoden zur Verfügung, wie etwas zu berechnen ist. Die Algorithmen werden mit Hilfe von Anweisungen oder Prozeduren formuliert. Für Programme, die in solch einer Programmiersprache geschrieben sind, ist typisch, dass sie nach einem Compilierungsvorgang ablauffähig sind, d. h., dass sie ohne besonderes, zusätzlich erforderliches Laufzeitsystem vom Rechnersystem abgearbeitet werden können.
- (Selbst-)*Reflexive Programmiersprachen* werden dadurch charakterisiert, dass sie selbstbezügliche Elemente enthalten, mit denen innerhalb eines Programms über dieses Programm Aussagen getroffen werden können. Dadurch kann sich das Programm während seiner Arbeit von selbst modifizieren bzw. Einfluss auf seine Umgebung nehmen.

- Bei den *Deklarativen Programmiersprachen* wird charakterisiert, was zu berechnen ist. Die meisten Programme, die in einer deklarativen Programmiersprache geschrieben sind, verlangen einen zusätzlichen Interpreter, der den Ablauf der Verarbeitung steuert und kontrolliert. Bei den deklarativen Programmiersprachen wird oftmals zwischen logikorientierten Programmiersprachen, Produktionsregelsprachen und objektorientierten Programmiersprachen unterschieden.<sup>6</sup>
  - Bei der *regel-orientierten Programmierung* arbeitet man mit regelhaften Ausdrücken, wobei diese Regeln die Gestalt Bedingung  $\Rightarrow$  Aktion haben, d. h., auf der rechten Seite stehen oftmals imperative Handlungsanweisungen. Typische Vertreter hierfür sind die Sprachen Meteor und OPS 5.
  - Die *objekt-orientierte Programmierung* basiert auf Informationseinheiten, die Datenkapselung, Datenabstraktion, dynamische Typbindung sowie Vererbungsmechanismen erlauben. Es werden Objekte bzw. Klassen von Objekten und deren hierarchische Beziehung untereinander beschrieben. Diese Objekte werden dadurch aktiviert, dass sie mittels objekt-inhärenter Methoden Nachrichten austauschen. Typische Vertreter der objektorientierten Programmiersprachen sind Smalltalk, Lavors, C++ und Java.<sup>7</sup>
  - Die *logik-orientierte Programmierung* basiert auf dem Programmieren mit logischen Kalkülen. Hauptausdrucksmittel ist die logische Implikation in der Überführung von Aussagen. PROLOG ist eine solche logik-basierte Programmiersprache.<sup>8</sup>

### 5.2.3 Dedizierte Programmiersprachen

Die Programmierung von Robotersystemen gewährleistet die Steuerung des Systems durch ein Programm mit expliziten Befehlen. Aber auch eine solche Programmiersprache muss entwickelt werden, wozu prinzipiell drei Wege eingeschlagen werden können:

- Vollständiger Neuentwurf einer Sprache,
- Weiterentwicklung einer vorhandenen Automatisierungs- oder Steuerungssprache oder
- Erweiterung einer schon vorhandenen, allgemein einsetzbaren Programmiersprache, um roboterorientierte Sprachelemente bzw. Prozeduren.

Diese verschiedenen Vorgehensweisen bringen bestimmte Vorteile:

- Bei einem vollständigen Neuentwurf der Sprache ist man frei von Sachzwängen und Implementierungsdetails vorgegebener Sprachen. Die neue Sprache kann unter Ver-

---

<sup>6</sup> Vgl. Schöning 2000.

<sup>7</sup> Vgl. Meyer 1988.

<sup>8</sup> Vgl. Heinemann und Weihrauch 1992.

meidung bekannter Schwachpunkte konzipiert werden, insbesondere können reichhaltige, roboterorientierte Datentypen realisiert werden.

- Die Weiterentwicklung einer schon vorhandenen NC-Sprache ist weniger aufwendig als ein vollständiger Neuentwurf. Verfügbare NC-Programmierer können schnell für die neuen Roboterprogrammiersprachen ausgebildet werden.
- Die Erweiterung einer schon vorhandenen und allgemein einsetzbaren Programmiersprache ist deutlich weniger aufwendig als ein vollständiger Neuentwurf, da viele Sprachelemente ohne Änderungen verwendet werden können.

Letzteres lässt sich wie folgt begründen: Über eine erweiterte Standardsprache lässt sich auf bereits vorhandene Konstrukte für die Definition komplexer Datenstrukturen, die arithmetischen und logischen Ausdrücke, die Organisation paralleler Abläufe, die Kommunikations- und EA-Schnittstellen sowie die Ablaufstrukturen zugreifen. Außerdem existieren oft schon vielfältige Bibliotheken von Unterprogrammen und Hilfsprogrammen. Die bereits vorhandenen Sprachmittel sind außerdem oft mächtiger und syntaktisch und semantisch ausgereifter als bei den NC-Sprachen. Die Ausgangssprache wird auch an anderen Stellen verwendet, so dass die Beherrschung der Programmiersprache als solches vorausgesetzt werden kann. Zum guten Schluss gestaltet sich die Integration in große Programmkomplexe leichter, da dieselbe Programmiersprache verwendet werden kann.

Zur Entwicklung von Anwendungen für Robotersysteme, insbesondere wenn der Einsatz von Sensoren und Effektoren gefordert ist, ist neben der rein geometrischen Bahnprogrammierung auch Logik zur Programmablaufsteuerung, Strukturierungsmöglichkeiten zur Beherrschung des Programmmfangs und Arithmetik zur Datenmanipulation notwendig. Deshalb liegt es nahe, *dedizierte Programmiersprachen* zur Robotersystemprogrammierung einzusetzen. So sind diese Programmiersprachen mit arithmetischen Fähigkeiten unterlegt, um in einem Roboterprogramm beispielsweise zur Sensordatenverarbeitung verwendet werden zu können. Höhere Sprachen unterstützen die Programmstrukturierung durch Unterprogrammtechnik und Block- und Task-Konzepte. Die Programmablaufsteuerung (bedingte Anweisungen, Fallunterscheidungen, Schleifen, etc.) kann durch Sensoreingaben gesteuert werden, so dass hier die Möglichkeit besteht, differenziert und problemadäquat auf die Umwelt just-in-time zu reagieren.

„Klassische“ Programmiersprachen wurden in erster Linie zur Darstellung von Algorithmen und logischen Abläufen entwickelt. Die Beschreibung räumlicher Verhältnisse und besonders die Beschreibung dreidimensionaler Bewegungen oder Orientierungen überfordern in der Regel die Vorstellungskraft des Anwenders. Deshalb wird auch die textuelle Robotersystemprogrammierung durch andere Methoden ergänzt. Das kann beispielsweise dadurch geschehen, dass einzelne der programmierten Raumpunkte eines Roboterprogramms in einem nachfolgenden Teach-In-Vorgang festgelegt werden.

So beispielsweise bei den Roboterprogrammiersprachen AL, VAL, VAL-II oder RAPT. RAPT wurde aus der Programmiersprache APT für NC-Maschinen (numerical-controlled machines) abgeleitet. ROBEX entstand aus EXAPT (EXtended APT).

Eine andere Möglichkeit ist die Konstruktion eines sogenannten *Umweltmodells*, das die räumlichen Verhältnisse beschreibt und auf dessen Bestandteile und Informationen mit Konstrukten der Robotersprache zugegriffen werden kann.

Die Ziele, die mit dem Einsatz von dedizierten Roboterprogrammiersprachen angestrebt werden, sind also im Wesentlichen

- die Beschreibung komplexer Aufgaben,
- das Treffen von on-line-Entscheidungen und
- die Ausführung sensorgeführter Aktionen.

Neben den oben angesprochenen speziellen Methoden unterscheidet sich die Robotersystemprogrammierung in einigen weiteren Punkten von der Programmierung in der allgemeinen Datenverarbeitung. Die Unterschiede sind bestimmt durch die interaktive Wechselwirkung mit der realen Umwelt. Zunächst muss der Zustand der Umwelt programmintern möglichst gut abgebildet und bei Zustandsänderungen konsistent und zeitnah angepasst werden. Bearbeitungs- und Verarbeitungsvorgänge sind nicht umkehrbar, so dass beim Programmtest auch der physikalische Anfangszustand der Umwelt wiederhergestellt werden muss. Roboterprogramme sind von ihrem physikalischen Kontext abhängig: Wegen der entsprechenden Eigenschaften der ausführenden Robotersysteme zeigen Teilprogramme ein unterschiedliches Verhalten in unterschiedlichen Bereichen des Arbeitsraumes (Ortsabhängigkeit) und sie zeigen ein unterschiedliches Verhalten bei unterschiedlichen Geschwindigkeiten durch dynamische Bahnverformung (Geschwindigkeitsabhängigkeit). Diese Eigenschaften erschweren Programmerstellung und Programmtest deutlich und schränken die Wiederverwendung von Lösungsansätzen bzw. die Übertragbarkeit von Lösungen ein.

Wie von der allgemeinen Datenverarbeitung gewohnt, kann der Entwickler auch in der Robotertechnik auf ein breites Spektrum von Programmiersprachen zurückgreifen und muss daher bei der Auswahl einer solchen Sprache unterschiedliche Kriterien zu ihrer Beurteilung heranziehen. Ein solches Merkmal betrifft die *Implementierung* und *Steuerungsanbindung*. Einige Programmiersprachen bestehen im Wesentlichen aus einem Unterprogrammpaket auf der Basis einer allgemeinen Programmiersprache, wie beispielsweise FORTRAN, PASCAL, C oder Java. Dadurch kann die Erstellung eines eigenen Compilers oder Interpreters umgangen werden.<sup>9</sup> Ein weiterer Vorteil ist die *Portabilität* solcher um Robotikspezifika erweiterten Sprachen. Die Portabilität ist allerdings nur in dem Maße gegeben, wie die zugrundegelegte Sprache portabel ist. Der Hauptnachteil dieses Ansatzes ist jedoch, dass das Anwenderprogramm nach jeder Änderung mit der Robotersteuerung zusammen gebunden und das Gesamtprogramm danach neu in den Steuerungsrechner geladen werden muss. Die meisten Roboterprogrammiersprachen sind deshalb Spezialsprachen, die eventuell in Anlehnung an allgemeine Programmiersprachen, auf die Bedürfnisse der Robotertechnik hin entworfen sind. Dazu steht ein eigener Compiler oder Interpreter zur Verfügung, der der Bewegungssteuerung vorgeschaltet ist. Viele Roboter-

---

<sup>9</sup> Vgl. Wirth 1996.



**Abb. 5.5** Klassifikation der Programmiersprachen

sprachen sind herstellerspezifisch und fest in die jeweiligen Steuerungen integriert. Der Austausch von Robotern gegen Modelle anderer Hersteller bedeutet somit auch implizit einen Wechsel der Programmiersprache. Anwender, die Robotersysteme verschiedener Hersteller betreiben, müssen daher über Programmier- und Systemkenntnisse für diese unterschiedlichen Systeme verfügen. Es gibt deshalb verschiedene Versuche zur Standardisierung. Ein vielversprechender Ansatz ist die Normierung eines *Zwischencodes*, den die Compiler der Robotersprachen als Zielsprache erzeugen. Fester Bestandteil der Einzelsteuerungen muss dann nur noch ein Interpreter sein, der den festgelegten Leistungsumfang dieses Zwischencodes beherrscht. Dadurch lässt sich die verwendete Programmiersprache stark von der verwendeten Steuerung entkoppeln. Ein normierter Zwischencode, beispielsweise der *IRDATA-Code*, wird in der VDI-Richtlinie 2863 festgelegt. Das wichtigste Beurteilungsmerkmal für Roboterprogrammiersprachen ist die Art der Programmierung, die eine Sprache ermöglicht. Die Art der Programmierung wurde in diesem Buch daher als Klassifikationskriterium verwendet (Abb. 5.5).

Die Klasse achs-orientierte Sprachen drücken die Bewegungsanweisungen *achs-orientiert* aus (Punkt-zu-Punkt-Bewegungen). In dieser Klasse stehen Koordinatentransformationen und kartesische Bewegungskommandos nicht zur Verfügung. Eine weitere Klasse bilden die *roboterorientierten Sprachen*. Hier stehen kartesische Lage- und Orientierungsbeschreibungen sowie kartesische Bewegungsbefehle zur Verfügung. Die Bewegungsanweisungen sind jedoch auf den Roboter bzw. das Werkzeug bezogen. Jede Aktion des Roboters muss einzeln und ausdrücklich formuliert werden. Objekte der Umwelt sind nicht direkt repräsentiert. Deshalb werden diese Sprachen auch als *explizite Roboterprogrammiersprachen* bezeichnet. Als *implizite Roboterprogrammiersprachen* werden Sprachen bezeichnet, die ein (Teil-)Umweltmodell besitzen. Die Programmanweisungen sind werkstückbezogen, d. h. Aufgaben werden als Aktionen formuliert, die mit dem Werkstück ausgeführt werden sollen. Die dazu notwendigen expliziten Bewegungsbefehle an den Roboter werden bei der Übersetzung generiert – sie stecken implizit in der werkstückbezogenen Formulierung. Als *aufgabenorientierte Programmiersysteme* werden Aktionsplanungssysteme bezeichnet, die in der Lage sind, Aufgabenbeschreibungen zu interpretieren und die notwendigen Schritte zur Ausführung zu planen.

Eine *explizite Roboterprogrammierung*, wie sie beispielsweise mit der Programmiersprache VAL-II möglich ist, kann als roboter- bzw. bewegungsorientiert bezeichnet werden. Jede Einzelbewegung und jede Datenmanipulation muss auch einzeln und dediziert programmiert werden. Ein Umweltmodell ist von der Sprachkonzeption her nicht vor-

gesehen. Selbstständige Programmernachrichten (implizierte Aktionen) finden nicht statt. Das heißt beispielsweise, dass der Entwickler selbst durch entsprechende Programmieranweisungen dafür sorgen muss, dass das Werkstück im Greifer abgelegt wird, bevor ein anderes geöffnet wird. Eine selbstständige Erkennung von Fehlersituationen findet wegen des fehlenden Umweltmodells ebenso nicht statt. Das bedeutet beispielsweise, dass der Zielpunkt einer Bewegung, der so programmiert wurde, dass er hinter einer Wand liegt, vom Programmier- und Steuerungssystem nicht als unzulässig erkannt werden kann. Der Anwender muss also auch mögliche Fehlersituationen voraussehen und gegebenenfalls selbst explizit Gegen- oder Notfallmaßnahmen programmieren. Explizite Programmiersysteme lassen sich andererseits mit denselben Methoden und Hilfsmitteln realisieren, wie sie zum Bau von Übersetzern und Interpretern allgemeiner Programmiersprachen verwendet werden. Da keine Aktionsplanungen stattfinden müssen, können alle roboter- und peripheriebezogenen Anweisungen direkt und mit vorhersehbarem Zeitverhalten durch die Robotersteuerung ausgeführt werden.<sup>10</sup>

Die wichtigsten Merkmale einer expliziten Roboterprogrammiersprache lassen sich in den folgenden Gruppen zusammenfassen.

- *Sprachtyp:* Compiler-Sprache, Interpreter-Sprache, der Sprachumfang ist erweiterbar, eignet sich für Parallelverarbeitung
- *Roboterspezifische und geometrische Datentypen:* Frame, Gelenkwinkel, Vektor, Transformation, Rotation, Bahn
- *Rotationsdarstellung:* Rotationsmatrix, Drehwinkel/Drehvektor, Eulersche Winkel, Roll-Nick-Gier-Winkel, Quaternionen
- *Kontrollstrukturen:* Marken, IF-THEN-ELSE, WHILE-DO, DO-UNTIL, CASE, FOR, BEGIN-END-Blöcke, Unterprogramme, Funktionen, COBEGIN-COEND für parallel ausführbare Anweisungen
- *Bewegungsarten:* Punkt-zu-Punkt, Linearinterpolation, Zirkularinterpolation, Spline-Interpolation, on-line-Bahnbeeinflussung
- *Ein-/Ausgabemöglichkeiten:* digitale Ein-/Ausgabe, analoge Ein-/Ausgabe
- *Sensor- und Werkzeugschnittstellen:* Sichtsystem, Kraft-Momenten-Sensor, Näherungssensoren, Effektorenkommandos
- *Entwicklungsumgebung:* Editor, Dateiverwaltung, Interpreter, Compiler, Simulator, Debugger, Hilfe-Funktionen, Makro-Eigenschaften, INCLUDE-Anweisungen, Kommandoprozeduren, Fehlerprotokollierung
- *Testmöglichkeiten:* Einzelschritt, Haltepunkte, Tracing, Speicherabild

Keine existierende Sprache besitzt jedoch alle angeführten Merkmale. Die Liste soll also somit zur Übersicht über die Möglichkeiten dieser Sprachklasse dienen und Grundlage zu vergleichenden Betrachtungen sein.

Bei der Verwendung einer *impliziten Programmiersprache* werden nicht die zur Lösung einer Aufgabe notwendigen Roboterbewegungen beschrieben, sondern die zur Lösung

---

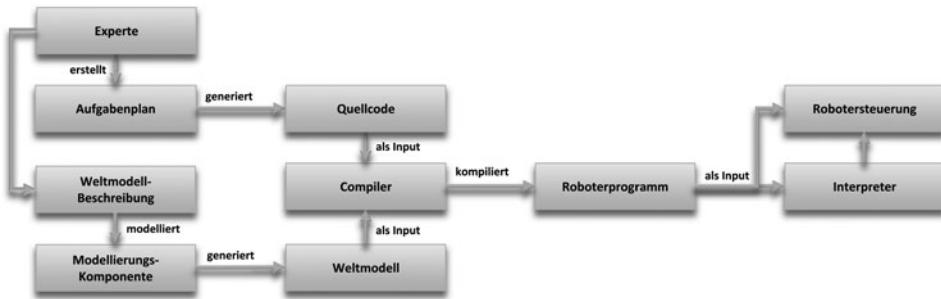
<sup>10</sup> Siehe auch Güsmann 1992.

notwendigen Zustände und Zustandsänderungen der Werkstücke und Zielobjekte. Beispielsweise wird durch diese Beschreibungsform „Füge Bolzen in Loch“ die Programmierung erleichtert, da jetzt mächtigere, aufgabenbezogene Ausdrucksmöglichkeiten zur Verfügung stehen, die der menschlichen Beschreibungsweise einer Aufgabe nahekommen. Die Planung der Details, d. h. die Realisierung in explizite Einzelanweisungen zur Beauftragung einer Robotersteuerung, wird vom Programmiersystem der Entwicklungsumgebung streckenweise übernommen. Der wichtigste Systembestandteil ist der Planer, der die impliziten Anweisungen in explizite Anweisungen umwandelt. Dazu ist ein Weltmodell erforderlich, das die Geometrie der Umweltbestandteile und einige ihrer physikalischen Eigenschaften beschreibt. Der Planer muss dazu in der Lage sein, die nächsten Schritte zu simulieren, um daraus dann das Bewegungsprogramm abzuleiten. Dazu ist es erforderlich, den augenblicklichen Zustand der Umwelt und die Beziehungen der Umweltobjekte untereinander intern darzustellen und fortzuschreiben. Speziell für diese Planung fallen dann folgende Einzelaufgaben an:

- die Erzeugung kollisionsfreier Bahnen,
- die Planung des Bewegungsverhaltens,
- die Effektorenplanung,
- die Koordinierung verschiedener Sensoren,
- die Planung von Feinbewegungen.

Bei der Implementierung derartiger impliziter Programmiersysteme treten zahlreiche Probleme auf. Dazu gehören der Aufbau eines Umweltmodells und die Gewährleistung einer möglichst hohen Datenkonsistenz bei der Anpassung dieses Modells an die sich ändernde Umwelt. Ein Teil der erforderlichen geometrischen Daten kann dazu aus speziellen CAD-Modellen übernommen werden, ein wesentlicher Teil muss allerdings auch von Hand eingegeben werden. Ein weiteres Problem ist die Behandlung von Unsicherheiten. Der Planer muss in Betracht ziehen, dass sein Umweltmodell nicht vollständig konsistent sein kann. Zur Reduzierung der Unsicherheiten muss die vorhandene Sensorik eingesetzt und koordiniert werden. Eine Überschätzung der Unsicherheiten erhöht allerdings den Planungsaufwand und die Programmlaufzeiten, eine Unterschätzung vermindert hingegen die Programmzuverlässigkeit.

*Aufgabenorientierte Programmiersprachen* gestatten ebenfalls die Formulierung von Robotersystemaufgaben in einer Sprache, die einer natürlichen Sprache (meist Englisch) ähnlich ist. Der Entwickler erstellt in dieser Sprache einen Aufgabenplan in Form eines Quellcodes und stellt dem System ein Umweltmodell zur Verfügung, das zuvor mit Hilfe einer speziellen Modellierungskomponente konstruiert wurde. Der Compiler übersetzt das Quellprogramm und berücksichtigt dabei die Daten des Weltmodells. Der Zielcode besteht aus Prozeduren einer expliziten Roboterprogrammiersprache. Das Weltmodell ist in einer Modell-Datenbank in Form eines Repository abgelegt, die die geometrische Gestalt der Objekte, deren Positionen, deren räumliche Beziehungen untereinander und deren Aufgaben- oder Verbindungszustand enthält. Dabei wird das Modell oftmals in Form gerichteter Graphen dargestellt, wobei die Knoten des Graphen die Objekte und die Kan-



**Abb. 5.6** Aufgaben-orientierte Programmiersprache

ten deren Beziehungen darstellen. Die Anweisungen der Programmiersprache zerfallen grob betrachtet in aufgabenspezifische und in allgemeine Anweisungen. Die aufgabenspezifischen Anweisungen lassen sich ihrerseits in drei Gruppen einteilen:

- Anweisungen, die eine Zustandsänderung bewirken,
- Effektor- und Sensor-Anweisungen,
- Handlungsanweisungen.

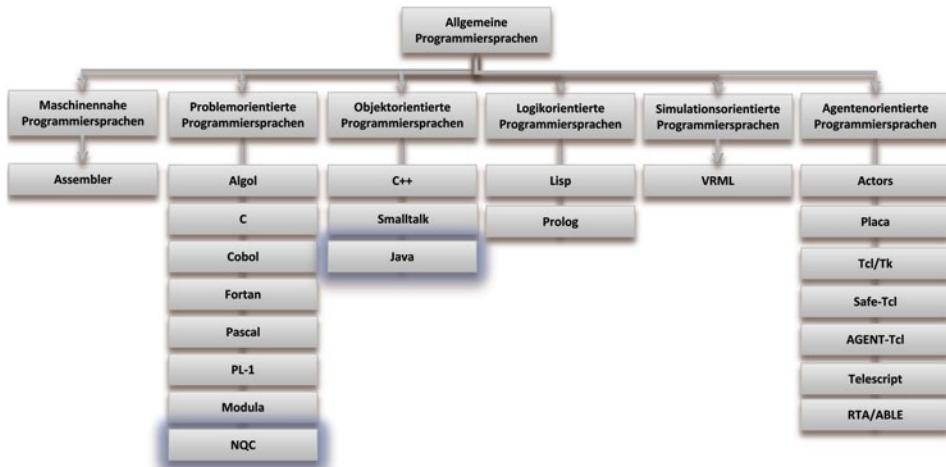
Anweisungen, die eine Zustandsänderung bewirken, beschreiben dabei Vorgänge, wie etwa das Positionieren von Teilen oder Roboterbewegungen (Abb. 5.6).

Der Kern der Compilierung einer impliziten Anweisung läuft wie folgt ab. Zu jedem Anweisungstyp existiert ein Schema, das vom Compiler abgearbeitet wird. Der erste Schritt ist die Syntaxprüfung und die Bewertung der Operanden und Plausibilitätskontrollen. Im nächsten Schritt werden *die* Voraussetzungen zur Ausführung geprüft. Anschließend kann der Ablauf in Einzelschritte zerlegt werden (Planung). Das schließt beispielsweise die Bahnplanung, die Kollisionsvermeidung und die Greifpunktbestimmung ein. Nach der erfolgreichen Festlegung der Einzelschritte wird das Weltmodell auf den jetzt erreichten Zustand gebracht.

#### 5.2.4 Allgemeine Programmiersprachen

Ein vergleichender Überblick, wie in der folgenden Abbildung, kann nur eine Orientierung bieten, da zum einen viele der Programmiersprachen unerwähnt bleiben, und zum anderen die Reihe der hybriden Programmiersprachen nicht voll zum Ausdruck kommt.

Bereits auf den ersten elektronischen Computern konnten verschiedene Programme laufen. Steuerbefehle wurden dabei über ein Programmierbrett eingegeben, auf dem je nach Aufgabe verschiedene Drähte in vorhandene Steckbuchsen gesteckt werden mussten. Jedes Programm bestand somit aus einer Kombination von Hunderten von Drähten, was eine hohe Fehleranfälligkeit implizierte und bei Verlust der handschriftlichen Unter-



**Abb. 5.7** Allgemeine Programmiersprachen

lagen nur schwer reproduzierbar war. In einem weiteren Entwicklungsschritt wurden diese Steckverbindungen auf Lochkarten nachgebildet. Das Terminal war nun eine spezielle Schreibmaschine, die anstelle von Buchstaben, entsprechende Löcher stanzen konnte. Als die Tasten mit speziellen Symbolen versehen wurden, war dann eine Art sprachliche Repräsentation möglich. Allerdings besteht die *Maschinensprache* eines Computers nur aus Folgen von Bits, die für den Anwender völlig unübersichtlich sind. Daher wurden mehrere Bits zu Gruppen zusammengefasst und mit Buchstaben und Ziffern bezeichnet. Diese Symbole sollten leicht verständlich entsprechende Befehle und Daten im Binärkode repräsentieren. Sie werden deshalb auch mnemonische Symbole genannt. Das war der Beginn maschinenorientierter Assemblersprachen. Beispiele sind die Bezeichnungen:

```

add ' Addition
sub ' Subtraktion.
  
```

Befehle in Assemblersprache bestehen aus Bezeichnungen für die durchzuführenden Operationen, die zu bearbeitenden Daten (Operanden) und aus Marken für Adressen. Assemblersprachen sind demnach auf den Befehlsvorrat einer ganz konkreten Maschine zugeschnitten. Ein Assemblerprogramm ist auf einem Computer nur dann ausführbar, wenn es in den Binärkode einer Maschinensprache übersetzt wird. Das Entwickeln von Anwendungen in solch einer Sprache setzt daher die genaue Kenntnis der Hardware des Zielrechners voraus. Der Vorteil solcher Assemblerprogramme besteht eindeutig in dem geringen Speicherbedarf und die damit verbundene schnelle Ausführung. Damit werden Nachteile einer recht hohen Abhängigkeit von konkreten Maschinen oftmals in Kauf genommen. Heute werden Assemblerprogramme für solche Anwendungen verwendet, die unbedingt einen unmittelbaren und schnellen Zugriff auf die Hardware erfordern, wie es beispielsweise bei der Entwicklung von Betriebssystemen der Fall ist (Abb. 5.7).

Bei schwierigen Aufgaben, beispielsweise in der Mathematik, in den Wirtschaftswissenschaften oder im Falle der Simulation von Prozessen wird sich ein Anwender kaum noch zusätzlich um die direkte Zuordnung von Werten und Speicherstellen kümmern können und wollen. Vielmehr ist dieser Anwender an der Lösung an sich interessiert und weniger an der Programmiersprache. Dieser Interessenwandel führte zu den *problemorientierten Programmiersprachen*. So wurde zur Bearbeitung mathematischer Aufgaben von IBM seit Mitte der 50er Jahre die Programmiersprache FORTRAN (engl. Formular Translator) entwickelt. Bei dieser Sprache besteht ein Programm aus einzelnen Anweisungen und Befehlen mit mathematischen Formeln und Ausdrücken, die dann Schritt für Schritt, d. h. sequentiell nacheinander ausgeführt werden. FORTRAN ist somit ein Beispiel für eine imperative (algorithmische bzw. operative) Programmiersprache. Für geschäftliche und wirtschaftswissenschaftliche Probleme wurde seit Ende der 50er Jahre die Programmiersprache COBOL (engl. Common Business Oriented Language) entwickelt. Auch diese Sprache ist ein Beispiel für eine problemorientierte imperative Programmiersprache. Man bezeichnet solche problemorientierten Sprachen im Fachjargon auch gerne als höhere Programmiersprachen, da sie nicht maschinenorientiert sind und sie daher nicht extra in eine Maschinensprache übersetzt werden müssen. Für diese Übersetzungsarbeiten von Quellcode einer höheren Programmiersprache in den Maschinencode existieren sogenannte Compiler oder Interpreter. Ein Compiler übersetzt ein Programm vor der Programmausführung und speichert den Maschinencode ab. Zur Laufzeit wird das Programm nur noch ausgeführt, so dass die Übersetzungszeit entfällt. Dagegen übersetzt ein Interpreter den Quellcode ohne Abspeicherung Befehl für Befehl, so dass Änderungen an diesem Befehlsvorrat schnell durchführbar sind, dadurch aber eine längere Übersetzungszeit erforderlich ist. Mit einem Compiler übersetzte Programme können ohne erneute Übersetzung beliebig oft ablaufen, während der Quellcode beim Interpreter jedesmal übersetzt werden muss. Solche problemorientierten Programme waren ursprünglich zeilenorientiert aufgebaut, wobei die einzelnen Programmzeilen mit einer laufenden Nummer versehen wurden, die in der Regel sequentiell und nacheinander abgearbeitet wurden. Der Programmablauf konnte durch Kontrollstrukturen und einige wenige Sprungbefehle (beispielsweise goto zum Anspringen von Programmzeilen) beeinflusst werden. Bei komplizierten und langen Programmen führte aber die häufige und unkontrollierte Verwendung des Sprungbefehls zu wachsender Unübersichtlichkeit und Fehleranfälligkeit. So entstand die Forderung nach strukturierten Programmstrukturen, die nunmehr nicht an Zeilen, sondern an streng voneinander separierten Programmblöcken orientiert sind. In den 60er Jahren wurde daher ALGOL (engl. Algorithmic Language) als ein erstes Beispiel einer solchen Programmiersprache entwickelt, die Teilaufgaben durch Unterprogramme (Prozeduren) verwirklichte und Kontrollstrukturen für die Reihenfolge ihrer Bearbeitung einsetzte.

Den Durchbruch strukturierten Programmierens erreichte man allerdings Anfang der 70er Jahre durch die Programmiersprache PASCAL, eine von Niklaus Wirth zunächst nur für Lehrzwecke konzipierte Sprache. Ein solches PASCAL-Programm ist klar in einen Vereinbarungs- und einen Anweisungsteil getrennt. Im Vereinbarungsteil werden der Name

des Programms und die Namen der Variablen mit ihren Datentypen angegeben. Zwischen den Schlüsselwörtern begin und end werden die Anweisungen formuliert, die vom Rechner entsprechend auszuführen sind. Entsprechend aufgebaute Teilprogramme lassen sich dann als Module zu komplexen Prozeduren zusammensetzen, deren Ablauf über spezielle Kontrollstrukturen bestimmt wird. Der modulare Aufbau ermöglicht die arbeitsteilige Bewältigung einer komplexen Aufgabe in klar getrennten Teilaufgaben. Mit der technischen Entwicklung neuer und leistungsfähigerer Computertypen wurden auch neue Betriebssysteme notwendig, um die technischen Abläufe der Maschine zu steuern und zu überwachen. Eine höhere Programmiersprache wie PASCAL, erwies sich dafür allerdings eher als ungeeignet. In den 80er Jahren waren bereits unterschiedliche Betriebs- und Arbeitssituationen mit Computern zu bewältigen. Die einfachste Betriebssystemart ist das Einzelplatzsystem (engl. Single User), beispielsweise eines PCs oder Laptops, der den Betrieb nur eines Programms (engl. Single-Tasking) zulässt. Häufig muss der Nutzer eines Einzelplatzsystems jedoch mehrere Aufgaben gleichzeitig bewältigen (beispielsweise Informationsabfragen, Rechnen, Schreiben). Dieses quasi-parallel Bearbeiten mehrerer Aufgaben in Form des Multitaskings wird durch ein Betriebssystem für einen Einprozessor-Computer dadurch bewältigt, dass einzelnen Programmen für wenige Millisekunden Rechenzeiten zugesprochen werden. Bei einem Großrechner wollen mehrere Anwender (engl. Multi-user) mit denselben Dateien zur gleichen Zeit und mit gleichen oder unterschiedlichen Programmen arbeiten. Auch in diesem Fall regelt ein Betriebssystem die Rechnerzuteilung an den einzelnen Arbeitsplätzen.

Die Programmiersprache, mit der solche modernen Betriebssysteme entwickelt werden konnten, war C.<sup>11</sup> 1973 wurde beispielsweise das Betriebssystem UNIX in dieser Sprache formuliert, um es besser auf neue Computer übertragen zu können.<sup>12</sup> Die gute Übertragbarkeit (Portierbarkeit) röhrt daher, dass C zwar wie eine Assemblersprache maschinennahes Programmieren erlaubt und damit Schnelligkeit ermöglicht, aber gleichzeitig über alle Strukturen einer höheren Programmiersprache verfügt. In den 80er Jahren entwickelte das American National Standards Institute (ANSI) sogar einen Standard für diese Programmiersprache, dem alle neueren Compiler folgten. Aber auch komplexe Standardanwendungen, wie beispielsweise Textverarbeitungen oder Tabellenkalkulationsprogramme oder Datenbanksysteme, wurden in C geschrieben. Das Kürzel C kommt daher, weil ein Vorgängertyp dieser Sprache Anfang der 70er Jahre B hieß. Ein C-Programm besteht nur aus Funktionen. Eine Funktion ist eine abgeschlossene Programmeinheit, die unter einem bestimmten Namen aufgerufen wird, Anweisungen an festgelegten Argumenten oder Parametern ausführt und genau einen oder keinen Wert als Ergebnis zurückliefert. Formal besteht eine Funktion daher aus einem Rückgabetyp, wie beispielsweise int als Repräsentant für ganze Zahlen (engl. integer), dem Funktionsnamen, einer Parameterliste in runden Klammern und dem eigentlichen Anweisungsblock in geschweiften Klammern.

---

<sup>11</sup> Vgl. Kernighan und Ritchie 1990.

<sup>12</sup> Vgl. Gulbins 1995.

```
// ermittelt den größten Wert zweier Ganzzahlen
int max (int a, int b)
{
    if (a > b)
    {
        return a;
    }
    else
    {
        return b;
    }
}
```

Diese Funktion `max` benötigt die zwei Parameter `a` und `b` vom Typ `int` und liefert als Ergebnis ebenfalls einen Wert vom Typ `int` zurück. Die Anweisung `return` verlässt die Funktion an der Stelle, an der sie zur Abarbeitung gelangt ist. Sie gibt dabei `a` zurück, falls `a` größer als `b` ist, andernfalls verlässt sie die Funktion an der nachfolgenden Stelle, um `b` zurückzugeben. Jede Anweisung wird mit einem Semikolon (`;`) abgeschlossen. Die geschweiften Klammern schließen den gesamten Anwendungsblock ab. Diese Funktionen sind in sich abgeschlossene Einheiten, in denen keine weiteren Funktionen eingeschachtelt werden dürfen. Das hat den Vorteil, dass alle Funktionen gleichberechtigt in einem Programm sind. Der Geltungsbereich der Variablen wird durch die geschweiften Klammern des Anweisungsblocks beschränkt und somit besteht auch keine Verwechslungsgefahr bezüglich namensgleicher Variablen in verschiedenen Funktionen. In jedem C-Programm muss die Hauptfunktion `main` genau einmal vorkommen. Sie ist sozusagen der Startpunkt eines jeden C-Programms, an dem der Compiler ansetzt, den Maschinencode erstellt und das Programm zur Ausführung bringt. Der Sprachumfang von C ist so klein gehalten, dass sie sehr effizient in eine Maschinensprache übersetzt werden kann. Nahezu alle benötigten Funktionen stehen in diversen Bibliotheken zur Verfügung und können bei Bedarf in ein C-Programm inkludiert werden. Insofern liegen die meisten Funktionen als fertige Module bereit, können gemäß dem Baukastenprinzip problemorientiert zusammengebaut werden und erlauben daher eine große Portabilität der C-Programme. Die C-Bibliotheken umfassen sowohl typische Funktionen für betriebssystem-spezifische Aufgaben (beispielsweise Speicherverwaltung, Prozess- und Umgebungssteuerung) als auch Funktionen höherer Programmiersprachen (beispielsweise mathematische Funktionen). Das Format von C-Programmen kann dabei festgelegt werden, wodurch man beispielsweise auf eine Zeilennummerierung verzichten kann. Dadurch lassen sich Programme flexibel gestalten, was allerdings nicht *a priori* zu übersichtlichen Programmen führen muss.

Die bisher besprochenen Programmiersprachen werden als prozedurale Sprachen zusammengefasst. Bei diesen Sprachen ist die Modellierung der Informationen von der Modellierung der Verarbeitung klar getrennt. Informationen werden dabei im Wesentlichen durch Daten (ganze Zahlen, boolesche Werte, Zeichen etc.) modelliert, um sie später in Variablen zu speichern. Das Verarbeitungsmodell der prozeduralen Sprachen besteht zum

einen in einer Modellierung der Informationen durch Zustände über Variablen und deren Daten und einer Modellierung der Verarbeitung in Form einer schrittweisen Änderung des Zustands. Zustandsänderungen werden durch Anweisungen herbeigeführt, wobei sich mehrfach verwendete Anweisungsfolgen zu Prozeduren zusammenfassen lassen. Insofern bilden die Prozeduren das zentrale Strukturierungsmittel prozeduraler Programme. Historisch und systematisch sind die prozeduralen Programmiersprachen durch eine zunehmende Abstraktion von der Hardware bestimmt. Daher spricht man auch oftmals von der

- Generation der Maschinensprachen
- Generation der Assemblersprachen
- Generation der höheren prozeduralen Sprachen.

Die erwähnten Beispiele von FORTRAN und COBOL über PASCAL bis C sind imperativisch bzw. algorithmisch orientiert. Dabei wird zwischen Daten und Algorithmen streng unterschieden. Der Entwickler muss daher genau darauf achten, dass immer nur die passenden Algorithmen auf die entsprechenden Daten angesetzt werden. Passen sie nämlich nicht zueinander, kann es zu verheerenden Programmstörungen kommen. Bei der Darstellung eines komplexen Problems ergeben sich daher gegebenenfalls komplexe Zusammenhänge zwischen Daten und passenden Algorithmen, die immer vor, während und nach der Entwicklung durchschaut werden müssen.

Die Simulation von Realität erfordert oftmals eine *simulationsorientierte Programmiersprache*, um dreidimensionale Umgebungen abilden zu können. Sie soll zudem als Programmcode im World Wide Web einsetzbar sein. HTML-Formate reichen dabei nicht aus, da sie ausschließlich auf textuelle Darstellung ausgerichtet sind. Verteilte virtuelle Umgebungen erwarten außerdem höhere Anforderungen an das Netzmanagement. Im Jahre 1994 entwickelten die Begründer des WWW eine erste Version des sogenannten VRML (engl. Virtual Reality Markup Language)-Konzepts, nach dem dreidimensionale Informationen als Inline-Datei in ein HTML-Dokument eingebunden werden konnten. VRML ist dabei ein Werkzeug zur Beschreibung geometrischer Körper und Flächen, ihrer Lagen im Raum, der Strukturen ihrer Oberflächen und der herrschenden Lichtverhältnisse. Bereits im Jahre 1995 lag mit VRML 1.0 eine gebrauchsfähige Version vor, die unabhängig vom HTML-Format war. Aus der VR Markup Language wurde eine VR Modelling Language, mit der beliebig komplexe dreidimensionale Szenen beschrieben werden konnten. Allerdings waren es statische virtuelle Welten ohne Interaktion mit dem Anwender. Den Schritt zu bewegten und interaktiven virtuellen Welten leistete VRML 2.0 von 1996. Damit ist VRML zwar komplexer als HTML, aber weniger komplex als Programmiersprachen wie C++ oder Java.<sup>13</sup> Allerdings hat VRML ein objektorientiertes Datenformat, das eine dreidimensionale Szene in elementare Bausteine auflöst. Der Zustand eines Objekts, wie beispielsweise eines Würfels, wird durch die Werte bestimmter Eigenschaften, wie Größe, Farbe oder Bewegungszustände, bestimmt. So soll es beispielsweise in einer interaktiven virtuellen Szene möglich sein, dass ein Anwender durch Mausklick den Farbzustand des

<sup>13</sup> Vgl. Alexandrescu 2001.

Objekts verändern kann. Ebenfalls sollen kausale Beziehungen zwischen den Objekten einer Szene bestehen. Wenn bei zwei ineinandergeschachtelten Würfeln der äußere verschoben wird (d. h. die Werte der Eigenschaft „Translation“ verändert werden), dann soll sich auch der innere Würfel entsprechend verschieben. Die Veränderung einer Eigenschaft ist ein Ereignis (engl. event), das andere Ereignisse an anderen Objekten der Szene (ursächlich kausal) auslöst. In VRML spricht man dabei von einer Route von Ereignisketten zwischen den entsprechenden Objekten. Im objektorientierten Datenformat von VRML werden die Beziehungen zwischen den Objekten einer virtuellen Szene in einem Szenengraphen abgebildet. Dabei werden die Objekte grafisch durch Knoten (engl. nodes) dargestellt. Knoten können für geometrische Basiskörper (beispielsweise Würfel, Kugeln) stehen, aber auch für ihre Gestalt und Erscheinung, für Licht, Sound, Bewegungen und andere Faktoren, die eine dreidimensionale Szene bestimmen. Formal ist ein Knoten durch einen Namen des Objekts und Felder (fields) für seine Eigenschaften eindeutig bestimmt, die, wie in objektorientierten Sprachen üblich, in geschweiften Klammern zusammengefasst werden. Die Namen von Knoten werden groß, die Namen von Feldern klein geschrieben.

```

Shape
{
    appearance Appearance
    {
        material Material { Color 010 }
    }
    geometry Box
    {
        size 123
    }
}

```

In einer komplexen Szene ist es oftmals sinnvoll, eine große Anzahl von Knoten in so genannten Gruppenknoten zusammenzufassen. Veränderungen an einem Gruppenknoten lassen sich damit einfach auf die gesamte Szene übertragen. So kann in VRML die Vererbungseigenschaft objektorientierten Programmierens ebenfalls angewendet werden. Formal besitzt ein Gruppenknoten ein Kind (engl. children)-Feld, in das die jeweiligen Knoten in eckigen Klammern eingefügt werden. Mittlerweile steht eine VRML-Bibliothek mit vorgefertigten Knotentypen zur Verfügung, mit denen auch komplexe multimediale Szenen konzipiert und entwickelt werden können. VRML 2.0 eröffnet aber auch die Möglichkeit, neue Prototypen zu konstruieren. Ein Prototyp ist ein Schema für Knoten und besteht aus einem Namen und einer Liste von Feldern. Durch Einsetzung von Daten (oder Knoten mit Daten) in den Feldern entsteht ein ganz konkreter Knoten. Vordefinierte Knotentypen und neue Prototypen entsprechen daher den Klassen in Java oder C++,

aus denen durch Einsetzung spezieller Daten konkrete Objekte entstehen.<sup>14</sup> Eine der bemerkenswertesten Eigenschaften von VRML 2.0 ist ihre Modellierung dynamischer Szenen und die Interaktion mit dem Anwender. Ein Ereignis wird als Datenveränderung in den Feldern eines Knotens interpretiert. Über eine Route kann dieses Ereignis an einen anderen Knoten übertragen werden, um in entsprechenden Feldern eine Datenveränderung, also ein weiteres Ereignis, auszulösen. So ändert beispielsweise das Ereignis `set`.  
`position` das Feld `position` eines Knotens, während das Ereignis `positon.changed` die Veränderung des Feldes mitteilt. Bei der Definition von Knotentypen und Prototypen werden daher Ereignisse unterschieden, die Felder ändern (`eventIn`) oder die Veränderung von Feldern der Umgebung mitteilen (`eventOut`). Eine Route beginnt mit dem Ereignis `eventOut` im Feld eines Knotens und mündet in einem Ereignis `eventIn` im Feld eines anderen Knotens. VRML umfasst Schlüsselwörter zur Bezeichnung von Elementen der zu beschreibenden Welten, wie Körper und Flächen, die mit Parametern über die Lage und Form versehen werden. Darüber hinaus gibt es Strukturelemente, wie Klammern und Schlüsselwörter zur Bezeichnung zusammengefasster Gruppen. Die ganze Beschreibung einer VRML-Szene wird als normaler ASCII-Text abgespeichert.

JavaScript wurde von Netscape als Script-Sprache für den Browser NetScape Navigator entwickelt. Der Name wurde in Anlehnung an die gleichzeitig erschienende Programmiersprache Java (Sun) gewählt. JavaScript ist dabei eine beschränkt objektorientierte Sprache, die zur Formulierung einfacher Skripte entwickelt wurde. Im Sinn der Kapselung enthalten ihre Objekte neben Datenfeldern auch die dazugehörigen Methoden. So gehören beispielsweise zum Objekt Kreis neben Daten wie der Konstanten pi auch mathematische Funktionen als Methoden. JavaScript wird in HTML-Dokumente eingebettet und kann diese mit prozeduralen Elementen versehen. Man kann mit JavaScript Java-Applets in einem HTML-Dokument steuern bzw. auf Funktionalität aus den Packages zugreifen. Gleichzeitig ist es, wenn auch nur in eingeschränktem Umfang, möglich, von Java-Applets aus JavaScript-Funktionalität zu nutzen. JavaScript liegt im Quellcode innerhalb eines HTML-Dokuments vor. Die Script-Anweisungen werden dabei vollständig interpretiert, wobei eine teilweise Vorübersetzung bereits beim Laden des Dokumentes stattfindet. Einschränkend gilt, dass für die Ausführung von JavaScript immer ein Browser vorausgesetzt wird. Eigenständige Anwendungen können daher nicht geschrieben werden. Außerdem kann praktisch nicht auf das Dateisystem des lokalen Rechners zugegriffen werden. Weiterhin gilt es zu beachten, dass gerade ältere Browser Probleme mit der Ausführung von JavaScript haben können. Neben arithmetischen und logischen Operationen sind in JavaScript Anweisungen beispielsweise für bedingte Fallunterscheidungen oder Wiederholung von Schleifen vorgesehen. Die Sprache JavaScript reicht jedoch nicht immer aus, um komplexe Daten zwischen verschiedenen Rechnern im World Wide Web auszutauschen.<sup>15</sup> Die Simulation virtueller Realitäten im Netz wird erst durch eine zusätzliche Verwendung einer höheren Programmiersprache wie Java möglich. Java-Programme müssen vor der

---

<sup>14</sup> Vgl. Herrmann 2001.

<sup>15</sup> Vgl. Koch 2001.

Anwendung in Script-Knoten übersetzt werden. Aus dem Quellprogramm entsteht dann der Java-Bytecode mit dem Dateiformat `.class`. Der Bytecode wird als externe Datei auf einem WWW-Server im Netz bereitgestellt. Ein Java-Programm besteht aus Objekten, die Datenfelder und Methoden enthalten. Objekte werden nach dem Schema von Klassen erzeugt. Knoten und Knotentypen bzw. Prototypen in VRML 2.0 entsprechen daher den Objekten und Klassen in Java.

Gerade die Entscheidungsfindung setzt oftmals nicht nur das Verarbeiten von Daten oder Informationen, sondern auch von Wissen voraus. Umgangssprachlich unterscheidet man gewöhnlich zunächst eine Information von der Nachricht. So kann beispielsweise die Information, dass in Stockholm der Nobelpreis vergeben wurde, in verschiedenen Nachrichten, und dort in verschiedenen Sprachen, übermittelt werden. Die Information bleibt also dieselbe, obwohl sich die Nachricht unterschiedlich gestaltet, beziehungsweise in unterschiedlicher Ausprägung übermittelt wird. Technisch-physikalisch können Nachrichten durch verschiedene Signale vom Morsen, Rundfunk, Fernsehen bis hin zur E-Mail übertragen werden. Insofern handelt es sich bei einer Nachricht um eine endliche Zeichenfolge, die eine Information übermittelt. Eine Nachricht wird dabei von einer Nachrichtenquelle codiert, über einen Kanal zu einem Empfänger geschickt und dort decodiert. Information bezieht sich auf die Bedeutung der Nachricht für den Empfänger. Wissen hingegen besteht aber nicht nur in der Anhäufung von Informationen. Wissen bezieht sich vielmehr auf die Fähigkeit, Informationen so zu verwerten dass damit gegebenenfalls Probleme gelöst werden können. Wissen ist nach dieser Auffassung Information plus Know-how. In den bisher besprochenen prozeduralen oder algorithmischen Programmiersprachen wurde dem Computer für eine Problemlösung mehr oder weniger genau mitgeteilt, welche Anweisungen, Funktionen oder Methoden auszuführen sind. Bei der computergestützen Wissensverarbeitung wird in einem Programm nur das Wissen über ein bestimmtes Problem deklariert. Der Rechner soll dann anhand dieses Programms selbstständig mit Hilfe dieses Wissens eine Lösung des Problems finden. Ein erstes Beispiel solcher nicht-prozeduralen Sprachen waren die strukturierten Abfragesprachen komplexer Datenbanksysteme, mit denen der Anwender das Arbeitsergebnis (beispielsweise eine Datenrecherche) beschreibt, um es dann vom Rechner in die notwendigen Anweisungen zur maschineninternen Ausführung übersetzen zu lassen.<sup>16</sup> Im Anschluss an die 3. Generation der höheren prozeduralen Sprachen wird von den 4-GL-Sprachen (engl. 4th Generation Language) der 4. Generation gesprochen wie beispielsweise SQL (engl. Structural Query Language).<sup>17</sup> In der 5. Generation geht es um die *Programmiersprachen der Wissensverarbeitung*, wobei in diesem Buch die zwei wichtigsten Vertreter kurz vorgestellt werden: Prolog und Lisp.<sup>18</sup>

Die ersten Grundgedanken, die später als Basis für die Entwicklung von PROLOG herangezogen wurden, reichen bis in das Jahr 1879. In diesem Jahr verfasste der deutsche Mathematiker und Philosoph Gottlob Frege seine Begriffsschrift, die inzwischen als ein

---

<sup>16</sup> Vgl. Misgeld 2001.

<sup>17</sup> Vgl. Date und Darwen 1998.

<sup>18</sup> Vgl. Mayer 1995.

Entwurf einer formalen Einheitswissenschaft angesehen ist. In ihr wurde versucht, die formale abstrakte Beschreibung von Gedanken in mathematische Notation zu bringen.

Die logischen Verhältnisse kehren überall wieder, und die Zeichen für die besonderen Inhalte können so gewählt werden, dass sie sich in den Rahmen einer Begriffschrift einfügen<sup>19</sup>.

Diese Notation wurde in den Nachfolgejahren – teils unter dem Deckmantel der Kritik – systematisch von Whitehead und Russel um wesentliche Basiselemente wie Funktion, Endlichkeit, Beweisbarkeit erweitert. In diesem Jahrhundert wandten sich die Mathematiker Jacques Herbrand, Thoralf Skolem und Kurt Gödel dieser formalen Weltsicht zu und entwickelten Verfahren, die die Beweisbarkeit beliebiger Sätze ermöglichen sollten. Es wurden die mathematischen Grundlagen gelegt, auf die die heutigen Theorembeweisverfahren im Prädikatenkalkül aufbauen. In den fünfziger Jahren wurden mit dem Auftauchen der ersten Rechner diese Beweisverfahren rechnergestützt durchgeführt. Alle früheren Versuche scheiterten an dem Umstand der „kombinatorischen Explosion“ und der algorithmischen Komplexität, da die einzelnen Varianten meistens nur durch systematisches Durchprobieren aller Varianten gefunden werden konnten. Erst um das Jahr 1965 machte J.A. Robinson mit seinem Resolventenprinzip einen großen Schritt nach vorne, indem er mit seiner Entwicklung die Komplexität derartiger Algorithmen wesentlich einschränken konnte. Parallel zu diesem eher mechanischen Beweisverfahren beschäftigte sich in Aberdeen eine Forschungsgruppe mit der Entwicklung eines intelligenten Antwort-Frage-Systems (ABET). Ein herausragendes Merkmal dieses Systems war dessen Eigenschaft, während seiner Arbeit – sozusagen interaktiv – sich neue Regeln hinzuzufügen. Dieser dynamische Regelzuwachs konnte dann die weiteren Konsultationen und damit die Lösungsfindung wesentlich beeinflussen. Im Jahre 1971 entwickelte Colmerauer und sein Forscherteam das Basissystem eines anderen Frage-Antwort-Systems; SYSTEM-Q. Dieses System baut auf Hornklauseln auf, nutzt das Resolventenprinzip und die Unifikation. Auch hier konnte das System interaktiv neue Regeln hinzufügen. Dieses Basissystem wurde anschließend zu einem Interpreter für eine eigenständige Programmiersprache entwickelt. Man gab diesem System den Namen PROLOG. In den Nachfolgejahren beschäftigte man sich an unterschiedlichen Orten mit dem Phänomen der logischen Programmierung. Kowalski führte die Gleichung

$$\text{Algorithm} = \text{Logic} + \text{control}$$

ein, in dem Sinne, dass Algorithmen immer zwei Komponenten implizit enthalten müssen: eine Logik-Komponente, die das Wissen über das zu lösende Problem spezifiziert, und eine Steuerungs- bzw. Kontrollkomponente, die die Lösungsstrategie für das Problem darstellt. Während bei konventionellen Programmiersprachen diese beiden Komponenten

---

<sup>19</sup> Begriffschrift und andere Aufsätze, hrsg. Von I. Angelelli, 2. Aufl. Darmstadt 1964.

stark vermischt und kaum zu trennen sind, sollen Logik-Programme nur eine Logikkomponente darstellen, während die Steuer- und Kontrollkomponenten dem System überlassen bleiben. Oder etwas praktischer formuliert: Alle problemspezifischen Informationen werden in einer Wissensbasis in Form von Horn-Klauseln gespeichert, die dann durch eine problemunabhängige Inferenzmaschine ausgewertet werden. In PROLOG (engl. Programming in Logic) wird das Wissen über ein Problem in einem System von wahren Aussagen (Fakten) und Regeln zusammengestellt.<sup>20</sup> Ein Problem wird im Sinne der Logik als eine Behauptung verstanden, für die das PROLOG-System selbstständig eine Problemlösung in Form eines Beweisschlusses sucht. Logisch besteht eine wahre Aussage aus einem oder mehreren Objekten, auf die eine Eigenschaft (Prädikat) oder Beziehung (Relation) zutrifft. Hat man auf diese Weise die Verhältnisse beschrieben, können Anfragen gestellt werden, auf die man dann Antworten, respektive Lösungen, erhält.

```

mensch(sokrates).                                /* Faktum */
sterblich(X) :- mensch(X).                      /* Regel */
? sterblich(sokrates).                           /* Anfrage */
Yes.                                              /* Antwort */

```

PROLOG ist eine prädiktative Programmiersprache. Die Namen von Variablen, an deren Stelle Namen von Objekten eingesetzt werden können, sollen groß geschrieben werden. In PROLOG wird der Schlussatz vor die Voraussetzungen gesetzt und durch das Schlussymbol:- abgetrennt. Mehrere Voraussetzungen werden mit Kommata aneinander gereiht. Durch geschicktes Einsetzen von Namen von Objekten für Variablen und Anwendung von logischen Regeln sucht das System einen Beweis, nach dem diese Behauptung aus der Wissensbasis ableitbar ist. Die Lösung einer Aufgabe findet das PROLOG-System durch eine kombinatorische Zurückführung (engl. Backtracking) auf die Wissensbasis: Es werden dabei alle Möglichkeiten solange ausprobiert, bis eine adäquate Lösung gefunden wird. Dabei wird versucht, Aussagen durch Variablenersetzung an vorausgesetzte Fakten anzulegen (unifizieren). In PROLOG kommt es darauf an, effiziente Algorithmen für diese Unifizierungen zu finden. Wissensverarbeitung bedeutet also primär Backtracking von Problemen auf eine Wissensbasis.

Während prädiktative Programmiersprachen der 5. Generation an der Logik orientiert sind, verwenden funktionale Programmiersprachen wie in der Mathematik Funktionen, die Abhängigkeiten von Symbolfolgen darstellen. Als Datenstrukturen zur funktionalen Darstellung von Algorithmen werden in der Programmiersprache LISP (engl. List Processing Language) Listen von definierten Symbolen verwendet. Die kleinsten (unteilbaren) Bausteine von LISP heißen Atome. Diese können Zahlen, Zahlenfolgen oder Namen sein. Aus den Atomen werden schrittweise neue symbolische Ausdrücke, sogenannte s-Ausdrücke, zusammengesetzt. Wenn x und y s-Ausdrücke sind, dann soll auch (x,y) ein S-Ausdruck sein. Aus dem Symbol Nil aus der Nachrichtentechnik für eine leere Symbolfolge

<sup>20</sup> Vgl. Clocksin und Mellish 1990.

und s-Ausdrücken werden dann verkettete Listen erzeugt. Wenn x ein s-Ausdruck und y eine Liste ist, dann soll auch (x.y) eine Liste sein. Zur algorithmischen Verarbeitung von Datenlisten werden einfache Grundfunktionen eingeführt. So liefert die Funktion `car` den linken Teil eines s-Ausdrucks, d. h.  $(\text{car } (x.y)) = x$ , während die Funktion `cdr` den rechten Teil ergibt, d. h.  $(\text{cdr } (x.y)) = y$ . Die Funktion `cons` vereinigt zwei S-Ausdrücke zu einem s-Ausdruck  $(\text{cons } xy) = (x.y)$ . Auf Listen angewendet liefert `car` das erste Element und `cdr` die restliche Liste ohne das erste Element. Anschaulich können Listen und s-Ausdrücke mit diesen Grundfunktionen auch als binär geordnete Bäume dargestellt werden. Funktionskompositionen wie beispielsweise  $(\text{car } . (\text{cdr } . x))$  drücken die Hintereinanderausführung zweier Funktionsanwendungen aus, wobei die innere Funktion zuerst ausgewertet wird. Bei mehrstöckigen Funktionen werden alle Argumente zuerst ausgewertet und dann die Funktion angewendet. Listen werden in der Regel als Anwendung einer Funktion aufgefasst. Dann bedeutet (ABCDEF), dass die Funktion A auf B, C, D, E und F anzuwenden ist. Oft ist es aber auch nützlich, Listen als geordnete Mengen von Symbolen aufzufassen. So macht es wenig Sinn (123451) als Anwendung der Funktion 1 auf die Argumente 2, 3, 4, 5 zu lesen, wenn es um eine Sortierungsaufgabe der Zahlen geht. In LISP wird daher das Symbol `quote` eingeführt, wonach die folgende Liste nicht als Funktionsanweisung, sondern als Aufzählung von Symbolen zu verstehen ist: `quote 1123451` oder `kurz= (123451)`. Dann ist laut Definition beispielsweise `car' 11231=1`, `cdr' 31= , (23)` und `cons 1' (23) =' (123)`.

Die Form einer Funktionsdefinition lautet: (De Name (p1 p2...pn) s-Ausdruck). Dabei ruft die Abkürzung De eine Definition auf. Name ist die Bezeichnung der Funktion, wobei p1, p2,..., pn ihre formalen Parameter sind. Der s-Ausdruck heißt Rumpf der Funktion und beschreibt die Funktionsanwendung mit den formalen Parametern. Wenn in einem Programm die Funktion Name in der Form (Name a1 a2...an) auftritt, dann sind im Rumpf der Funktion die formalen Parameter pi (= Variable) durch die entsprechenden aktuellen Parameter ai (= Konstante) zu ersetzen und der so veränderte Rumpf der Funktion auszuwerten. Als Beispiel wird die Funktion Drei definiert, die das dritte Element einer Liste berechnet: (De Drei(liste) (`car(cdr(cdr liste))`)). Die Funktionsanordnung `Drei' (415)` ersetzt dabei im Rumpf der Funktion Drei die formalen Parameter durch (`car(cdr(cdr' 1 415))`). Die Auswertung liefert dann den Wert 5 als drittes Element der vorgelegten Liste. Um Bedingungen formulieren zu können, werden neue Atome wie beispielsweise T für „wahr“ (engl. true) und NIL für „falsch“ und neue Grundfunktionen wie beispielsweise equal zum Vergleich zweier Objekte eingeführt: (`equal 12`) = NIL. Bedingungen und Voraussetzungen werden durch con (engl. bedingung) angezeigt. Ein LISP-Programm ist allgemein eine Liste von Funktionsdefinitionen und ein s-Ausdruck, der mit diesen Funktionen ausgewertet wird. Auf der Maschinenebene werden LISP-Programme überwiegend durch Interpreter zur Ausführung gebracht. Mittlerweile existieren auch LISP-Maschinen, die Rechenzeit ersparen.

Der Grundgedanke der *objektorientierten Programmierung* besteht darin, die Daten mit passenden Funktionen als Objekte zusammenzufassen. Eine Programmausführung gestal-

tet sich demnach als ein System miteinander kooperierender Objekte. Diese Objekte haben einen eigenen Zustand, besitzen eine gewisse Lebensdauer und empfangen, beziehungsweise bearbeiten Nachrichten. Bei der Verarbeitung dieser Nachrichten kann das Objekt seinen Zustand ändern, Nachrichten an andere Objekte verschicken, neue Objekte erzeugen oder aber existierende Objekte löschen. Objekte sind also autonome Ausführungseinheiten, die unabhängig voneinander und parallel arbeiten können. Eines der basalen Ziele der objektorientierten Programmierung besteht darin, eine möglichst gute softwaretechnische Modellierung der realen Welt zu unterstützen und somit eine gute Integration von realer und softwaretechnischer Welt zu ermöglichen. So verhalten sich Objekte wie Gegenstände der materiellen Welt. Insbesondere haben die Objekte eine Identität, indem ein solches nicht gleichzeitig an zwei Orten sein kann. Ein Objekt kann sich zwar ändern, bleibt dabei aber dasselbe Objekt.

Den Schritt zur objektorientierten Programmierung machte in den achtziger Jahren (1983) Bjarne Stroustrup mit der Sprache C++. Dabei leitet sich der Name C++ von der eher nebensächlichen Eigenschaft ab, dass der Operator `++` in der Verbindung `x++` die Zählervariable `x` nach dem Schema `x + 1` hochzählt. Historische Vorläufer waren die Programmiersprachen SIMULA und SMALLTALK. Den Durchbruch schaffte erst C++ wegen ihrer großen Standardisierung und weiten Verbreitung. Zwar gab es auch in der Programmiersprache C bereits den Begriff der Struktur, in der verschiedene Daten in verschiedenen Variablen abgelegt werden können, in objektorientierten Sprachen wie C++, wird dieser Strukturbegriff allerdings zum Klassenbegriff erweitert. Eine Klasse ist nach dieser Auffassung eine Struktur, in der nicht nur Daten abgelegt werden, sondern auch die Funktionen, die diese Daten modifizieren können. Eine Klasse wird in C++ durch das Schlüsselwort `class` realisiert. Es leitet den Beginn einer Klassendeklaration ein. Der prinzipielle Aufbau einer Klasse sieht wie folgt aus:<sup>21</sup>

```
class konto
// Vereinbarung einer Klasse namens Konto
{
    public:
        // Deklarationen der öffentlichen
        // Variablen und Funktionen der Klasse
    private:
        // Deklarationen der privaten
        // Variablen und Funktionen der Klasse
    protected:
        // Deklarationen der abgeleiteten
        // Variablen und Funktionen der Klasse
};
```

---

<sup>21</sup> Vgl. Stroustrup 2002.

Die Schlüsselworte `public`, `private` und `protected` klassifizieren die Informationen innerhalb der Klasse:

- öffentlicher Bereich (`public`): ist allen zugänglich
- privater Bereich (`private`): darauf dürfen nur Funktionen innerhalb der Klasse zugreifen
- geschützter Bereich (`protected`): dieser Bereich verhält sich für abgeleitete Klassen wie `public`, für alle anderen jedoch wie `private`.

Eine Klasse ist also nur eine Art Definitionsschema für die Erzeugung von Instanzen, oder anschaulich gesprochen, eine Art Bauplan für konkrete Objekte. In der objektorientierten Programmierung werden Funktionen auch als Methoden bezeichnet. In einem Objekt sind also Datenfelder und die dazu passenden Methoden zu einer Einheit zusammengefasst. Man spricht in diesem Zusammenhang von der Kapselung der Datenfelder und den passenden Methoden in Objekten. Datenfelder sind dadurch nach außen hin abgeschirmt und können nur durch erlaubte Zugriffsmethoden erreicht werden. Damit ist der Programmfluss der verwendeten Variablen eindeutig geregelt. Die Implementierung der Methoden einer Klasse befindet sich zumeist außerhalb der Klassenvereinbarung.

```
float konto::einzahlen(float betrag)
{
    // hier folgt die konkrete Implementierung
    // einer Funktion einzahlen als Memberfunktion
    // der Klasse konto
}
```

Die Zugehörigkeit zu einer Klasse wird dem Compiler syntaktisch durch den Klassennamen und zwei folgenden Doppelpunkten `::` angegeben. Um mit diesen Objekten in einem C++ Programm arbeiten zu können, muss, wie in einem C-Programm, die Hauptfunktion `main` genau einmal vorkommen. Sie löst wie in einem C-Programm die Bearbeitung durch den Computer aus. Analog zu C-Bibliotheken mit ihren C-Funktionen lassen sich in einer C++-Bibliothek Klassen zusammenstellen, um standardisierte Schnittstellen für bestimmte Objekte bereitzustellen. Insofern stehen damit fertige Module für die standardisierte Montage eines Programms zur Verfügung, hier aber nicht nur für Methoden, sondern gerade für Datenfelder und ihre Methoden in gekapselten Objekten.

Datenfelder speichern das Wissen, respektive Eigenschaften, eines Objekts, Methoden dessen Verhaltens. Die Philosophie der Kapselung besagt also, dass objektorientiertes Programmieren nicht nur Wissen oder Methoden getrennt anbietet, sondern Wissen zusammen mit dem nötigen Know-how. Viele Objekte weisen große Gemeinsamkeiten auf. Häufig verfügen sie über die Datenfelder und Methoden anderer Objekte, die nur erweitert wurden. Speziell hierzu können aus einer Klasse neue Klassen abgeleitet werden, die automatisch alle Daten und Methoden des Grundtyps haben, ohne dass sie nochmals extra definiert werden müssten. Anschaulich gesprochen werden Daten und Methoden

einer Oberklasse (Elternklasse, engl base class) an ihre Unterklassen (Kindklassen, engl. child class) vererbt. Neben der Kapselung ist die Vererbung damit ein weiteres und vor allem zentrales Rationalisierungsprinzip objektorientierter Programmierung. Der einmal erstellte Programmcode der Oberklassen ist nämlich wiederverwendbar. Nur noch Ausnahmen und Erweiterungen müssen neu programmiert werden. Das erspart Zeit, reduziert Fehler und kommt der Pflege und Wartung von Programmen entgegen.

Java ist zunächst eine streng objektorientierte Programmiersprache, die sich gegenüber C++ durch weitere Rationalisierungen, Standardisierungen und Sicherungsmechanismen auszeichnet. Java ist aber vor allem eine Trias, bestehend aus

- Sprache,
- Plattform,
- Programmierschnittstelle (Application Interface, API),

die ursprünglich für vernetzte Geräte der Unterhaltungselektronik entwickelt wurde. Die Programmiersprache Java basiert vollständig auf objektorientierten Konzepten, was den Einsatz von modernen Modellierungsmethoden gestattet, sowie die Wiederverwendung von Software-Komponenten ermöglicht. Bezuglich der Syntax ähnelt Java den Sprachen C respektive C++. Allerdings gibt es auch eine Vielzahl von Unterschieden, insbesondere durch fehlende, kompliziert erscheinende Konstrukte, die häufig zu Fehlern und deren zeitraubenden Suche verantwortlich waren. Zu diesen Konstrukten gehören in erster Linie die Zeiger (engl. Pointer). Auf diese kann in Java verzichtet werden, da sie durch eine konsequente Verwendung von Objekten nicht mehr benötigt werden. So wird in diesem Zusammenhang die Speicherverwaltung gänzlich dem Java-System überlassen, das mit Hilfe des Garbage Collection diese Aufgabe selbstständig übernimmt. Dies bringt den Vorteil mit sich, dass Objekte nicht unbenutzt im Speicher ihr Dasein fristen und diesen für andere Objekte blockieren. Ein weiterer Grund, warum man auf Zeiger verzichtet hat, liegt darin begründet, dass Java über ein ausgeklügeltes Sicherheitskonzept und eine integrierte Netzwerk-Unterstützung verfügt. Java ist dabei eine verhältnismäßig einfache Sprache, da sie über einen relativ kleinen Sprachumfang verfügt, der außerdem noch einigermaßen leicht zu erlernen ist. Die wohl wichtigste Vereinfachung besteht sicherlich darin, dass Java keine Zeiger verwendet. Vielmehr übernimmt Java die Referenzierung und Dereferenzierung von Zeigern automatisch. Ebenso führt Java automatisch Garbage Collection durch, wodurch der Entwickler von fast allen Aspekten der Speicherverwaltung befreit wird.

Um die Portabilität von Java-Anwendungen gewährleisten zu können, definierte man eine virtuelle Maschine, für die auf den heutigen üblichen Hardware-Plattformen effiziente Interpretierer-Implementierungen zur Verfügung stehen. Diese Tatsache macht einen großen Teil der Portabilität aus. Jedoch geht Java noch einen Schritt weiter, indem es sicherstellt, dass keine implementierungsabhängigen Aspekte der Sprachspezifikation existieren. So wird beispielsweise bei Java explizit die Größe jedes primitiven Datentyps festgelegt, ebenso wie dessen arithmetisches Verhalten. Das 100 % Pure-Java-Programm von Sun hilft dabei sicherzustellen, dass der Code portabel ist gemäß dem Motto: „Write

once, Run Anywhere (Einmal schreiben und überall ausführen)“. Java ist eine interpretierte Sprache. Dabei werden Java-Programme mit einem Compiler in den sogenannten Byte-Code für die virtuelle Maschine (JVM) übersetzt. Diese Maschine verfügt über eine recht kleine Menge von Instruktionen. Der Bytecode wird von einem Interpretierer ausgeführt und kann damit auf jeder Plattform verwendet werden. Zur Steigerung der Ablaufgeschwindigkeit verfügen einige Interpretierer über sogenannte Just-in-Time-Compiler, die den Java-Byte-Code zur Laufzeit in nativen Maschinencode übersetzen. Java ist eine streng objektorientierte Programmiersprache. In einem objektorientierten System besteht eine Klasse aus einer definierten Ansammlung von Daten, sowie aus Methoden, die mit diesen Daten zusammenarbeiten. Daten und Methoden beschreiben den Zustand und das Verhalten eines Objekts. Klassen werden in einer Hierarchie angeordnet, so dass Unterklassen das Verhalten von übergeordneten Klassen (Superklassen) erben können. Eine solche Klassenhierarchie besitzt stets eine Stammklasse (Wurzel). Diese Klasse verfügt über das allgemeine Verhalten des Systems. Die Grundelemente einer objektorientierten Sprache sind

- Kapselung,
- Vererbung und
- Polymorphismus.

Kapselung bedeutet, dass die Daten eines Objektes nur über definierte Methoden zugänglich sind. Das heißt, dass die Daten, und somit die internen Strukturen in den Objekten gekapselt sind. In diesem Zusammenhang spricht man auch von data hiding. Vererbung bedeutet, dass Objekte Eigenschaften und Methoden von anderen Objekten erben und damit übernehmen können. Klassen sind dabei über eine „ist-eine“-Verbindung miteinander verbunden und bilden die oben angesprochene Klassenhierarchie, an deren Wurzel die einfachste, an ihren Enden die spezialisierten Klassen liegen. Java wird mit einem sehr umfangreichen Satz von Klassen ausgeliefert und ist in Paketen organisiert. So stellt Java beispielsweise Klassen zur Verfügung, mit denen Komponenten grafischer Anwendoberflächen entwickelt werden können (java.awt-Paket), mit denen die Ein- und Ausgaben behandelt werden können (java.io-Paket), sowie Klassen, die die Netzwerkfunktionalität gewährleisten (java.net-Paket). Die Object-Klasse (java.lang-Paket) bildet dabei den Stamm der java-Klassenhierarchie.

Java ist eine dynamische Sprache. Jede Klasse kann zu jeder Zeit in einen laufenden Java-Interpretierer geladen werden. Diese dynamisch geladenen Klassen können dann dynamisch instantiiert werden. Maschinenspezifische (native) Bibliotheken können ebenfalls dynamisch geladen werden. Java ist auch eine verteilte Sprache. Das bedeutet, dass sie Netzwerke auf einer sehr hohen Ebene unterstützt.<sup>22</sup> Beispielsweise ermöglicht die URL-Klasse, sowie die verwandten Klassen im java.net-Paket, das Lesen entfernter Dateien oder Ressourcen. In ähnlicher Art und Weise erlaubt es das RMI-API (Remote Method Invoca-

<sup>22</sup> Vgl. Kurose und Ross 2002.

tion), dass ein Programm Methoden entfernter Objekte aufruft, so, als wären es lokale Objekte: Ein Client ruft eine Methode auf, der Server führt sie aus und liefert „lediglich“ das Ergebnis zurück. Java wurde für die Entwicklung sehr zuverlässiger oder robuster Software entworfen. Der Verzicht auf Zeiger, die automatische Speicherverwaltung, die strenge Objektorientierung mit gleichzeitiger strenger Typprüfung, der Verzicht auf Operator, Overloading und multipler Vererbung, sowie die Einschränkung der automatischen Typumwandlung, eliminiert „von Hause aus“ ganz bestimmte Arten von Programmierfehlern, was der Entwicklung zuverlässiger Software zu Gute kommt. Java ist eine sichere Sprache. Auf der untersten Ebene können beispielsweise Java-Programme keine Zeiger auf den Arbeitsspeicher erzeugen, Arrays überfluten oder gar Speicher außerhalb von deren Grenzen lesen. Indem ein direkter Zugriff auf den Speicher unterbunden wird, wird eine große und sehr unangenehme Klasse von Sicherheitsattacken ausgeschlossen. Weiterhin überprüft der Java-Interpretierer jeden ihm unbekannten Code. Die Verifikationsschritte stellen dabei sicher, dass der Code richtig geformt ist, d. h., dass er keinen Über- oder Unterlauf des Stacks verursacht, oder illegale Byte-Codes enthält. Eine weitere Sicherheits-Ebene stellt das sogenannte Sandkasten-Modell dar. Unbekannter Code wird dabei in einem „Sandkasten“ platziert, in dem er sicher „spielen“ kann, ohne der realen Welt Schaden zufügen zu können. Wird ein Applet oder ein anderer Code in diesem Sandkasten ausgeführt, gelten eine Reihe von Einschränkungen. Eine dieser Einschränkungen besteht beispielsweise darin, dass der Code keinerlei Zugriff auf das lokale Dateisystem hat. Zusätzlich enthält der Interpretierer einen sogenannten Bytecode-Verifier, der das Programm vor dem Ablauf nach einigen Regeln hin überprüft. Um Applets ein größeres Maß an Funktionalität und Flexibilität zukommen zu lassen, werden bereits in der Version 1.1. die signierten Applets eingeführt. Solche Applets sind mit einer digitalen Signatur versehen und können dieselben Zugriffsrechte wie lokale Applets zugeteilt bekommen. Java ist eine interpretierte Sprache. Dennoch ist die Geschwindigkeit für die Ausführung interaktiver oder netzwerkbasierter Anwendungen mehr als ausreichend. Zur weiteren Leistungssteigerung beinhaltet viele Java-Interpretierer die „Just-in-Time“-Compiler, die Java-byte-Codes zur Laufzeit in den Maschinencode einer bestimmten CPU übersetzen. Sun behauptet zu Recht, dass die Leistung von in Maschinencode übersetztem Bytecode fast so gut ist wie echter C- oder C++-Code. Gerade für Anwendungen, die als Applets in einem Web-Browser ablaufen, ist es sinnvoll, dass ein Applet in mehrere einzelne Prozesse bzw. Threads unterteilt werden kann, die dann unabhängig voneinander ablaufen. Java ist eine Multithread-fähige Sprache, die deren Verwendung recht einfach gestaltet. Damit kann sich beispielsweise ein Thread um die Aktualisierung der Bildschirmausgaben kümmern, während ein anderer Prozess über einen TCP/IP-Socket mit einem Server kommuniziert.<sup>23</sup> Neben den in Web-Seiten eingebundenen Applets können in Java aber auch völlig eigenständige Programme geschrieben werden, die unabhängig von einem Web-Browser ablaufen können. Letztere bezeichnet man als Applikationen. Um Applikationen laufen zu lassen, benötigt man entweder einen Byte-Code-Interpretierer wie Java, der Bestandteil des Java Development Kit

---

<sup>23</sup> Siehe auch Corner und Stevens 2000.

(JDK) ist, oder aber einen Compiler, der aus dem Byte-Code ein echtes Binärprogramm für eine bestimmte Plattform generiert. Solche Compiler werden teilweise mit den verfügbaren integrierten Entwicklungsumgebungen ausgeliefert. Das folgende Beispiel zeigt das klassische Java-Programm „Hello World“.

```
public class HelloWorld{
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

Dieses Programm besteht, wie jedes Java-Programm, aus einer Klassendefinition vom Typ `public`. Die Klasse enthält eine Methode namens `main()`, die den Einstiegspunkt für alle Java-Anwendungen darstellt, d. h. an diesem Punkt beginnt der Java-Interpretierer mit der Ausführung des Programms. `main()` selbst besteht nur aus einer Zeile, die die Nachricht „Hello World!“ ausgibt. Dieses Programm muss in einer Datei gesichert werden, die den gleichen Namen besitzt wie die `public`-Klasse, wobei an den Namen noch die Erweiterung `.java` angehängt werden muss. Um das Programm zu kompilieren, kann `javac` verwendet werden: `javac HelloWorld.java`. Dieser Befehl erzeugt die Datei `HelloWorld.class` im aktuellen Verzeichnis. Um das Programm auszuführen, wird der Java-Interpreter `java` benutzt: `java HelloWorld`.

### 5.2.5 Programmietechniken

Im Bereich der Robotik wird man nicht nur mit Problemstellungen im „Kleinen“ konfrontiert, die dann eine Realisierung mit Hilfe relativ kurzer Programme bedingen. Vielmehr stößt man bei der Entwicklung intelligenter Robotersysteme sehr schnell an die Grenzen des Kleinen, und die Softwareentwicklung gereicht zu einer Programmierung im Großen, bei der erst Tausende von Programmcodezeilen die komplexe Lösung ergeben.<sup>24</sup> Die Programmierung im Großen unterscheidet sich von der Programmierung im Kleinen nicht nur in quantitativer, sondern auch in qualitativer Hinsicht. Ein kleines Programm wird meist nur von einem einzelnen Entwickler erstellt, der zudem oft der einzige Benutzer dieses Programms ist. An der Entwicklung eines großen Systems sind dagegen mehrere Entwickler beteiligt, und das System soll anschließend von vielen Benutzern oder in mehreren Robotersystemen eingesetzt werden. Ein großes Softwaresystem hat demnach einen anderen Lebenszyklus, der sich über mehrere Monate oder sogar Jahre erstreckt. Dadurch muss ein solches System im Laufe der Zeit bezüglich der sich ändernden Anforderungen laufend angepasst werden. Damit sind bei der Software nicht primär die reine Laufzeit und Speichereffizienz wichtig, sondern eher Aspekte, wie Ergonomie, Zuverlässigkeit, Sicher-

<sup>24</sup> Siehe auch Frank 1991.

heit, Erweiterbarkeit, Übertragbarkeit und so weiter. Die meisten Probleme beim Übergang vom Kleinen zum Großen bereitet jedoch die logische Komplexität der zu lösenden Aufgabe und der resultierenden Software: Während bei der Programmierung im Kleinen die Problemstellung und das Programm noch gut überschaubar sind, hat man es bei der Programmierung im Großen mit deutlich komplizierteren Aufgaben und Lösungen zu tun. Aus diesen Beobachtungen folgt, dass eine Programmierung im Großen kein kreativ-künstlerisch-schöpferischer Akt mehr sein kann (wie es oft bei der Programmierung im Kleinen der Fall ist), sondern eher als systematischer Entwicklungsprozess gestaltet werden muss (siehe auch Kapitel über das Vorgehensmodell).

Dabei sind die folgenden Aspekte wichtig:

- *Strukturierte Vorgehensweise*: Auf der Basis allgemein anerkannter Prinzipien müssen bewährte Methoden und Techniken sowie entsprechende Werkzeuge eingesetzt werden.
- *Steuerung und Kontrolle*: Die Entwicklung muss hinsichtlich der zeitlichen Vorgehensweise und der dabei entstehenden Systemstruktur gesteuert und kontrolliert werden. Die zeitliche Organisation erfordert ein zielorientiertes Vorgehensmodell, bei dem eine Reihe von abgegrenzten Schritten in einer wohldefinierten Reihenfolge ausgeführt wird. Die strukturelle Organisation muss den Prozess auf die Realisierung eines klar strukturierten Systems ausrichten, das aus einer Sammlung von Komponenten besteht.
- *Problemorientierung*: Die Entwicklung muss berücksichtigen, dass die zu erstellende Software eine Lösung darstellt, also eine Antwort auf eine Problemstellung ist. Die entstehende Lösung sollte sich also nicht primär am technisch Machbaren orientieren, sondern an den Anforderungen aus der Problemstellung. Darüber hinaus sind wirtschaftliche Aspekte, wie Kosten, Zeitaufwand und Produktqualität zu beachten, ebenso wie der Aufwand zur Installation, Betrieb und Wartung der Software.

Die Erkenntnis, dass die Entwicklung von Software planmäßig vorgehen sollte, ist dabei nicht neu. Bereits im Jahre 1968 fanden diverse Veranstaltungen statt, die sich mit dieser Problematik beschäftigten. Auf diesen Tagungen wurde insbesondere der Begriff des *Software Engineering* geprägt, um die Forderung nach einer ingenieurmäßigen Erstellung von Software zu betonen.<sup>25</sup> Software Engineering wird ins Deutsche zumeist als Softwaretechnik übersetzt; in manchen Publikationen bezeichnet Softwaretechnik aber nur die rein technischen Fragestellungen, während Software Engineering zusätzlich die organisatorischen Aspekte der Softwareerstellung umfasst. In diesem Buch werden die beiden Begriffe zunächst gleichgesetzt, um dann später diese Gleichsetzung in Richtung eines *Robotik Engineering* auszubauen. Seit Anfang der 70er Jahre setzte sich langsam aber stetig eine Reihe grundlegender Prinzipien für die Softwareentwicklung durch. Auf dieser Basis entstanden unterschiedliche *Vorgehensmodelle*, die sowohl den zeitlichen Ablauf eines Entwicklungsprozesses festlegen, den zugehörige *Notationen* und die zum Einsatz kommen-

---

<sup>25</sup> Siehe auch Kahlbrandt 1998.

den Werkzeuge vorschlagen, mit denen die entstehenden Entwürfe und Softwarekomponenten modelliert bzw. entwickelt werden können. Von besonderem Interesse gerade im Bereich der Robotik sind heute objektorientierte Vorgehensmodelle, die in allen Phasen des Entwicklungsprozesses auf das Objekt- und Klassenprinzip setzen. Der Erfolg objektorientierter Ansätze manifestiert sich im immer stärkeren Einsatz objektorientierter Programmiersprachen, wie insbesondere Java, und entsprechender Modellierungssprachen, wie insbesondere der *Unified Modeling Language (UML)*.

Eine systematische und problemorientierte Softwareentwicklung muss sich an bestimmten Prinzipien orientieren, also bestimmten Handlungsgrundsätzen folgen. In vielen Bereichen der Informatik werden Strukturen und Vorgehensweisen der realen Welt durch *Modelle* nachgebildet. Entscheidend bei der Modellierung ist, dass man die Realität nicht in allen ihren Einzelheiten erfasst, sondern Details weglässt, die für den jeweiligen Zweck unwichtig sind. Mit einer Modellierung geht also eine gewisse und einkalkulierte Abstraktion von den realen Verhältnissen einher, wodurch die *logische Komplexität* des Problems und seiner Lösung reduziert wird. Je nach Zweck des Modells sind unterschiedliche Abstraktionsniveaus möglich, also eine starke oder nur eine schwache Vereinfachung der Realität. Ein Modell besitzt zudem eine *Notation*, also eine Beschreibungssprache, in der es dargestellt wird. Zur Modellierung von *Abläufen* werden zum Beispiel

- *Maschinenmodelle (Automaten)* in einer grafischen Darstellung oder in mathematisch-formaler Notation,
- *Petri-Netze*, vornehmlich in einer grafischen Darstellung, sowie
- *Struktogramme* und *Programmablaufplane*, ebenfalls mit grafischen Darstellungen

eingesetzt.<sup>26</sup> Zur Modellierung von *Datenstrukturen* werden beispielsweise

- *attributierte Graphen* mit einer Darstellung in grafischer Form oder in Mengenschreibweise,
- *Entity-Relationship-Modelle* in einer grafischen Darstellung und
- *formale Sprachen* mit einer Darstellung durch Grammatiken in Backus-Naur-Form,
- *Syntaxdiagramme* oder reguläre Ausdrücke benutzt.

Zur integrierten Modellierung von *Datenstrukturen* und *Abläufen* dienen *Objekte*, die Dinge der realen Welt mit ihren Eigenschaften repräsentieren. Dargestellt werden Objekte durch Konstrukte einer Programmiersprache, beispielsweise *Java*, oder in grafischer Form, beispielsweise *UML-Diagramme*. Modelle zu bilden ist insbesondere wichtig, wenn man Software für Robotersysteme erstellen muss. Ein Modell verdeutlicht hier die relevanten Bestandteile einer Problemstellung und dient somit als Grundlage für das zu realisierende Softwaresystem. Bei der Systementwicklung selbst kann dann eine Folge von Soft- und Hardwaremodellen erarbeitet werden, aus denen sich schließlich das einsatzfähige Robotersystem ergibt.

---

<sup>26</sup> Siehe auch Baumgarten 1996.

Die Strukturierung eines klassischen Robotersystems, bestehend aus Hard- und Software, betrifft seine Architektur, legt also fest, wie das System aufgebaut ist. Ein solches System sollte nicht monolithisch sein, also nicht aus einem „großen Block“ bestehen, sondern es sollte sich aus mehreren wohlunterschiedenen Teilsystemen zusammensetzen, die jeweils eine Teilaufgabe erfüllen und zur Lösung der Gesamtaufgabe kooperieren. Hieraus ergibt sich eine übersichtliche Systemstruktur, mit der man auch bei großen Hard- und Softwaresystemen die logische Komplexität gut in den Griff bekommen kann. Eine Strukturierung ist in unterschiedlichen Detailierungsgraden möglich. Eine Grobstrukturierung zerlegt das System in nur wenige große Teile, eine Feinstrukturierung in viele kleine Teile. Eine grobe Strukturierung wird zum Beispiel als einer der ersten Schritte während der Problemanalyse vorgenommen. Man kann hierfür auf standardisierte Strukturierungsmuster zurückgreifen, wie sie auch im Bereich der Robotik bereits vorliegen und unter den Stichworten *Entwurfsmuster* und *komponentenbasierte Entwicklung* skizziert werden.<sup>27</sup> Eine feine Strukturierung ergibt sich beispielsweise spätestens bei der objektorientierten Analyse und Design. Ein Beispiel, wie man komplexe Systeme grob strukturieren kann, stellt die *Drei-Schichten-Architektur* (engl. three tier architecture) dar. Diese Architektur berücksichtigt, dass eine Software nicht nur die eigentlichen Verfahren zur Lösung eines Anwendungsproblems implementieren muss, sondern dass auch die zugehörigen Daten strukturiert zu speichern sind, und dass den Anwendern eine komfortable Möglichkeit zum Zugriff auf die Anwendung als auch den Daten gegeben werden muss. Entsprechend gliedert man die Software in drei Teile, die sich jeweils mit einer dieser Teilaufgaben befassen:

- die Benutzeroberfläche (Präsentation),
- das Anwendungsprogramm (Geschäftslogik) und
- die Datenhaltung (Persistenz).

Da die Benutzeroberfläche auf die Dienste der Geschäftslogik und die Geschäftslogik wiederum auf die Dienste der Datenhaltung zugreift, spricht man auch von drei Schichten, die aufeinander aufbauen. Die Funktionalitäten der drei Schichten können auch auf verschiedene Prozessoren in einem Rechnernetz oder einem Robotersystem verteilt werden, die dann nach dem *Client-Server-Prinzip* zusammenarbeiten. Das Prinzip der Strukturierung ist eng verbunden mit den Prinzipien der *Modularisierung* und der *Hierarchiebildung*. Sie bestimmen, wie die einzelnen Komponenten eines strukturierten Systems auszusehen haben bzw. wie man bei der Strukturbildung systematisch vorgehen kann.

Das Prinzip der *Modularisierung* basiert auf dem Modulbegriff: Ein Modul ist eine abgeschlossene Einheit mit einer eindeutigen Schnittstelle und einem Modulinneren mit Verfahrensvorschriften und/oder Datenstrukturen. Die Schnittstelle definiert die Operationen („Funktionen“), die auf dem Modul ausgeführt werden können, mit ihren Namen, Parametern und Rückgabetypen. Das Modul kann nur über diese Schnittstelle benutzt

---

<sup>27</sup> Siehe auch Gamma et al. 1996.

werden, denn allein sie ist nach außen „sichtbar“, nicht jedoch die eigentliche Implementierung der Funktionen und Datenstrukturen. Module sind damit gekapselt und realisieren das Geheimnisprinzip (Information Hiding). Das Geheimnisprinzip bedeutet in vielen Fällen nicht unbedingt, dass der innere Aufbau des Moduls nicht bekannt sein darf, sondern vielmehr, dass er nicht bekannt sein muss, um das Modul benutzen zu können. Insofern werden die Implementierungsdetails hinter einer wohldefinierten Schnittstelle verborgen. Ein modulares System besteht aus einer Anzahl von Modulen, die über ihre Schnittstellen Nachrichten und Aufträge austauschen und somit kooperieren. Anschaulich gesprochen ist das System also nach Art eines Bausteinkastenprinzips zusammengesetzt. Ein Modul kann hier bei Bedarf gegen ein anderes Modul mit gleicher Schnittstelle und Funktionalität ausgetauscht werden. An Module und modulare Systeme werden die folgenden grundlegenden Forderungen gestellt:

- *Transparenz*: Ein Modul sollte überschaubar sein, also nur eine relativ kleine Anzahl von Funktionen anbieten.
- *Kohäsion*: Ein Modul sollte einen möglichst starken inneren Zusammenhalt (eine so genannte starke *Kohäsion*) besitzen, das heißt, alle seine Funktionen und Datenstrukturen sollten in einem engen logischen Zusammenhang zueinander stehen.
- *Schwache Kopplung*: Der Zusammenhang zwischen den Modulen (die *Modulkopplung*) sollte dagegen relativ schwach sein, das heißt, sich auf möglichst wenige Funktionsaufrufe beschränken.
- *Abgeschlossenheit*: Anders formuliert, sollte ein Modul möglichst in sich abgeschlossen sein und weitgehend unabhängig von seiner Umgebung arbeiten.

Eine Systemstruktur kann realisiert werden, indem man eine Hierarchie von Schichten konstruiert, also eine Folge von Abstraktionsebenen, die aufeinander aufbauen. Man geht dabei bevorzugt von der Grob- zur Feinstruktur (oder kurz: Top-Down) über, indem man die Hauptaufgabe schrittweise in immer kleinere Teilaufgaben zerlegt. Charakteristisch für eine so entstehende Funktionshierarchie ist, dass die Funktionen Abstraktionsschichten zugeordnet sind. Jede Schicht bildet dabei eine Basis für die nächsthöhere Schicht, indem sie Schnittstellen mit Diensten bereitstellt, die von ihrem oberen Nachbarn genutzt werden können. Umgekehrt gilt, dass jede Schicht nur die Schnittstellendienste der unmittelbar darunter liegenden Schicht benutzen darf; die interne Realisierung dieser Schicht und alle Schichten weiter unterhalb sind für sie nicht sichtbar. Die Schichten sind also, wie Module, durch ihre Schnittstellen gekapselt. Alternativ zur Top-Down-Methodik ist auch eine Bottom-Up-Arbeitsweise möglich: hier baut man, ausgehend von primitiven Funktionen, immer komplexere, mächtigere Funktionen auf. Bei der praktischen Systementwicklung bedeutet das oft eine Abstraktion von den Eigenschaften der realen Hardwaremaschine und deren Realisierung in Form eines Robotersystems mit erweitertem Funktionsumfang. Schrittweise wird so die Kluft zwischen der primitiven realen Basismaschine und dem komfortablen System überbrückt, wobei die Funktionen der Basismaschine „veredelt“ werden. Der Top-Down-Ansatz ermöglicht allerdings meist eine systematischere

Vorgehensweise als der Bottom-Up-Ansatz: Der Ausgangspunkt von Top-Down ist ein übersichtliches System aus wenigen Einheiten, die schrittweise nach Bedarf verfeinert werden. Bei Bottom-Up muss man dagegen zunächst eine Vielzahl primitiver Teilmodule erstellen und sie dann sinnvoll zusammensetzen. Dabei ist es gerade bei komplexen Problemstellungen schwierig, den Überblick zu behalten. Das Prinzip der Hierarchiebildung wird nicht nur bei der Programmierung eines Systems von Funktionen angewandt, sondern beispielsweise auch

- bei der Definition eines Systems von Klassen in einem objektorientierten Programm-Paket unter Bildung einer Klassenhierarchie,
- bei der Realisierung von Kommunikationsprotokollen zur Steuerung der Datenkommunikation in einem Rechnernetz und insbesondere
- beim Systementwurf wo beispielsweise bei der Problemanalyse die komplexen Prozesse schrittweise in immer kleinere Prozesse zerlegt werden.

In vielen Anwendungsbereichen hat die Hierarchiebildung nicht nur einen strukturellen, sondern auch einen zeitlichen Aspekt: Der zeitliche Aspekt betrifft die Vorgehensweise bei der Konstruktion eines Systems, bei der die Schichten eine nach der anderen definiert werden. Der strukturelle Aspekt betrifft die resultierende Systemarchitektur, die sich aus den gebildeten Schichten zusammensetzt und somit eine übersichtliche hierarchische Struktur besitzt.

Der Begriff der *Wiederverwendung* bezeichnet das Prinzip, einmal entwickelte Systemkomponenten möglichst oft und in möglichst vielen unterschiedlichen Zusammenhängen einzusetzen. Der Vorteil dabei ist, dass weniger Entwicklungsarbeit zu leisten ist und dass das resultierende System einen reduzierten Umfang hat und damit übersichtlicher wird. Die Wiederverwendbarkeit ist bereits bei der *Modularisierung* ein wichtiges Motiv: Ein Modul (beispielsweise ein Unterprogramm oder ein Objekt) stellt Dienste bereit, den ein oder mehrere Dienstnehmer wiederholt nutzen können, ohne ihn jeweils selbst realisieren zu müssen.

Weitere Anwendungsbereiche, bei denen die Wiederverwendung eine Rolle spielt, sind die folgenden:

- *Funktions- und Klassenbibliotheken* stellen Programmstücke bereit, die Programmierer in ihre Programme einbinden können. In Klassenhierarchien greifen abgeleitete Klassen auf Attribute und Methoden zurück, die bereits in Basisklassen definiert wurden.
- Bei der *komponentenbasierten Programmierung* verteilt man die Funktionalität eines Programms auf eine Reihe von Softwarebausteinen. Man kann dabei auch auf externe Bausteine zurückgreifen, braucht also nicht alles von Grund auf neu zu erstellen. Diese Technik wird beispielsweise bei JavaBeans, bei den Enterprise JavaBeans (EJB), beim klassischen Component Object Model (COM) der Firma Microsoft oder auch bei der Implementierung verteilter Client-Server-Systeme mit Hilfe einer Middleware benutzt.

- Die komponentenbasierte Programmierung wird heute oft durch Techniken der *visuellen Programmierung* unterstützt, bei der die Komponenten mit Hilfe eines grafischen Werkzeugs kombiniert werden.

Ein *Entwurfsmuster* (engl. design pattern) beschreibt eine bestimmte Vorgehensweise bei der Definition von Klassen und bei der Gestaltung ihrer Zusammenarbeit. Entwickler können sich bei der Ausarbeitung ihrer Programme an solchen Mustern orientieren. Ein typisches Beispiel hierfür ist das *Model-View-Controller-Muster* (MVC) zur Strukturierung interaktiver Systeme. Das Muster legt eine Dreiteilung der Systemstruktur fest, nämlich in ein Modell, das die interne Darstellung der Daten durch Objekte definiert, einen View, der die Darstellung der Daten an der Benutzeroberfläche beschreibt, und einen Controller, der Benutzereingaben zur Bearbeitung der Daten behandelt. Mit Entwurfsmustern hängen *Frameworks* zusammen. Ein Framework ist ein Programmrahmen in einer höheren Programmiersprache, der die allgemeine Form des Zusammenspiels von Programmteilen (zum Beispiel Objekten) vorgibt. Die Programmteile, die für die jeweilige Anwendung spezifisch sind, sind dabei nicht definiert; sie müssen durch Programmierer ausprogrammiert und in den vorgegebenen Rahmen „eingehängt“ werden.<sup>28</sup>

---

## 5.3 Brainware

Das zentrale Anliegen der folgenden Abschnitte liegt in der Steigerung des Intelligenzquotienten eines Robotersystems, dem sogenannten systemischen Intelligenz-Quotienten ( $\text{IQ}_S$ ).

### 5.3.1 Artifizielles Leben und artifizielle Intelligenz

Aus der Beschreibung der verschiedenen Sensoren und Akten, die in der Robotik zur Verfügung stehen, wurde deutlich, dass die Sensoren ausschließlich bestimmte physikalische Eigenschaften wahrnehmen können, die in irgendeinem, zunächst vagen Zusammenhang mit der Information stehen, die das Robotersystem tatsächlich benötigt. So gibt es keine „Hindernissensoren“, nur Mikroschalter, Sonarsensoren, Infrarotsensoren usw., deren Signal die An- oder Abwesenheit eines Objekts angibt. Es ist aber durchaus möglich, dass der Mikroschalter ein Signal auslöst, nur weil der Roboter über eine Bodenwelle fährt, oder ein Sonarsensor meldet ein Hindernis aufgrund von Übersprechen, oder ein Infrarotsensor entdeckt möglicherweise ein Objekt nicht, weil es schwarz ist und deshalb kein Infrarotsignal reflektiert. Unter Berücksichtigung dieser Vorbehalte und Einschränkungen lassen sich die Technologien der Artifiziellen Intelligenz und des Artifiziellen Lebens einsetzen, um aus rohen Sensordaten sinnvolle Informationen zu gewinnen. Eine

---

<sup>28</sup> Siehe auch Booch 1991.

vielversprechende Strategie ist dabei, den Roboter aus Erfahrung *lernen* zu lassen, durch Probehandeln, durch Erfolge und Misserfolge.

Die Technologien des Artifiziellen Lebens und der Künstlichen Intelligenz sind heute dabei, aus den Forschungslabors herauszutreten, um in vielen Bereichen der Technik Fuß zu fassen und eine zunehmende Verbreitung zu finden.<sup>29</sup> Besonders gewinnbringend ist hierbei ein transdisziplinärer Technologieansatz, der auf erworbenes Wissen im Gebiet der Theorie und Technik der künstlichen Intelligenz und des artifiziellen Lebens zurückgreift. Dieser Ansatz vereint die einzelnen Erkenntnisse und bildet damit eine harmonische Basis für die Entwicklung von rechnerbasierter Intelligenz, die als Grundlage für hybride intelligente Robotersysteme mit hohen systemischen IQ ( $IQ_s$ ) dient. Ein Ansatz, artifizielle Systeme zu entwickeln, besteht darin, diese nicht nur den Schritten der Evolution folgend zu entwickeln, sondern auch dieses Entwickeln selbst einer wie auch immer gearteten Evolution zu überlassen. Die Disziplin, die mit diesem Ansatz experimentiert, heißt *Artificial Life*, Artifizielles Leben (AL). Es handelt sich um eine Disziplin, angesiedelt zwischen Biologie, Physik, Chemie, Informatik und Kognitionswissenschaft<sup>30</sup>, deren Wurzeln bis zur Mitte des 20. Jahrhunderts zurückgehen, die aber erst seit den 90er Jahren verstärkt wieder in Erscheinung tritt.<sup>31</sup>

Die bekannten Formen des „natürlichen“ Lebens haben eine gemeinsame biologische Basis, die Kohlenstoffchemie.<sup>32</sup> Der genetische Code ist universell. Alle bekannten natürlichen Wesen unterliegen dabei der Evolution. Dies bezeichnen die AL-Forscher als „Leben, wie es ist“. Was sie interessiert, ist allerdings ein „Leben, wie es sein könnte“, unabhängig von der Kohlenstoffchemie, betrachtet als ein informationeller Prozess, der in einem Computer simulierbar ist. Gemäss dieser Auffassung sucht man nach den charakteristischen Strukturen und Prozessen des Lebens. Das Wesen des Lebens hängt ihrer Ansicht nach nicht von seiner materiellen Basis ab, sondern von seinen strukturellen Eigenschaften, wie Selbstorganisation, Anpassung, Evolution oder Wechselwirkung. Leben ist demnach ein abstraktes Phänomen, das in biologischen Systemen realisiert ist, vielleicht aber auch anders realisiert sein könnte. Anders als Biologen, die Tiere auseinandernehmen, um sie zu erforschen, will man sie zusammensetzen, beziehungsweise von evolutionären Programmen zusammensetzen lassen. Was dabei generiert werden soll, sind Systeme, die lebensähnliches Verhalten zeigen.

Speziell für den Bereich der Robotik hat man aus diesem Forschungszweig die so genannten evolutionären Algorithmen übernommen. Hierbei definiert man zunächst ein Ziel, das im evolutionären Prozess erreicht werden soll. Dabei kann es sich sowohl um eine bestimmte Gestalt eines artifiziellen Systems handeln als auch um eine bestimmte Fähigkeit, die dieses System haben soll (zum Beispiel herumzufahren, ohne vom Tisch zu

---

<sup>29</sup> Vgl. Scheife 1991.

<sup>30</sup> Vgl. Münch 1992.

<sup>31</sup> Vgl. Haun 2004.

<sup>32</sup> Vgl. Schrödinger 1951.

fallen), egal wie und in welcher Gestalt es dies zuwege bringt. Dann benötigt man Ausgangsmaterial, an dem der evolutionäre Prozess ansetzen kann, ein Robotersystem, das sich verändern soll, oder auch nur Bauteile desselben, aus denen eine Konstruktion am Ende des Prozesses entstehen soll. Das Ganze kann sich in Form einer Computersimulation oder anhand realer Systeme abspielen. Auf das Ausgangsmaterial wirken nun zufällig generierte artifizielle Mutationen ein, das heißt, die Ausgangsmaterialien werden zufällig verändert. Ist dieser Prozess vollzogen, stellt man eine Rangliste auf. Zuoberst stehen diejenigen Systeme, die dem vorgegebenen Ziel am nächsten kommen, zuunterst jene, die am weitesten von ihm entfernt sind. Dann schließt man einen gewissen Prozentsatz der schlechtesten Ergebnisse von der weiteren Evolution aus. Das ist das *survival of the fittest* der Evolutionsbiologen. Dieser Prozess wird so lange wiederholt, bis das gewünschte Ergebnis erreicht ist. Man kann diesen Prozess auch subtiler gestalten, indem man etwa, den Erkenntnissen der Genetik folgend, Systeme „verheiratet“ und sie die Hälfte ihrer Eigenschaften tauschen lässt.<sup>33</sup> Bisweilen führt die künstliche Evolution zu überraschenden und für die praktische Verwendung eher untauglichen Konstruktionsprinzipien, so etwa zur Fortbewegung durch Purzelbaumschlagen. Erfolgreich wird sie dagegen bei der Optimierung der Steuerelemente von Robotern verwendet.

Inzwischen hat sich auch eine eigene Disziplin etabliert, die sich innerhalb der Robotik mit dem evolutionären Ansatz befasst, die Evolutionäre Robotik (evolutionary robotics).

Der Begriff der „Artifiziellen Intelligenz“ wird in diesem Buch in gleich verschiedenen Bedeutungen verwendet: Zum einen bezeichnet er eine wissenschaftliche Disziplin und zum anderen eine Eigenschaft von Robotersystemen. In diesem Abschnitt steht die erste Bedeutung im Vordergrund, indem man unter Artifizieller Intelligenz ein Forschungs- und Arbeitsgebiet versteht, dessen Gegenstand es ist, Leistungen mit Hilfe hardware- und/or softwaretechnischer Systeme zu realisieren, die folgende Bedingung erfüllt: ihre Hervorbringung erfordert nach allgemeinem Verständnis eine bestimmte (menschliche) Intelligenz.

Die deutsche, eher schlechte Übersetzung lautet: „Künstliche Intelligenz“, zumal man mit „künstlich“ viel zu häufig „unvollkommen“ assoziiert.<sup>34</sup> Zum einen wird sich die Unvollkommenheit der artifiziellen Systeme immer mehr verringern, wenngleich die Frage dennoch dann offen bleibt: Woran erkennt man die Künstlichkeit von Systemen, wenn diese dem Original in funktionaler und visueller Hinsicht zum Verwechseln ähnlich sind?<sup>35</sup>

Das Themenangebot der Artifiziellen Intelligenz (AI) als solches und speziell deren Inhalte scheinen immer umfangreicher zu werden. Dieses Buch versucht in diesem Abschnitt

<sup>33</sup> Siehe auch Dobzhansky 1937.

<sup>34</sup> Vgl. Irrgang und Klawitter 1990.

<sup>35</sup> Vgl. Lunze 1995.

diesen Trend umzukehren und eine prägnante, zugängliche und praxisnahe Behandlung dieses Themas zu bieten, die besonders für die Anforderungen der Robotik geeignet ist. Um aber prägnant zu sein und verständliche Ansichten der Themen zu bieten, ist es unerlässlich, dass bestimmte Themen ausgelassen oder nur gestreift werden. In den folgenden Abschnitten dieses Kapitels werden auch Methoden und Techniken vorgestellt, mit denen sich *Wissen* darstellen und verarbeiten lässt, wobei sich der Aspekt der Verarbeitung primär auf ein solches Schlussfolgern bezieht, um Probleme lösen zu können.

Aufgrund der immer kürzer werdenden Innovations- und Entwicklungszyklen der einzelnen Wissenschaftsdisciplinen haben sich bereits mehrere relativ selbständige, dennoch eng miteinander verbundene Teilgebiete herauskristallisiert. Ihre interdisziplinäre Verflechtung kann in den bereits bekannten Ebenen veranschaulicht werden:

- Ebene der *Konzeptionalisierung*: Theoretische Grundlagen der Wissenschaftsdisziplinen.
- Ebene der *Formalisierung*: Methoden und Techniken
- Ebene der *Implementierung*: Basiswerkzeuge und Programmiersprachen<sup>36</sup>
- Ebene der *Konkretisierung*: Simulationen und konkrete Anwendungen

Im Rahmen der *Konzeptionalisierung* bedient man sich der Errungenschaften und Erkenntnissen unterschiedlicher Wissenschaftsdisziplinen. Dazu gehören unter anderem die Philosophie, die Mathematik, die Psychologie, die Linguistik als auch die Robotik selbst. So findet man wesentliche Grundlagen zum Themengebiet des Denkens in der *Philosophie* (seit 428 v. Chr.). In der griechischen Antike begann die Entwicklung des logischen Denkens mit Sokrates, Plato und vor allem Aristoteles (428 ist das Geburtsjahr Platos). Sokrates formulierte als erster das Problem algorithmischer Vorgehensweisen. Aristoteles entwickelte die Syllogismen, die eine formale Deduktion erlaubten, allerdings nahm er an, dass es auch so etwas wie intuitives Schließen gebe. René Descartes (1596–1650) vertrat die Auffassung, dass es einen Teil des Denkens (mind) gebe, der streng nach physikalischen Prinzipien funktioniert. Die Konsequenz ist, dass es keinen freien Willen mehr gibt. Descartes vertrat deshalb die Ansicht, dass es noch einen anderen Teil des Denkens, genannt *Seele* oder *Geist*, existiere, der nicht den Gesetzen der Physik unterworfen ist. Diese Auffassung nennt man deshalb *Dualismus*. Wilhelm Leibniz (1646–1716) vertrat als erster die konsequent materialistische Auffassung, dass alles in der Welt, einschließlich Gehirn und Denken, nach physikalischen Gesetzen funktioniert. Er baute auch als erster ein mechanisches Gerät, das geistige Operationen ausführen sollte. Leider produzierte diese Maschine wenig interessante Ergebnisse, da die zugrundeliegende Logik schwach war. In Bezug auf das Themengebiet Wissen liefert die *empiristische* Bewegung (Francis Bacon, 1561–1626, John Locke, 1632–1704) die Auffassung, dass alles Verstehen durch sinnliche Wahrnehmung entsteht. David Hume (1711–1776) führte das *Induktionprinzip* ein. Danach werden

---

<sup>36</sup> Siehe auch Schönthal und Nemeth 1990.

die allgemeinen Regeln des Denkens dadurch erworben, dass immer wieder Assoziationen zwischen ihren Elementen gebildet werden. Diese Auffassung wurde von Bertrand Russell (1872–1970) durch den *logischen Positivismus* vertieft. Alles Wissen kann danach durch logische Theorien charakterisiert werden, die mit *Beobachtungsaussagen* verknüpft sind. Diese wiederum entsprechen den Sinneswahrnehmungen. Rudolf Carnap und Carl Hempel versuchten mit ihrer *Bestätigungstheorie* die Verbindung zwischen den Beobachtungsaussagen und den logischen Theorien näher zu beschreiben. Der für die Robotik so wichtige Zusammenhang zwischen Wissen und Handeln ist ebenfalls in der Philosophie behandelt. So stellte bereits Aristoteles in der *Nikomachischen Ethik* fest, dass Handlungen nach den notwendigen (und verfügbaren) Mitteln zum Erreichen eines Ziels und nicht nach dem Ziel selbst entschieden werden. Aristoteles' Ansatz wurde von Newell und Simon in ihrem GPS implementiert. Die wichtigste Methode in GPS ist die Mittel-Ziel-Analyse. Sie ist allerdings unzureichend, wenn mehrere Aktionen in einer Situation zur Auswahl stehen oder wenn gar keine Aktion möglich ist. Arnauld, ein Schüler Descartes', und John Stuart Mill haben dazu eine verallgemeinerte Entscheidungstheorie entwickelt (Mill: „Utilitarismus“). Die *Mathematik* als naturwissenschaftliche Disziplin liefert die Logik, das Unvollständigkeitstheorem als auch Gesetze zur Berechenbarkeit. So hat als erster George Boole (1815–1864) eine formale Sprache zur Darstellung der Logik eingeführt und war damit Begründer der mathematischen Logik. Dieser noch beschränkte Ansatz wurde von Gottlob Frege (1848–1925) zu einer vollen Logik erster Ordnung ausgebaut. Alfred Tarski (1902–1983) lieferte dazu mit der Theorie der Referenz eine Semantik. David Hilbert (1862–1943) stellte das *Entscheidungsproblem* auf. Es stellt die Frage, ob es einen Algorithmus gibt, der die Wahrheit irgendeiner logischen Aussage, die die natürlichen Zahlen involviert, feststellen kann. Kurt Gödel (1906–1978) zeigte 1930, dass es eine berechenbare Prozedur gibt, die jede wahre Aussage in der Logik erster Ordnung von Frege und Russell beweisen kann, dass aber diese Logik nicht das Prinzip der mathematischen Induktion beschreiben kann, das notwendig ist, um die natürlichen Zahlen zu charakterisieren. Später zeigte er in seinem Unvollständigkeitstheorem, dass jede formale Sprache, die die Eigenschaften der natürlichen Zahlen beschreiben kann, wahre Aussagen enthält, die nicht entscheidbar sind, d. h. ihre Gültigkeit kann durch keinen Algorithmus bewiesen werden. Alan Turing (1912–1954) charakterisierte dann mit seinem Maschinenmodell 1936 die berechenbaren Funktionen, und zeigte zugleich, dass es Funktionen gibt, die mit diesem Modell nicht berechenbar sind. Die Komplexitätstheorie beschäftigt sich seit Mitte 1960 mit Komplexitätsklassen, nämlich Funktionen mit polynomieller Laufzeit und Funktionen mit exponentieller Laufzeit. Probleme mit polynomieller Laufzeit wurden als *behandelbar*, die mit exponentieller Laufzeit als *nicht behandelbar* betrachtet. Zur gleichen Zeit wurde festgestellt, wie sich Probleme auf andere reduzieren lassen, so dass die Lösung des einen Problems auch eine Lösung des darauf reduzierten Problems liefert. Das war die Grundlage zur Definition von Komplexitätsklassen. Steven Cook (1971) und Richard Karp (1972) entwickelten die Theorie der NP-Vollständigkeit und zeigten die Reduzierbarkeit von NP-vollständigen Problemen aufeinander. Das erste Konzept von Wahrscheinlichkeit und

damit der Beginn der Entwicklung der Wahrscheinlichkeits- und Entscheidungstheorie wurde von Gerolamo Cardano (1501–1576) entwickelt, damals noch, um die möglichen Ergebnisse von Spielen zu beschreiben. Danach wurde die Wahrscheinlichkeitstheorie unverzichtbarer Bestandteil der quantitativen Wissenschaften, denn sie konnte zur Beschreibung unsicherer Messungen und unvollständiger Theorien verwendet werden. Sie wurde vor allem von Pierre Fermat (1601–1665), Blaise Pascal (1623–1662), James Bernoulli (1654–1705), Pierre Laplace (1749–1823) und Thomas Bayes (1702–1761) weiterentwickelt. Insbesondere Bayes Theorem wurde für das Schließen mit unsicherem Wissen in der KI bedeutsam. John von Neumann und Oskar Morgenstern (1944) kombinierten die Wahrscheinlichkeitstheorie mit der Nützlichkeitstheorie und schufen somit die Entscheidungstheorie. Damit war es möglich, gute (nützliche) Aktionen von schlechten (nutzlosen) zu unterscheiden. Die wissenschaftliche *Psychologie* begann mit Hermann von Helmholtz (1821–1894) und seinem Schüler Wilhelm Wundt (1832–1920). Insbesondere Wundt führte die *experimentelle Psychologie* ein und eröffnete das erste Labor für experimentelle Psychologie. Es wurden sorgfältig kontrollierte Experimente durchgeführt, bei denen die Probanden perzeptuelle und assoziative Aufgaben durchführen und dabei ihre eigenen gedanklichen Prozesse beobachten mussten. John Watson (1878–1958) und Edward Lee Thorndike (1874–1949) setzten gegen den Subjektivismus der experimentellen Psychologie den *Behaviorismus*. Danach wird jede Theorie über mentale Prozesse abgelehnt mit der Begründung, dass Introspektion keine zuverlässigen Daten liefern könne. Als einzigen Untersuchungsgegenstand erlaubten die Behavioristen die Stimulus-Response-Beziehung, die Wahrnehmungen mit Aktionen in Beziehung setzt. Mentale Konstrukte, wie Wissen, Annahmen, Ziele oder Schlussfolgerungen, wurden als unwissenschaftlich abgetan. Mit William James (1842–1910) begann die kognitive Psychologie und erhielt den stärksten Auftrieb durch das Buch *The Nature of Explanation* von Kenneth Craik (1943). Er behauptete, dass Annahmen, Ziele und Schlussfolgerungen nützliche Komponenten einer Theorie des menschlichen Verhaltens sein könnten. Craik spezifizierte drei Hauptschritte, die ein wissensbasierter Agent durchführt: 1) Übersetzung des Stimulus in eine interne Repräsentation, 2) Manipulation der Repräsentation durch einen kognitiven Prozess und dadurch Erzeugung einer neuen Repräsentation, 3) Rückübersetzung der erzeugten Repräsentation in Aktion. Die *Computertechnik* ist sicherlich mit dem Namen Alan Turing verbunden, der mit seinem Team 1940 den ersten Computer überhaupt baute, den *Heath Robinson*, dessen Zweck die Dechiffrierung der deutschen Nachrichten war. Sein Nachfolger war der *Colossus*, gebaut 1943. Davon waren bei Kriegsende 10 Stück in Betrieb. Der erste programmierbare Computer war der Z-3, den Konrad Zuse 1941 baute. Zuse erfand die Gleitkomma-zahlen für den Z-3 und entwickelte bis 1945 die Programmiersprache *Plankalkül*. Der Z-3 arbeitete mit Röhren. Zuses Entwicklung war wissenschaftlich begründet, das Militär kümmerte sich wenig darum. In den USA wurde der erste Computer, der ABC, von John Atanasoff und Clifford Berry 1940–1942 an der Iowa State University gebaut. Die Entwicklungen des *Mark I, II* und *III* an der Harvard University durch das Team um Howard Aiken, und des *ENIAC* an der University of Pennsylvania durch das Team um John Mauchly und John Eckert waren durch das Militär initiiert. Der ENIAC war der erste elektronische

digitale Allzweck-Computer. Der Nachfolger *EDVAC* war der erste, der gespeicherte Programme verarbeiten konnte. Der richtige Durchbruch gelang mit dem IBM 701, der 1952 von Nathaniel Rochester gebaut wurde. Er war der erste kommerziell genutzte Computer. Seit dieser Zeit hat die Computertechnik einen großen Aufschwung genommen mit sehr hohen Steigerungsraten in der Leistungsfähigkeit der Maschinen. Es gab auch Vorläufer der modernen Computer. Der Abakus ist beispielsweise ca. 7000 Jahre alt. Pascal baute im 17. Jahrhundert die erste mechanische Addier- und Subtrahiermaschine. Sie wurde von Leibniz 1694 weiterentwickelt. Charles Babbage (1792–1871) konzipierte eine Maschine zur Berechnung von Logarithmentafeln, aber er baute sie nicht. Stattdessen entwarf er die Analytical Engine, die einen adressierbaren Speicher besaß, Programme speichern und bedingte Sprünge ausführen konnte. Ada Lovelace schrieb Programme für die Analytical Engine. Sie spekulierte damals darüber, dass die Maschine Schach spielen und Musik komponieren könnte. Noam Chomsky gilt mit seinem Buch *Syntactic Structures* (1957) als Begründer der modernen *Linguistik*. Er setzte sich darin mit der behavioristischen Sprachauffassung von Skinner auseinander. Er weist nach, dass die behavioristische Theorie keine Erklärung des Phänomens der Kreativität in der Sprache liefert. Kreativität in der Sprache bedeutet, dass ein Mensch Sätze verstehen und selber bilden kann, die er vorher nie gehört oder gelesen hat. Chomskys Theorie lieferte eine Erklärung und sie war formal, d. h. programmierbar. In der Folgezeit entwickelten sich KI und Linguistik zwar unabhängig voneinander, es gab aber viele Querbeziehungen, insbesondere in Form der Verarbeitung natürlicher Sprache oder Computerlinguistik. Im Rahmen der Konzeptionalisierung muss auch das Gebiet der Robotik erwähnt werden, das inzwischen von Handhabungsmechanismen in der industriellen Fertigung bis hin zu integrierten Hand-Auge-Systemen reicht. Es berührt technische Fragen, die über den Rahmen der KI hinausreichen. Eigentlich ist es das letztendliche Ziel der KI, autonome, integrierte Systeme zu schaffen, die alle Merkmale der artifiziellen Intelligenz in sich vereinigen. Solche Roboter müssten sowohl über Sensoren als auch über Effektoren verfügen und in der Lage sein, sich aktiv mit ihrer Umwelt auseinanderzusetzen. Aufgrund eines inneren dreidimensionalen Modells ihrer Umgebung sollten sie sich richtig orientieren und planvoll verhalten können. Es ist bisher noch nicht gelungen, ein alle Aspekte der KI umfassendes technisches System zu schaffen, obwohl bereits wichtige Ergebnisse auf diesem Wege erzielt wurden. Auch die in dem vorliegenden Buch dargestellten Methoden zur Steigerung des systemischen Intelligenzquotienten sind letztlich als Softwarekomponenten für integrierte Roboter relevant und finden ihren Ausgangspunkt in der Ebene der Konzeptionalisierung.

Der Kern fast aller Arbeitsgebiete in der Ebene der *Formalisierung* ist die *automatische Wissensverarbeitung*. Sie stellt sowohl die Mittel zur Wissensrepräsentation auf dem Rechner als auch die Methoden zur Handhabung und Manipulation von Wissen bereit. Wesentliche Impulse erhält dieses Gebiet der KI von der kognitiven Psychologie. Deshalb hat sich auch im Grenzbereich zwischen KI und Psychologie ein besonderes Arbeitsgebiet, die *kognitive Modellierung*, herausgebildet. So werden beispielsweise semantische Netze sowohl in der Psychologie als auch in der KI als eines der grundlegenden Modelle für die Darstellung von Wissens- bzw. Gedächtnisstrukturen verwendet. Eng verbunden mit

der Wissensverarbeitung ist das Gebiet der Expertensysteme, das ebenfalls eine eigene Methodologie herausgebildet hat. Menschliches Wissen wird im Verlaufe der geschichtlichen Entwicklung über Generationen hinweg und auch im täglichen Leben vorwiegend in natürlicher Sprache kommuniziert und gespeichert.<sup>37</sup> Darüber hinaus besteht eine enge Beziehung zwischen Sprache und Denken. Aus diesem Grunde spielt die *automatische Verarbeitung der natürlichen Sprache* eine entscheidende Rolle in der KI. Für den Aufbau einer Wissensbasis sind Lernen und Wissenserwerb erforderlich. *Lernvorgänge* reichen von der einfachen Einspeicherung von Fakten (Auswendiglernen) bis hin zum induktiven Lernen und zur Verallgemeinerung bzw. automatischen Abstraktion von Begriffen. Zu den Methoden der Extraktion von Wissen aus einer vorhandenen Wissensbasis und die zielgerichtete Verwendung von Wissen beim Lösen von Problem- bzw. Aufgabenstellungen zählen die Verfahren zur automatischen Problemlösung ebenso wie die der automatischen Ausführung von Schlussfolgerungen. Letztere werden in Anlehnung an den englischen Sprachgebrauch auch als *Inferenzverfahren* bezeichnet (abgeleitet von „to infer“: schließen, schlussfolgern). Den Vorgang des rationalen Denkens und seine Automatisierung bezeichnet man ebenfalls nach dem englischen Sprachgebrauch als *Reasoning*.

In der *Implementierungsebene* hat die automatische Ausführung von streng logischen (deduktiven) Schlussfolgerungen dazu geführt, dass die entsprechenden Methoden direkten Eingang in eine der beiden Basisprogrammiersprachen der KI, in PROLOG, gefunden haben. Beziüglich der Weiterentwicklung der speziellen KI-Programmiersprachen lassen sich derzeit drei Trends ausmachen:

- verstärkte objektorientierte Programmierung,
- Parallelisierung von Sprachkonzepten,
- Fusionierung verschiedener Sprachkonzepte in Form von Kombinationen (wie beispielsweise logischer und funktioneller Programmierung oder objektorientierter und imperativer Programmierung).

Neben den stärker theoretisch orientierten Gebieten der KI gibt es eine Reihe von Bereichen der KI, die ihren Schwerpunkt in der *Konkretisierungsebene* finden und damit konkrete Anwendungen in den Vordergrund stellen. Eine praktische Bedeutung haben inzwischen die Expertensysteme, die Systeme zur automatischen Verarbeitung natürlicher Sprache und die Robotertechnik erlangt.

Auch in Bezug auf die Künstliche Intelligenz lohnt es sich, am Ende dieses Abschnitts einen Blick auf deren Geschichte zu werfen<sup>38</sup>, geht doch aus dieser in logischer und praktischer Konsequenz das Cognitive Computing hervor.<sup>39</sup> Dabei liegt die Geburtsstunde der KI im Jahre 1956, obwohl der Wunsch und das Bestreben, artifizielle menschliche Wesen zu schaffen, schon in den Welten der Mythen und Legenden zum Ausdruck kommt. Im 4.

---

<sup>37</sup> Siehe auch Mitchie und Johnston 1985.

<sup>38</sup> Siehe auch McCorduck 1989.

<sup>39</sup> Siehe auch Kräme 1994.

Jahrhundert wurden von Aristoteles durch seine aufgestellten Syllogismen die logischen Grundlagen dafür geschaffen, was später Boole und Frege zu modernen Logikkalkülen ausbauen sollten. Im 17. und 18. Jahrhundert wurden bereits die ersten mechanischen Rechenmaschinen von den Mathematikern Leibnitz und Pascal gebaut. Aber auch im 18. und 19. Jahrhundert beschäftigten sich vor allem Descartes und LaMettrie mit dem Verhältnis zwischen Mensch und Maschine bzw. Geist und Materie. Descartes vertrat einen dualistischen Ansatz, indem er von einer Trennung zwischen Geist und Materie ausging. Hingegen vertrat der Materialist LaMettrie die Auffassung, daß der Mensch im Grunde genommen nichts anderes sei als eine komplizierte Maschine. Die KI-Entwicklung war auch möglich, weil in den zwanziger und dreißiger Jahren wichtige Arbeiten zur mathematischen Logik verfasst wurden. Mathematiker wie Gödel, Church, Post, Markov und Turing entwickelten Begriffe wie Berechenbarkeit, Entscheidbarkeit, Vollständigkeit und Beweisbarkeit. In dieser Zeit erarbeitete man sich auch eine mathematisch exakte Definition des Algorithmenbegriffs, was grundlegend für die spätere Rechentechnik und Informationsverarbeitung war. Am Ende der vierziger Jahre prognostizierten bereits Turing und Shannon die Entwicklung künstlich-intelligenter Systeme.<sup>40</sup> In den sechziger Jahren fand ein Übergang von der reinen Zahlenmanipulation zur Verarbeitung beliebiger Zeichen und Symbole statt. Dieser Schritt wurde mit der Entwicklung der ersten Symbolmanipulations-Sprache (IPL) vollzogen. Ebenfalls bedeutsam für KI, speziell für die Verarbeitung der natürlichen Sprache, war die Entwicklung der Theorie der formalen Sprachen durch Chomsky. Zu dieser Zeit entstanden auch die ersten Spielprogramme (Schach, Dame etc.). Es lag also nahe, die bisherigen Erfahrungen zu konzentrieren.<sup>41</sup> Im Jahr 1956 fand die Dartmouth-Konferenz statt, in der eine erste Bestandsaufnahme durchgeführt, Zielstellungen formuliert und der Begriff „Artificial Intelligence“ erstmals von Wissenschaftlern wie M. Minsky, J. McCarthy, A. Newell, H.A. Simon u. a. definiert wurde.<sup>42</sup> Seit dieser Zeit lässt sich die Historie der KI in sechs Epochen einteilen:<sup>43</sup>

- Klassische Epoche
- Romantische Epoche
- Moderne Epoche
- Postmoderne Epoche
- Epoche der Renaissance
- Epoche des Cognitive Computings

In der *klassischen Epoche* (1955–1965) suchte man nach Prinzipien und Methoden, um beliebige Probleme lösen zu können. So entstand beispielsweise der „General Problem Solver (GPS)\“, der Spielprobleme lösen konnte. Als eine der zentralen Problemlösungsformen

<sup>40</sup> Siehe auch Turing 1994.

<sup>41</sup> Siehe auch Rose 1986.

<sup>42</sup> Siehe auch Barr und Feigenbaum 1982.

<sup>43</sup> Siehe auch Kurzweil 1993.

bestand diese Vorgehensweise in der zielgerichteten Durchforstung des Suchraumes. Hierzu wurden leistungsstarke Heuristiken entwickelt, die die Suche effizient gestalteten. Ernüchtert musste man jedoch am Ende dieser Epoche eingestehen, dass die Entwicklung eines allgemeinen Problemlösungsprogramms nicht gelungen war. Auf dieser Grundlage wurden auch die ersten Formelmanipulationssysteme entwickelt, die beispielsweise symbolische Berechnungen unbestimmter Integrale durchführen konnten (SAINT). Ebenfalls zu dieser Zeit begann man, sich mit natürlichsprachlichen Prozessen zu beschäftigen, und diese mit Hilfe einfacher Mustervergleichs- und Transformationssysteme (sogenanntes Pattern Matching) abzubilden. Man versuchte in der *romantischen Epoche* (1965–1975) sein Heil in der Spezialisierung und entwickelte grundlegende Methoden und Techniken für relevante Teilelemente der artifiziellen Intelligenz. Die Repräsentation von Wissen und das Suchen in Teilsuchbäumen stand im Mittelpunkt des Interesses. Erwähnenswert sind u. a. das Resolutionsverfahren für den automatischen Theorembeweis (Robinson), das Frame-Konzept für die Wissensrepräsentation (Minsky), die Semantischen Netze (Quillian) und Grammatikformalismen (Woods). Erste Frage-Antwort-Systeme und integrierte Sprachverarbeitungssysteme wurden entwickelt. Die ersten Implementierungen von LISP und PROLOG entstehen. Aber auch in dieser Epoche gelang nicht der erhoffte Durchbruch, was die Praxistauglichkeit der abgelieferten Arbeiten betrifft. So setzte sich immer mehr die Erkenntnis durch, dass die Entwicklung universeller Problemlöser mit den vorhandenen Mitteln nicht möglich sein wird. Dem Wissen schenkte man hingegen mehr Beachtung, da diesem eine zentrale Rolle in KI-Systemen zuzukommen schien. Letzteres fand seinen Niederschlag in den ersten Expertensystemen. Im Jahre 1969 fand die erste internationale KI-Konferenz (IJCAI) statt, die von da an regelmäßig alle zwei Jahre durchgeführt werden sollte. In der *modernen Epoche* (1975–1994) entdeckte man die Bedeutung des problemspezifischen Wissens und stellte fest, dass die Fähigkeit eines Programmes, Probleme effizient zu lösen, nicht so sehr von den formalen Prinzipien und Strategien abhängt.<sup>44</sup> Vielmehr werden die Art des Problemlösens und die Qualität der gefundenen Lösung von dem des Problemlösungsverfahrens zugrundeliegendem Wissen beeinflusst. Die wissensbasierten Systeme traten also immer mehr in den Vordergrund. In diesem Zusammenhang setzte man sich auch mit Produktionsregelsystemen auseinander. Aufbauend auf dem Frame-Konzept von Minsky entstanden zahlreiche Frame-Repräsentationssprachen. Im Bereich der Spracherkennungssysteme findet ein Übergang von den Frage-Antwort-Systemen zu natürlichsprachlichen Schnittstellen zu Datenbanken statt. Erste Systeme zur automatischen Übersetzung verlassen die Labors und finden ihren praktischen Einsatz. Man wandte sich verstärkt den praktischen Problemen der realen Welt zu und befragte menschliche Experten gemäß dem Ansatz „in the knowledge lies the power“, wie sie in der Praxis Probleme angingen und dort nach Lösungen suchten. Die anschließend entwickelten Programme, die sich bei der Lösungsforschung ähnlich diesen konsultierten menschlichen Experten verhielten, bezeichnete man von nun an als „echte Expertensysteme“. Es

---

<sup>44</sup> Vgl. Leidlmaier 1988.

entstanden immer mehr sogenannte KI-Werkzeuge (Toolkits), die sich zur Entwicklung von Expertensystemen als auch für die Entwicklung natürlichsprachlicher Systeme eigneten. Shells oder leere Expertensysteme wurden entwickelt, die nur noch aus einer Wissenserwerbs- und Wissensverarbeitungs-Komponente bestanden. Mit dieser zunehmenden Verbreiterung der Expertensysteme rückte auch das Problem des Wissenserwerbs, d. h. die Bereitstellung von Basiswissen immer mehr in den Vordergrund des Interesses. Auf dem Gebiet der Programmiersprachen kam es zu einer Standardisierung: COMMON LISP bei LISP und Clocksin-Mellish-Standard bei PROLOG.<sup>45</sup> Vor allem zwei Entwicklungsströmungen hinterließen zum Teil bahnbrechende Erkenntnisse aus dieser Epoche. Einerseits hatte man leistungsfähige Methoden für die Erfassung und Darstellung des Wissens gefunden. Insbesondere entwickelte man ein Verständnis dafür, wie man Zusammenhänge im Rechner abbilden konnte. Andererseits war es gelungen, Ergebnisse der Grundlagenforschung auch der Praxis zur Verfügung zu stellen. Die KI-Bewegung wurde auch sehr stark von dem japanischen Projekt FGCS (Fifth Generation Computer Systems) stimuliert. Die KI-Forschung erlebte in allen Ländern einen starken Aufschwung.<sup>46</sup> Die *postmoderne Epoche* (1995 bis 2000) stand ganz im Zeichen der kommerziellen Anwendung und des Vertriebs von KI-Werkzeugen und Expertensystem-Shells. Aber auch Systeme zur automatischen Übersetzung oder natürlichsprachliche Schnittstellen hatten inzwischen die Labors verlassen und wurden kommerziell genutzt. Es entstanden Rechner mit Nicht-Neumann-Architektur, die speziell für KI-Belange konzipiert waren. Auf dem Gebiet der Programmiersprachen zeichnete sich ein Trend ab: Logische und funktionale Programmierung wuchsen zusammen. In der Epoche der Renaissance erfolgte eine Fortsetzung des „going practices“. Die Entwicklungen fanden ihren Einsatz in ganz konkreten, praxisrelevanten Problemstellungen.<sup>47</sup> Dabei zeichneten sich folgende Entwicklungstendenzen ab:

- Zuwendung zu Alltagsproblemen (engl. real world computing)
- Implementierung von Systemen zur Einbindung der Alltagssprache
- Zusammenführung von akustischen Erkennungssystemen mit linguistischen Sprachverarbeitungssystemen
- Hinwendung zu verteilten KI-Systemen (Agentensysteme)
- Gegenseitige Integration von Bildverarbeitung, Wissensverarbeitung und Sprachverarbeitung
- Perfektionierung von konnektionistischen Modellen als Gegenströmung zur symbolischen KI
- Interdisziplinäre Zusammenarbeit mit anderen Wissenschaftsdisziplinen, insbesondere mit der traditionellen Informatik

<sup>45</sup> Vgl. Mayer 1995.

<sup>46</sup> Siehe auch Partridge 1989.

<sup>47</sup> Siehe auch Dreyfuß 1989.

- Harmonische Integration der bisher monolithischen Teiltechnologien in komponentenorientierte Baukästen
- Parallel zu dieser Entwicklung befasste man sich nun auch mit den sozialökonomischen Auswirkungen der KI bzw. mit den ethischen Problemen, die der Einsatz von KI-Anwendungen im militärischen Bereich oder im Bereich der Geheimdienste aufwirft.

In der derzeitigen Epoche des *Cognitive Computings* zeichnet sich ein weiterer Quantensprung in Richtung systemische Intelligenz ab, indem die unterschiedlichen Teiltechnologien der Artifiziellen Intelligenz so modelliert werden, dass sie sich harmonisch miteinander kombinieren lassen. Das geht sogar so weit, dass sich die einzelnen Technologien gegenseitig unterstützen, indem sie beispielsweise miteinander kommunizieren oder aber deren produzierten Ergebnisse gegenseitig validieren bzw. verifizieren. Dieser neue Impuls der Kombination bisher getrennter Technologien basiert dabei nicht auf einem weiteren Paradigmenwechsel, sondern auf der Vereinigung solcher Paradigmen.

### 5.3.2 Systemische Intelligenz

Eine systemische Intelligenz ist ein neues gedankliches Konstrukt auf der Basis von hardware- und softwaretechnologischen Möglichkeiten und Fähigkeiten unter Einbezug einer Brainware. Eine solche Intelligenz ist ein besonderes systempsychologisches Potential der Spezies Roboter, ein Potential, das nach Maßgaben noch zu bestimmender Faktoren ausgewertet werden kann.

Trotz intensiver Erforschung der „Intelligenz“ in den vergangenen hundert Jahren gehen die Auffassungen darüber, was unter Intelligenz zu verstehen ist, weit auseinander. Ein Grund für diese Uneinigkeit entspringt wohl der Tatsache, dass Intelligenz als Begriff keinen allgemein anerkannten, objektiven Inhalt besitzt. Anders als bei der Größe, beim Alter oder beim Gewicht ist es nicht möglich, ein einzelnes Merkmal direkt zu beobachten, um dann etwas über die Intelligenz eines Menschen sagen zu können. Stattdessen muß Intelligenz aus dem Verhalten, z. B. beim Lösen von Problemen oder bei der Bewältigung neuer Situationen, erschlossen werden. Der Begriff der systemischen Intelligenz wird daher in zwei verschiedenen Bedeutungen verwendet: Einerseits systembezogen im Sinne von intelligenten Aktionen, Aktivitäten oder Handlungen wie beispielsweise die Entdeckung einer neuen Energiequelle, der Entwicklung eines neuen Problemlösungsverfahrens oder der Komposition von einzelnen Aktionsschritten. Andererseits prozessbezogen, um die Prozesse zu beschreiben, denen diese intelligenten Handlungen entspringen.

Offensichtlich liegen jedem Definitionsversuch unterschiedliche Auffassungen und Perspektiven von Intelligenz zugrunde, die sich auch in den unterschiedlichen Forschungstraditionen widerspiegeln. Grundsätzlich wird auch beim Ansatz der systemischen Intelligenz auf den Ansatz der Informationsverarbeitung, den psychometrischen Ansatz und den Ansatz des Systemlebenszyklus zurückgegriffen. Diese Ansätze betrachten Intelligenz aus

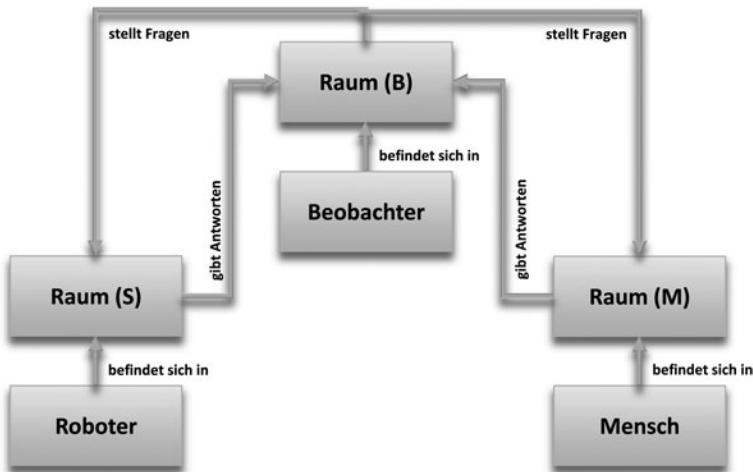
unterschiedlichen Blickwinkeln und versuchen demnach, auf ganz verschiedene Fragen eine Antwort zu geben:

- Gemäss des *Informationsverarbeitungsansatzes*, der sich aus der experimentellen Psychologie entwickelte, werden die grundlegenden Prozesse kognitiver Leistung erfasst. Dazu werden Reaktionszeiten und Gedächtnisleistungen gemessen und untersucht, wie das System auf unterschiedliche Informationen reagiert und in welcher Weise es Gelerntes verarbeitet.
- In der *Psychometrie* geht es um die Messung intelligenter Merkmale auf der Basis von Tests. Psychometrische Intelligenztheorien wenden spezielle statistische Verfahren an, um Intelligenztests zu analysieren. Daraus werden dann Schlussfolgerungen über die Struktur der Intelligenz abgeleitet.
- Der *Ansatz des Systemlebenszyklus* beschäftigt sich mit der Intelligenzentwicklung im Verlauf des Lebenszyklus eines Robotersystems.

Unabhängig der durch diesen neuen Ansatz heraufbeschworenen Kontroversen vertritt dieses Buch den Standpunkt, dass die Intelligenz als Anpassungsfähigkeit an die Anforderungen einer heterogenen und sich wandelnden Umgebung aufzufassen und nicht als angeborene Persönlichkeitseigenschaft zu zementieren ist.

In Bezug auf die Entwicklung des neuen systemischen Intelligenzquotienten ( $IQ_S$ ), wird ebenfalls auf die bisher erarbeiteten Konzepte zurückgegriffen. So definierte der deutsche Psychologe William Stern erstmalig den Begriff *Intelligenzquotient* (IQ). Ein Defizit des bis damals gängigen „Intelligenzalters“ (IA) von Binet bestand nämlich darin, dass der absolute Intelligenz- Rückstand oder -Vorsprung über den tatsächlichen Leistungsstand wenig aussagt. So muss beispielsweise das Defizit eines fünfjährigen Kindes mit einem IA von 3 als gravierender bewertet werden als das Defizit eines zehnjährigen Kindes mit einem IA von 8, obwohl der absolute Rückstand in beiden Fällen zwei Jahre beträgt. Der IQ von Stern berücksichtigte dies, indem das Intelligenzalter durch das Lebensalter geteilt wurde. Um auf ganzzahlige Werte zu gelangen, wurde der Wert später mit 100 multipliziert. Doch auch der so definierte Intelligenzquotient hat Nachteile, da sich die kognitiven Leistungen mit zunehmendem Alter nicht mehr verbessern, das Lebensalter, durch das dividiert wird, jedoch kontinuierlich ansteigt. Für Erwachsene ist daher ein IQ, der Intelligenzalter mit Lebensalter in Beziehung setzt, völlig sinnlos. Dieses Problem löste der Amerikaner *David Wechsler*, der sich in erster Linie für die Intelligenz erwachsener Menschen interessierte. Er führte 1932 den IQ als „Abweichungsquotienten“ ein. Prinzipiell hat das von ihm entwickelte Vorgehen – Ermittlung der Abweichung zwischen individuellem Leistungswert und dem Leistungsmittelwert der korrespondierenden Altersgruppe – noch heute Gültigkeit. Dieser Abweichungswert wird nach wie vor als Intelligenzquotient bezeichnet, obwohl es sich strenggenommen nicht um einen Quotienten handelt.

Um die eher allgemeinen Komplikationen, die mit einer Definition eines Begriffes und die Schwierigkeiten im Speziellen mit dem Begriff der „Intelligenz“ zu vermeiden, kann



**Abb. 5.8** Turing Test

man zwei Wege gehen. Der erste Weg besteht darin, dass man ein operationales Kriterium angibt, anhand dessen sich ein System beurteilen lässt, ob es intelligent ist oder nicht.

Hierzu stellt man folgendes Gedankenexperiment vor: In zwei abgeschlossenen Räumen  $R_M$  und  $R_S$  befinden sich ein Mensch M bzw. das auf Intelligenz zu beurteilende technische Robotersystem S. Weiter sei ein menschlicher Beobachter B vorhanden, der mit M und S kommunizieren kann, ohne durch bloßen Augenschein oder sprachliche Eigenheiten bereits feststellen zu können, wo sich M bzw. S befinden. Wenn es dem Beobachter B durch noch so geschickte Aufgabenstellung oder Anfragen an M und S und Beurteilung der erzielten Resultate nicht zu unterscheiden gelingt, in welchem Raum sich M bzw. S befinden, dann kann man mit Recht behaupten, daß S ebenso intelligent wie M ist (Abb. 5.8).

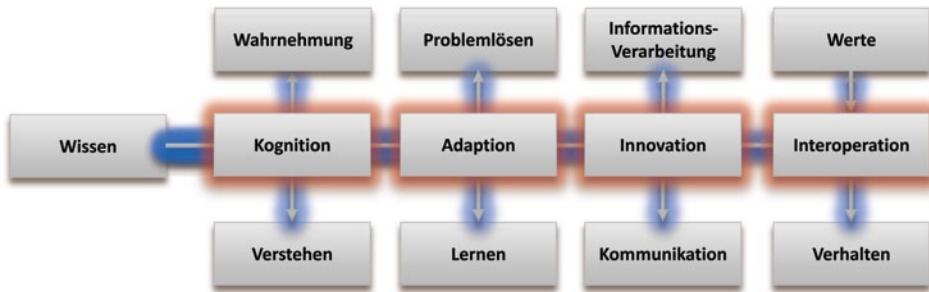
Die zweite Möglichkeit, die einen konstruktiveren Charakter trägt, besteht darin, die *Fähigkeiten* und *Merkmale* zu definieren, die ein Robotersystem System aufweisen muss, um intelligent zu sein. Eine Fähigkeit ist dabei ein gegenwärtig verfügbares und abrufbares Potential, etwas zu leisten. Als wichtigste *Charakteristika* dieser Art sind die folgenden Fähigkeiten anzusehen:

- *Symbolverarbeitungsfähigkeit*: die Fähigkeit, mit beliebigen Symbolen (nicht nur mit Zahlen) operieren zu können
- *Wissensrepräsentationsfähigkeit*: der Besitz eines inneren Modells der äußeren Welt, was die Fähigkeit zur Wissensrepräsentation impliziert

- *Schlußfolgerungsfähigkeit*: die Fähigkeit, das gespeicherte Wissen zweckentsprechend einzusetzen und insbesondere, vernünftige Schlußfolgerungen aus dem Wissen ziehen zu können
- *Abstraktionsfähigkeit*: die Fähigkeit zur Verallgemeinerung (Abstraktion) und zur Spezialisierung (d. h. zur Anwendung allgemeiner Zusammenhänge auf konkrete Sachverhalte)
- *Problemlösungsfähigkeit*: die Fähigkeit, sich planvoll zu verhalten und entsprechende Strategien zum Erreichen der Ziele bilden zu können (Problemlösefähigkeit im allgemeinsten Sinn)
- *Anpassungsfähigkeit*: Anpassungsfähigkeit (Adaptivität) an verschiedene, u. U. sich zeitlich ändernde Situationen und Problemumgebungen
- *Lernfähigkeit*: Lernfähigkeit verbunden mit dem Vermögen, partiellen Fortschritt oder Rückschritt einschätzen zu können
- *Orientierungsfähigkeit*: die Fähigkeit, auch in unscharf bzw. unvollständig beschriebenen oder erkannten Situationen sich orientieren und handeln zu können
- *Mustererkennungsfähigkeit*: die Fähigkeit zur Mustererkennung (Besitz von Sensoren) und zur aktiven Auseinandersetzung mit der Umwelt (Besitz von Effektoren)
- *Kommunikationsfähigkeit*: die Fähigkeit, über ein Kommunikationsmittel von der Komplexität und Ausdrucksfähigkeit der menschlichen Sprache zu verfügen
- *Intersystemische Fähigkeit*: Verstehen anderer Systeme
- *Autonomiefähigkeit*: die Fähigkeit, sich seine eigenen Verhaltensregeln entweder selbst zu geben oder sie zumindest in Interaktion mit der Umwelt zu ergänzen, zu verändern oder durch neue zu ersetzen
- *Körperlich-kinästhetische Fähigkeit*: Fertigkeiten der motorischen Bewegung und Koordination
- *Intrasystemische Fähigkeit*: Verstehen des eigenen Systems, Selbstbewusstsein, Entwicklung eines Identitätsbewusstseins<sup>48</sup>

Die genannten Merkmale sind natürlich nicht unabhängig voneinander und auch nicht vollständig. Sie beleuchten, wie die Strahlen mehrerer Scheinwerfer eine Qualität, in deren Brennpunkt das Phänomen „Intelligenz“ steht. Die zuerst genannte Fähigkeit zur *Symbolmanipulation* muss als besonders fundamental angesehen werden, da sie eine notwendige Voraussetzung für alle anderen Merkmale darstellt. Die Methoden zur automatischen Symbolmanipulation finden heute wegen ihrer grundlegenden Wichtigkeit Ausdruck in allen höheren Programmiersprachen. In denen der artifiziellen Intelligenz stehen sie im Vordergrund. Insofern wird in den folgenden Ausführungen dann einem Robotersystem die Eigenschaft der „systemischen Intelligenz“ zugesprochen, wenn es einen wesentlichen Teil der spezifizierten Merkmale, und mindestens die ersten drei, in sich vereinigt. Um den Turing-Test aus dem voran gegangenen Kapitel zu bestehen, muss ein artifizielles System folgende Fähigkeiten haben:

<sup>48</sup> Vgl. Dennett 1994.



**Abb. 5.9** Systemische Intelligenzkriterien

- *Kommunikationsfähigkeit* in Form einer Verarbeitung natürlicher Sprache;
- *Wissensrepräsentationsfähigkeit* zur Speicherung von Information vor und während der Kommunikation;
- *Schlussfolgerungsfähigkeit* zur Beantwortung der Fragen aus dem gespeicherten Wissen und um neue Schlüsse zu ziehen;
- *Lernfähigkeit*, um sich an neue Umstände anzupassen und typische Muster zu entdecken.

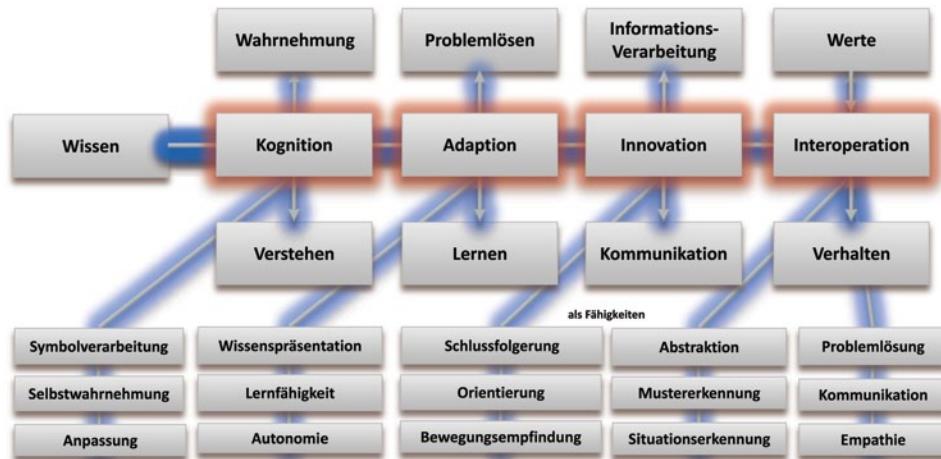
Bei der Realisierung künstlich intelligenter Systeme kann man zwei prinzipiell verschiedene Vorgehensweisen unterscheiden. Zum einen kann man versuchen, das menschliche Denken zu erforschen und zu modellieren (*Simulationsmethode*). Zum anderen kann man sich rein phänomenologisch an Merkmalen der Intelligenz (wie den oben genannten) orientieren, ohne primär danach zu fragen, wie diese Fähigkeiten beim Menschen zustande kommen. In diesem Fall ist das Ziel erreicht, wenn sich das System intelligent verhält, ohne unbedingt die menschlichen Denkprozesse zu kopieren (*phänomenologische Methode*) (Abb. 5.9).

Es sind vor allem vier Kriterien, die gemeinsam den Zusammenhang einer wie auch immer gearteten systemischen Intelligenz ausmachen: Kognition, Adaption, Innovation und Interoperation. *Kognition* ist die allgemeine Fähigkeit, sich selbst als System, sein relevantes Umfeld, sowie seine Umwelt zu erkennen, zu antizipieren, zu reflektieren und das daraus bestehende Wissen zu erweitern oder zu revidieren. Kognition als Fähigkeit bzw. Kriterium stellt die enge Beziehung zwischen dem Verstehen und der Wahrnehmung im Sinne des hermeneutischen Zirkels dar. Das Vorverständnis, die Annahmen über den Gegenstand der Wahrnehmung, ist ein Ausschnitt des bestehenden Wissens über die Umwelt und bestimmt die erste Perspektive. Die weitere Wahrnehmung bestätigt oder verwirft dieses Verständnis. Der äußere Kognitionszyklus bezieht das bestehende implizite und explizite Wissen eines Systems über die Umwelt mit ein. Dieses Wissen bereichert das Verständnis, schafft neue Perspektiven, intensiviert die Wahrnehmung und macht sich damit selbst zum Objekt der Revision und Erweiterung. Der Zweck der Kognition eines Systems besteht darin, dessen Verständnis und Wissen über sich selbst, die Umwelt und das Umfeld, in dem es interoperiert, zu kontrollieren, zu revidieren und aktiv zu gestalten, damit

es seine erforderliche Varietät erkennen und antizipieren kann.<sup>49</sup> Kognition schafft damit neues Wissen um zu interoperieren. Um effektive Verhaltens- bzw. Interoperationsmuster zu entwickeln, sind Systeme darauf angewiesen, ihre Umwelt zu kennen. Dazu sammeln sie Daten, Informationen und Heuristiken, um alles zusammen entsprechend zu interpretieren. *Adaption* ist die Fähigkeit, auf Probleme flexibel zu reagieren, das bestehende Wissen anzuwenden und sich seiner Umwelt anzupassen. Eine Adaption stellt, ausgehend von Problemstellungen, die enge Beziehung zwischen dem Lernen und dem Finden von Problemlösungen her. Diese Wechselbeziehung beschreibt ein operatives Lernen (Single-Loop-Lernen). Es handelt sich um eine Suche nach effizienteren Vorgehensweisen, die durch bestehende Konzepte begrenzt ist. Beispiele sind das Lernen durch Versuch und Irrtum und das Lernen durch Instruktion. Dieses Lernen geht von bestehenden Erfahrungen aus, die bereits in einem engen Zusammenhang mit vorangehenden Problemlösungen stehen. Adaption nutzt alles Wissen des Systems, welches für den Gehalt der Lernprozesse sinnvoll erscheint. Der Einbezug neuen Wissens erhellt den Problemzusammenhang und erlaubt das Eingehen auf antizipierte Problemstellungen. Neues Wissen deckt unweigerlich die Grenzen der bestehenden Konzepte und Vorgehensweisen auf, stellt Routinen in Frage und ermöglicht so ein konzeptionelles Lernen (Double-Loop-Lernen). Es lassen sich dadurch schneller umfassende und effektive Problemlösungen entwickeln. Der Zweck der Adaption eines Systems besteht darin, bestehendes Wissen durch Lernprozesse direkt in Fähigkeiten umzusetzen, damit es für aktuelle und antizipierte Problemlösungen nutzbar wird. Im Sinne des Komplexitätsmanagements bedeutet dies, im relevanten Umfeld das Gesetz der erforderlichen Varietät konzeptionell und operativ sowie effektiv und effizient zu erfüllen. Der Adoptionszyklus nutzt Wissen. Die Konzepte des systemischen Lernens setzen sich mit der Fähigkeit der Adaption auseinander und zeigen, wie diese in einem kontinuierlichen Prozess in einem System vor sich gehen kann. Hervorzuheben ist die Kybernetische Methodik, welche das Lernen im Kontext der Problemsituation konzeptiell ermöglicht. *Innovation* ist die Fähigkeit, neues Wissen zu schaffen und zu instrumentalisieren, um so die Umwelt aktiv mitzugestalten. Eine Innovation geht von der engen Beziehung zwischen Informationsverarbeitung und Kommunikation in einem bestimmten Kontext innerhalb und außerhalb des Systems aus. Es fordert Intuition und Emotion. Eine intensive Kommunikation in allen derzeit technisch möglichen Formen verbindet die verschiedenen Informationsverarbeitungsprozesse. Der Zweck der Innovation ist das Versorgen der Systeme mit einem fließenden Strom an potenziellen Innovationen. Auch die Innovation schafft neues Wissen. *Interoperation* ist die Fähigkeit, Wissen und erworbene Fähigkeiten direkt wirklichkeitswirksam werden zu lassen und zweckmäßig in den Verhaltensmustern zu verankern. Zwischen Werten und dem beobachtbaren Verhalten besteht eine enge Beziehung, welche die Interoperation innerhalb eines bestimmten Handlungszusammenhangs beschreibt. Werte formen durch ihren direkten Einfluss auf Einstellungen und Verhaltensregeln ein bestimmtes Verhalten. Dieses wirkt wertvermittelnd, weil es Werte systemkulturell überträgt. Entsprechen die geäußerten Werte nicht den authen-

---

<sup>49</sup> Siehe auch Newell 1990.



**Abb. 5.10** Systemische Intelligenzkriterien und Fähigkeiten

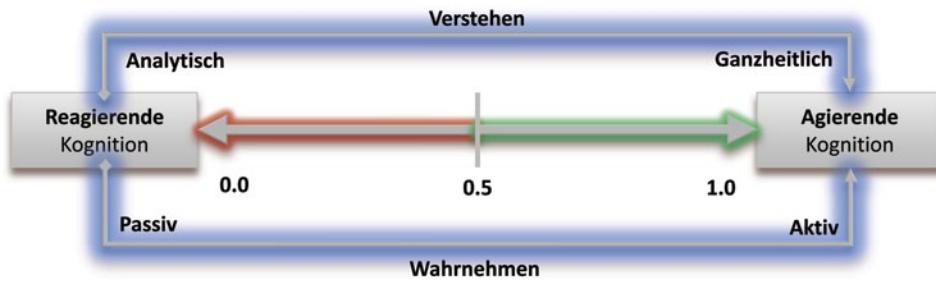
tischen Werten, scheitert das Verankern des intendierten Verhaltens. Stimmen die geäußerten Werte mit den authentischen überein, entwickeln sich die erwünschten Verhaltensmuster. Die zu beobachtenden Verhaltensmuster sind konsequent, bestätigen die Authentizität der Werte und verstärken deren Wirkung. Eine systemische Realisation überprüft die Zweckmäßigkeit der bestehenden Verhaltensmuster, Verhaltensregeln und Einstellungen, indem sie den Handlungszusammenhang erweitert und diesem Zusammenhang neues Wissen gegenüberstellt (über Zusammenhänge, Absichten, Fähigkeiten, Möglichkeiten, Konzepte oder Vorgehensweisen). Die Erkenntnisse führen zu einer Weiterentwicklung der Verhaltens- bzw. Interoperationsmuster, der Verhaltens- und Interoperationsregeln, der Einstellungen oder ganzer Verhaltens- bzw. Interoperationskonstellationen. Über die Beurteilung der Zweckmäßigkeit und darüber, in welchem Ausmaß der Einbezug neuen Wissens geschehen kann, bestimmen unmittelbar die in dem System gelebten Werte. Der Zweck der Realisation besteht darin, unter stetem Einbezug neuen Wissens die Verhaltens- bzw. Interoperationsmuster und Verhaltens- bzw. Interoperationskonstellationen zu entwickeln.

Systemische Intelligenz ist aus dieser Perspektive betrachtet, die Fähigkeit eines Systems, sein Umgebung zu erkennen, zu antizipieren, sich dieser anzupassen, diese mitzustalten und damit auf sie einzuwirken oder diese Umgebung zu verlassen, um in eine andere zu wechseln. Die Netzwerkstruktur systemischer Intelligenz entsteht durch die Kombination der beschriebenen Fähigkeiten bzw. Kriterien Kognition, Adaption, Innovation und Interoperation. Das sich selbstorganisierende Netzwerk systemischer Intelligenz zeichnet sich aus durch die gemeinsame, gleichzeitige Wirkung seiner neun Elemente Wahrnehmung, Verstehen, Lernen, Problemlösung, Informationsverarbeitung, Kommunikation, Werte, Verhalten und Wissen (Abb. 5.10).

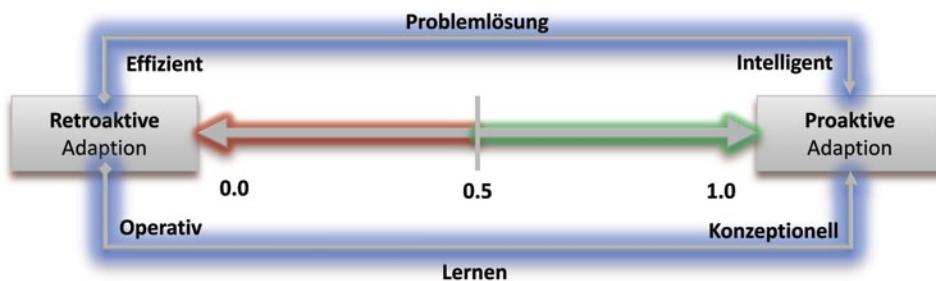
Der Zusammenhang und das Niveau systemischer Intelligenz eines Robotersystems entstehen erst durch das Verbinden der Fähigkeiten Kognition, Adaption, Innovation und Interoperation. Je ausgeprägter die einzelnen Fähigkeiten sind und je intensiver ihr Zusammenhang Wirkungen entfalten kann, desto höher wird das Niveau systemischer Intelligenz, ausgedrückt in einem gesteigerten Intelligenzquotienten. Systemische Intelligenz ist dabei nicht nur ein Prozess. Sie ist vielmehr ein selbstorganisierender und selbstreferentieller Zusammenhang, den ein System nutzen kann. Jedes Element und jede Ebene ist gleichzeitig Wirkungs- und Zweckursache. Jede Veränderung des Inhaltes eines Elementes verändert gleichzeitig die Inhalte des ganzen Zusammenhangs. Systemische Intelligenz ist iterativ und evolutionär. Sie nutzt Ungewissheit und Ambiguität als Gestaltungsfreiraum. Kognition, Adaption, Innovation und Interoperation wirken gleichzeitig, überlappend und wechselwirkend. Neue Informationen (Kognition) oder neue Problemstellungen (Innovation) schlagen sich beispielsweise in verbesserten Prozessen des Systems (Adaption) nieder. Die Verhaltensmuster entwickeln sich entsprechend (Interoperation). Die veränderten Verhaltensmuster werden wahrgenommen und beurteilt (Kognition), was zu weiteren Anpassungen (Adaption und Interaktion) oder zu weiteren neuen Konzepten führen kann, die umgesetzt werden (Innovation und Interoperation). Systemische Intelligenz kann demzufolge nicht verordnet werden. Eine Diagnose, wie die Kognition, die Adaption, die Innovation und die Interoperation konstituiert sind und zusammenhängen, ist im Rahmen der Ermittlung des systemischen Intelligenzquotienten möglich. Dadurch entsteht ein Verständnis für das bestehende Niveau systemischer Intelligenz. Die Diagnose deckt Stärken und Schwächen der Fähigkeiten systemischer Intelligenz in ihrem spezifischen Umfeld auf.

Für jede Ebene existieren dabei stets zwei Extrempole, ein Zielpol und ein Basispol, die innerhalb eines Quadranten eines Koordinatensystems dargestellt werden können. Auf den Achsen finden sich die Ausprägungen, welche die Grundelemente der betreffenden Ebene qualifizieren. Für jedes Grundelement wird eine Bewertung auf einer Skala von Null (Basispol) bis Eins (Zielpol) vorgenommen. Die Multiplikation der Bewertungen ergibt den Summanden des betreffenden Zyklus. Die Summe der vier Summanden, gemeinsam mit der Bewertung des Gesamtzusammenhangs (ebenfalls von eins bis zehn), ergibt schließlich den Intelligenzquotienten. Zum Intelligenzniveau des betreffenden Systems führt die Division des erreichten Intelligenzaggregates durch das maximale Intelligenzaggregat von 90 (neunmal die Höchstwertung). Das Intelligenzniveau kann dann in Prozentpunkten angegeben werden (maximal 100) und entspricht dann dem systemischen IQ. Eine sinnvolle Aussagekraft erhält die Maßzahl des Niveaus systemischer Intelligenz vor allem durch eine Vergleichsmöglichkeit im Ablauf. Die Bewertung der einzelnen Grundelemente sollte aus einer standardisierten, regelmäßigen Ermittlung hervorgehen.

Den Basispol der Kognition bildet dabei eine *reagierende Kognition*. Sie zeichnet sich durch eine passive Wahrnehmung und ein analytisches Verstehen aus. Zweck der Kognition sind die Bestätigung oder die Revision des notwendigen Wissens und die Lenkung des Systems. Der Zielypus, die *agierende Kognition*, ist geprägt durch eine aktive Wahr-



**Abb. 5.11** Kognition

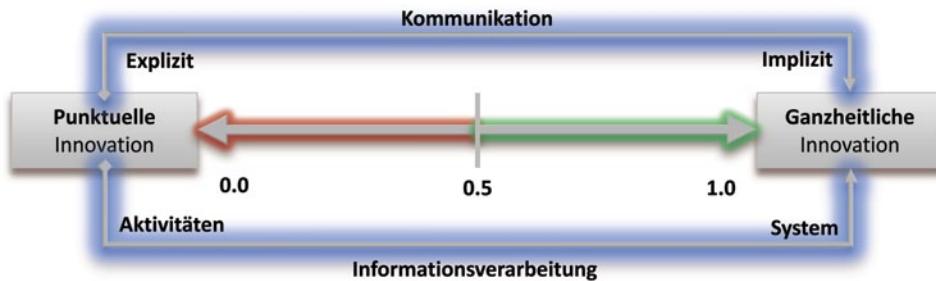


**Abb. 5.12** Adaption

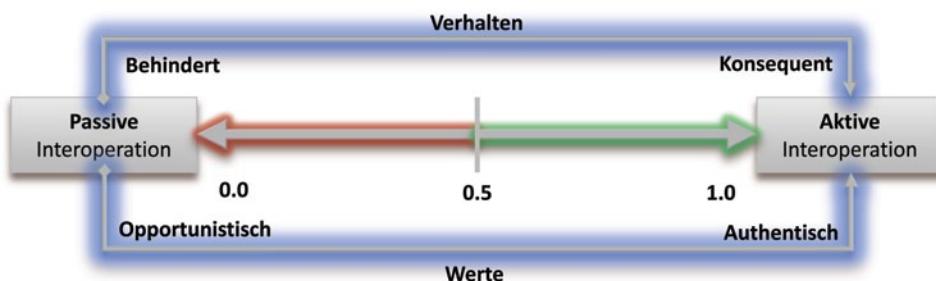
nehmung und ein ganzheitliches Verstehen. Das System ist vernetzt mit seiner Umwelt und gestaltet dieses aktiv mit (Abb. 5.11).

Der Basispol der Adaption heißt *retroaktive Adaption*. Seine Problemlösungen orientieren sich am Kriterium der Effizienz und die Lernprozesse sind operativer Natur. Sie gehen über gegebene Zusammenhänge nicht hinaus, wenn es nicht unmittelbar notwendig ist. Diese Art der Adaption ist eher sporadisch, denn sie wird erst ausgelöst, wenn gravierende Fähigkeitslücken entdeckt werden. Sie versucht primär, bestehende Hindernisse durch den Einbezug von Erfahrungswissen zu überwinden. Der Zielpol, *proaktive Adaption*, orientiert sich zusätzlich an der Effektivität von möglichen Problemlösungen. Die Lernprozesse sind operativer und konzeptioneller Natur. Im Vordergrund steht das Wahrnehmen neuer Möglichkeiten. Dieser Typus der Adaption ist kontinuierlich, denn er geht von potenziellen Problemen aus. Er ist bestrebt, stets das neueste Wissen durch alle möglichen Lernformen einzubeziehen, um Hindernisse gar nicht aufkommen zu lassen (Abb. 5.12).

Bei der Innovation wird auf der Basis möglichst allen verfügbaren impliziten und expliziten Wissens versucht, durch intensive Informationsverarbeitungs- und Kommunikationsprozesse neues Wissen innerhalb des Systems zu entwickeln. Dieses Wissen beeinflusst die Modelle des Systems. Den Basispol bildet die *punkuelle Innovation*. Sie zeichnet sich vor allem dadurch aus, dass sich die einzelnen Verarbeitungsprozesse vor allem in der Dimension der Aktivitäten bewegen. Sie reagiert damit auf eine Nachfrage nach neuem Wissen und bezieht vor allem explizites Wissen in die Entwicklung ein. Der Zielpol ist die *ganzheitliche Innovation*. Sie ist geprägt durch umfassende Verarbeitungsprozesse, die



**Abb. 5.13** Innovation



**Abb. 5.14** Interoperation

sich nicht nur vorwiegend auf die Abarbeitung der Aktivitätenliste, sondern auch auf die Gestaltung des Systems als Ganzes beziehen. Ihr Zweck ist das Hervorbringen von Innovationen aus eigenem Antrieb. Sie bezieht alles verfügbare Wissen mit ein und gibt dem System die Möglichkeit, ihr Umfeld aktiv mitzugestalten (Abb. 5.13).

Bei der Interoperation trägt das neue Wissen direkt zur Entwicklung eines Verhaltens bei, das den normativen Kern bestärkt und dem Zweck des Systems entspricht. Den Basisspol bildet die *passive Interoperation*. Das beabsichtigte Verhalten orientiert sich an opportunistischen Werten und sieht sich in seiner Umsetzung durch bestimmte ungünstige Umstände innerhalb bestehender Verhaltenskonstellationen stark behindert. Ohne direkt empfundene Notwendigkeit fehlt die Motivation, die ungünstigen Umstände zu beseitigen und die Verhaltenskonstellationen optimieren zu wollen. Ihren Zweck sieht diese Art der Realisierung solange in der Verwendung ihres Erfahrungswissens, bis der Einbezug neuen Wissens unumgänglich ist.

Die *aktive Interoperation* orientiert sich an authentischen Werten und kompromisslos am normativen Kern. Ihr beabsichtigtes Verhalten ist auf die Konsistenz mit den Werten ausgerichtet und wird konsequent umgesetzt. Umstände, die legitime Absichten behindern, werden eigendynamisch beseitigt. Ihren Zweck findet diese Art der Interoperation in der sofortigen Umsetzung neuesten Wissens (Abb. 5.14).

Damit sind alle vier Ebenen oder Fähigkeiten systemischer Intelligenz abschließend diagnostiziert. Ein schnelles und zweckmäßiges Zusammenwirken der einzelnen Ebenen systemischer Intelligenz bereichert ein System mit einer Fähigkeit, die ihm eine intensive

Auseinandersetzung mit der Umweltdynamik ermöglicht. Damit befähigt die systemische Intelligenz das System, sein Umfeld zu erkennen, zu antizipieren, sich diesem anzupassen, es mitzugestalten und es gegebenenfalls, bei Gefährdung der Existenz, zu wechseln. Ein hohes Niveau systemischer Intelligenz kommt erst durch das Zusammenführen von Kognition, Adaption, Innovation und Interoperation zustande.

Mit diesem Begriff der systemischen Intelligenz kann man ein Robotersysteme formal durch die Menge der Aktionen  $A = \{a_1, a_2, a_3, \dots, a_n\}$  definieren, die es in einer Umwelt ausführen kann. Diese Umwelt wiederum ist dadurch gekennzeichnet, dass sie sich über die Zustände  $S = \{s_1, s_2, s_3, \dots, s_n\}$  definiert. Bezüglich der Aktionen des Robotersystems können das der einfache Griff oder die Ortsveränderung des Robotersystems sein (bis hin zu komplexen sensomotorischen Handlungsfolgen, je nach Granularität der Modellierung). Der Umweltzustand kann der isolierte Zustand eines Montageobjekts sein, bis hin zu momentanen Raum-Zeit-Konfigurationen mehrerer interagierender Robotersysteme, die entweder ein gemeinsames Ziel verfolgen, oder aber gegeneinander konkurrieren. Dabei muss unter Umständen berücksichtigt werden, dass das Robotersystem schon wegen seiner beschränkten Sensorik die Umweltzustände nur teilweise kennt und sie durch seine Aktionen auch nur teilweise beeinflussen kann. Erschwerend kann hinzukommen, dass das Robotersystem zudem auch im Normalfall nicht weiß, welches die Gesamtheit der Konsequenzen seines Handelns ist bzw. welches die fernsten Aktionen anderer Robotersysteme sind, die auf den von ihm beobachteten Zustand Einfluss haben (bzw. seine Entscheidung für eine Aktion beeinflussen sollten). Diese Beschränkung findet ihren unmittelbaren Niederschlag darin, dass im Normalfall nicht klar ist, inwieweit (also bezüglich welcher Aspekte) das Robotersystem sein internes Wissen nach jeder Interoperation aktualisieren muss. Ausgestattet mit diesen beiden Definitionen für S und A kann man ein Robotersystem mathematisch als ein System auffassen, das eine Abbildung  $S \rightarrow A$  vornimmt, d. h. bei einem gegebenen Zustand eine Aktion vornimmt. Dabei kann noch Unterschieden werden zwischen dem deterministischen Fall, in dem aus jedem Paar der Produktmenge  $S \times A$  ein ganz bestimmter neuer Zustand folgt und dem nichtdeterministischen Fall, bei dem unterschiedliche Folgezustände auftreten können, die jeweils eine bestimmte Wahrscheinlichkeit haben. In beiden Fällen folgt aus der Aktion des Robotersystems eine Zustandsänderung der Umwelt, die nun ihrerseits wieder dazu führt, dass das Robotersystem weitere Aktionen (nämlich die auf den jeweils neuen Zustand passende) ausführt.

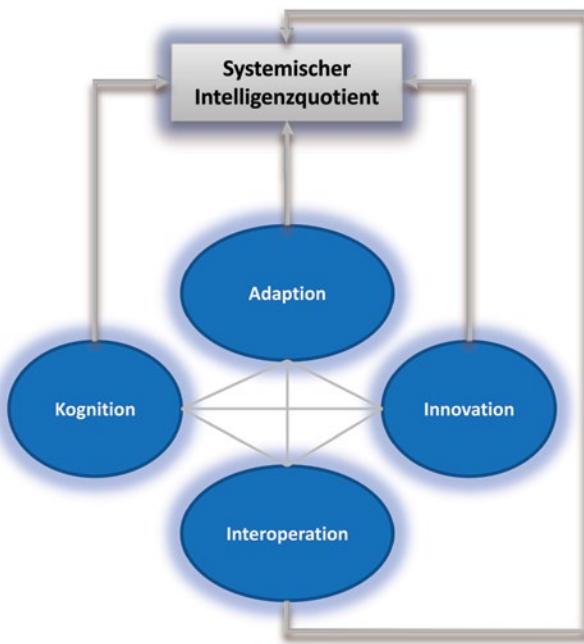
Die bisher erarbeitete elementare Definition eines solchen Robotersystems dient als erste Annäherung an die Modellierung des Verhaltens von Robotersystemen, bedeutet sie doch, dass das Robotersystem im Sinne eines fest programmierten (Zustands)automaten auf eine vorgelegte Situation immer wieder gleich reagiert. Es hat vermöge seiner direkten Verknüpfung von Eingabe und Ausgabe allerdings nicht die Freiheit, etwas anderes zu tun. Will man also intelligente Robotersysteme modellieren, erfordert dies unweigerlich eine Erweiterung des bisher erarbeiteten mathematischen Modells. Dazu stattet man die Robotersysteme mit generellen Intentionen I aus, d. h. man billigt ihnen zu, gewisse Absichten zu verfolgen. Um dies zu tun, versuchen die Robotersysteme, Ziele zu erreichen, wobei der Zustand zwischen Absicht und Zielerreichung auch als systemischer Wunsch W

aufgefasst werden kann. Verbal lässt sich dieser Sachverhalt dann so ausdrücken, dass das Robotersystem versucht ist, diesen Wunsch durch entsprechende Interoperationen zu erfüllen. Weiterhin verfügen intelligente Robotersysteme über gewisse interne Vorstellungen bezüglich der Merkmale und den Zustand der Welt V. Diese bisher erarbeiteten mathematischen Erweiterungen führen zu einer sogenannten IWV-Architektur. Dabei bezeichnet S den Zustand des vom Robotersystem erfassten Umweltausschnitts. Es kann sich dabei um optisch erfassbare Zustandsvariable handeln oder aber auch um Kräfte, Temperaturen etc. Für die Ausstattung des Robotersystems mit einem hohen Intelligenzquotienten sind nun zwei Funktionen von wesentlicher Bedeutung. Die Funktion  $f_{IQ}: S \rightarrow P$  bildet den erfassten Umweltzustand in so genannte Perzepte P ab. Perzepte sind die direkten Resultate des systemischen Wahrnehmungsvorgangs. Diese Funktion ist beispielsweise dafür zuständig, aus dem enorm komplexen auf das Robotersystem einströmenden Datenvolumen einer Kamera die entscheidenden Merkmale bzw. Objekte zu extrahieren. Dabei kann  $f_{IQ}$  selbst eine sehr komplexe, mehrstufige Abbildung darstellen, die sich in Abhängigkeit von S und/oder über die Zeit ändert. Für ein intelligentes Robotersystem wird zudem gefordert, dass sich  $f_{IQ}$  zur Lebenszeit des Robotersystems permanent entwickelt. Dies impliziert, dass das Robotersystem die Strukturen seines eigenen Wahrnehmungssystems aufbaut. Die geeignete repräsentierten Perzepte P sind nun die Eingangsgröße für eine Funktion  $g_{IQ}$ , welche die aktuellen Perzepte in Aktionen abbildet. Diese Funktion  $g_{IQ}$  bestimmt im Wesentlichen den Intelligenzquotienten des Robotersystems. Dabei ist diese Funktion parametriert mit geeigneten Variablen, deren Werte sich aus den aktuellen Wünschen, Zielen und Umweltzuständen ableiten, die sich ihrerseits auf ein raum-zeitliches Gedächtnis stützen können. Die Struktur der Abbildung  $g_{IQ}$  kann aber ebenso auch durch das aktuelle Perzept beeinflusst werden. Weiterhin ist diese Funktion nicht statischer Natur, sondern kann sich aufgrund von Erfahrungen stets anpassen, indem es beispielsweise die getroffenen Entscheidungen einer gewissen „Fitness-Betrachtung“ unterzieht. Mathematisch betrachtet modifiziert sich  $g_{IQ}$  sozusagen selbst. Sowohl  $f_{IQ}$  als auch  $g_{IQ}$  sind üblicherweise von aufwändiger algorithmischer Struktur, zeitabhängig und nichtlinear. Zu ihrer Implementierung werden daher mathematische Formalismen benötigt, die stark nichtlineare Abhängigkeiten modellieren können. Dies sind typischerweise unscharfe Fuzzy Regelbasen, neuronale Netze oder genetische Algorithmen.

Damit wird auch klar, welche Größen zur Erzielung einer hohen systemischen Intelligenz vom Entwickler oder dem Robotersystem selbst beeinflussbar sind:

- $0 > IQ_S < 1.0$ : Die Funktionen  $f_{IQ}$  als auch  $g_{IQ}$  sind vom Entwickler des Robotersystems fest vorgegeben und können nicht verändert werden. Das Robotersystem kann also auf Umwelteinflüsse bzw. Zustandsänderungen reagieren, aber immer in derselben Weise. Der systemische IQ ( $IQ_S$ ) wird also nahe bei der 1 liegen.
- $1.0 > IQ_S < 1.5$ : Die Funktion  $g_{IQ}$ , also die Abbildung vom Perzept auf die Aktion, ist entweder parametrisiert (und diese Parameter können vom Robotersystem selbst verändert werden) oder aber es stehen verschiedene Funktionen  $g_{IQ}$  zur Verfügung, die das Robotersystem auswählt. Der systemische IQ ( $IQ_S$ ) wird sich zwischen 1.0 und 1.5 bewegen.

**Abb. 5.15** Modell der Systemischen Intelligenz



- $1.5 > IQ_s < 1.7$ : Die Wünsche  $W$  können selbständig modifiziert werden. Dies kann dadurch geschehen, dass Wünsche mit Zuständen verknüpft werden, in der Weise, dass aus der Erfahrung der Vergangenheit bewertet wird, wie nützlich die Formulierung eines Wunsches zur Verfolgung einer Intention ist. Eine Änderung der Bewertung der Nützlichkeit hat sofort eine Änderung der Wünsche zur Folge. Der systemische IQ ( $IQ_s$ ) wird sich zwischen 1.5 und 1.7 bewegen.
- $1.7 > IQ_s < 1.8$ : Die Menge der Perzepte  $P$  wird verändert, damit nimmt das Robotersystem die Umwelt anders wahr, die inneren Repräsentationen und die Ziele verändern sich automatisch so, dass sie für eine bessere Intentionenverfolgung sorgen. Dies kann insbesondere durch eine Veränderung von  $f_{IQ}$  geschehen. Der Vergleich mit einem Säugling ist offensichtlich, dessen Sichtvermögen sich in den ersten Monaten dramatisch ändert und damit erst zielgerichtetes Handeln ermöglicht. In gleicher Weise kann sich auch  $g_{IQ}$  verändern, also aus den gleichen Perzepten andere Aktionen folgen. Der systemische IQ ( $IQ_s$ ) wird sich zwischen 1.7 und 1.8 bewegen.
- $1.8 > IQ_s < 1.99$ : Die Mengen  $I$  und  $A$  werden verändert. Robotersysteme können also ihre übergeordneten Intentionen und ihr Handlungsrepertoire ändern. Das Verhalten des Robotersystems kann sich vollständig wandeln. Der systemische IQ ( $IQ_s$ ) wird sich zwischen 1.8 und 1.99 bewegen (Abb. 5.15).

Das Modell der systemischen Intelligenz artifizieller System ist bewusst einfach gehalten, da durch einige wenige Bausteine der Intelligenzquotient errechnet werden kann. Die Berechnungen an den einzelnen Schaltstellen richten sich nach Gewichtsfaktoren.

Als Legitimation für die Ausstattung der Robotersysteme mit einem hohen systemischen Intelligenzquotienten soll die *Kogniogenese* beitragen, die sich mit der Entwicklung von Eigenschaften, wie Erkennen, Wahrnehmen oder der Fähigkeit zur Erkenntnis auseinandersetzt. Diese belegt, dass sich „Unintelligente“ Wesen wie Tiere und Pflanzen sehr wohl an Veränderungen anpassen, die durch Umweltfaktoren hervorgerufen werden, etwa durch die Jahreszeiten. Die Evolution bietet einen gewaltigen Katalog an homöostatischen Lösungen dieser Aufgabe. Der zeitweilige Verlust der Blätter, das Zurücklassen von Sporen, der Winterschlaf und die Metamorphosen der Insekten, das sind nur einige von vielen möglichen Beispielen. Das Dilemma ist jedoch, dass die von der Erbinformation festgelegten Regelungsmechanismen nur jenen Veränderungen gewachsen sind, von denen sie in Tausenden vorausgegangener Generationen herausselektiert wurden. Die Präzision des Instinktverhaltens wird nutzlos, wenn es gilt, neue Aufgaben zu lösen, die von der Gattung noch nicht bewältigt wurden und somit nicht genetisch verankert sind. Die Pflanze, die Bakterie oder das Insekt verfügen somit über Reaktionsweisen auf Veränderungen, die ihnen von Anfang an eingebaut sind. Systemtheoretisch im Sinne dieses Buches gesprochen, ist ein solches System hinsichtlich des Bereichs möglicher Veränderungen, die es um seines eigenen und des Fortbestands der Gattung willen regulativ bewältigen muss, deterministisch veranlagt, d. h. vorprogrammiert. Solche Veränderungen haben meistens rhythmischen Charakter (Wechsel von Tag und Nacht, jahreszeitliche Veränderungen, Wechsel von Ebbe und Flut), zumindest aber einen zeitlich begrenzten Charakter (das Auftauchen eines Raubtieres löst fertige Schutzmechanismen aus: Flucht oder Erstarrung im „Scheintod“). Kommt es zu Veränderungen, die den Organismus durch die Programmierung von unvorhergesehenen Instinkten aus seinem Umweltgleichgewicht reißen, dann erweist sich die Antwort des Systems als unzureichend, und es beginnt eine Krise. Einerseits erhöht sich die Sterblichkeit der unangepassten Organismen gewaltig, und zugleich privilegiert der Selektionsdruck gewisse neue Formen (Mutanten) – was schließlich dazu führen kann, dass für das Überleben unerlässliche Reaktionsweisen dem genetischen Programm einverleibt werden. Andererseits entsteht eine außergewöhnliche Chance für Organismen, die mit einem Gehirn ausgestattet sind, mit einem Teilsystem also, welches nach Bedarf das Handlungsprogramm wechselt kann (Selbstprogrammierung durch Lernen). Ohne sich auf deterministische Interoperationsprogramme zu stützen, passt sich nunmehr der Organismus entweder der veränderten Umwelt (die Ratte lernt, den Ausgang aus dem Labyrinth zu finden) oder die Umwelt sich selbst an (der Mensch errichtet die Zivilisation). Es gibt natürlich auch die dritte Möglichkeit, dass der Organismus verliert, wenn er, nachdem er ein falsches Modell der Situation entwickelt hat, die Anpassung nicht erreicht und untergeht. Mit deterministischen Interoperationsprogrammen ausgestattete Systeme wissen alles von vornherein, solche, die über Lernfähigkeiten verfügen, müssen erst das richtige Verhalten erlernen. Die Annehmlichkeiten der ersten Lösung erkauft sich der Organismus mit seiner Beschränktheit, die der letzteren mit dem Risiko. Im Regelfall ist das Fassungsvermögen für Interoperationsprogramme beschränkt, weshalb die Menge der von vornherein programmierten Handlungen nicht sonderlich groß sein kann. Das Lernen ist dagegen mit einer Zeit der „Novizenschaft“ verbunden, in der das System in hohem Maße



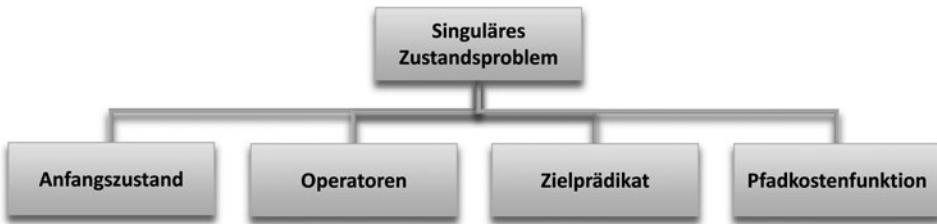
**Abb. 5.16** Problemlösungsprozess

Irrtümern ausgesetzt ist, die ihn rasch sehr viel kosten können, inklusive des Verlusts der Daseinsberechtigung bzw. bei Organismen des Lebens. Gewiss haben sich deshalb in der Tierwelt bis heute jene beiden Haupttypen erhalten. So gibt es dort Milieus, in denen sich ein deterministisches und stereotypes, aber „von der Wiege an gekonntes“ Verhalten besser auszahlt als die Mühe des kostspieligen Lernens aus eigenen Fehlern. Dieser Tatsache zum Trotz neigen viele dazu, die Bedeutung der Intelligenz zu überschätzen und sie als einen „Wert an sich“ zu betrachten. So kann man sich beispielsweise eine Ratte vorstellen, die nicht in der Lage ist zu lernen, wie sie mit größter Vorsicht eine ihr dargebotene Nahrung probiert. Eine „intelligente“ Ratte, die gelernt hat, dass die Nahrung sich immer zur selben Zeit am selben Ort befindet, hat scheinbar eine größere Überlebenschance. Falls aber diese Nahrung giftig ist, überlebt die vorsichtige Ratte, die nichts gelernt hat, dank ihres instinktiven Misstrauens, die „intelligente“ Ratte, die sich vollfrisst, verendet. Demnach privilegiert nicht jede Umwelt eine Intelligenz. Ganz allgemein gesprochen ist die Extrapolation von Erfahrungen in der irdischen Umwelt überaus zweckmäßig. Es sind jedoch auch solche Umwelten denkbar, in denen dieses Merkmal zum Nachteil wird. Folglich ist die Kognogenese eine Möglichkeit, aber keineswegs ein unausweichliches Phänomen; sie ist eine der besten Lösungen, aber nicht immer und nicht für alle Welten die optimale Lösung.

### 5.3.3 Problemlösungsverfahren der systemischen Intelligenz

Menschen als auch artifizielle Systeme, wie Roboter, sehen sich mit dem Umstand konfrontiert, in bestimmten Situationen Probleme lösen zu müssen. Je nach Ausstattung des Systems, wird sich diese Problemlösung unterschiedlich gestalten und damit die Qualität bzw. Werthaltigkeit der Lösung unterschiedlich bemessen sein. Unabhängig davon, strukturiert sich die Problemlösung in einem ersten Ansatz in die Phasen der Zielformulierung, Problemformulierung, Lösungssuche und der Lösungsanwendung (Abb. 5.16).

Die *Zielformulierung*, ausgehend von der aktuellen Situation, ist der erste Schritt beim Problemlösen. Ein Ziel wird als eine Menge von Weltzuständen betrachtet, in denen das Ziel erfüllt ist. Aktionen werden als Übergänge zwischen Weltzuständen betrachtet. Bei der *Problemformulierung* wird entschieden, welche Aktionen und welche Zustände betrachtet werden sollen. Sie folgt unmittelbar auf die Zielformulierung. Hat eine Problemlösungskomponente mehrere mögliche Aktionsalternativen mit unbekanntem Ausgang, dann kann sie eine Entscheidung dadurch herbeiführen, dass sie verschiedene mögliche

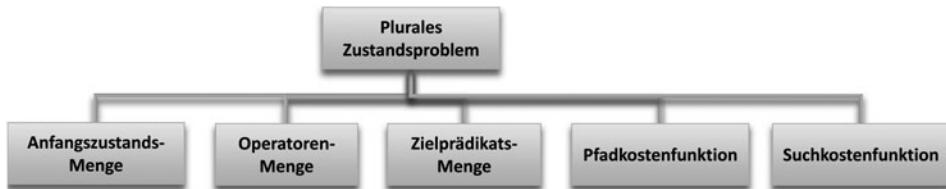


**Abb. 5.17** Singuläres Zustandsproblem

Aktionsalternativen, die zu Zuständen mit bekanntem Wert führen, untersucht, und dann diejenige Alternative wählt, die zu dem Zustand mit dem besten Wert führt. Dieser Vorgang wird als *Lösungssuche* bezeichnet. Ein solcher Suchalgorithmus nimmt eine Problembeschreibung als Eingabe und liefert eine Lösung in Form einer Aktionsfolge. Die Lösung kann dann in der *Ausführungsphase* ausgeführt werden. Bereits bei der Problemformulierung kann man unterschiedliche Problemtypen unterscheiden.

- *Singuläres Zustands-Problem*: Die Problemlösungskomponente weiß, in welchem Zustand sie sich befindet, d. h. die Umgebung ist zugänglich, und sie weiß, was jede Aktion bewirkt. Dann kann sie, ausgehend von ihrem Zustand, für eine Aktionsfolge vorausberechnen, in welchem Zustand sie nach Ausführung der Aktionen sein wird.
- *Plurales Zustands-Problem*: Die Problemlösungskomponente weiß nicht, in welchem Zustand sie sich befindet, d. h. die Umgebung ist unzugänglich oder nur beschränkt zugänglich, aber sie weiß, was jede Aktion bewirkt. Dann kann sie trotzdem das Erreichen eines Zielzustands vorausberechnen. Dazu muss sie aber über eine Menge von Zuständen schlussfolgern, nicht nur über einzelne Zustände.
- *Kontingenzen-Problem*: Die Problemlösungskomponente kann nicht im Voraus eine bestimmte Aktionsfolge berechnen, die zu einem Zielzustand führt, vielmehr benötigt sie während der Ausführung der Aktionen Sensorinformationen um zu entscheiden, welche von mehreren möglichen Aktionen zum Ziel führt. Statt einer Folge von Aktionen braucht sie also einen Baum von Aktionen, in dem jeder Zweig, von der Wurzel bis zu einem Blatt, eine Aktionsfolge darstellt, und jede Aktionsfolge behandelt ein kontingen tes Ereignis.
- *Explorations-Problem*: Die Problemlösungskomponente kennt die Auswirkungen ihrer Aktionen nicht. In diesem Fall muss sie experimentieren (try and error) um herauszufinden, was ihre Aktionen bewirken und welche Art von Zuständen es gibt. Dies kann sie nur in der realen Welt tun. Sie kann aber dabei „lernen“, d. h. sich eine interne Repräsentation der Welt aufbauen und diese für weitere Problemlösungen nutzen.

Zur Modellierung eines *singulären Zustands-Problems* sind folgende Informationen erforderlich: Anfangszustand, Menge möglicher Aktionen, ein Zielprädikat und eine Pfadkostenfunktion (Abb. 5.17).



**Abb. 5.18** Plurales Zustandsproblem

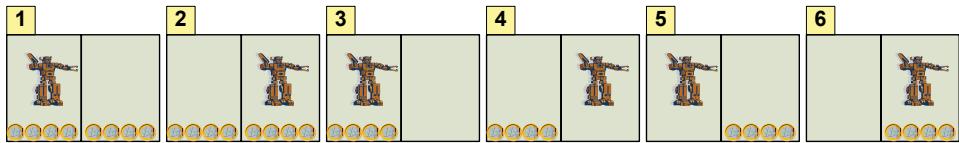
Der Anfangszustand ist der Zustand, von dem die Problemlösungskomponente weiß, dass sie sich in ihm befindet. Eine Aktion wird als *Operator* (alternativ: Nachfolgefunktion) bezeichnet in Verbindung mit dem Zustand, zu dem sie von einem gegebenen Zustand aus führt. Der Anfangszustand und die Menge der Aktionen definieren den *Zustandsraum* des Problems. Er besteht aus der Menge aller Zustände, die vom Anfangszustand aus durch irgendwelche Aktionsfolgen erreichbar sind. Ein *Pfad* im Zustandsraum ist eine Aktionsfolge, die von einem Zustand zu einem anderen führt. Das *Zielprädikat* kann der Problemlöser auf einen Zustand anwenden um zu testen, ob er ein Zielzustand ist. Die *Pfadkostenfunktion* ordnet jedem Pfad im Zustandsraum einen Kostenwert zu. Dieser ist die Summe der Kosten jeder einzelnen Aktion entlang des Pfades. Instanzen dieses Datentyps bilden die Eingaben für Suchalgorithmen. Die Ausgabe eines Suchalgorithmus ist eine Lösung, d. h. ein Pfad vom Anfangszustand zu einem Zustand, der das Zielprädikat erfüllt.

Bei *pluralen Zustands-Problemen* werden die Zustände durch Zustandsmengen ersetzt. Eine Aktion wird durch eine Menge von Operatoren repräsentiert, die die Folgezustandsmenge spezifizieren. Ein Operator wird auf eine Zustandsmenge angewandt, indem er auf jeden einzelnen Zustand in der Menge angewandt wird und die Ergebnisse vereinigt werden. Anstelle des Zustandsraums erhält man einen *Zustandsmengenraum*, und ein Pfad in diesem Raum führt über Zustandsmengen. Eine Lösung ist ein Pfad zu einer Zustandsmenge, die nur aus Zielzuständen besteht (Abb. 5.18).

Die *Performanz* einer Problemlösung wird durch drei Größen gemessen:

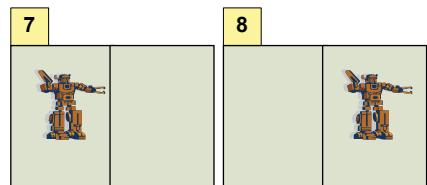
- Wird eine Lösung gefunden?
- Wie hoch sind die Pfadkosten?
- Wie hoch sind die Suchkosten?

Die Gesamtkosten der Problemlösung bilden die Summe aus Pfadkosten und Suchkosten. Gerade bei komplexen Problemstellungen kann man durch ein Weglassen bzw. Ignorieren von nicht erforderlichen Details die Problemmodellierung einfacher gestalten. Dieser Schritt des Weglassens von Details bezeichnet man als *Abstraktion*. Eine solche Abstraktion ist sinnvoll, wenn so viele Details wie möglich weggelassen werden, aber die Gültigkeit des Problemmodells dennoch erhalten bleibt.



**Abb. 5.19** Roboterwelt als Gelddeponien

**Abb. 5.20** Roboterwelt und Zielzustände



Als Beispiel für eine solche zustandsbasierte Problemmodellierung diene eine Roboterwelt, die aus zwei Gelddeponien bestehe. An jeder Deponie kann sich sowohl Geld befinden oder nicht, dort kann auch ein Robotersystem vorhanden sein oder nicht (Abb. 5.19).

In dieser Welt gibt es acht verschiedene Zustände und das Robotersystem kann drei verschiedene Aktionen ausführen: *Links*, *Rechts* und *Geld sammeln*, wobei idealtypisch davon ausgegangen wird, dass die Sammelaktion seitens des Robotersystems einwandfrei funktioniert. D. h. konkret, dass in dieser modellierten Welt keine Systemstörungen zugelassen werden. Das Ziel ist, alles Geld zu sammeln, d. h. das Ziel ist in einem der Zustände 7 und 8 erfüllt (Abb. 5.20).

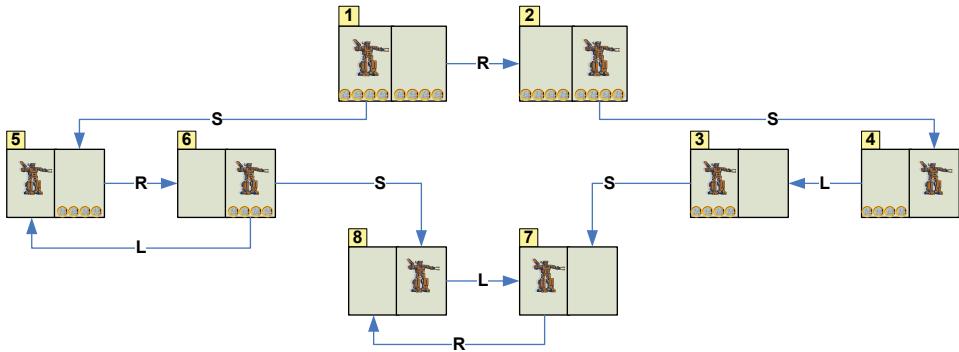
Zunächst wird das Ein-Zustands-Problem mit vollständiger Information und korrekt arbeitendem Sammelroboter betrachtet. Die Problemmodellierung ergibt:

- **Zustände:** Einer der acht Zustände der obigen Abbildungen.
- **Operatoren:** Nach links bewegen (L), nach rechts bewegen (R), Geld sammeln (S).
- **Zielprädikat:** Kein zurückgelassenes Geld an beiden Deponien.
- **Pfadkosten:** Jede Aktion hat die Kosten 1.

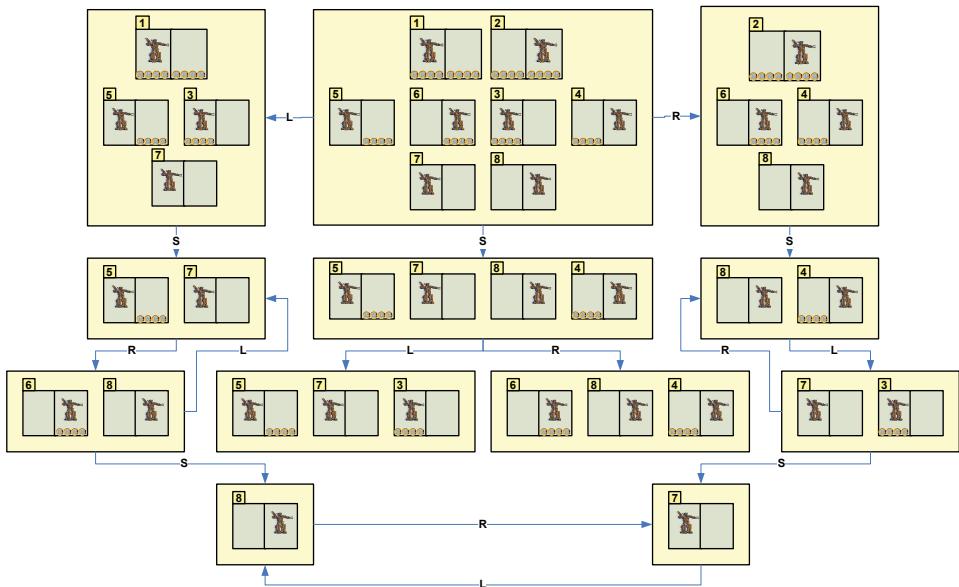
Die folgende Abbildung zeigt den vollständigen Zustandsraum mit allen möglichen Pfeilen (Abb. 5.21).

Nun wird das Mehr-Zustands-Problem betrachtet, bei dem das Robotersystem keinen Sensor hat. Die Problemmodellierung unter diesen Gegebenheiten ergibt nun:

- **Zustandsmengen:** Teilmengen der acht Zustände
- **Operatoren:** Nach links bewegen (L), nach rechts bewegen (R), Geld sammeln (S).
- **Zielprädikat:** Alle Zustände in der Zustandsmenge haben kein Geld an beiden Deponien.
- **Pfadkosten:** Jede Aktion hat die Kosten 1 (Abb. 5.22).



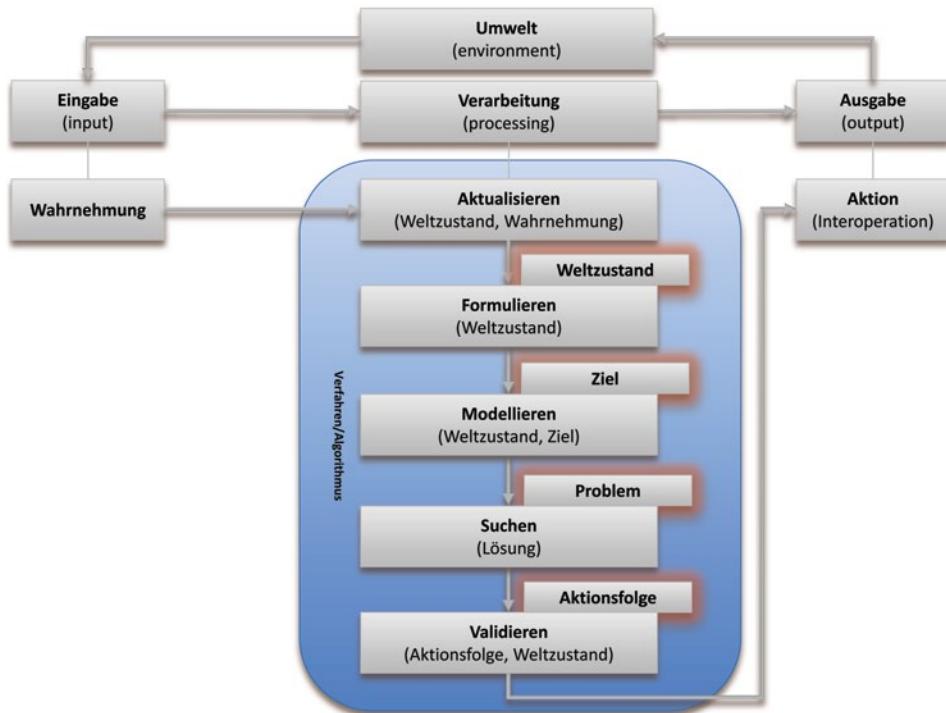
**Abb. 5.21** Roboterwelt und Zielzustandsraum



**Abb. 5.22** Modell eines pluralen Zustandsraumes

Daran lässt sich erkennen, dass der Startzustand durch die Menge aller Zustände gegeben ist, dass das Robotersystem über keinen Sensor verfügt, um seine Position festzustellen.

Im nächsten Abschnitt werden nun *suchorientierte Ansätze* vorgestellt, bei denen aus einer eher zustandsorientierten Sichtweise durch Anwendung eines Operators von einem gegebenen Zustand aus eine Menge anderer Zustände erzeugt werden. Durch diesen Erzeugungsprozess wird die Zustandsmenge sozusagen *expandiert*. Das Wesen der Suche ist, einen Zustand aus einer Menge auszuwählen und die anderen für einen eventuellen

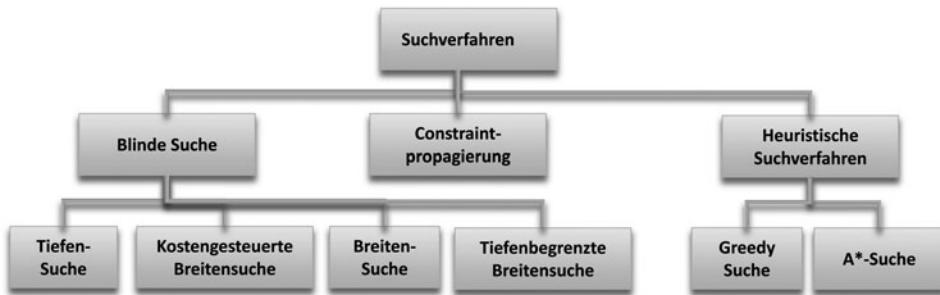


**Abb. 5.23** Lösungsverfahren eines pluralen Zustandsproblems

späteren Gebrauch zurückzustellen, nämlich dann, wenn die getroffene Auswahl nicht zum Erfolg führt. Die Wahl des als nächsten zu expandierenden Zustands wird durch eine *Suchstrategie* bestimmt. Der Suchprozess kann als Aufbau eines *Suchbaums*, der über den Zustandsraum gelegt wird, gedacht werden. Die Wurzel des Suchbaums ist der *Suchknoten*, der dem Anfangszustand entspricht. Die Blätter des Baums entsprechen Zuständen, die keine Nachfolger im Baum haben, entweder weil sie noch nicht expandiert wurden oder weil sie keine Nachfolger haben. In jedem Schritt wählt der Suchalgorithmus einen Blattknoten zum Expandieren aus (Abb. 5.23).

Suchverfahren bilden bis in diese Tage eine der wichtigsten Forschungsgegenstände der Artifiziellen Intelligenz. Sie waren von Anfang an eine Schlüsseltechnologie, um intelligentes Verhalten bei der Problemlösung zu simulieren. Gemäß des handlungsorientierten Ansatzes der Robotik geht es dabei vor allem um die Suche nach möglichen Handlungen und deren Auswirkungen im Hinblick auf ein wohldefiniertes Ziel (Abb. 5.24).

Fast immer existiert dabei eine Auswahl von Handlungsalternativen, die das Robotersystem zu analysieren hat, um dann die vielversprechendste auszuwählen. Der kritische Punkt stellt dabei die richtige Wahl der anzuwendenden Suchstrategie dar. Eine solche Suchstrategie lässt sich anhand der folgenden Kriterien bestimmen:



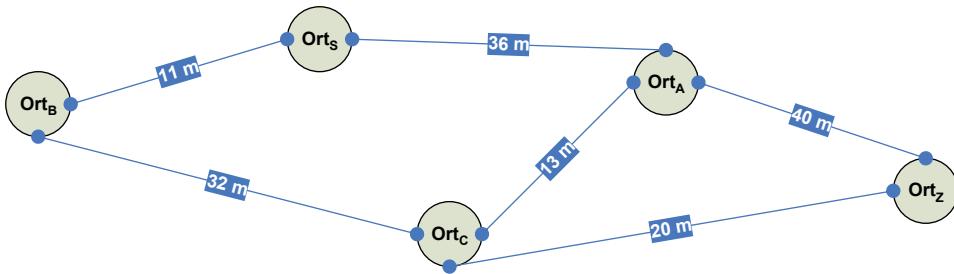
**Abb. 5.24** Suchverfahren

- *Vollständigkeit*: Findet die Suchstrategie garantiert eine Lösung, wenn es eine gibt?
- *Zeitbedarf*: Wieviel Zeit benötigt die Suchstrategie, um eine Lösung zu finden?
- *Speicherplatzbedarf*: Wieviel Speicherplatz benötigt die Suchstrategie für die Suche?
- *Optimalität*: Findet die Suchstrategie die beste Lösung, wenn es mehrere Lösungen gibt?

Wenn Lebewesen Probleme zu lösen versuchen, wägen sie im Allgemeinen zwischen alternativen Vorgehensweisen ab, um dann eine davon anzuwenden. So beispielsweise in Spielsituationen, in denen es darum geht, den Gegner zu schlagen, existieren in der Regel verschiedene Möglichkeiten, um einen Spielzug auszuführen. Der Sinn der meisten Spiele besteht ja gerade darin, in jeder Situation den für die Zielerreichung bestmöglichen Zug zu finden. Dabei muss der gewählte Spielzug für die aktuelle Spielsituation nicht unbedingt die beste Alternative darstellen. Vielmehr ist vielleicht dieser Zug auf das globale Ziel ausgerichtet, das Spiel final zu gewinnen. Eine solche Vorgehensweise lässt sich auch auf andere Gebiete übertragen. So nutzen beispielsweise Mediziner ihr Erfahrungswissen, um aufgrund bestimmter Krankheitssymptome die wahrscheinlichste der möglichen und sinnvollen Diagnosen zu erstellen. Dazu betrachten sie nicht einzelne Symptome, sondern setzen alle Erscheinungen einer Krankheit mit der Reihenfolge ihres Auftretens und ihren Wechselwirkungen in Beziehung. Aber nicht nur in Spezialgebieten finden Faustregeln ihre Anwendung, um Lösungen für bestimmte Probleme zu finden. Auch im täglichen Leben kommen Situationen vor, die es erforderlich machen, aus einer Menge von unübersichtlich vielen Handlungsalternativen eine auszuwählen.

Fahrt von S nach Z: Ein Robotersystem muss einen Weg von einem S (tandort) zu einem Z(iel) zurücklegen. Nunmehr wird angenommen, dass sich alle möglichen Verbindungen zwischen S und Z mit Hilfe einer einfachen Karte darstellen lassen. In dieser Karte sind die einzelnen Verbindungen mit den entsprechenden Entfernungswerten versehen (Abb. 5.25).

Suchprobleme dieser Art lassen sich in ein allgemeines Problemlösungsmodell überführen, um es dort mit der Graphentheorie zu beschreiben. Dazu verwendet man einen so-



**Abb. 5.25** Zustandsgraph

genannten *Zustandsgraphen*. Ein Zustandsgraph besteht aus einer Menge von Knoten und einer Menge von gerichteten Kanten. Die Kanten stellen mögliche Aktionen dar und die Knoten entsprechen den Zuständen. Dieses Modell trägt dem Umstand Rechnung, dass das o. a. Problem sich auszeichnet durch:

- einen definierten Anfangszustand
- einen gewünschten Zielzustand
- mögliche Aktionen zum Übergang von einem zum nächsten Zustand

Der ausgezeichnete Startknoten S repräsentiert eine gegebene Ausgangssituation. Entsprechende Zielknoten korrespondieren mit möglichen Lösungen eines Problems. Jeder Zustand, außer dem Startzustand, entspricht einer Sequenz von Aktionen, dem Weg zum Ziel. Zu beachten ist, dass auch mehrere Wege zu einem Knoten führen können.

Ein Graph besteht aus einer Menge N von Knoten und einer Menge E von Kanten:  $G = (N, E)$ , dabei besteht E aus Paaren von Knoten:  $E \subseteq N \times N$ . Ein Knoten  $k_N$  ( $k$  aus  $N$ ) wiederum ist eine Datenstruktur mit fünf Komponenten:

- Der Zustand des Zustandsraums, dem  $k_N$  entspricht;
- derjenige Knoten im Suchbaum, der  $k_N$  erzeugt hat (der *Vaterknoten* von  $k_N$ );
- der zur Erzeugung von  $k_N$  verwendete Operator;
- die Zahl der Knoten auf dem Pfad von der Wurzel zu  $k_N$  (die *Tiefe* von  $k_N$ );
- die Pfadkosten des Pfades vom Anfangszustand bis zu dem  $k_N$  entsprechenden Zustand.

Solche Karten sind typische Repräsentanten der Zustandsgraphen. Der Startknoten für die spezifische Problemstellung ist Ort S, der Zielknoten entspricht dem Ort Z. Die Menge der möglichen Aktionen bei diesem Problem ist durch die vorgegebenen Verbindungsstücke zwischen den Orten definiert. Dieses klassische Beispiel ist in besonderer Weise dazu geeignet, das Verhalten alternativer Suchstrategien verständlich zu machen und ihre Unterschiede zu verdeutlichen. Dabei wird ein Suchproblem  $S = (Z_{\text{start}}, N, E, Z_{\text{ziel}})$  wie folgt definiert:

Ein Suchproblem  $S = (Z_{\text{start}}, N, E, Z_{\text{ziel}})$  ist definiert durch einen Graphen  $(N, E)$ , wobei  $z_{\text{start}} \in N$  und  $\text{ziel}: N \in \{W, F\}$  gilt. Gesucht wird dann eine Folge von Knoten  $k_0, k_1, \dots, k_n$  mit den Eigenschaften:  $z_{\text{start}} = k_0, (k_i, k_{i+1}) \in E$  und  $Z_{\text{ziel}}(k_n) = W$ .

Ein durch den Graphen explizit dargestellter Zustandsraum enthält also alle möglichen Handlungsfolgen in jeder denkbaren Situation der Problemstellung. Sie entsprechen den Wegen von der Problemstellung bis hin zu Situationen, die entweder zum Ziel geführt, oder aber in einer Sackgasse geendet haben. Zunächst stellt jede Folge von Teilstrecken, die hintereinander gewählt werden kann, eine potentielle Lösung dar. Als unsinnig werden jedoch solche Varianten ausgeschieden, die Schleifen enthalten. So darf beispielsweise die Strecke von Ort  $S$  über die Orte  $A, C$  und  $B$  nicht mehrmals beschritten werden.

Für die folgende Vorab-Algorithmisierung seien die folgenden Variablenvereinbarungen getroffen:

- $K_p$  ist die Liste der potentiellen Kandidaten der Wege, die es zu untersuchen gilt.
- $Z_1$  ist der Ausgangszustand
- $Z_E$  ist eine Zustandsvariable, die angibt, ob ein Ziel erreicht (wahr) oder nicht nicht erreicht (falsch) ist.
- $z$  und  $s$  sind Zustände bzw. die Knoten des Graphen
- $p$  sind die Wege durch den Zustandsraum, wobei ein Weg eine definierte Folge von durch Kanten verbundene Knoten darstellt.

**ALGORITHMUS:** Suche

$K_p := [z_1];$

$Z_E := \text{falsch}$

**WIEDERHOLE**

Wähle aus  $K_p$  einen Weg:  $p := z_1, z_2, \dots, z_m$  aus

$Z_E := \text{ziel}(z_m)$

**WENN NICHT  $Z_E$  DANN TUE**

**BERECHNE** alle Zustände, die von  $z_m$  in einem Schritt erreicht werden können:  $[s_1, \dots, s_k]$

**LÖSCHE**  $p$  aus  $K_p$  heraus

**FÜGE** Wege  $z_1, z_2, \dots, z_m s_1, \dots, z_1, z_2, \dots, z_m s_k$  in  $K_p$  ein

**END-WENN**

**BIS**  $Z_E$

**LIEFERE**  $p$  **ZURÜCK**

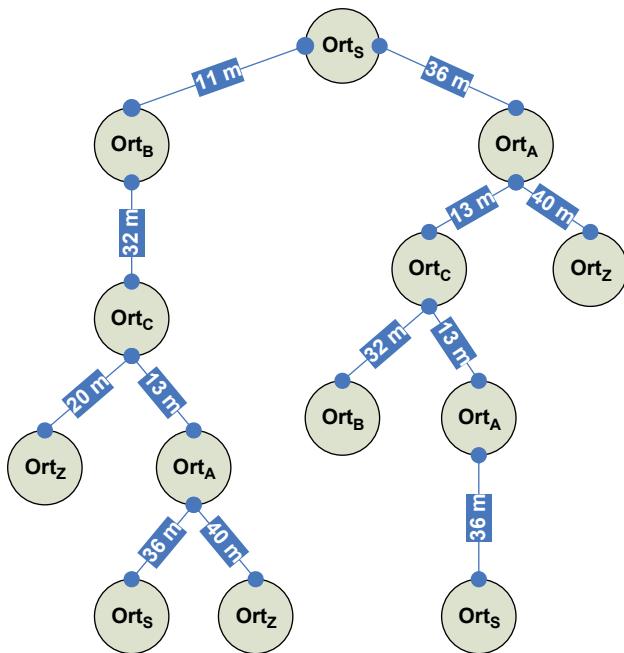
**ENDE-ALGORITHMUS**

In den meisten Fällen gibt es alternative Handlungsfolgen, um das Ziel zu erreichen oder eine Lösung zu finden. Beispielsweise könnte man das Ziel Ort<sub>Z</sub> vom Standort Ort<sub>S</sub> aus über Ort<sub>B</sub> und Ort<sub>C</sub> anfahren oder die Route über Ort<sub>A</sub> und Ort<sub>C</sub> wählen. Da man jedoch immer bestrebt ist, einen möglichst kurzen Weg zu finden, wird es in der Regel nur eine,

nämlich die effizienteste und damit in diesem Fall die beste Lösung geben. Um den Zustandsraum einigermaßen übersichtlich zu halten, wandelt man den Zustandsgraphen in einen *Suchbaum* um. Diese Form der Darstellung wird deshalb Suchbaum genannt, da sie, wenn auch auf den Kopf gestellt, wie ein Baum nach oben hin eine zunehmende Verästelung aufweist. An der Baumwurzel ist demnach der Ausgangszustand des Problems dargestellt, die Enden der äußeren Zweige, die Blätter, repräsentieren entweder Lösungen für das Problem, oder aber Zustände, die Sackgassen darstellen. Im Unterschied zu Zustandsgraphen ist der Baum gerichtet, d. h. die miteinander verbundenen Knoten stehen als Vorgänger und Nachfolger in einer eindeutigen Beziehung zueinander. Diese Beziehungen werden durch Kanten dargestellt, die den Schritten entsprechen, die das System durchläuft, um zu diesen Knoten zu gelangen.

In Bezug auf die Robotik kann man diese Suchbäume auf auch die Suche nach einer der Problemstellung adäquaten Handlungsfolge anwenden. Bei dieser Umwandlung von einem Zustandsgraphen in einen Suchbaum von Handlungen, handelt es sich erneut um ein Gedankenexperiment mit dem Ziel eines kognitiven Modells. Dabei werden als Zustände die einzelnen Situationen abgebildet, in denen jeweils bestimmte Handlungen möglich sind. Man bezeichnet diese Zustände dann auch als *Konsequenzen*. Damit wird hervorgehoben, dass die Situationen als Ergebnisse des Handelns angesehen werden. Dies gilt auch für die im Handlungsfeld als allererste berücksichtigte Situation. Sie ist Ergebnis noch weiter davorliegender Handlungen. Die Konsequenzen werden *Zeitpunkten*  $t_1, t_2, \dots, t_n$  zugeordnet. Dies ist nicht im zeitlichen Sinne gemeint. Konsequenzen des „gleichen Zeitpunktes“ sind darin „gleich“, dass sie alternative Ergebnisse der Handlungen der vorangegangenen Konsequenz sind. Das dem Feld unterlegte „Zeitraster“ ist nur modellhaft gemeint. Den Handlungen sind anstelle Kosten sogenannte *Wirkwahrscheinlichkeiten* zugeordnet, die durch die Kanten in Form von „Handlungspfeilen“ repräsentiert werden. Sie geben an, mit welchen Wahrscheinlichkeiten jeweils welche Konsequenz des folgenden Zeitpunkts erreicht würde, wenn die Handlung ausgeführt werden würde. Um die Umwandlung zu vereinfachen, geht man davon aus, dass der *Beginn* einer Handlung dadurch gekennzeichnet ist, dass an diesen Stellen der Aktivitätsfluss aufgrund einer Entscheidung auch in eine andere Richtung hätte fortgesetzt werden können. Für das *Ende* einer Handlung gilt entsprechend: es ist die Stelle im Aktivitätsfluss, an der wiederum die Möglichkeit besteht zu entscheiden, den Aktivitätsfluss in eine andere Richtung zu lenken. Insofern sind nur solche „möglichen“ Entscheidungen zu beachten, die der Handelnde bzw. das Robotersystem *vor* Beginn seiner Aktivität als (auf dem Weg) mögliche vorgesehen hat. Ganz bewusst werden damit nicht alle zukünftig möglichen Handlungen und Konsequenzen berücksichtigt. Vielmehr ist es das Ziel selbst, aufgrund dessen viele bzw. die meisten der potentiell möglichen Handlungen außer Betracht bleiben. Gerade die Ergebnisse aus der Forschung der Künstlichen Intelligenz haben gezeigt, dass dies offenbar eine außerordentlich hoch zu bewertende Fähigkeit des Menschen ist, für bestimmte Ziele eben nicht alle potentiell möglichen Handlungsmöglichkeiten in Betracht zu ziehen (Abb. 5.26).

Jede Strategie, diesen Baum zu durchsuchen, ist mit einer bestimmten Reihenfolge verbunden, in der die Knoten aufgesucht werden. Der Suchraum muss im Allgemeinen komplett durchsucht werden. Wird in einem Suchraum jeder mögliche Zustandsübergang

**Abb. 5.26** Suchbaum

untersucht, so bezeichnet man dies als *vollständige Suche*. Verfahren, die eine vollständige Suche betreiben, sind etwa die Tiefensuche (engl. depth first search) oder die Breitensuche (engl. breadth first search) (Abb. 5.27).

Beide Verfahren arbeiten den Suchbaum nach einer bestimmten Systematik ab, beispielsweise von links nach rechts oder von oben nach unten. Während die Tiefensuche möglichst schnell sehr tief in den Baum vorzudringen versucht, bearbeitet die Breitensuche zunächst alle Knoten einer Ebene des Suchbaumes, bevor sie deren Nachfolger untersucht.

**ALGORITHMUS:** Tiefensuche

$K_p := [z_1]$ ;

$Z_E := \text{falsch}$

**WIEDERHOLE**

Sei  $K_p = [z_1, z_2 \dots z_n, p_2, p_3, \dots, p_n]$

$p := z_1, z_2 \dots z_m$

$Z_E := \text{ziel}(z_m)$

**WENN NICHT  $Z_E$  DANN TUE**

Berechne alle Zustände, die von  $z_m$  in einem Schritt errechnet werden können:  $[s_1, \dots, s_k]$

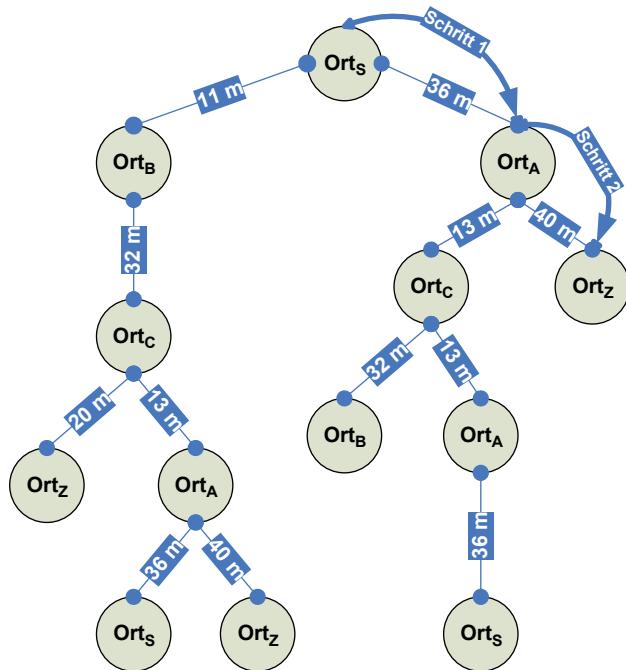
$K_p := [z_1, z_2 \dots z_n, s_1, \dots, z_1, z_2 \dots z_n, s_k, p_2, p_3, \dots, p_k]$

**END-WENN**

**BIS  $Z_E$**

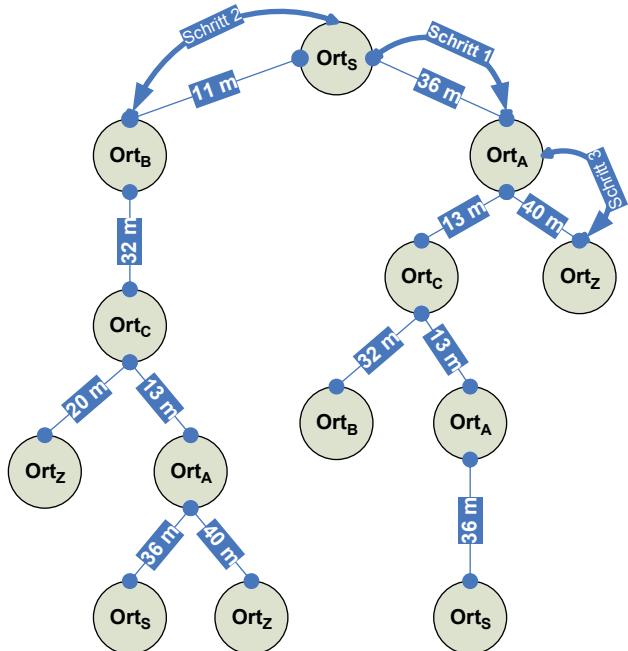
**LIEFERE  $p$  ZURÜCK**

**ENDE-ALGORITHMUS**

**Abb. 5.27** Tiefensuche

Bei der *Tiefensuche* ergibt sich eine Abarbeitungsreihenfolge der Baumknoten von *Ort<sub>S</sub>*  $\Rightarrow$  *Ort<sub>A</sub>*  $\Rightarrow$  *Ort<sub>Z</sub>* und bei der *Breitensuche* von *Ort<sub>S</sub>*  $\Rightarrow$  *Ort<sub>A</sub>*  $\Rightarrow$  *Ort<sub>B</sub>*  $\Rightarrow$  *Ort<sub>Z</sub>*. Damit sind eindeutige Fahrtrouten definiert.

Beide Verfahren, Tiefensuche wie auch Breitensuche, würden von *Ort<sub>S</sub>* aus über *Ort<sub>A</sub>* nach *Ort<sub>Z</sub>* fahren und eine Strecke von 76 m zurücklegen. Bei Anwendung der Tiefensuche würde man zwei, bei der Breitensuche allerdings drei Suchschritte benötigen. Zu beachten ist, dass beide exemplarisch angezeigten Strategien die Entfernungsaangaben der einzelnen Abschnitte bei ihrer Vorgehensweise in keiner Weise berücksichtigen. Bei einer *kostengesteuerten Breitensuche* wird immer der Knoten mit den niedrigsten Pfadkosten als nächster expandiert, weshalb unter bestimmten Voraussetzungen die kostengünstigste Lösung gefunden wird. Bei einer *tiefenbegrenzten Suche* wird die Tiefe des Suchraumes beschnitten. Jeder Pfad im Suchbaum wird höchstens bis zu einer vorgegebenen Tiefe entwickelt. Dass die Tiefensuche in diesem Fall bereits nach zwei Schritten eine Lösung findet, ist reiner Zufall. Wäre der Baum an der Wurzel horizontal gespiegelt, so würden beide Strategien andere Ergebnisse erzielen. Insofern wird der Zustandsraum bei beiden Verfahren sozusagen „blind“ abgearbeitet und zwar solange, bis ein Zielzustand erreicht ist. Das garantiert zwar in jedem Fall das Auffinden einer Lösung, die Effizienz des Verfahrens wird jedoch offensichtlich sehr stark von der Gestalt des Suchraums beeinflusst. Im schlechtesten Fall kann es vorkommen, dass der gesamte Suchbaum durchsucht werden muss, bevor eine erste Lösung gefunden wird. Durch diese *Blindheit* bzw. Orientierungslosigkeit der vollständigen Suche wird oft sehr viel unnötige Zeit damit verbracht, in den zur Lösungsfundung

**Abb. 5.28** Tiefensuche

eigentlich irrelevanten Teilen des Suchbaumes zu suchen. Insofern eignet sich die vollständige Suche vor allem für Probleme mit überschaubaren Zustandsräumen, ist jedoch i. d. R. aufgrund der Größenordnung, die Zustandsräume bei praktisch zu lösenden Problemen annehmen können, völlig ungeeignet (Abb. 5.28).

Im Allgemeinen hängt die Effizienz von Lösungsansätzen im Wesentlichen davon ab, den Zustandsraum auf eine sinnvolle Größe einzuschränken.

**ALGORITHMUS:** Breitensuche

 $K_p := [z_1];$ 
 $Z_E := \text{falsch}$ 
**WIEDERHOLE**

 Sei  $K_p = [z_1, z_2 \dots z_n, p_2, p_3, \dots, p_n]$ 
 $p := z_1, z_2 \dots z_m$ 
 $K_p := \text{ziel}(z_m)$ 
**WENN NICHT  $Z_E$  DANN TUE**

 Berechne alle Zustände, die von  $z_m$  in einem Schritt errechnet werden können:  $[s_1, \dots, s_k]$ 
 $K_p := [p_2, p_3, \dots, p_k, z_1 z_2 \dots z_n s_1, \dots, z_1, z_2 \dots z_n s_k,]$ 
**END-WENN**
**BIS  $Z_E$** 
**LIEFERE  $p$  ZURÜCK**
**ENDE-ALGORITHMUS**

Beide Verfahren zeichnen sich durch ihre inhärenten Vor- und Nachteile aus: Die wichtigen Eigenschaften zur Beurteilung solcher Vor- und Nachteile sind:

- Terminiertheit
- Korrektheit
- Komplexität

Ein Suchverfahren heißt *terminiert*, wenn es stets eine Lösung findet, vorausgesetzt, es existiert auch eine Lösung. Tiefensuche kann nicht-terminiert sein. Ist im Suchraum ein unendlich tiefer Ast enthalten, so läuft die Tiefensuche irgendwann in diesen Bereich hinein und verlässt ihn nie wieder, bleibt sozusagen in diesen Unendlichkeiten gefangen. Ist der Suchraum hingegen endlich, dann ist auch die Tiefensuche terminiert. Breitensuche ist terminiert. Das liegt darin begründet, dass die Breitensuche den Suchraum derart durchläuft, dass jeder Knoten innerhalb endlicher Zeit erreicht wird. Diese Eigenschaft bezeichnet man auch als Fairness eines Suchverfahrens. Insofern hat die Breitensuche auch in unendlich tiefen Suchräumen keine Probleme. Ein Suchverfahren heißt *korrekt*, wenn ein vom Suchverfahren vorgeschlagener Weg tatsächlich eine Lösung für das Problem darstellt.

Im Allgemeinen haben Probleme mehrere Lösungen. Daher ist oftmals die Frage nach der Qualität einer gefundenen Lösung nicht immer sinnvoll zu beantworten. Eine mögliche Definition der Qualität könnte über die Zahl der benötigten Schritte erfolgen. Legt man dieses Kriterium der Beurteilung der beiden Verfahren zugrunde, so zeigt sich, dass die Breitensuche immer die kürzeste Lösung liefert, da der Suchraum Ebene für Ebene durchsucht wird. Allerdings kann dafür die Tiefensuche eine Lösung sehr schnell finden. Richtet man den Grad der Komplexität an dem benötigten Speicherbedarf aus, dann stellt man zunächst fest, dass dieser Speicherbereich durch die Größe der Liste der möglichen Wege bestimmt wird. Die Tiefensuche hat bezüglich einer solch aufgefassten Komplexität gegenüber der Breitensuche Vorteile. Die ermittelten Vorteile für die Tiefensuche (Komplexität) und die Vorteile der Breitensuche (Fairness, Terminiertheit) lassen sich kombinieren. Das Verfahren, dass daraus entsteht, führt eine Tiefensuche durch, wobei aber die Tiefe der Kandidaten beschränkt wird, d. h. man erlaubt nur das Expandieren von Kandidaten bis zu einer bestimmten Tiefe. Wird dabei keine Lösung gefunden, lockert man die Tiefeneinschränkung, man erlaubt eine tiefere Suche. Dieses Verfahren wird als schrittweise Vertiefung (iterative Deeping) bezeichnet. Es zeigt ein zeitliches Verhalten wie die Breitensuche und bezüglich des Speicherbedarfs verhält es sich wie die Tiefensuche.

Es gilt noch ein weiteres Suchverfahren zu nennen, das ebenfalls die Vorteile der Tiefen- und Breitensuche kombiniert. Bei der *bidirektionalen Suche* wird simultan vorwärts vom Anfangszustand aus und rückwärts vom Zielzustand aus gesucht, bis sich die beiden Suchen in der Mitte treffen. Ein sinnvoller Einsatz der bidirektionalen Suche ist allerdings an gewisse Anforderungen geknüpft:

**Tab. 5.1** Suchverfahren

Kriterium	Breiten-suche	Kosten-ge-steuerte Breiten-suche	Tiefen-suche	Tiefen-begrenzte Suchen	Bidirektionales Suche
Zeit	$f^t$	$f^t$	$f^m$	$f^l$	$f^{t/2}$
Speicherplatz	$f^t$	$f^t$	$f^*m$	$f^*l$	$f^{t/2}$
Optimal	Ja	Ja	Nein	Nein	Ja
Vollständig	ja	Ja	Nein	Ja, falls $l > t$	Ja

- Es muss möglich sein, die Vorgänger eines Knotens zu erzeugen. Diese sind für die Rückwärtssuche die zu erzeugenden Nachfolger. Die Vorgänger eines Knotens  $n$  sind alle Knoten, die  $n$  als Nachfolger haben.
- Alle Operatoren müssen umkehrbar sein. Dann sind die Menge der Vorgänger und die Menge der Nachfolger identisch. Das Berechnen der Vorgänger kann aber schwierig sein.
- Wenn es mehrere Zielknoten gibt, muss die Rückwärtssuche im Sinne eines Mehr-Zustands-Problems durchführbar sein. Dies ist immer möglich, wenn die Zielzustände explizit gegeben sind. Gibt es nur eine Beschreibung des Zielknotens durch ein Zielprädicat, dann wird die Definition von Zuständen schwierig.
- Die Prüfung, ob ein neu generierter Knoten bereits im anderen Teil des Suchverfahrens vorkommt, muss effizient möglich sein.
- Es muss möglich sein zu entscheiden, welche Art der Suche in beiden Richtungen ablaufen soll.

Um die einzelnen Suchverfahren vergleichen zu können, wird ein hypothetischer Zustandsraum angenommen, in dem jeder Zustand zu genau  $f$  Folgezuständen expandiert werden muss. Diese Anzahl  $f$  wird als *Verzweigungsfaktor* der Zustände bezeichnet. Die Anzahl der insgesamt erzeugten Knoten des Suchbaums ist ein Maß für den Zeit- und Speicherbedarf. In der Tiefe 0 gibt es dabei einen Knoten (die Wurzel), in der Tiefe 1  $f$  Knoten, in der Tiefe 2  $f^2$  Knoten usw. und damit allgemein in der Tiefe  $t$   $f^t$  Knoten. Liegt eine erste Lösung in der Tiefe  $l$ , dann ist die Zahl der maximal zu erzeugenden Knoten:

$$1 + f + f^2 + f^3 + \dots + b^t = (K(f^t))$$

Dabei gilt der Zusammenhang, dass Zeit- und Speicherbedarf komplexitätsmäßig gleich groß sind ( $K(f)$ ), denn im Worst Case (ungünstigster Fall) müssen alle Knoten der Tiefe  $l$  gespeichert werden, im günstigeren Fall immerhin die Knoten der Tiefe  $l-1$ . Insgesamt ergibt sich also für die einzelnen Suchverfahren folgende Tabelle (Tab. 5.1):

Das Verfahren der *Constraintpropagierung* lässt sich an einem einfachen Zuordnungsproblem deutlichen. Ein typisches Zuordnungsproblem stellt das Acht-Damen-Problem dar: Acht Damen sollen auf einem Schachbrett so positioniert werden, dass sie sich nicht gegenseitig schlagen können. Solche Zuordnungsprobleme sind dadurch ausgezeichnet,

dass bestimmten Variablen Werte zugeordnet werden müssen. Die Zuordnung unterliegt aber bestimmten Restriktionen oder Beschränkungen (*Constraints*). Gemäß geltender Schachregeln muss eine Lösung für das Problem diese Restriktionen erfüllen:

- In jeder Zeile muss exakt eine Dame stehen.
- In jeder Spalte muss exakt eine Dame stehen.
- Pro Diagonale darf höchstens eine Dame positioniert werden.

Insofern werden bei dieser Problemstellung die Zustände nicht erst beim Erreichen hinsichtlich der gewünschten Anforderungen überprüft, sondern die Anforderungen an die gewünschte Lösung schon als Wissen in den Suchprozess mit einbezogen. Durch die Berücksichtigung zusätzlicher Anforderungen kann die Menge der alternativen Suchwege im Suchraum noch weiter eingeschränkt werden. Dabei ist zu beachten, dass die Wahl einer einzigen Alternative oft die Möglichkeit bietet, weitere Alternativen einzuschränken, und deren Einschränkung führt wiederum zur Einschränkung anderer. Diese sich fortpflanzende Einschränkung bezeichnet man als *Propagierung*. Um Zielanforderungen in den Suchprozess mit einzubeziehen, muss das entsprechende Wissen einerseits formalisiert und entsprechend repräsentiert werden. Andererseits müssen Mechanismen bereitgestellt werden, die aus dem explizit vorhandenen Wissen (durch Streichung von Alternativen) neues implizites Wissen inferieren können.

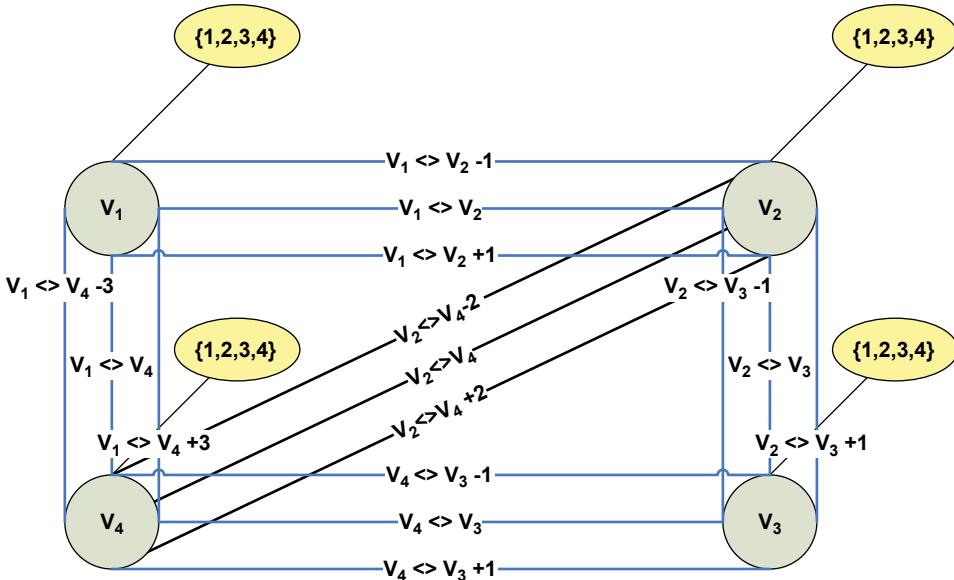
In einem Constraint Satisfaction Problem sind die Zustände durch die Werte einer wohldefinierten Menge von Variablen definiert und das Ziel ist durch eine Reihe von Constraints formuliert, die diese Variablenwerte erfüllen müssen. Dabei existieren verschiedene Typen von Constraints. Da sie praktisch Relationen zwischen Variablen darstellen, können sie nach ihrer *Stelligkeit* in ein-, zwei-, drei- usw. stellige Constraints unterteilt werden. Sie können *hart* oder *weich* sein. Jede Variable  $V_i$  hat einen Wertebereich  $D_i$ , eine Menge möglicher Werte für die Variable. Der Wertebereich kann kontinuierlich oder diskret sein. Ein einstelliges Constraint spezifiziert die zulässige Teilmenge des Wertebereichs, ein zweistelliges Constraint spezifiziert die zulässige Teilmenge des kartesischen Produkts zweier Wertebereiche. In diskreten Wertebereichen können Constraints einfach durch Aufzählung der zulässigen Werte oder Wertetupel definiert werden. Zunächst werden also die vier Damen durch die Variablen  $V_1, V_2, V_3$  und  $V_4$  beschrieben. Dann kann zunächst die Voraussetzung gelten, dass je eine Dame einer Spalte zugeordnet sein muss, um nicht auf einem Feld zu stehen, von dem aus sie mit einem Zug eine der anderen Damen schlagen kann. Die Lösung besteht darin, den vier Variablen solche Werte zuzuordnen, die entsprechend den Reihen des Schachfeldes mit einer Zahl zwischen 1 und 4 numeriert sind. Zu Beginn der Aufgabe kommt die gesamte Wertemenge der möglichen Reihen  $\{1,2,3,4\}$  als Belegung der Variablen in Frage. Nun muss gefordert werden, dass sich jede der Damen in einer anderen Reihe befinden muss. Dies kann durch entsprechende Beschränkungsrelationen für jede der Damen ausgedrückt werden. So gilt beispielsweise für  $V_1: V_1 \neq V_2, V_1 \neq V_3$  sowie  $V_1 \neq V_4$ . Ebenso lässt sich für die beiden Diagonalen, die vom Standort von  $V_1$  ausgehen, Relationen erstellen, welche die Position einer Dame gegenüber den Positionen der ande-

ren Damen einschränken. Durch diese Überlegungen ergeben sich weitere Beschränkungen für  $V_1$ :  $V_1 <> V_2 + 1$ ,  $V_1 <> V_3 + 2$ ,  $V_1 <> V_4 + 3$ ,  $V_1 <> V_2 - 1$ ,  $V_1 <> V_3 - 2$ ,  $V_1 <> V_4 - 3$ .

Alle diese Anforderungen in Form von Beschränkungsrelationen lassen sich mit einem Constraintgraphen zusammenfassend darstellen. Ein solcher Constraintgraph besteht aus einer Menge von Knoten und Kanten. Die Knoten entsprechen Variablen, also in diesem Fall den Damen, und die Kanten repräsentieren die Beschränkungen oder Constraints, die zwischen den Variablen bestehen. Die folgende Darstellung entspricht dem Ausgangszustand gemäß der Aufgabenstellung. Die Knoten enthalten in diesem initialen Zustand den gesamten Wertebereich als mögliche Belegungen für die Variablen. Zu beachten ist, dass das Vier-Damen-Beispiel lediglich die Modellierung zweistelliger Beschränkungsrelationen erfordert. Jedoch ist diese Technik hinsichtlich der Anzahl der in die Relation einbezogenen Variablen (Stelligkeit) theoretisch nicht eingeschränkt. Die Frage, wann die Beschränkungsrelationen sinnvoll angewendet werden können, ist einfach zu beantworten: Immer, wenn auf einer Seite des Ungleichheitszeichens ein einzelner Wert (Singleton) zu finden ist, kann man aus der anderen Variablen diesen Wert herausstreichen. Zu Beginn des Suchprozesses wird eine der Variablen ausgewählt und ihr ein Wert aus dem Wertebereich zugewiesen. Dazu wird üblicherweise eine Heuristik angewendet. Beispielsweise wählt man diejenige Variable aus, die den kleinsten Wertebereich aufweist. Die Idee, die hinter dieser Vorgehensweise steckt, ist die, dass ein kleiner Wertebereich eine Abhängigkeit von vielen anderen Variablen anzeigen vermag. Außerdem wird durch die Wahl eines kleinen Wertebereichs die Menge der Alternativen im Zustandsraum minimiert. Falls mehrere Variablen als Alternativen für diese Auswahl in Betracht kommen, wird von ihnen diejenige Variable ausgewählt, die mit den meisten Beschränkungsrelationen verbunden ist (First-fail principle) und somit potentiell viele Propagierungen auslöst.

Im obigen Beispiel wird  $V_1$  also mit dem Wert 1 instanziert. Die Wahl dieser Belegung führt alsdann zu der Propagierung von Einschränkungen im Wertebereich anderer Variablen. Das bedeutet im konkreten Fall, dass auch Werte anderer Variablen, die an demselben Constraint beteiligt sind, das Constraint nicht mehr erfüllen. Sie werden daher weggelassen oder ignoriert. Dieser Prozess pflanzt sich über alle Constraints fort, deshalb bezeichnet man ihn auch als *Constraintpropagierung* (Constraint propagation). Durch diese Propagierung der Belegung der Variablen  $V_1$  mit dem Wert 1 werden aus dem Wertebereich von  $V_2$  die Werte 1 und 2 gestrichen, aus dem Wertebereich von  $V_3$  die Werte 1 und 3 und aus dem Wertebereich von  $V_4$  die Werte 1 und 4. Im nächsten Schritt wird  $V_2$  mit dem Wert 3 belegt. Durch die Propagierung ergibt sich bei  $V_3$  ein leerer Wertebereich. Da dies ein Widerspruch darstellt, muss die Suche einen Schritt zurückgehen, d. h. ein Backtracking durchführen. Dadurch erhält man den modifizierten Constraintgraph (Abb. 5.29):

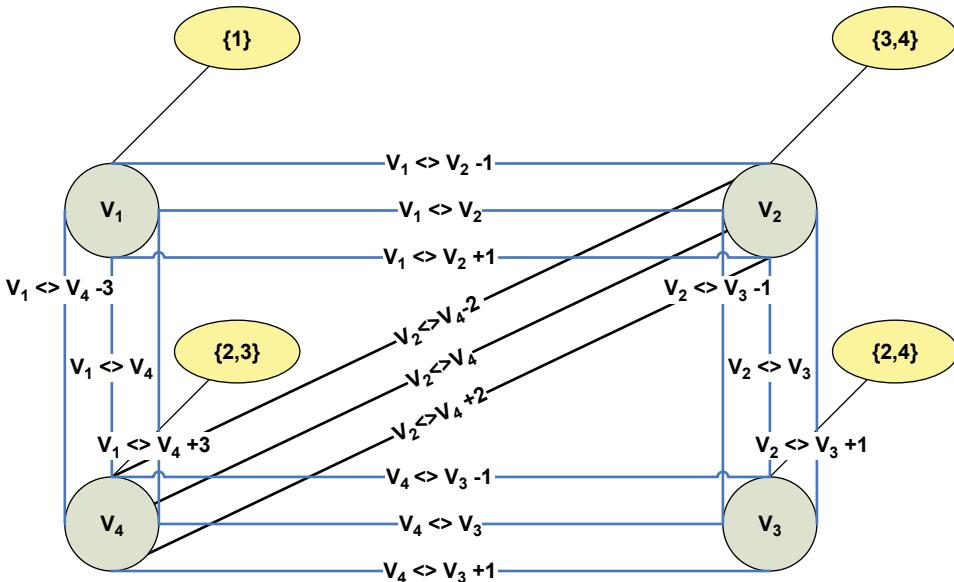
Während der Propagierung kommen Beschränkungsrelationen zur Ausführung, was man ähnlich wie bei Regeln als „Feuern“ von Relationen bezeichnet. Durch die erste Wertbelegung einer Variablen und anschließendem constraint propagation verändert sich der Constraintgraph. Operational besteht die Schwierigkeit darin, nicht nur den Graphen aufzubauen und die Propagierungen zu verwalten, sondern auch die „abgefeuerten“ Beschränkungsrelationen durch das Backtracking sozusagen zu reanimieren. Durch die ab-



**Abb. 5.29** Constraintgraph für das Vier-Damen-Problem

wechselnde Ausführung von Propagierung, Backtracking und Werteberechnungen gelangt man schließlich zu einer Lösung. Aufgrund der Einbeziehung von Anforderungswissen in Form von Beschränkungsrelationen braucht nur ein kleiner Teil des Zustandsraums durchsucht zu werden. Besonders erwähnenswert ist dabei die sehr zielgerichtete Vorgehensweise der Suche. Dies liegt vor allem in den Propagierungen der Werte und Einschränkungen begründet. Ist einmal ein Weg eingeschlagen und damit eine Werteberechnung vorgenommen, so vermeidet die Propagierung eine Verzweigung im Suchbaum. Vielmehr erfolgt eine solche Verzweigung nur dann, wenn Backtrackingschritte ein Zurücksetzen der Suche erforderlich machen. Typisch für den Einsatz von Constraints ist, dass man beim Erreichen von Zuständen mit unmöglichen Sachverhalten den Baum nicht weiter durchlaufen muss, sondern durch Zurücksetzen (Backtracking) den Zustandsraum in erheblichem Maße einschränken kann (Abb. 5.30).

Blinde Suchverfahren, wie beispielsweise Tiefen- und Breitensuche, scheitern häufig gerade bei komplexeren Suchproblemen an den damit verbundenen großen Zustandsräumen. Oftmals werden aber auch zur Problemlösung zusätzliche Kriterien oder Heuristiken benötigt, die den Suchraum strukturieren oder eingrenzen. So bekommt man bei vielen Problemstellungen Zusatzinformationen, wie Entfernung, Kosten, Materialverbrauch etc. quasi mit den Problem- bzw. Fragestellungen mitgeliefert. Solche problem- und damit lösungsrelevanten Informationen werden bei der Breiten- und Tiefensuche jedoch nicht berücksichtigt. Komplexe Problemstellungen werden von Lebewesen auch durch die Anwendung effizienter Suchstrategien, den sogenannten Heuristiken, gelöst. So benutzen Menschen dazu Faustregeln, welche sie intuitiv oder gemäß ihrem Erfahrungswissen an-



**Abb. 5.30** Veränderter Constraintgraph

wenden, um die beste Lösung für eine bestimmte Problemsituation zu finden. Jeder Mediziner unterzieht einen Patienten nur solchen Tests, die ihm aufgrund der auftretenden Symptome als sinnvoll erscheinen, und der Autofahrer, der möglichst schnell von München nach Hamburg gelangen möchte, wird dazu bestimmt zunächst keine Waldwege, sondern sicherlich Autobahnen benutzen. In diesem Abschnitt werden daher Suchverfahren vorgestellt, die Zusatzinformationen über Problem- oder Fragestellungen mitberücksichtigen.

Der Begriff *Heuristik* stammt dabei aus dem Griechischen und bedeutet etwa „finden“, „entdecken“. Umgangssprachlich bezeichnet man mit Heuristiken Faustregeln, die Systemen als Entscheidungshilfe dienen, um situativ diejenigen Aktionen im Zustandsraum auszuführen, die anscheinend am ehesten zu einer Lösung führen. Die Begrenzung des Suchraums auf eine sinnvolle Größenordnung beeinflusst die Effizienz solcher Programme beträchtlich. Heuristiken werden im allgemeinen dann eingesetzt, wenn ein Problem keine exakte Lösung liefert, wie etwa in der medizinischen Diagnostik, oder bei Problemen, für die zwar eine exakte Lösung existiert, die Größe des Zustandsraums aber ein effizientes Auffinden verhindert. Die Anwendung von Heuristiken zur Problemlösung in Zustandsräumen bezeichnet man dann als *heuristische Suche*. Heuristiken erlauben es, den gesamten Zustandsraum auf den Teilbereich zu beschränken, der für die Zielerreichung sinnvolle Handlungsketten beinhaltet.

Während das Strukturieren des Suchraums bei der Tiefen- und Breitensuche noch auf eine Modifikation der Reihenfolge der Abarbeitung der Kandidaten hinauslief, impliziert eine Eingrenzung des Suchraums, dass die Zweige im Suchraum abgeschnitten werden.

Dazu verwenden Heuristiken i. a. eine Evaluierungsfunktion, beispielsweise in der Ausprägung einer Kostenfunktion, welche eine Bewertung des aktuellen Zustands und der alternativen Folgezustände im Suchraum ermöglicht. Folglich kann man jedem Zustand im Zustandsraum eine reelle Zahl zuordnen, die entweder seinen Nutzen oder die mit seiner Erreichung verbundenen Kosten abschätzt.

Bei einer Kostenbetrachtung der Fahrt von S nach Z werden alle in der angenommenen, aktuellen Situation A erreichbaren Folgezustände B, C, D, E in eine geordnete Menge abgebildet:  $f_{\text{kosten}}(D) > f_{\text{kosten}}(B) > f_{\text{kosten}}(E) > f_{\text{kosten}}(C)$ . Die Ordnung gibt die Reihenfolge vor, in der die Knoten abgearbeitet werden sollten.

Die Wahl der anzuwendenden Kostenfunktion hängt von der jeweiligen Problemstellung selbst ab.

Bei der Fragestellung, wie man am schnellsten auf den Gipfel eines Berges gelangt, können zwei Aspekte in eine Kostenfunktion unmittelbar einfließen: die Höhe des momentanen Standortes und der Höhengewinn, welcher mit dem nächsten Schritt unmittelbar verbunden ist.

Für die meisten Problemstellungen existieren oftmals gleich mehrere Kostenfunktionen, die unterschiedliche Wirkung intendieren und sich dadurch auch durch eine Qualität auszeichnen. Generell spricht man von einer heuristischen Funktion und bezeichnet diese dann entsprechend der Information, die in sie eingeflossen ist. So besteht eine Möglichkeit darin, zu schätzen, wie weit man noch von einer akzeptablen Lösung entfernt ist. Zur Steuerung der Suche kann dann eine *heuristische Funktion h* eingesetzt werden, die folgendes steuert: Je besser die heuristische h-Bewertung eines Kandidaten, desto näher liegt der Kandidat an einer Lösung. Diese heuristische Funktion *h* bietet allerdings keine Garantie für eine optimale Suche. Sie stellt daher nur eine *Evaluierungs- oder Schätzfunktion* dar, denn die exakte Entfernung zum Ziel ist nicht immer ermittelbar. Eine weitere Möglichkeit besteht darin, dass man den bisher aufgebrachten Aufwand berücksichtigt. Hier bewertet man mit der heuristischen Funktion *g* nicht, wie weit man vom Ziel noch entfernt ist, sondern den Aufwand, der zum Erreichen der Zwischenlösung erforderlich war. Jedoch muss an dieser Stelle erwähnt werden, dass nicht immer die Berücksichtigung von Kostenfunktionen automatisch zur besten Lösung führt. Eine mögliche Heuristik ist die *Heuristik des nächsten Nachbarn* (Nearest Neighbour Heuristic).

Eine Nächster Nachbar-Betrachtung einer Fahrt von S nach Z sieht wie folgt aus: So lange das Ziel noch nicht erreicht ist, wähle aus den noch nicht besuchten Städten die zur aktuellen Stadt am nächstgelegene Stadt aus.

Die Voraussetzung für die Anwendbarkeit dieser Heuristik ist die Aufwandschätzung *g* für die Zustandsübergänge. Man sucht immer den nächstgelegenen Nachbarknoten, den Knoten also, den man mit minimalen Kosten erreichen kann und der bisher noch nicht

besucht wurde. Bei dieser Heuristik ist ein Zurückgehen und Suchen nach Alternativen im Suchraum nicht vorgesehen. Insofern ist diese Strategie unvollständig und wird nicht immer die Optimallösung finden.

**ALGORITHMUS:** Nächster Nachbar

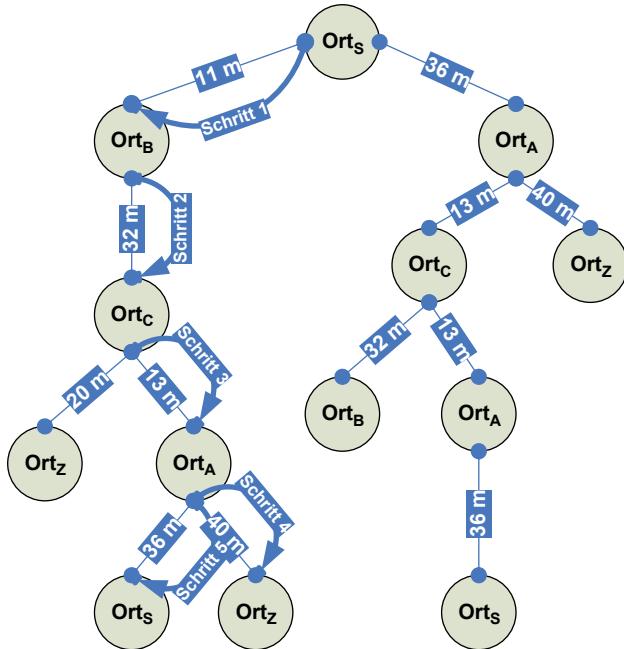
```

P :=  $z_1$ 
 $z_E := \text{Falsch}$ 
WIEDERHOLE
  Sei  $p = z_1, z_2, \dots, z_n$ 
   $z_E := \text{ziel}(z_n)$ 
  WENN NICHT  $z_E$  DANN TUE
    Suche den nächstgelegenen Nachbarn von  $z_n$ :  $X$ 
     $P := z_1, z_2, \dots, z_n, X$ 
  END-WENN
  BIS Ziel_erreicht
  LIEFERE  $p$  ZURÜCK
ENDE-ALGORITHMUS
```

Eine weitere einfache Strategie, um den Suchraum einzuschränken, besteht darin, in jedem Schritt der Suche die Kosten der Folgezustände des aktuellen Zustandes miteinander zu vergleichen und durch Wahl des „besten“ Folgezustands die Suche fortzusetzen. Diese Heuristik setzt die Bewertungsfunktion  $h$  für den Abstand zum Ziel voraus. Im Gegensatz zur Nachbar-Strategie erlaubt *Hill Climbing* das Durchsuchen mehrerer Kandidaten. Bildlich gesprochen schneidet diese Heuristik im Suchraum alle diejenige Zweige ab, bei denen der Nachfolger schlechter als sein Vorgänger ist. Im Beispiel der Straßenkarte entspricht dies immer demjenigen Knoten  $X$ , welcher vom momentanen Standort aus mit der geringsten Kilometeranzahl erreicht werden kann:  $f_{\text{kosten}}(X) > f_{\text{distanz}}(\text{aktueller Knoten}, X)$ . Folglich wäre mit dieser Strategie eine Fahrtstrecke  $S \Rightarrow B \Rightarrow C \Rightarrow A \Rightarrow Z$  ermittelt worden, was eine Fahrstrecke von 96 km bedeutet sowie einen Suchaufwand von fünf Schritten. Diese Vorgehensweise kann auch als eine durch lokale Messungen erweiterte Tiefensuche betrachtet werden. In der Literatur wird sie als *Steepest-Ascent-Hill-Climbing-Verfahren* bezeichnet. Die Anwendung dieses Verfahrens führt in dem gewählten Beispiel zwar zu mehr Kilometern als bei einer vollständigen Suche, jedoch kann die Qualität dieses Ergebnisses nicht verallgemeinert werden.

Die Bezeichnung Hill Climbing ist zurückzuführen auf die Vergleichbarkeit dieser Heuristik mit einem „blindlen“ Bergsteiger: Um den Weg zum Gipfel zu finden, tastet dieser seine unmittelbare Umgebung ab. Aus der gewonnenen lokalen Information wählt er den steilsten Anstieg aus, um seinen nächsten Schritt auszuführen. Diese Vorgehensweise wird solange wiederholt, bis seine direkte Umgebung keinen Anstieg mehr aufweist (Abb. 5.31).

Das Problem, das diese Vorgehensweise mit sich bringt, ist offensichtlich: Vermehrt vielversprechende Wege, die steilen Anstiege werden bevorzugt und führen zu einem loka-

**Abb. 5.31** Hill-Climbing

len Maximum. Aufgrund des streng vorwärtsorientierten Verhaltens ist damit die Lösungsfundung aber nicht immer gewährleistet.

**ALGORITHMUS:** Hill Climbing

$Z_E := \text{Falsch}$

**WIEDERHOLE**

Wähle aus Liste einen Weg:  $P := Z_1, Z_2, \dots, Z_m$

$Z_E := \text{ziel}(Z_n)$

**WENN NICHT  $Z_E$  DANN TUE**

Berechne alle Zustände  $[s_1, \dots, s_k]$  die von  $Z_n$  in einem Schritt erreicht werden können und für die gilt  $h(z_n) \geq h(s_i)$

Füge die Wege  $Z_1 Z_2 \dots Z_m s_1$  in Liste :=  $[p_2, p_3, \dots, p_k, Z_1 Z_2 \dots Z_n s_1, \dots, Z_1 Z_2 \dots Z_n s_k]$  ein

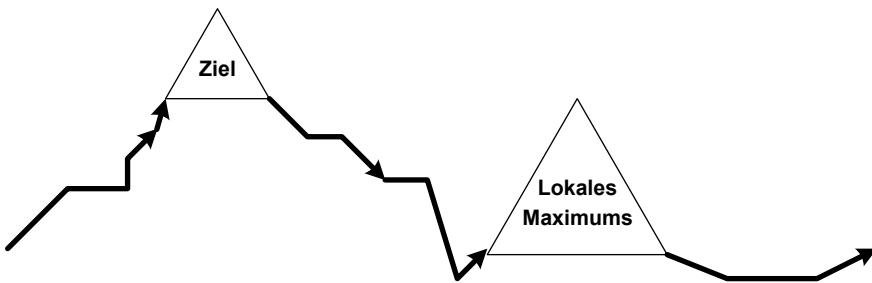
**END-WENN**

**BIS** Ziel\_erreicht

**LIEFERE**  $p$  **ZURÜCK**

**ENDE-ALGORITHMUS**

Auswege aus diesem Dilemma sind Backtracking und Look Ahead. *Backtracking* steht für das schrittweise Zurücksetzen im Suchbaum bis zu nicht weiter verfolgten Alternativen, die gleich gut oder nur geringfügig schlechter waren als die ursprünglich gewählte Fort-



**Abb. 5.32** Lokales Maximum

setzung. *Look Ahead* bezieht sich auf die Vorausschau über mehrere Zustandsübergänge, bevor die Entscheidung über eine Fortsetzung der Suche getroffen wird (Abb. 5.32).

Genau dieses Problem tritt auch beim klassischen Beispiel des Schiebepuzzles auf: In vielen Fällen müssen Plättchen, die sich schon auf ihrer richtigen Position befinden, wieder verschoben werden, um sich auf das endgültige Zielfeld zuzubewegen. In diesem Fall würde die Anwendung von Hill Climbing versagen, da diese Strategie nicht zwischen lokalen und globalen Maxima unterscheiden kann.

Eine weitere Variante der Breitensuche stellt die Heuristik der sogenannten *Best-First-Suche* dar. Wie der Name schon sagt, versucht dieses Verfahren, in jedem Schritt denjenigen Folgeschritt durchzuführen, der insgesamt mit den geringsten Kosten verbunden ist. Dazu wird eine spezielle Kostenfunktion  $\int_{\text{Strecke}}(X)$  eingeführt. Das Verfahren besteht darin, dass man aus der Menge der Kandidaten nicht einen beliebigen Weg, sondern denjenigen wählt, dessen Bewertung am besten ausfällt. Beste Bewertung heißt hier: Es wird der Kandidat gewählt, der gemäß der Schätzung  $h$  einem Ziel am nächsten ist. In Bezug auf das Beispiel ermittelt dieses Suchverfahren die Anzahl der Meter, die auf einer Fahrtroute vom Standort S bis zum Ort X bisher zurückgelegt wurden: Strecke (C) = distanz (S, B) + distanz (B, C) = 11 + 32 = 43. Nach sechs Suchschritten wird die Fahrtroute S  $\Rightarrow$  B  $\Rightarrow$  C  $\Rightarrow$  Z mit einer Streckenlänge von 63 m ermittelt.

Im Gegensatz zum Steepest-Ascent-Hill-Climbing-Ansatz, bei dem eine lokale Be trachtung von Alternativen erfolgt, werden hier alle möglichen Folgezustände X von bisher untersuchten Teilstrecken berücksichtigt. Die entsprechende Kostenfunktion könnte demnach wie folgt definiert sein (Abb. 5.33):

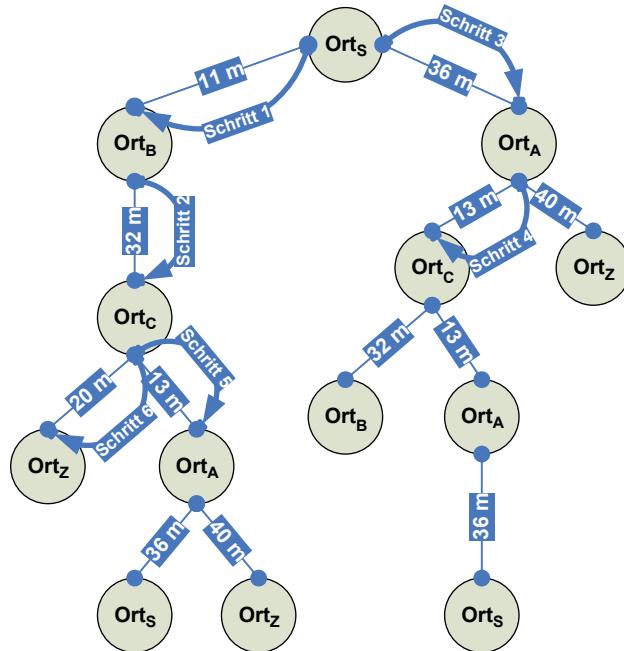
$$\int_{\text{Kosten}}(X) = \text{strecke}(\text{Aktueller Knoten}) + \text{distanz}(\text{Aktueller Knoten}, X)$$

Um in jedem Fall den kürzesten Weg zu finden, müssen alle diejenigen Teilstrecken weiter untersucht werden, welche kürzer sind als der bisher kürzeste Weg zu einer Lösung.

**ALGORITHMUS:** Bestensuche

$Z_E := \text{Falsch}$

**WIEDERHOLE**

**Abb. 5.33** Best First Suche

Wähle den besten Kandidaten aus Liste aus:

$P := Z_1, Z_2, \dots, Z_m$

**WENN**  $Z_n$  ist Zielzustand **DANN**  $Z_E := \text{WAHR}$

$\text{Ziel\_erreicht} := \text{ziel}(Z_n)$

**ANSONSTEN TUE**

Berechne alle Zustände  $[s_1, \dots, s_k]$  die von  $Z_n$  in einem Schritt erreicht werden können

**LÖSCHE**  $p$  in Liste und füge die Wege  $Z_1 Z_2 \dots Z_m S_1$  inListe :=  $[p_2, p_3, \dots, p_k, Z_1 Z_2 \dots Z_m S_1, \dots, Z_1, Z_2 \dots Z_m S_k, ]$  ein

**END-WENN**

**BIS**  $Z_E$

**LIEFERE**  $p$  **ZURÜCK**

**ENDE-ALGORITHMUS**

Die Bestensuche strukturiert den Suchraum nicht nach der Tiefe, wie dies bei der Breitensuche der Fall ist, sondern nach der Qualität der partiellen Lösung. Im Gegensatz zur Nachbar-Suche bricht die Bestensuche nicht blind in einen Zweig des Suchbaums ein, sondern steuert die Reihenfolge der Abarbeitung der Kandidaten nach deren Qualität. Insoweit strukturiert die Bestensuche den Suchraum, ohne ihn dabei zu reduzieren.

Eine weitere Verbesserungsmöglichkeit bei einer Best-First-Suche bietet sich in der zusätzlichen Einschränkung des Suchraums. Dazu wird eine Schätzfunktion eingeführt, welche abschätzen soll, wie weit es vom momentanen Standort aus noch bis zur Lösungsfindung ist. Damit wird eine erweiterte Kostenfunktion  $\int_{\text{Kosten}}(X)$  eingeführt, die sich aus zwei Komponenten zusammensetzt:  $\int_{\text{Kosten}}(X) = g(x) + h(x)$ . Die Funktion  $g$  beschreibt die Kosten  $g(x)$  für den zurückgelegten Weg vom Startknoten  $S$  bis zum Knoten  $X$  und die Funktion  $h$  schätzt die Kosten  $h(x)$  von  $X$  aus bis zum Ziel  $Z$  ab. Für das betrachtete Beispiel könnte als Schätzfunktion beispielsweise die Funktion  $\int_{\text{luftroute}}(x, z)$  definiert werden, die angibt, wie groß die direkte Entfernung von der Stadt  $X$  zum Zielort  $Z$  ist, also wieviel Meter noch mindestens zurückgelegt werden müssen. Damit ist man in der Lage, in jedem Zustand der Suche in Verbindung mit der bisherigen zurückgelegten Strecke eine Schätzung für Länge der Fahrtroute durchzuführen. Somit ergibt sich die Gesamtfunktion:  $\int_{\text{Kosten}}(X) = \text{strecke}(x) + \text{luftroute}(x, z)$ . Zu beachten ist, dass mit Verwendung einer Schätzfunktion wesentlich weniger Schritte zur Lösungsfindung notwendig sind als ohne. Überträgt man das Verhalten von Heuristiken auf praktische Problemstellungen, so ist die Qualität ihrer Lösung und die Effizienz ihrer Suche oft unzureichend. Nimmt man beispielsweise einen autonomen Roboter, der in einer gefährlichen Umgebung operieren soll, so kann schon ein einziger falscher Schritt zu einer Katastrophe führen. Folglich sind die Anforderungen, welche eine Heuristik unbedingt erfüllen muss, in den meisten Fällen höher als bei einfacher Hill-Climbing- oder Best-First-Suche. Es reicht dann nicht mehr aus, irgendeinen Weg zum Ziel zu finden, sondern die ausgewählte Heuristik muss einfach den kürzesten Weg garantieren.

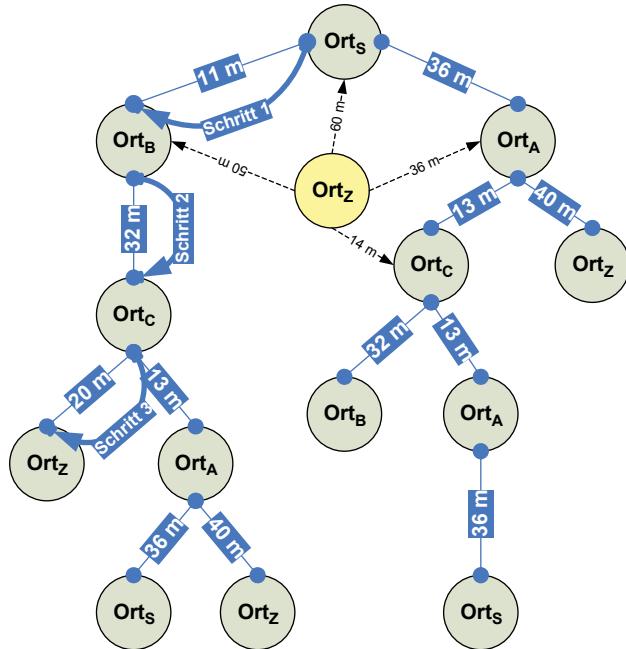
Heuristiken, die immer den kürzesten Weg zum Ziel finden, falls ein solcher Weg existiert, nennt man zuverlässig: Eine Funktion  $h$  zur Bewertung eines Knotens heißt demnach dann zuverlässig, wenn sie die Kosten, um zum Ziel zu kommen, nicht überschätzt. Für jeden Knoten  $n$  und jeden Zielknoten  $z$ , der von  $n$  aus erreichbar ist, gilt dann:  $h(n) \leq g(z) - g(n)$ . Bei Berücksichtigung folgender Luftlinien:  $Z \Leftrightarrow A: 36 \text{ m}$ ;  $Z \Leftrightarrow S: 60 \text{ m}$ ;  $Z \Leftrightarrow B: 50 \text{ m}$ ;  $Z \Leftrightarrow C: 14 \text{ m}$  ergibt sich eine Route  $S \Rightarrow B \Rightarrow C \Rightarrow Z$  mit einer Länge von 77 m.

Betrachtet man etwa Problemstellungen, bei denen die Zustandsübergänge gleich viel „kosten“, so ist auch die Breitensuche eine zulässige Strategie, da sie zunächst alle Wege der Länge  $n$  untersucht, bevor sie Wege der Länge  $n + 1$  betrachtet. Für den praktischen Einsatz sind jedoch effiziente Verfahren notwendig. Eine der wohl bekanntesten zulässigen Heuristiken ist der sogenannte  $A^*$ -Algorithmus des  $A^*$ -Suchverfahrens, der die heuristischen Funktionen  $g$  und  $h$  voraussetzt:

- $g$ : Kosten für den bisherigen Weg
- $h$ : Abstandsmaß für den Abstand des aktuellen Knotens zum Ziel

Um die Eigenschaften dieses Verfahrens zu erklären, muss die

$$\int_{\text{Kosten}}(X) = g(x) + h(x)$$

**Abb. 5.34** Bestensuche

leicht modifiziert werden:  $f^*(\text{Kosten})(X) = g^*(x) + h^*(x)$ .  $g^*$  beschreibt die Kosten des kürzesten Pfades  $g^*(X)$  vom Startknoten zum Knoten  $X$  im Zustandsraum.  $h^*$  ist eine Schätzfunktion, deren Wert  $h^*(X)$  kleiner oder höchstens gleich groß ist wie die Kosten des kürzesten Wege von  $X$  zum Zielzustand. Dadurch, dass die verwendete Schätzfunktion niemals die tatsächlichen Kosten zum Zielknoten übersteigt, ist garantiert, dass immer der kürzeste aller Wege zum Ziel gefunden wird. Man verwaltet dazu zwei Mengen von Knoten (Zuständen):

- Offene Knoten: Knoten, die bereits bewertet, aber noch nicht expandiert wurden
- Geschlossene Knoten: Knoten, die bewertet und expandiert wurden

Dabei wird mit „expandieren“ das Berechnen der Nachfolger bezeichnet, die in einem Schritt erreichbar sind. Eine solche Funktion stellt auch die im Beispiel der Straßenkarte verwendete Funktion  $\text{luftroute}(x)$  dar, da eine mögliche Streckenverbindung zwischen einem Knoten  $X$  und dem Zielknoten  $Z$  in jedem Fall größer oder gleich der Entfernung  $\text{luftroute}(x, z)$  sein muss (Abb. 5.34).

Für den oben angesprochenen Spezialfall gleich großer Zustandsübergangskosten kann man sogar die Breitensuche als A\*-Heuristik bezeichnen und zwar im Fall  $h^*(X) = 0$  für alle  $X$ . Die Implementierung erfolgt oftmals in der Form, dass man Zeiger zwischen den einzelnen Knoten vorsieht, die von einem Knoten auf seinen Nachfolger und von allen Knoten aus **GESCHLOSSEN** und **OFFEN** auf deren jeweiligen besten Vorgänger verweisen. Immerhin gewährleistet der A\*-Algorithmus, dass jeder bereits bearbeitete Knoten maximal einen Vorgänger hat. Stößt man nun auf der Suche auf einen besseren Vorgänger eines

Knotens, dann setzt man den Zeiger auf den besten Vorgänger und propagiert die neue, bessere Bewertung auf alle Folgezustände und sukzessive auf deren Folgenzustände. Wenn die angewandte Schätzfunktion zulässig ist, dann wird mit dem A\*-Algorithmus immer eine Optimallösung gefunden, d. h. wenn sie die tatsächlichen Kosten nicht überschätzt. Dies wird sich allerdings nicht immer realisieren lassen, da über ein Problem nicht zwingend genügend Informationen vorliegen müssen bzw. sich eruieren lassen. Abschließend ist noch zu bemerken, dass Art und Komplexität der Kostenfunktion zu sehr aufwendigen Berechnungen führen können. Deshalb sollte die Einschränkung des Suchraumes nicht alleiniges Kriterium bei der Wahl der Heuristik sein.

```

ALGORITHMUS: A*
OFFEN: = [  $z_1$  ]
GESCHLOSSEN: = [ ]
 $z_e$  := Falsch
WIEDERHOLE
    X := Bester Knoten aus OFFEN
    WENN x ist Ziel DANN  $z_e$  := WAHR
    ANSONSTEN TUE
        LÖSCHE c in OFFEN
        FÜGE x in GESCHLOSSEN ein
        ZUGRIFF := alle Nachfolger von x
        FÜR ALLE y ∈ ZUGRIFF
            IF y ∈ OFFEN DANN
                Vergleiche den neuen Weg zu y mit bisherigem und
                lösche den schlechteren
            ODER WENN y ∈ GESCHLOSSEN DANN
                Vergleiche neuen Weg zu y mit bisherigem und
                lösche den schlechteren
            ANSONSTEN Füge y in OFFEN ein
    END-WENN
    ENDE-FÜR ALLE
    ENDE-WENN
BIS  $z_e$ 
LIEFERE p ZURÜCK
ENDE-ALGORITHMUS

```

Nunmehr gilt es, sich den *planungsbasierten Ansätzen* zuzuwenden, bei denen intelligentes Verhalten ein zielgerichtetes Planen voraussetzt. Während es aber beim Schlussfolgern darum geht festzustellen, ob ein bestimmter Sachverhalt vorliegt oder nicht, ist das Ziel des Planens ein anderes. Gegeben seien ein vorliegender Zustand und die Beschreibung eines erwünschten Zielzustands. Die Planungsaktivität besteht dann darin, eine Folge von Aktionen zu erstellen, deren Ausführung den vorliegenden Zustand in einen Zustand überführt, in der die Zielbeschreibung zutrifft.

Neben der Fähigkeit, Schlussfolgerungen ziehen bzw. lernen zu können, ist das zielgerichtete Planen etwas, in dem sich intelligentes Verhalten in besonderer Weise manifestiert. Während es aber beim Schließen darum geht festzustellen, ob ein bestimmter Sachverhalt vorliegt oder nicht, ist das Ziel des Planens ein anderes. Gegeben sind meist ein vorliegender Zustand und die Beschreibung eines erwünschten Zielzustands. Die Planungsaktivität besteht dann darin, eine Folge von Aktionen zu erstellen, deren Ausführung den vorliegenden Zustand in einen Zustand überführt, in der die Zielbeschreibung zutrifft. Um welche Art von Zuständen und möglichen Aktionen es sich dabei handelt, ist zunächst noch völlig offen. Einen der ersten Versuche, diese Fähigkeiten in einem Roboter zu realisieren, stellt das System STRIPS dar. Die grundsätzliche Arbeitsweise dieses Systems soll als Beispiel benutzt werden, um die Wirkprinzipien eines Verfahrens mit Operatorauswahl zu illustrieren. Es wird eine Roboterwelt betrachtet, die aus mehreren, durch Türen miteinander verbundenen Zimmern besteht, in denen sich verschiedene Objekte befinden. Die Aufgabe des Robotersystems ist es, diese Objekte in einer bestimmten Weise zu manipulieren. Das Weltmodell des Roboters wird mit Hilfe einer Menge von prädikatenlogischen Formeln beschrieben. Im Verlaufe der Aktionen des Robotersystems entsteht eine ganze Folge auseinander hervorgehender Modelle. Jedes dieser Modelle beschreibt einen Problemzustand. Im Anfangszustand kann die Roboterwelt durch folgendes Modell  $M_0$  charakterisiert werden (Kleinbuchstaben bezeichnen Variable; freie Variable sind als allquantifiziert zu betrachten):

Zustand	Term	Beschreibung
$M_0$	STUHL(OBJ <sub>1</sub> )	Objekt OBJ <sub>1</sub> ist ein Stuhl
	TISCH(OBJ <sub>2</sub> )	Objekt OBJ <sub>2</sub> ist ein Tisch
	ORT(OBJ <sub>1</sub> , RAUM <sub>2</sub> )	OBJ <sub>1</sub> befindet sich im Raum 2
	ORT(OBJ <sub>2</sub> , RAUM <sub>3</sub> )	OBJ <sub>2</sub> befindet sich im Raum 3
	ORT(ROB <sub>1</sub> , RAUM <sub>1</sub> )	Der Roboter 1 befindet sich im Raum 1
	LINK(RAUM <sub>1</sub> , RAUM <sub>2</sub> , TÜR <sub>1</sub> )	Raum 1 und Raum 2 sind mit der Tür 1 verbunden
	LINK(RAUM <sub>2</sub> , RAUM <sub>3</sub> , TÜR <sub>2</sub> )	Raum 2 ist mit Raum 3 mit der Tür 2 verbunden
	LINK(r <sub>1</sub> , r <sub>2</sub> , d) $\leftrightarrow$ LINK(r <sub>2</sub> , r <sub>1</sub> , d)	Die LINK-Relation ist bezüglich der ersten beiden Argumente symmetrisch

Als eine mögliche Handlung wird nun folgende Operation durchgeführt:

Operation	Term	Beschreibung
O <sub>1</sub>	GEHE (d, r <sub>1</sub> , r <sub>2</sub> )	Das Robotersystem geht durch die Tür d von Raum 1 nach Raum 2

Zunächst muss das Robotersystem sein Modell auf folgende Anwendungsbedingungen überprüfen:

Bedingungen	Term	Beschreibung
$B_{1\_1}$	$\text{ORT}(\text{ROB}_1, r_1)$	Das Robotersystem befindet sich aktuell im Raum 1
$B_{1\_2}$	$\text{LINK}(r_1, r_2, d)$	Der Raum 1 ist durch eine Tür mit Raum 2 verbunden

Das Ausführen dieser Operation führt unweigerlich zu Streichungen des Terms  $\text{ORT}(\text{ROB}_1, \text{RAUM}_1)$  und damit insgesamt zu einer Modellmodifikation. Gleichzeitig wird die Modellwelt um folgenden Term erweitert:  $\text{ORT}(\text{ROB}_1, \text{RAUM}_2)$ .

Als weitere Handlung wird nun folgende Operation durchgeführt:

Operation	Term	Beschreibung
$O_2$	$\text{TRANS}(o, d, r_1, r_2)$	Das Robotersystem transportiert das Objekt o durch die Tür d aus dem Raum 1 in den Raum 2

Zunächst muss auch hier das Robotersystem sein Modell auf folgende Anwendungsbedingungen überprüfen:

Bedingungen	Term	Beschreibung
$B_{2\_1}$	$\text{ORT}(O_1, r_1)$	Das Robotersystem befindet sich aktuell im Raum 1
$B_{2\_2}$	$\text{ORT}(\text{ROB}_1, r_1)$	Das Robotersystem befindet sich aktuell im Raum 1
$B_{2\_3}$	$\text{LINK}(r_1, r_2, d)$	Der Raum 1 ist durch eine Tür mit Raum 2 verbunden

Das Ausführen dieser Operation führt unweigerlich zu Streichungen des Terms

$$\text{ORT}(O_1, \text{RAUM}_1) \text{ und } \text{ORT}(\text{ROB}_1, \text{RAUM}_1)$$

und damit insgesamt zu einer Modellmodifikation. Gleichzeitig wird die Modellwelt um folgenden Term erweitert:

$$\text{ORT}(O_1, \text{RAUM}_2) \text{ und } \text{ORT}(\text{ROB}_1, \text{RAUM}_2)$$

Man kann diese Operationen mit dem Muster der Produktionsregeln vergleichen. Die Anwendungsbedingungen OP1 und OP2 enthalten gleichfalls Variablen, die zu belegen sind. Sie stellen Bedingungen dar, weil sie nach Belegung der Variablen in Aussagen übergehen, die anhand des jeweiligen Zustandes der Roboterwelt (des jeweiligen Modells) überprüft werden müssen. Nach einer Operatoranwendung wird nicht das gesamte Weltmodell des Robotersystems neu formuliert (das wäre bei umfangreichen Weltmodellen praktisch gar nicht durchführbar), sondern es werden nur die Änderungen gegenüber dem vorhergehenden Zustand festgehalten. Die Technologien der Artifiziellen Intelligenz stellen hierfür verschiedene Ansätze zur Verfügung:

- Eine generelle Annahme bezüglich der Beschreibung von Roboterhandlungen besteht darin, dass bei jeder Aktion (diese wird durch eine Operation beschrieben) immer nur diejenigen Charakteristika der Welt zu ändern sind, die explizit in der Definition der entsprechenden Operation vorkommen. Diese Annahme ist in der Literatur unter der Bezeichnung *STRIPS-assumption* eingegangen. Sie besagt beispielsweise, dass durch die Operation GEHE nur der Ort des Roboters, aber nicht die Lage der anderen Objekte verändert wird; ebenso werden durch die TRANSPORT-Aktion nicht die Geometrie der Räume oder die bestehenden Türverbindungen beeinflusst usw.
- Von *McCarthy* und *Hayes* wurde eine Situationslogik entwickelt, die für die Beschreibung dynamischer Sachverhalte geeignet ist. Ein wesentliches Charakteristikum dieser Logik ist die Einführung von zusätzlichen Situationsvariablen in die Prädikate. Das Problem der Spezifikation des Teiles der Welt, der gegenüber bestimmten Operationen stabil bzw. invariant bleibt (sogenanntes *Frame Problem*) wird in der Situationslogik durch Formulierung von speziellen Axiomschemata gelöst, die entsprechend ihres Zwecks als *Frame-Axiome* bezeichnet werden.

Es ist bei den weiteren Ausführungen zu dem obigen Beispiel zu bedenken, dass hier ein relativ komplexes Problem (Beschreibung einer Roboterwelt) auf einige wenige prädikatenlogisch formulierte Zusammenhänge reduziert wurde. In den Operationen ist keine besondere Argumentstelle für das Robotersystem vorgesehen, da nur ein Robotersystem kommt. In einer Welt mit mehreren Robotern wäre eine solche Argumentstelle natürlich erforderlich, um angeben zu können, welcher Roboter wohin geht, oder welcher Roboter etwas transportiert. Im obigen Beispiel ist relativ leicht zu sehen, wie diese Welt um weitere Räume und Objekte erweitert werden kann. Demgegenüber ist die Behandlung mehrerer interagierender Roboter wesentlich komplizierter, da die Aktivitäten der Roboter i. a. nicht unabhängig voneinander sind und sich deshalb die Notwendigkeit für die Abstimmung der Roboterhandlungen untereinander und für Konfliktlösungen ergibt. Die Aufgabe des Roboters bzw. das Ziel für seine Aktionen kann als prädikatenlogischer Ausdruck formuliert werden, den der anzustrebende Zustand der Roboterwelt erfüllen muss. So kann der Auftrag an den Roboter, dafür zu sorgen, dass sich ein Stuhl im Raum RAUM 3 befindet, ausgedrückt werden durch:

$$Z_0 : \exists STHUL(o) \wedge ORT(o, Raum_3)$$

Der Versuch, das Zielprädikat  $Z_0$  unmittelbar anhand des Modells  $M_0$  zu erfüllen, führt nicht zum Erfolg. Denn das einzige Objekt  $OBJ_1$ , auf welches das Prädikat STUHL( $OBJ_1$ ) zutrifft, erfüllt nicht das Prädikat ORT( $OBJ_1, RAUM_3$ ). Da es, zumindest in der hier betrachteten Roboterwelt, keine Operatoren gibt, die an der Art der Objekte etwas verändern (beispielsweise einen Tisch in einen Stuhl umfunktionieren), wird als alleinige Differenz zwischen Zielprädikat  $Z_0$  und Modell  $M_0$  der Ausdruck ORT( $OBJ_1, RAUM_3$ ) festgehalten. Als nächstes gilt es festzustellen, welche Operation diese Differenz beseitigen könnte („means-ends-analysis“). Das ist die Operation  $OP_2$ , da nur der Transport eines Objektes etwas an dessen Lage ändert. Allerdings ist die Anwendungsbedingung für diesen Opera-

tor noch nicht erfüllt, so dass sich ein neues Ziel  $Z_1$ , ergibt, das in der Erfüllung eben dieser Bedingung besteht:

$$Z_1 : \text{ORT}(\text{OBJ}_1, r_1) \wedge \text{ORT}(\text{ROB}_1, r_1) \wedge \text{LINK}(r_1, \text{RAUM}_3, d).$$

Mit der Belegung  $r_1 = \text{RAUM}_2$  kann zwar der erste Teilausdruck des Ziels  $Z_1$  im Modell  $M_0$  erfüllt werden, es verbleibt aber eine weitere Differenz bei dieser Belegung, nämlich

$$\text{ORT}(\text{ROB}_1, \text{RAUM}_2) \wedge \text{LINK}(\text{RAUM}_2, \text{RAUM}_3, d).$$

Durch Vergleich dieser Differenz mit den Operatoren findet man, dass in diesem Fall  $\text{OP}_1$  für die Beseitigung der Differenz geeignet ist. Die Erfüllung der Anwendungsbedingung des Operators  $\text{OP}_1$  führt aber zu einem weiteren Teilziel  $Z_2$ :

$$Z_2 : \text{ORT}(\text{ROB}_1, \text{RAUM}_1) \wedge \text{LINK}(\text{RAUM}_1, \text{RAUM}_2, d)$$

Dieser Ausdruck kann in  $M_0$  unmittelbar durch die Belegung  $d = \text{TÜR}_1$  verifiziert werden. Damit wird der Operator  $\text{OP}_1$  anwendbar; als Ergebnis entsteht ein neuer Weltzustand, der durch das Modell  $M_1$  beschrieben wird:

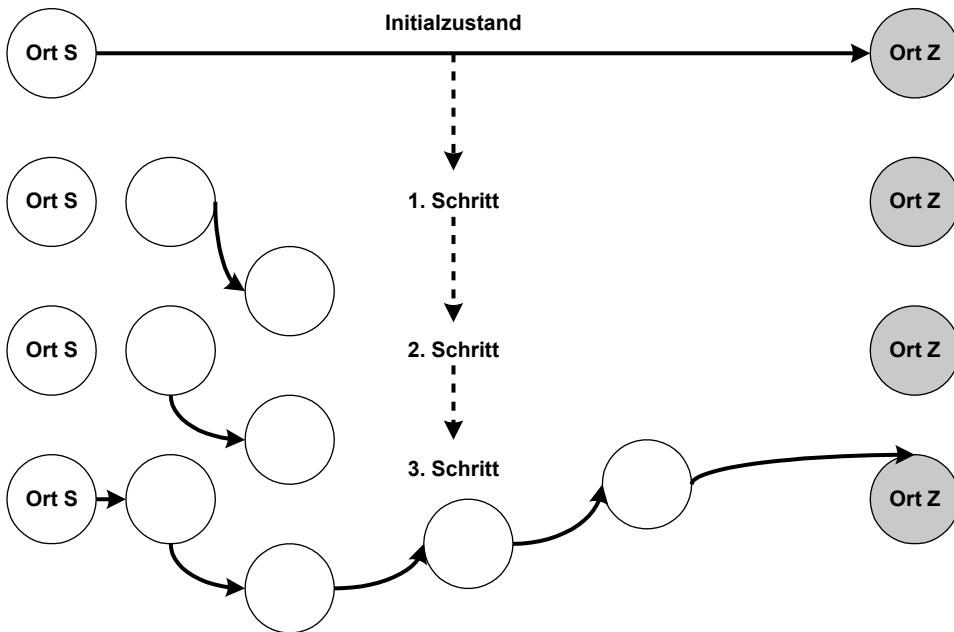
$$M_1 : \text{ORT}(\text{OBJ}_1, \text{RAUM}_2) \mid \text{ORT}(\text{ROB}_1, \text{RAUM}_2).$$

Alles Übrige bleibt unverändert. Im Modell  $M_0$  wird die als Ziel  $Z_0$  formulierte Anwendungsbedingung für den Operator  $\text{OP}_2$ ;  $\text{TRANSPORT}(o, d, r_1, r_2)$  mit der Variablenbelegung  $o = \text{OBJ}_1$ ,  $r_1 = \text{RAUM}_2$ ,  $r_2 = \text{RAUM}_3$  und  $d = \text{TÜR}_2$  erfüllbar. Nach Anwendung des Transportoperators entsteht der Weltzustand  $M_2$ ,

$$M_2 : \text{ORT}(\text{ROB}_1, \text{RAUM}_3) \mid \text{ORT}(\text{OBJ}_1, \text{RAUM}_3).$$

Alles Übrige bleibt wiederum gegenüber Modell  $M_0$  unverändert. Wenn dieses Modell der Welt erreicht ist, kann das ursprüngliche Zielprädikat  $Z_0$  mit  $o = \text{OBJ}_1$  verifiziert werden. Damit ist der angestrebte Zustand der Welt durch Operationen des Robotersystems erreicht.

Bei der Suche nach Lösungen in einem formal spezifizierten Zustandsraum ist jede Situation, welche durch die Ausführung von Aktionen erreicht werden kann, durch bestimmte Bedingungen gekennzeichnet, die diese Lösung hinreichend beschreiben. Dies gilt für alle Zustände entlang eines Weges im Suchbaum. Bei den bisher beschriebenen heuristischen Ansätzen geht man davon aus, dass, ausgehend von einem gegebenen Startzustand, durch die sequentielle Verknüpfung von Aktionen immer eine Lösung gefunden wird. Manche Situationen erlauben diese Vorgehensweise jedoch nicht, da Handlungsalternativen nicht explizit bereitstehen, sondern erst implizit erzeugt werden müssen. Daher könnte alter-



**Abb. 5.35** Teillösungen

nativ die Erforschung eines Zustandsraums auch so organisiert sein, dass die Lösung eines Problems gerade in der Suche nach Schritten besteht, zusammengenommen dann einen durchgängigen Lösungsweg bilden. In der Praxis treten diese Art von Problemen vor allem im Bereich der Planung auf, aber auch bei dem Versuch, Handlungsweisen von anderen abzuschätzen, was eine Planung derselben voraussetzt.

Auch beim Reiseproblem mit Mittel-Zum-Zweck-Verfahren sei wieder das Problem gegeben, die Verbindung wegen einer Reise von einem Ort S zu einem Ort Z zu suchen. Zur Lösung solcher Planungsaufgaben muss ein wissensbasiertes System spezielle Kenntnisse haben, beispielsweise wie groß die Entfernung zwischen allen in Frage kommenden Zwischenstationen auf der Reiseroute von Ort S nach Ort Z sind. Darüber hinaus muss das System über Wissen verfügen, welche Verkehrsmittel zum Erreichen eines konkreten Ortes zur Verfügung stehen und welche wann am sinnvollsten zu nehmen sind. Am besten lassen sich dazu Faustregeln definieren, die angeben, dass es günstig ist, 1) das Flugzeug für Strecken größer als 500 km zu nehmen, 2) die Bahn für Strecken über 100 km zu nehmen, 3) das Auto für Strecken zwischen 15 km und 100 km zu nehmen, 4) das Taxi für Strecken zwischen 1 und 15 km zu nehmen und 5) zu gehen, falls die zurückzulegende Strecke unter 1 km lang liegt (Abb. 5.35).

Diese Art der Beschreibung kann in einer speziellen Datenstruktur, einer sogenannten Mittel-zum-Zweck-Tabelle festgehalten werden. In ihr sind demnach alle zur Verfügung stehenden Mittel mit ihrem Verwendungszweck aufgelistet. Für das obige Beispiel sind die Mittel durch die Verkehrsmittel und die Zwecke durch die zurückzulegenden Entfernun-

gen definiert. Für den Zweck, eine Strecke von über 500 km zurückzulegen, kann sowohl auf das Flugzeug als auch auf die Bahn zurückgegriffen werden. Ein wissensbasiertes System verwendet nun die oben aufgelisteten Faustregeln, um für das konkrete Anwendungsbeispiel aufgrund der zurückzulegenden Distanz zu entscheiden, welches Verkehrsmittel am besten eingesetzt werden sollte. Dabei ist es wichtig, dass ein Flugzeug nur dann genommen werden kann, wenn man sich am Flughafen befindet und dass man mit der Bahn nur dann fahren kann, wenn man sich auf einem Bahnhof befindet. Dies impliziert, dass jede Aktion an bestimmte Vorbedingungen gebunden ist. Hier wird offensichtlich, dass der Vorgang des Planens nichts anderes als die Suche nach einer Sequenz von möglichen Operationen darstellt, die von einem Ausgangszustand zu einem definierten Zielzustand führt. Dabei lässt sich eine Priorisierung berücksichtigen, die sicherstellt, dass die wichtigsten oder unbedingten Schritte zuerst geplant werden.

Diese Strategie der Problemlösung nennt man *Mittel-zum-Zweck-Analyse*. Ihre Einführung geht zurück in das Jahr 1957 und dort auf den von den Wissenschaftlern Newell, Shaw und Simon entwickelten „Allgemeinen Problemlöser (engl. general problem solver)“. Ein typisches Anwendungsbereich für diese Art von Suche ergibt sich in der Robotik. Hier ist es beispielsweise von großem Interesse, Bewegungen von Robotern oder Roboterarmen so zu planen, dass sie möglichst präzise und schnell Positionen ansteuern und dabei Objekte greifen oder bewegen können. Um eine entsprechende Mittel-zum-Zweck-Tabelle zu erstellen, werden oftmals Operatoren als Mittel für bestimmte Zwecke verwendet, die zur Aufgabenlösung beitragen können. Zur Bewältigung solcher Aufgaben lässt sich ein Algorithmus entwickeln, dessen Wissen aus der Mittel-zum-Zweck-Tabelle, sowie der vorhandenen Operatormenge besteht.

**ALGORITHMUS:** Mittel-zum-Zweck

```

# zStart ist der Startzustand
# zZiel
WENN zStart = zZiel DANN STOP
ANSONSTEN
    Bestimme Unterschied zwischen zStart und zZiel
    WIEDERHOLE
        Wähle einen noch nicht getesteten OPERATOR zur
        Überbrückung dieses Unterschieds
        WENN OPERATOR nicht vorhanden DANN FEHLER
        BESCHREIBE Vor- und Nachbedingungen
        Mittel-zum-Zweck (Vorbedingungen, Nachbedingungen)
        WENN Mittel-zum-Zweck (Vorbedingungen,
        Nachbedingungen) MÖGLICH DANN
            LIEFERE (Vorbedingung, OPERATOR, Nachbedingung)
        END-WENN
    END-WENN
END-WIEDERHOLE
ENDE-ALGORITHMUS
```

Der Algorithmus liefert als Ausgabe eine Operatorfolge zur Überführung des Startzustands in den Zielzustand. Dazu bestimmt er zunächst den wichtigsten Unterschied zwischen dem Start- und Zielzustand und versucht, diesen zu überwinden, indem er einen entsprechenden, noch nicht angewendeten Operator auswählt. Existiert ein solcher, so werden Vor- und Nachbedingungen spezifiziert, und der Algorithmus ruft sich selbst mit zwei neuen Teilproblemen auf, die auf die gleiche Weise bearbeitet werden. Nach erfolgreichem Lösen aller Teilprobleme wird eine Liste von Operatoren zurückgegeben, die zu einer Gesamtlösung führen. Diese Liste repräsentiert sozusagen einen Plan zur Lösung des Problems.

### 5.3.4 Symbolische Verfahren

*Intelligentes Verhalten* kann durch die Bearbeitung von Symbolen erzeugt werden. Dies ist einer der grundlegenden KI-Lehrsätze. Symbole sind Zeichen, die Objekte oder Ideen der realen Welt darstellen und auf einem Computer in Form von Zeichenketten oder Zahlen repräsentiert werden können. Bei diesem Ansatz wird ein Problem durch eine Gruppe von Symbolen dargestellt und dann der geeignete Algorithmus entwickelt, um diese Symbole zu verarbeiten. Das *Paradigma der Symbolverarbeitung* besagt, dass nur ein Symbolsystem über die erforderlichen und ausreichenden Mittel für allgemeines intelligentes Handeln verfügt. Dieser Gedanke, dass Intelligenz aus der aktiven Bearbeitung von Symbolen herrührt, war das Fundament, auf dem ein Großteil KI-Forschungsarbeit beruhte. Die Forscher konstruierten intelligente Systeme, indem sie Symbole für Mustererkennung, Schlussfolgerungen, Lernen und Planen einsetzten. Die Geschichte hat gezeigt, dass Symbole zwar für Schlussfolgerung und Planen zweckmäßig sind, dass sich Mustererkennung und Lernen jedoch besser mit anderen Ansätzen angehen lassen. Es gibt einige typische Methoden der Handhabung von Symbolen, die sich als sinnvoll für die Problemlösung erwiesen haben. Der gebräuchlichste Ansatz besteht darin, die Symbole in Formulierungen von Produktionsregeln in der Form von Wenn-dann-Vorschriften einzusetzen, die mit Hilfe von speziellen Schlussfolgerungstechniken, der Vorwärts- und der Rückwärtsverkettung, verarbeitet werden.<sup>50</sup> Beim Vorwärtsverketten leitet das System neue Informationen aus einer gegebenen Menge von Eingabedaten ab. Beim Rückwärtsverketten trifft es Schlussfolgerungen, ausgehend von einem bestimmten Zielzustand. Eine Methode zur Symbolverarbeitung kann auch der Aufbau eines semantischen Netzwerks sein, in dem die Symbole und die Konzepte, die sie darstellen, zu einem Wissensnetzwerk verknüpft werden, das dann dazu genutzt werden kann, neue Beziehungen zu bestimmen. Ein weiterer Formalismus ist ein Frame (Rahmen), in dem zusammengehörige Attribute eines Konzepts zu einer Struktur mit Slots (Feldern) gruppiert werden und von einer Reihe verwandter Prozeduren verarbeitet werden. Ändert man den Wert eines einzigen Slots, kann man eine komplexe Abfolge damit verbundener Prozeduren auslösen, während das

<sup>50</sup> Vgl. Haun 2000

Wissen, das durch die Frames dargestellt wird, in einen konsistenten Zustand überführt wird. Symbolverarbeitende Verfahren stehen im kognitiven Prozess auf einer relativ hohen Stufe. Aus der Sicht der Kognitionswissenschaft entspricht die Symbolverarbeitung dem bewussten Denken mit einer expliziten Wissensdarstellung, wobei das Wissen selbst untersucht und bearbeitet werden kann.<sup>51</sup>

Während Symbolverarbeitung und bewusstes Denken naheliegende Forschungsbeziehe sind, verfolgt eine andere Gruppe von Wissenschaftlern die Erforschung der Intelligenz mit einem *nichtsymbolischen Ansatz* nach dem Muster des menschlichen Gehirns. Diese subsymbolischen Verfahren werden zwar im nächsten Abschnitt gesondert behandelt, jedoch sollen an dieser Stelle bereits einige wenige Anmerkungen zwecks eines besseren Überblicks erfolgen. Ein subsymbolisches Verfahren, das immer mehr an Popularität gewinnt, ist bekannt unter dem Namen *Konnektionismus*. Neuronale Netzwerke haben weniger mit der Theorie der Symbolverarbeitung zu tun, die von formaler mathematischer Logik angeregt ist, sondern eher damit, wie sich menschliche oder natürliche Intelligenz äußert. Menschen tragen in ihren Köpfen neuronale Netzwerke, bestehend aus Hunderten von Milliarden von Gehirnzellen, den Neuronen, die durch adaptive Synapsen verbunden sind, welche als Schaltsysteme zwischen den Neuronen dienen. Artifizielle neuronale Netzwerke werden dieser gewaltigen Architektur des menschlichen Gehirns nachgebildet. Sie verarbeiten Informationen nicht durch die Manipulation von Symbolen, sondern indem sie riesige Mengen von Rohdaten parallel verarbeiten. Neuronale Netzwerke unterschiedlicher Fassung benutzt man zur Segmentierung, Gruppierung und Klassifizierung von Daten und zur Erstellung von Prognosemodellen. Diese Funktionen werden von einer Reihe von Prozessoren ausgeführt, die die grundlegenden Aktionen zwischen echten Neuronen nachbilden. Wenn das Netzwerk lernt oder Wissen vermittelt bekommt, werden die Verbindungsgewichte zwischen den Prozessoren entsprechend der wahrgenommenen Beziehungen zwischen den Daten verändert. Im Vergleich zu symbolverarbeitenden Systemen leisten neuronale Netzwerke kognitive Funktionen auf einem relativ niedrigen Niveau. Das Wissen, das sie sich durch den Lernprozess aneignen, wird in den Verbindungsgewichten gespeichert und kann nicht einfach untersucht oder verändert werden. Die Fähigkeit neuronaler Netzwerke, von ihrer Umgebung zu lernen und sich anzupassen, ist jedoch eine entscheidende Funktion, die intelligente Softwaresysteme erfüllen müssen. Aus Sicht der kognitiven Wissenschaft verkörpern neuronale Netzwerke eher die unbewussten Vorgänge im menschlichen Gehirn, die der Mustererkennung und Sinnesverarbeitung zugrunde liegen. In den Anfängen herrschte in der KI-Forschung der symbolverarbeitende Ansatz vor, doch heute gibt es eine größere Ausgewogenheit und die Stärken der konnektionistischen Systeme werden eingesetzt, um die Schwächen der Symbolverarbeitung auszugleichen.

- *Regelbasierte Ansätze:* Zu den ältesten und bewährtesten Formen der Wissensrepräsentation gehören die sogenannten WENN-DANN-Regeln. Einerseits stellen sie Wissen in

---

<sup>51</sup> Vgl. Varela 1990.

einer gut verständlichen Weise dar, andererseits lassen sich (deterministische) Regeln mit Hilfe der klassischen Logik adäquat verarbeiten. Die regelbasierten Systeme fundieren also auf gut verstandenen und erprobten Techniken mit Tradition. Gerade in klar strukturierten Bereichen, bei denen es lediglich auf 0–1-Entscheidungen ankommt, haben sie sich bestens bewährt.

- *Konnektionistische Ansätze*: Eine besondere Eigenheit intelligenten Verhaltens ist die Lernfähigkeit. Menschen lernen aus Erfahrungen, aus Beispielen, aus Versuchen, durch Unterweisung, aus Büchern usw. Inzwischen ist es gelungen, verschiedene Formen menschlichen Lernens zu modellieren und dies für den Einsatz in wissensorientierten Systemen auszunutzen. In diesem Buch werden sowohl Neuronale Netze als auch Fuzzy-Ansätze vorgestellt. Letztere deshalb, weil es in vielen Anwendungsfällen erforderlich sein wird, Unsicherheiten oder Vagheiten in dem verwendeten Wissen mit sogenannten Sicherheitsfaktoren oder Wahrscheinlichkeitswerten zu belegen. Dabei sind natürlich entsprechende Inferenzmechanismen notwendig, die diese numerischen Werte verarbeiten.
- *Agenten* können nicht nur ihre Umgebung wahrnehmen und darauf reagieren, sondern sind in der Lage, ihr Agieren zu planen, zu kommunizieren und aus Erfahrung zu lernen. Schon an dieser Aufgabenstellung erkennt man deren hybriden Charakter, da sie zu deren Erfüllung sowohl Produktionsregelwissen verarbeiten, als auch mit eher vagem Wissen umgehen, als auch lernfähig sein müssen.

Wie kann man als ein menschliches Wesen das Ganze der Intelligenz erfassen? Offenbar nur unter der Bedingung, dass diesem selbst so etwas wie eine „Logik“ zugrunde liegt. Tatsächlich ist die Intelligenz das, was sie ist, nur Kraft der ihr inhärenten Eigenschaften, die ihrerseits kein An-sich-Seiendes, sondern logischer Natur sind: So kann auch das Fallgesetz selbst nicht fallen, das Gesetz der Beschleunigung sich nicht selbst beschleunigen, die Bauvorschrift für ein Robotersystem ist selbst noch kein Robotersystem. Die folgenden Verfahren sind, wie überhaupt wissenschaftliche Theorien, zunächst einmal von Wissenschaftlern erdachte Modelle. Wären sie freilich nichts weiter als das, so wären sie überhaupt überflüssig. Von Nutzen können sie demnach nur insoweit sein, als dass durch die in ihnen entfalteten Möglichkeiten zugleich etwas von der Logik der Intelligenz zum Vorschein kommt und dieses sich wiederum gewinnbringend zum Bau von intelligenten Robotersystemen verwenden lässt.

Um Daten, Information und Wissen verarbeiten zu können, müssen diese Konzepte durch Symbole dargestellt und entsprechend verarbeitet werden. Das Wissen wird dabei oft in Form von Regeln, logischen Formeln, semantischen Netzen, Frames oder ähnlichen Konzepten dargestellt. Nahezu alle Darstellungsmethoden beruhen auf der Verwendung von Symbolen zur Identifikation elementarer Objekte. Ein Symbol ist dadurch gekennzeichnet, dass es als solches „für etwas steht“, d. h. mit dem Symbol wird ein Gegenstands-bewusstsein intentional erzeugt. Komplexe Objekte werden dann durch zusammengesetzte Strukturen auf Basis von Symbolen dargestellt und verarbeitet. Dabei gilt das Verarbeitungsprinzip der algebraischen Berechnung, das besagt, dass ein Symbol nur ein Symbol

in der Form sein kann, wie es für das zu verarbeitende System erkennbar ist. Bereits aus dieser Tatsache lassen sich basale Grundsätze ableiten, die für die Robotik und dort in der Ausprägung des Symbolverarbeitungsansatzes wichtige Orientierungspunkte sind:<sup>52</sup>

- Der Mensch als ein Organismus in einer informationsbeladenen Umgebung, nimmt aus letzterer Informationen auf, um sie intern zu repräsentieren und durch Symbolmanipulation zu verarbeiten.
- Kognition basiert auf Informationsverarbeitung.
- Informationsverarbeitung beruht auf der Manipulation von Symbolen im Rahmen symbolischer Repräsentationen.
- Symbole werden durch Regeln manipuliert, die sich nur auf die syntaktische Form der Symbole und nicht auf die Semantik beziehen.
- Die Angemessenheit der Funktionsweise eines solchen kognitiven Systems ist dann bereits gegeben, wenn es zu einer adäquaten Repräsentation der realen Welt und zu einer erfolgreichen Lösung gestellter Probleme führt.

Der erste Grundsatz ist die Basis des Cognitive Computing und auch des konnektionistischen Ansatzes.

Beim späteren konnektionistischen Ansatz wird allerdings Informationsverarbeitung nicht als Manipulation von Symbolen nach auf einer Syntax aufbauenden Regel realisiert, wie später noch zu sehen sein wird.

Wissen ist hierbei darstellbar durch endliche Strukturen, die aus diskreten atomaren Symbolen zusammengesetzt sind, in Übereinstimmung mit einer endlichen Anzahl syntaktischer Relationen. Systeme des Symbolverarbeitungsansatzes vereinen sowohl die funktionalistischen als auch die realistischen Aspekte der Simulation in sich. Einerseits soll explizit nicht die physische Realisierung der Informationsverarbeitung durch diese Systeme nachgeahmt werden. In diesem Sinne sind es funktionalistische Simulationen. Andererseits ist es das erklärte Ziel, mit den Systemen die Informationsverarbeitung kognitiver Systeme zu simulieren. Dies macht den realistischen Aspekt der Simulation kognitiver Systeme aus. Diese Etikettierung von Simulationen als realistisch oder funktionalistisch ist dabei stets relativ zu der jeweils betrachteten Ebene oder Schicht des simulierten Realitätsausschnitts. Der Nutzen von Simulationen ist gerade darin zu sehen, dass es nicht notwendig ist, das zu simulierende System (Original) in allen Einzelheiten nachzuahmen. Wäre dies nötig, so bedeutet das eine vollständige Replikation des Originals im simulierten System. Dies würde jedoch die Idee der Simulation ad absurdum führen. Insofern haben funktionalistische Simulationen kognitiver Systeme zum Ziel, ein adäquates Verhalten zu produzieren, um eine bestimmte Aufgabe zu lösen. Das Ziel dabei ist jedoch nicht, eine strukturtreue Nachbildung der kognitiven Prozesse des Originals zu erreichen.

---

<sup>52</sup> Vgl. Rolf Struve. *Wissensbasierte Systeme – Utopie und Realität*. In: Journal for General Philosophy of Science. Ausgabe 23/Nr. 2. S. 315–322. 1992.

## Produktionsregelsysteme

Unter einem *Produktionsregelsystem* wird ein auf Produktionsregeln basiertes und damit wissensbasiertes System mit Fähigkeiten zur Schlussfolgerung bzw.- Inferenzausführung verstanden. Die folgenden Anforderungen werden an ein solches System gestellt: Fähigkeit zur Wissensrepräsentation bzw. -manipulation, Fähigkeit zur effektiven Inferenzausführung, Selbsterklärungsfähigkeit, Fähigkeit zum Wissenserwerb, anwenderfreundliche Kommunikation, Einsetzbarkeit auch für komplexe Aufgabenstellungen. Die Anforderungen bezüglich der Einsetzbarkeit für spezifische, hinreichend komplexe Aufgaben in einer Problemdomäne bringt folgende Herausforderungen mit sich: großer Problemraum, Erfordernis von Spezialwissen, was heterogen und unscharf vorliegt und das sich von Zeit zu Zeit ändert. Wissen kann nicht dabei nur auf eine deklarative Art repräsentiert werden, d. h. als eine Menge von Aussagen, die wahr sind, sondern auch in Form von Regeln, die zum Ausdruck bringen, was ein Robotersystem tun, beziehungsweise welche Schlussfolgerungen es in unterschiedlichen Situationen ziehen soll. Insofern lassen sich die Probleme, zu deren Lösung Produktionsregelsysteme besonders geeignet sind, wie folgt charakterisieren:

- Die Problemstellungen verfügen über einen hohen Grad an Indeterminiertheit. Daraus ergibt sich die Gefahr einer kombinatorischen Explosion, die eine vollständige und exakte Lösung aus Zeit- und Platzgründen nahezu unmöglich macht.
- Es liegen bezüglich der Problemstellungen keine mathematisch klar umrissenen Strukturen vor. Die Inhalte stammen oft aus dem alltäglichen Leben und sind entsprechend unzulänglich, unvollständig, schlecht strukturiert, unsicher, manchmal sogar inkonsistent und falsch. Trotzdem können menschliche Experten auch für solche Probleme oft brauchbare Lösungsansätze liefern.

Regelbasierte Systeme behandeln solche Probleme dadurch, dass sie das über eine Problemdomäne vorhandene Wissen explizit *deklarativ* repräsentieren.

Mögliche Arten des Wissens sind:

- Wissen über elementare Tatsachen und die Art und Weise, wie Schlussfolgerungen gezogen werden,
- Wissen über spezielle Situationen,
- Wissen über Vorgehensweisen, Algorithmen und Kontrollstrukturen,
- Wissen über allgemeine Gesetzmäßigkeiten und Alltagswissen.

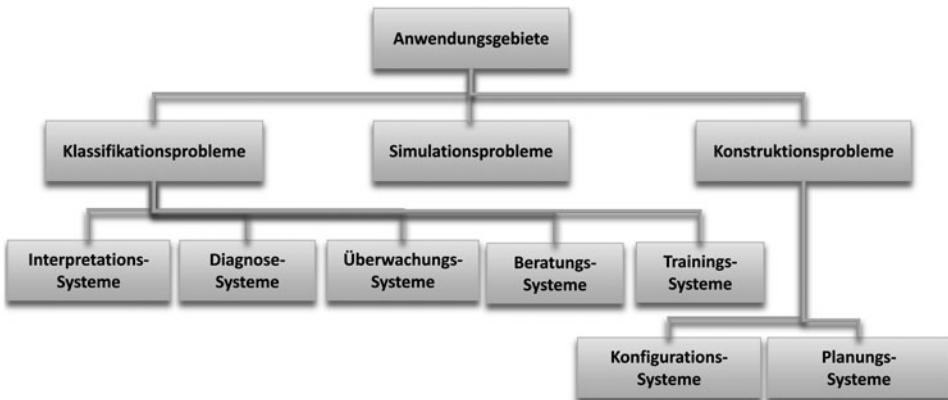
Eine Variante von regelbasierten Systemen stellen die sogenannten *Expertensysteme* dar. Das Ziel von Expertensystemen ist es, Wissen und Fähigkeiten aus einem definierten Fachgebiet zur Verfügung zu stellen, über die sonst nur Experten verfügen. Ein Expertensystem ist demnach ein Programm, das in einem definierten Anwendungsbereich die spezifischen Problemlösungsfähigkeiten eines menschlichen Experten erreicht oder gar übertrifft. Grundsätzlich bezeichnet man als Experten einen speziell ausgebildeten und

erfahrenen Fachmann, der auf einem abgegrenzten Fachgebiet über umfassendes Wissen verfügt. Darüber hinaus ist er in der Lage, mit diesem Wissen aktiv umzugehen, er kann es anwenden, d. h. er kann aus seinem Wissen Schlussfolgerungen ziehen, sowie einmal getroffene Entscheidungen erläutern und begründen. Dabei kann sich sein fachliches Wissen aus sehr vielen Elementen zusammensetzen: Fakten, Regeln, Allgemeinwissen, Erfahrungswissen, Intuition und Kreativität. Aus dieser Sicht handelt es sich bei einem Experten um eine Person, die aufgrund der folgenden Eigenschaften in der Lage ist, ein an ihn herangetragenes Problem aus einem bestimmten Bereich zu lösen. Ein Experte besitzt Wissen über den Problembereich, kann Wissen erwerben und damit hinzu lernen, kann Probleme lösen, kann seine Vorgehensweise beim Problemlösen erklären, kann Wissen reorganisieren, handelt in den meisten Fällen effizient, kann aber auch Regeln übertreten, kann die eigene Kompetenz beurteilen und sich ggf. mit Anstand zurückziehen.<sup>53</sup> Schon anhand dieser Beschreibung wird deutlich, dass wahre Experten sehr rar und teuer sind. Das Wissen des Experten bildet dabei die Grundlage für sein Handeln. So verfügt der Experte über eine Begriffswelt und kennt den Zusammenhang zwischen diesen Begriffen. Er arbeitet aber auch mit vagem Wissen. Darüber hinaus kennt der Experte die Zusammenhänge zu anderen Wissensgebieten und hat Hintergrundwissen über bestimmte Sachverhalte. Letzteres bezeichnet man als Common-Sense-Wissen. Wo er selbst nicht unbedingt Wissen abrufen kann, verfügt er oftmals über Verweise auf Wissensquellen, wie beispielsweise Bücher, Fachartikel etc. Daher versucht man, deren Wissen mit Hilfe von Programmen einem größeren Kreis von Anwendern jederzeit zur Verfügung zu stellen. Aber auch die Experten selbst bedienen sich der Expertensysteme, indem letztere dazu eingesetzt werden, um Experten bei ihrer Arbeit zu unterstützen. Zentrales Anliegen hierbei ist es, fachspezifisches Wissen eines Experten als wissensbasiertes System zu simulieren. Das Ergebnis ist auch hier ein Expertensystem:

- in dem Wissen über ein spezielles Fachgebiet gespeichert wird (Fähigkeit zur Wissensrepräsentation bzw. -manipulation),
- das aus dem gespeicherten Wissen Schlussfolgerungen ziehen kann (Fähigkeit zur effektiven Problemlösung bzw. Inferenzausführung),
- das Lösungen zu konkreten Problemen dieses Fachgebietes produziert (Fähigkeit des Erreichens von Expertenniveaus bezüglich der Leistungsfähigkeit),
- das die Lösungswege selbst erklärt (Fähigkeit zur Selbsterklärung, was Reflexion über eigene Inferenzprozesse voraussetzt),
- das mit dem Anwender kommuniziert (Fähigkeit zur nutzerfreundlichen Kommunikation mit dem Anwender),
- und das sich problembezogen einsetzen lässt (Fähigkeit zur Einsetzbarkeit für spezifische, hinreichend komplexe Aufgaben auf einem Gebiet, das sich durch folgende Eigenschaften auszeichnet: Problemraum groß, Spezialwissen erforderlich, Wissen oft heterogen und/oder unscharf und/oder zeitlich ändernd).

---

<sup>53</sup> Siehe auch Dörner 1976.



**Abb. 5.36** Aufgabenstellungen von Produktionsregelsystemen

Produktionsregelsysteme unterscheiden sich damit grundlegend von konventionellen Software-Systemen. Zwar werden auch in konventionellen Systemen, beispielsweise Datenbanken, Informationen mit speziellen Wissensinhalten gespeichert. Doch im Gegensatz zu Produktionsregelsystemen sind Datenbanken nicht in der Lage, dieses Wissen selbst weiterzuverarbeiten. Insofern setzt sich die Intelligenz eines konventionellen Systems aus der Summe der implementierten Algorithmen und Datenstrukturen zusammen.<sup>54</sup> Da gewöhnlich auch keine eigene Wissensrepräsentation vereinbart ist, kann dieses Wissen auch nicht direkt angesprochen werden. Von einem Produktionsregelsystem erwartet man dagegen, dass der Anwender das gespeicherte Wissen jederzeit anzeigen lassen und modifizieren kann. Die Intelligenz eines Produktionsregelsystems wird dem Anwender durch die Expertenfunktionen, sowie durch die Methoden der Wissensverarbeitung zur Verfügung gestellt.

Damit verbunden ist die geforderte Eignung für folgende Aufgabenstellungen: Interpretation und Deutung aus beobachtbaren Daten, Diagnose und Therapie, Konfiguration, Konstruktion und Entwurf, Fehlerortung, Reparatur, Planung, Überwachung und Monitoring, Prognose, Klassifikation und Erkennung, Instruktion und Training, Beratung und Konsultation sowie Koordinierung und Steuerung.

So lassen sich Produktionsregelsysteme auf unterschiedliche Aufgabenstellungen bzw. Problemtypen ansetzen (Abb. 5.36).

*Interpretationsysteme* sind in der Lage, aus beobachtbaren Daten Objekt- oder Situationsbeschreibungen abzuleiten. Bei *Diagnosesystemen* übernehmen Produktionsregelsysteme die Funktion, aufgrund von bestimmten Anzeichen (den Symptomen) Fehler in einem System aufzufinden und möglicherweise Vorschläge für geeignete Aktionen (Therapievorschläge, Reparaturanweisungen etc.) zu erarbeiten. So wird beispielsweise in einem Expertensystem aufgrund von beobachteten Symptomen diagnostiziert, welche Krankheit

<sup>54</sup> Vgl. Ottmann und Widmayer 2002.

vorliegen kann und entsprechend dieser Diagnose ein entsprechendes Rezept erstellt. Ein anderes Produktionsregelsystem spürt aufgrund der festgestellten Störung eines technischen Systems die Fehlerquelle auf. Ein solches Produktionsregelsystem klassifiziert ein vorhandenes Problem mit Hilfe des Wissens in der Wissensbasis, d. h., es versucht, dieses Wissen auf eine ganz konkrete Situation anzuwenden. Die Daten, die das Problem umschreiben, werden entweder durch den Anwender im Rahmen einer Konsultation erfasst, beziehungsweise von ihm im Rahmen eines Dialogs erfragt, oder sie stammen direkt aus den betrieblichen Prozessen. Das Problem wird anhand des vorliegenden Datenmaterials zunächst klassifiziert, und erst dann kann eine entsprechende Diagnose gestellt werden. Die Diagnose kann immer nur aus den im System vorhandenen Diagnosemöglichkeiten stammen. Sollte das System eine eindeutige Zuordnung vornehmen können, dann ist eine eindeutige Diagnose möglich. Ob letztlich die Diagnose zutrifft, muss der Anwender selbst entscheiden. Sollte hingegen keine eindeutige Zuordnung möglich sein, d. h., das System kann das Problem keiner vorhandenen Problemklasse zuordnen, so erfolgt ein Diagnosevorschlag mit Vorbehalt. Letzteres bezeichnet man auch als „eine Diagnose mit Produktionsregelsystem-üblichem Vorbehalt“. *Überwachungssysteme* dienen zur Steuerung von automatisierten Regelprozessen, indem Vergleiche von Beobachtungen mit Sollwerten vorgenommen werden. Zusätzlich gewährleisten solche Systeme die Einhaltung kritischer Systemparameter und die Beseitigung der Abweichungen von vorgegebenen Normalzuständen. Bei *Beratungssystemen* liegt die primäre Aufgabe in der Unterstützung beim Analysieren von komplexem und umfangreichem Datenmaterial. Dabei steht die Beratungsfunktion im Vordergrund. Der Anwender klärt im Rahmen eines Dialoges mit dem Expertensystem und dessen inhärentem Wissen aus der Wissensbasis einen konkreten Sachverhalt. Dazu beschreibt er durch Eingaben das anstehende Problem und erhält von der Wissensbasis aufgrund der gespeicherten Fakten und Regeln eine entsprechend ausgestaltete Beratung. Auch hier wird die Bewertung der Lösungen dem Anwender überlassen. So übernehmen Produktionsregelsysteme die Beratung der Kunden bei einem Autokauf. Der Kunde gibt auf Anfragen des Systems die gewünschten Eigenschaften seines späteren Autos ein. Die Beratungsleistung dieses Systems besteht dann darin, dass das System dem Kunden anhand der vorgegebenen Kriterien einen passenden Wagen vorschlägt. *Trainingssysteme* erteilen Instruktionen zur Anleitung und Unterweisung auf einem bestimmten Spezialgebiet bzw. in der Bewältigung komplizierter Situationen. *Design- oder Entwurfssysteme* umfassen das Entwerfen oder Zusammenstellen von Systemen aus Einzelkomponenten.

Beispiele hierfür sind der architektonische Entwurf eines Bürogebäudes, die Konstruktion eines Flugzeugs oder die Konfiguration von EDV-Systemen.

Gerade dem letzten Anwendungsbeispiel ist es zu verdanken, dass solche Systeme oftmals als Konfigurationssysteme bezeichnet werden. Dabei sind die Entwurfsfunktionen dem Anwender im Dialog zugänglich. Der eigentliche Entwurfsprozess gestaltet sich dann dahingehend, dass die spezifizierten Anforderungen auf dem Bildschirm direkt als Entwurf dargestellt werden. *Planungssysteme* dienen dem Entwurf einer Folge von Aktionen zum

Erreichen eines Ziels, wohingegen *Simulationssysteme* aus einem Ausgangszustand Folgezustände herleiten. Exemplarisch lassen sich für den Einsatz von Produktionsregelsystemen in Robotersystemen folgende Aufgabentypen angeben:

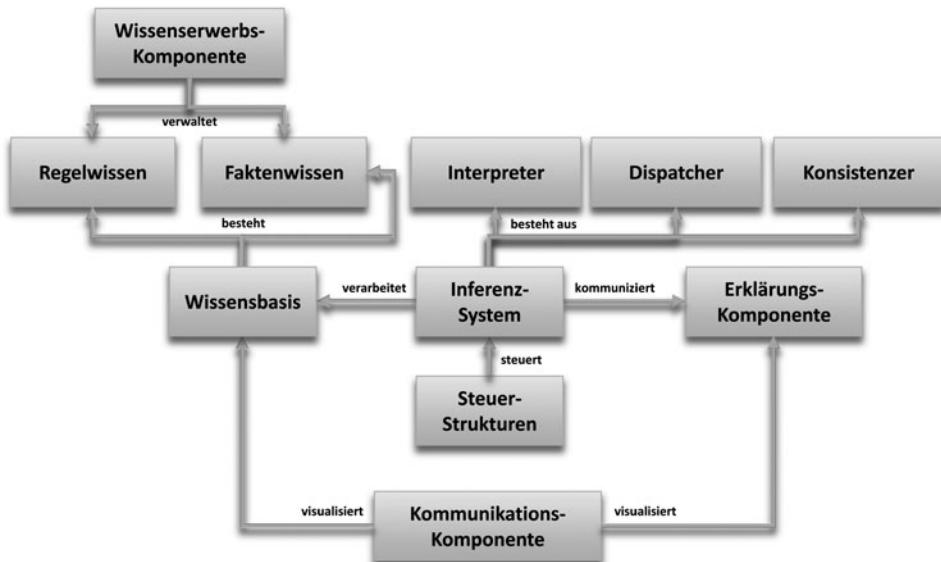
- *Kontrolle*: Regelung des Systemverhaltens gemäß der Spezifikation
- *Entwurf*: Konfiguration von Robotersystemen anhand von Anforderungen
- *Diagnose*: Herleitung von Problemfällen aus Beobachtungen
- *Lernen*: Diagnose und Korrektur von Fehlern und Fehlverhalten
- *Interpretation*: Herleitung von Situationsbeschreibungen
- *Überwachung*: Vergleich von Echtzeitverhalten und Erwartungen
- *Planung*: Auswahl und Anordnung von zielorientierten Interoperationen
- *Vorhersage*: Herleitung wahrscheinlicher Folgen aus gegebenen Situationen
- *Präskription*: Empfehlung und Behebung von Fehlfunktionen
- *Empfehlung*: Bestimmung von Interoperationsalternativen
- *Simulation*: Modellierung der Interoperationen von Robotersystemen bzw. deren Komponenten

In einem idealisierten Produktionsregelsystem lassen sich im Wesentlichen fünf Komponenten unterscheiden:

- *Wissensbasis*: Sie enthält sowohl Fakten als auch Regelwissen über das Anwendungsbereich.
- *Inferenzsystem*: Dieses System findet auf Anfragen bzw. Konsultation Lösungen zu einem bestimmten Problem.
- *Kommunikationskomponente*: Diese Komponente ermöglicht die Kommunikation mit dem System in einer nutzerfreundlichen Sprache.
- *Erklärungskomponente*: Sie übernimmt die Erklärung, warum und auf welche Weise eine bestimmte Lösung gefunden oder nicht gefunden wurde.
- *Wissenserwerbskomponente*: Diese Komponente unterstützt die Eingabe, Wartung und Editierung von Wissenselementen.

Das Zusammenwirken dieser einzelnen Komponenten ist in der folgenden Abbildung dargestellt. Im Allgemeinen enthält ein Expertensystem die in der Abbildung aufgezeigten Komponenten, die sich allerdings bezüglich ihrer Ausprägung je nach System unterscheiden. Diese Realisierung von Expertensystemfunktionen über einzelne, in sich abgeschlossene Komponenten bringt den Vorteil mit sich, dass Weiterentwicklungen komponentenorientiert vorgenommen werden können. Jedes Expertensystem benötigt im Kern mindestens eine Wissensbasis sowie eine Inferenzkomponente. Aber auch die Ausgestaltung der Schale, bestehend aus Erklärungs-, Dialog-, Wissensakquisitions- und Problemlösungskomponenten entscheidet über die Leistungsfähigkeit und damit über die „Intelligenz“ des Expertensystems (Abb. 5.37).

In der *Wissensbasis* ist das gesamte Wissen des Expertensystems gespeichert. Neben dem Expertenwissen enthält sie auch das spezifische Wissen jeder einzelnen Expertensys-



**Abb. 5.37** Komponenten von Produktionsregelsystemen

temfunktion. Insofern sind dort nicht nur Fakten, d. h., Datensätze, sondern auch Regeln, d. h., Vorschriften zur Wissensverarbeitung gespeichert. Die Wissensbasis enthält also Regeln und Fakten, um das Wissen zu organisieren und zu strukturieren. Die Darstellung eines solchen Wissens lässt sich, grob betrachtet, in vier Kategorien einteilen:

- deklaratives Wissen: Fakten und Relationen,
- prozedurales Wissen: Verfahren und Vorschriften,
- Kontrollwissen: Steuerungsverfahren zur Verarbeitung von deklarativen und prozeduralen Wissen,
- vages Wissen: Erfahrungswissen und Heuristiken.

Der Inhalt einer solchen Wissensbasis kann grob unterteilt werden in:

- fachspezifisches Expertenwissen, das sich während einer Arbeitssitzung des Anwenders ändern kann,
- auf ein konkretes Problem bezogenes Faktenwissen, das der Anwender während der Konsultation eingibt,
- Zwischen- und Endergebnisse, die das Produktionsregelsystem während der Arbeitssitzung erarbeitet hat und dem Anwender mitteilt.

Die Wahl der für die Wissensrepräsentation in Produktionsregelsystemen verwendeten Repräsentationsschemata hängt vom Anwendungsgebiet bzw. von der Aufgabenklasse ab.

Fakten, die deskriptives Wissen über Objekte, Situationen, Zustände darstellen, können durch Frames, semantische Netze oder durch prädikatenlogische Ausdrücke dargestellt werden.

Semantische Netze stellen eine Form der Repräsentation von Wissen im Gedächtnis dar, die semantische Beziehungen von Bedeutungseinheiten modellieren. Entitäten (abstrakte oder materielle Objekte der beobachteten Welt) werden als Einheiten in Form von Knoten innerhalb des semantischen Netzes als Graphen repräsentiert. Verbindungen als Kanten zwischen den Einheiten repräsentieren Relationen zwischen den Entitäten. Als Mechanismen zur Weiterleitung von Information in solchen Netzmodellen wird unter anderem die Aktivierungsausbreitung verwendet. Bei der Aktivierungsausbreitung senden Einheiten im Netzwerk, die eine Mindestaktivierung aufweisen, Aktivierungen an benachbarte Einheiten. Dabei kann sich die Aktivierungsausbreitung nur bis zu direkt assoziierten Knoten über das gesamte Netzwerk erstrecken, wobei das Ausmaß der Aktivierung mit dem Abstand von der Ausgangseinheit abnimmt.

Zur Unterstützung der Inferenzprozesse können Fakten durch zusätzliche Eigenschaften angereichert werden:

- *Aktualität* (recency): Diese gibt den Grad der Neuheit eines Fakts an. Dabei kann man beispielsweise auf eine Heuristik zurückgreifen, die davon ausgeht, dass die aktuellsten Fakten auch gleichzeitig die bedeutsamsten sind.
- *Gewissheitsgrad* (certainty): Diese gibt den Grad der Sicherheit oder Gewissheit an, mit der ein Fakt gilt. Gewöhnlich verwendet man hierzu Wahrscheinlichkeiten oder Akzeptanzgrade, die subjektive Maße für die Gültigkeit von Wissenselementen darstellen.

Das *Regelwissen* wird in Produktionsregeln in der allgemeinen Struktur

< Faktum >< Bedingung >  $\Rightarrow$  < Zielausdruck >

bzw.

< Muster >< Bedingung >  $\Rightarrow$  < Ziel >

oder

< Muster >< Bedingung >  $\Rightarrow$  < Aufgabe >

spezifiziert. Dabei muss das Muster die gleiche Grundstruktur wie die Fakten haben. Bei Expertensystemen mit logischem Unterbau sind Muster und Bedingung nicht voneinander getrennt, sondern vielmehr in einem einzigen Ausdruck, der Prämisse, konzentriert. Als Ziel kommen Objekt- bzw. Situationsbeschreibungen oder Aktionen in Frage. Man

bezeichnet solche Ziele im Fachjargon auch oftmals als Kontext. Sehr häufig werden Produktionsregeln zur kontrollierten Beeinflussung der Steuerstrategie des Inferenzsystems benutzt. In diesem Fall ist der auf der rechten Seite der Regel stehende Ausdruck eine Aufgabe, die den Ablauf der einzelnen Abarbeitungsschritte des Systems beeinflusst.

Das Inferenzsystem verarbeitet diese Strukturen mit geeigneten Methoden, beispielsweise logischen Inferenzen, Vererbungsmechanismen oder Constraintpropagierung. Das Wissen wird in einem Produktionsregelsystem auf verschiedene Arten repräsentiert. Inhaltlich lässt sich Wissen in Objektwissen, d. h. Wissen über die Gegenstände eines Problembereichs, und Kontrollwissen, d. h. Wissen über die Art der Lösung und Abarbeitung, unterteilen. Das Kontrollwissen ist seiner Natur nach Metawissen im Verhältnis zum Objektwissen. Bei Systemen auf Basis einer Prädikatenlogik sind die Fakten und Bedingungen nicht voneinander getrennt, sondern zu einem einzigen Ausdruck, der sogenannten Prämisse, zusammengefasst. Als Zielausdrücke kommen Objekt- bzw. Situationsbeschreibungen oder auch Aktionen in Frage. Solche Produktionsregeln werden auch zur Steuerung des Inferenzsystems verwendet. In diesem Fall ist der auf der rechten Seite der Regel stehende Ausdruck beispielsweise eine Aufgabe (Task), die den Ablauf der einzelnen Abarbeitungsschritte des Systems verändert. Als Verarbeitungsrichtungen für die Regeln kommen sowohl die *Vorwärtsverkettung* (forward chaining) als auch die *Rückwärtsverkettung* (backward chaining) in Frage. Bei Regeln, die in Vorwärtsrichtung angewendet werden, stehen auf der rechten Seite sehr oft Aktionen, die Veränderungen in der Wissensbasis durchführen (Eliminieren, Verändern von Fakten bzw. Regeln, Konstruktion neuer Wissenselemente). Die Verarbeitungsrichtung, die von den Fakten ausgeht und auf ein bestimmtes Ziel hin orientiert ist, wird als *Bottom-Up-Verarbeitung* bezeichnet.

Bei der Vorwärtsverkettung wird die Lösung von einem Anfangszustand in Richtung eines Zielzustandes gesucht. Dabei wird von problembezogenen Fakten bzw. Daten ausgegangen. Die Inferenzkomponente prüft, welche Regeln aufgrund der eingegebenen Fakten zu beachten sind. Da von Daten ausgegangen wird, spricht man auch von einem datengesteuerten Inferenzmechanismus. Diese Strategie wird vor allem dann angewendet, wenn die Anzahl der möglichen Lösungen sehr groß und das Ziel eher unbekannt ist. Hier muss also eine mögliche Lösung erst gesucht bzw. konstruiert werden.

#### **ALGORITHMUS:** Vorwärtsverkettung

##### **WIEDERHOLE**

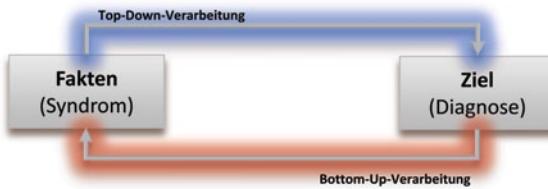
1. **Finde** alle Regeln, deren Bedingungen (IF-Teil) erfüllt sind.
2. **Wähle** unter der Verwendung der Konfliktlösungsstrategien eine Regel aus.
3. **Führe** die Aktionen durch und ändere ggf. den aktuellen Arbeitsspeicher.

**BIS** (keine Regeln mehr ausgelöst werden können

**ODER** bis zum HalteSymbol im Arbeitsspeicher)

##### **ENDE-ALGORITHMUS**

**Abb. 5.38** Verarbeitungsrichtungen



Bei Regeln, die in Rückwärtsrichtung angewendet werden, sind auf der rechten Seite keine Aktionen möglich. Hier müssen die Zielausdrücke die gleiche Grundstruktur haben, wie die Fakten auf der linken Seite. Andernfalls wäre eine Rückwärtsverkettung nicht möglich. Bei der Rückwärtsverkettung wird die Lösung vom Zielzustand in Richtung Anfangszustand gesucht. Hier wird von einer bekannten oder vermuteten Lösung ausgegangen. Dann werden Fakten und Regeln gesucht, mit denen die Hypothese im Rahmen eines Verifikationsprozess bestätigt oder im Rahmen eines Falsifikationsprozesses widerlegt werden kann. Da hierbei von den möglichen Lösungszielen ausgegangen wird, spricht man auch von einer zielgerichteten Strategie. Sie wird vor allem in solchen Anwendungsfällen eingesetzt, wo die möglichen Ergebnisse bekannt und deren Anzahl noch überschaubar sind.

**ALGORITHMUS:** Rückwärtsverkettung

**WIEDERHOLE**

**WENN** ein Ziel unter den anfänglichen Fakten ist  
**DANN** ist das Ziel bewiesen.

**ANSONSTEN**

**Finde** eine Regel, die genutzt werden kann, um das Ziel zu folgern  
 UND versuche jede Vorbedingung dieser Regel zu erfüllen.

**WENN** alle Vorbedingungen wahr sind  
**DANN** ist das Ziel auch wahr.

**BIS** ein Ziel bewiesen ist.

**ENDE-ALGORITHMUS**

Die Verarbeitungsrichtung von einem vorgegebenen Ziel zu den Fakten hin wird als *Top-Down-Verarbeitung* bezeichnet (Abb. 5.38).

Die *Ziele* können zur Steuerung des Inferenzsystems ähnlich wie die Fakten durch zusätzliche Eigenschaften angereichert werden:

- **Kosten:** Damit erfolgt eine Abschätzung des zeitlichen bzw. speichertechnischen Aufwandes, der erforderlich ist, das entsprechende Ziel zu erreichen.
- **Priorität:** Gibt das Maß für die Wichtigkeit des zu erreichenden Ziels an und bildet so im Inferenzsystem die Grundlage für die Reihenfolge der Zielbearbeitung.

- *Neuheitsgrad*: Dieser bestimmt die zeitliche Aktualität des Ziels. Im Regelfall geht man davon aus, dass dem zuletzt generierten Ziel der höchste Neuheitsgrad zukommt.
- *Status*: Diese Angabe drückt aus, welche Stellung ein Ziel im Verarbeitungsprozess besitzt. Dies betrifft beispielsweise die Einschätzung, ob das Ziel aktiv, suspendiert (ausgesetzt), gelöst oder verlassen werden kann.

Sowohl bei der Vorwärts- und Rückwärtsverkettung, der Tiefen- und Breitensuche, als auch bei anderen, hier nicht behandelten Algorithmen, kann Metawissen eingesetzt werden. Das sind hier Informationen aus dem Ableitungs- bzw. Schlussfolgerungsprozess, die beim Ableiten als Zwischenergebnisse oder Weginformationen in Form neuer Regeln abgelegt werden können. Sollte sich das gleiche Problem erneut stellen, wird auf diese Zwischenergebnisse der Wegeinformationen zurückgegriffen. Solche Algorithmen werden oftmals als universelle Inferenzverfahren in den Produktionsregelsystemen implementiert. Ein Inferenzverfahren als Universalalgorithmus ist dabei recht einfach, da die Schlussfolgerungen immer nach dem gleichen Algorithmus gezogen werden. Jeder, der das implementierte Inferenzverfahren benutzt, kann dann auf diesen Universalalgorithmus zugreifen. Dadurch reduziert sich der Entwicklungsaufwand beim Erstellen einer Anwendung auf das Formulieren der Problembeschreibung.

Die *Inferenzkomponente* verkörpert den Schlussfolgerungsmechanismus eines Produktionsregelsystems. Diese Komponente besteht aus mehreren miteinander wechselwirkenden Subsystemen:

- *Interpreter*: Diese Subkomponente ist für die unmittelbare Regelanwendung zuständig. Diese verfügt über einen Pattern-Matcher zur Verarbeitung der linken Seite einer Produktionsregel und einen Exekutor, der die rechte Seite der Produktionsregel ausführt.
- *Dispatcher*: Dieser nimmt die Ziel- bzw. Regelauswahl vor, regelt Konfliktlösungen und koordiniert die zu bearbeitenden Aufgaben.
- *Konsistenz*: Diese Komponente stellt sicher, dass das neu erschlossene Wissen als Hypothese mit der vorhandenen Wissensbasis verglichen und die Konsistenz aufrecht erhalten wird.

Das Inferenzsystem greift dabei auf sogenannte *Steuerstrukturen* zurück, um die Ziele, Aufgaben, Zwischenergebnisse und Hypothesen zu speichern und ggf. zu manipulieren. Insofern stellen diese Steuerstrukturen eine Art operatives Gedächtnis des Produktionsregelsystems dar. Folgende Typen von Steuerstrukturen kommen zur Anwendung:

- *Kellerspeicher* (Stack): Hierbei werden die Ziele in Form von Listen oder Baumstrukturen gespeichert und nach dem Last-In-First-Out-Prinzip abgearbeitet, d. h., dass das zuletzt aufgenommene Element zuerst abgearbeitet wird. Diese Struktur ist besonders für eine Regelverarbeitung nach der Methode der Tiefe-zuerst-Suche geeignet. Allgemein bezeichnet man bei einer solchen Baumstruktur die Anzahl der Kanten, die ein Knoten

von der Wurzel des Baumes entfernt ist, als die Tiefe des Knotens. Derjenige terminale Knoten eines Baumes, der die größte Tiefe besitzt, bestimmt zugleich die Tiefe des Baumes. Bei der Entwicklung eines solchen Baumes gibt es zwei verschiedene Strategien. So kann man, ausgehend von der Wurzel, zunächst alle ihre Nachfolger generieren, in dem man der Reihe nach alle Nachfolger der gleichen Tiefe 1 bildet. Im nächsten Schritt werden auf die gleiche Art alle Nachfolger der Tiefe 2 gebildet, um dann zur nächsten Tiefenstufe überzugehen. Dieses Vorgehen bezeichnet man als Breite-zuerst-Suche. Die andere Möglichkeit besteht darin, dass man von der Wurzel ausgehend nur einen Knoten der Tiefe 1 erzeugt und von diesem aus sofort immer nur einen Nachfolger pro Tiefenstufe erzeugt. Dieses Verfahren bezeichnet man demzufolge als Tiefe-zuerst-Suche.

- **Warteschlangen** (Queues): Auch hier werden Ziele in Form von Listen oder Bäumen aufbereitet, um sie dann allerdings nach *First In-First Out- Prinzip* abzuarbeiten, d. h. das zuerst aufgenommene Ziel wird auch zuerst bearbeitet. Diese Methode eignet sich vor allem für die *Breite-zuerst-Suche*.
- **Agenda:** Hier werden Aufgaben und Ziele in Form von Warteschlangen gespeichert, in denen die Aufgaben und Ziele nach Prioritäten geordnet sind. Hierbei werden die Elemente mit der höchsten Priorität zuerst abgearbeitet.
- **Bäume:** Die Darstellung von Zielen und Aufgaben in Form von Baumstrukturen wird vor allem dann verwendet, wenn die Ziele in einer Hierarchie von Unter- und Oberzielen organisiert sind.
- **Blackboards:** Als solche bezeichnet man zentrale Speicherplätze, die in mehrere Regionen unterteilt sind, die wiederum verschiedenen Abstraktionsniveaus zugeordnet sein können. Eine solche Steuerstruktur ist dann von Vorteil, wenn sich beispielsweise mehrere Produktionsregelsysteme eine Problemstellung teilen und damit eine Ablage von Zwischenergebnissen notwendig wird. Die beteiligten Produktionsregelsysteme können diese Steuerstrukturen zum Austausch der Lösungsergebnisse nutzen.

Für die Verfolgung einer Steuerstrategie in einem Produktionsregelsystem existieren verschiedene Möglichkeiten. Je nach den in einem Inferenzprozess konkurrierenden Wissenselementen kann man zwischen den folgenden Methoden wählen:

- **Konfliktlösung:** Wenn mehrere Regeln alternativ angewendet werden können, muss eine Entscheidung getroffen werden, welche dieser Alternativregeln zur Ausführung gelangen soll. Eine solche Konfliktlösung kann nach verschiedenen Prinzipien erfolgen:
  - *Refraktion:* Gemäß dieses Prinzips dürfen Regeln nur einmal über dieselben Daten angewendet werden. Als identische Regelinstantierungen gelten solche Regeln, deren Bedingungsteil identisch ist.
  - *Datenvorsortierung:* Hierbei werden die Daten zunächst nach einem bestimmten Kriterium, beispielsweise nach dem Aktualitätsgrad, vorsortiert, um dann die Regeln zur Ausführung zu bringen, die mit den Daten höchster Rangordnung zur Übereinstimmung gebracht werden können.

- *Inferenzzuweisung*: Regeln lassen sich dahingehend vorsortieren, indem man ihnen eine bestimmte Inferenzgewichtung zuweist. Eine solche Gewichtung drückt die logische Aussagekraft einer Regel aus. Eine solche Aussagekraft kann beispielsweise davon bestimmt sein, indem man spezielleren Regeln gegenüber allgemeineren Regeln den Vorzug einräumt.
- *Filterung*: Hierunter versteht man eine Vorauswahl von Regeln bzw. Fakten für den Inferenzprozess, in dem diese selektieren Regeln aus der Regelmenge eliminiert werden. Man unterscheidet hierbei zwischen einer Regelfilterung und Datenfilterung.
  - *Regelfilterung*: Hier werden die Regeln nach der Art der Ziele gruppiert, die sie erfüllen. Es werden nur diejenigen Regeln für den Inferenzprozess herangezogen, die für ein vorgegebenes Ziel auch wirklich relevant sind.
  - *Datenfilterung*: Dabei werden nur solche Fakten zur Inferenz zugelassen, die eine vorgegebene Zulässigkeitsschwelle überschreiten. Diese Schwelle kann beispielsweise durch einen bestimmten Aktualitätsgrad oder durch ein bestimmtes Akzeptanzmaß definiert sein.
- *Metaregeln*: Bei dieser Methode wird die Steuerstrategie durch Verwendung eines übergeordneten Regelapparates beeinflusst, der in die Arbeit des Dispatchers aktiv einwirkt, indem Veränderungen in der Agenda vorgenommen werden.

Eine nicht zu unterschätzende Rolle für die Akzeptanz und damit für die praktische Anwendung eines Produktionsregelsystems spielt die Schnittstelle zwischen Anwender und System, die sogenannte *Kommunikationskomponente*. Hier ist gefordert, neben der Einhaltung software-ergonomischer Aspekte eine möglichst natürliche und unkomplizierte Kommunikation mit dem Produktionsregelsystem zu ermöglichen. Die Formen der Kommunikation zwischen Anwender und System reichen von graphisch unterstützten, menügesteuerten Dialogen, über formalsprachliche Mittel bis hin zu natürlich-sprachiger Kommunikation.

Der Wissenserwerb und dessen Repräsentation mit Hilfe der *Wissenserwerbskomponente* ist sicherlich nicht nur ein Engpass in der Entwicklung von Produktionsregelsystemen, sondern der Garant für das Gelingen solcher Projekte überhaupt. Im einfachsten Fall werden sprachliche Mittel über Editoren zur Modellierung von Fakten und Regeln verwendet. Die Wahl der Mittel hängt auch von der Art der Wissensquellen ab, die man zu Rate ziehen muss. Als Wissensquellen kommen in Frage: Textwissen aus Büchern, Dokumentationen, Aufzeichnungen etc., Sammlungen von Daten und Fakten und das Expertenwissen von Spezialisten.

Ebenfalls für die Akzeptanz und für die praktische Tauglichkeit wichtig ist das Vorhandensein einer der Problemdomäne adäquaten *Erklärungskomponente*. Sie setzt das Vorhandensein von Metawissen und die Fähigkeit des Produktionsregelsystems voraus, über seine eigenen Inferenzprozesse Auskunft zu geben. Generell gilt, dass der Weg auf dem die Lösung bzw. Entscheidung gefunden wurde, für den Anwender transparent bzw. nachvollziehbar sein muss. Das System muss seine eigenen Schlussfolgerungen und Problemlösungsschritte erklären bzw. begründen können.

## Fuzzy Logik

Der Begriff der *Fuzzy-Logik* taucht in der klassischen Literatur in zweierlei Deutungen auf. Eine eher enge Deutung beschreibt damit ein im mathematischen Sinne logisches System mit dem Ziel, Modelle für die Erscheinungsformen der menschlichen Beweis- und Entscheidungsführung aufzustellen. Diese Erscheinungsformen werden darin in symbolischer Form eher annähernd als exakt festgelegt. Hingegen beschreibt die weite Deutung damit eine Theorie sogenannter unscharfer Mengen und liefert damit eine Lehre von Mengen mit unscharfen Begrenzungen. Die Bedeutung dieser Theorie liegt zum einen darin begründet, dass außerordentlich viele natürliche Mengen eher unscharf als scharf begrenzt sind. Zum anderen trägt diese Theorie der Tatsache Rechnung, dass die Gesetze der klassischen Mengenlehre über die Boolesche Algebra auf die klassische Logik und Schaltalgebra übertragbar sind und sich die Gesetze der Theorie unscharfer Mengen daraus ableiten lassen. Unabhängig dieser Deutungen stellt die Fuzzy Logik eine Erweiterung des binärlogischen Kalküls dar. Darin werden die in der klassischen binären Logik möglichen Wahrheitswerte „wahr“ und „falsch“ einer Aussage um weitere Zwischenzustände, wie beispielsweise „unbestimmt“ ergänzt.

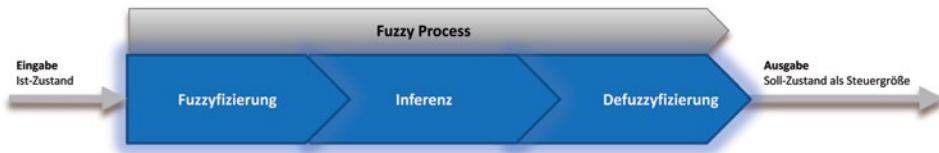
Ein Kalkül ist ein Schema, mit dem sich aus Zeichenketten durch Anwendung von Regeln neue Zeichenketten gewinnen lassen. Wesentlich dabei ist, dass es nur auf die äußere Form der Zeichenketten ankommt und nicht auf eine inhaltliche Bedeutung, weswegen man auch von einem Formalismus spricht.

Um unscharfe Konzepte modellieren zu können, werden Fuzzy-Mengen eingeführt. Mit solchen Fuzzy-Mengen können sprachliche Konzepte mit einer Funktion ausgedrückt werden, die jedem Element einer Grundmenge G eine Zugehörigkeit zuordnet. Das modellierte Konzept wird als *linguistische Variable* und deren Ausprägung als *linguistische Terme* bezeichnet.

So wird beispielsweise das modellierte Konzept der „Geschwindigkeit“ als linguistische Variable Geschwindigkeit eingeführt, deren Ausprägungen „niedrig“ oder „hoch“ durch die gleichnamigen linguistischen Termen niedrig und hoch bezeichnet werden.

Jeder Fuzzy-Menge kann hierbei ein linguistischer Term zugeordnet werden. Mit Hilfe der die Fuzzy-Menge beschreibenden Funktion wird das Maß für die Zugehörigkeit zur Menge und somit der entsprechende Begriff definiert. Fuzzy-mengen werden oftmals als Dreiecksfunktionen, Gaußfunktion oder als Trapezfunktionen modelliert. Die Wahl einer geeigneten Funktion hängt von der Problemstellung und dem zu modellierten Sachverhalt ab. Dabei werden Fuzzy-Mengen über eine Zugehörigkeitsfunktion  $\mu$  definiert, die für jedes Element der Grundmenge G den Grad der Zugehörigkeit zur Fuzzy-Menge angibt.

In Abgrenzung zur Wahrscheinlichkeit gibt die Fuzzy-Menge Auskunft über die Unsicherheit, insbesondere über die Zugehörigkeit eines Elements aus einer Grundmenge zu einem unscharfen Begriff. Die Wahrscheinlichkeit hingegen ist eine empirisch bestimmte oder formal abgeleitete Größe für das Auftreten eines Ereignisses.



**Abb. 5.39** Fuzzy-Regler

Da die Fuzzy-Logik auf Mengen von Zugehörigkeitswerten operiert, werden die logischen Operationen *Konjunktion*  $\wedge$ , *Diskunktion*  $\vee$  und *Negation*  $\neg$  auf die Fuzzy-Logik überführt. Dort werden sie als Mengen-Operationen Schnitt  $\cap$ , Vereinigung  $\cup$  und Komplement  $M$  aufgefasst.

*Fuzzy-Logik* ist eine Methode, die das menschliche Verhalten zur Darstellung und Manipulation von Daten nachahmt. Für die meisten Probleme gibt es zwei verschiedene Arten der Kenntnis: die objektive Kenntnis (z. B. in Form eines mathematischen Modells) und die subjektive Kenntnis. Bei der subjektiven Kenntnis ist es unmöglich, das Problem mit genauen mathematischen Formeln darzustellen. Vielmehr lassen sich die Probleme nur mit Worten ungefähr beschreiben. Zur Lösung solcher Probleme versucht man durch Erfahrung empirische Regeln herauszufinden. Diese Regeln haben ein gewisses Maß an Ungenauigkeit. Man spricht daher auch von „weichen Regeln“. Bei der Fuzzy-Logik werden die Eingangsdaten ( $x_i$ ) nach ihren Eigenschaften bestimmten Gruppen zugeordnet, aus denen dann die Ausgangsdaten ( $y_j$ ) gewonnen werden. Die dazu verwendeten Regeln  $f$  sind weiche Regeln, d. h. bei ungenauen Daten werden diese nur mit einer gewissen Wahrscheinlichkeit einer bestimmten Gruppe zugeordnet. Die Regeln werden von Experten aufgestellt oder sie werden aus numerischen Daten gewonnen. Bei der Anwendung im technischen Bereich können solche Regeln als eine Sammlung von Produktionsregeln in der Form von „Wenn-Dann“-Aussagen bzw. -Anweisungen dargestellt werden. Eine solche Regel könnte dann z. B. so aussehen: „Wenn  $u_1$  sehr warm und  $u_2$  relativ niedrig ist, dann drehe  $v$  etwas nach rechts“. Zur Anwendung solcher Regeln in einer Fuzzy-Logik müssen die gemessenen Eingangsdaten zunächst in einem *Fuzzifier* bestimmten Gruppen zugeordnet werden. Auf diesen linguistischen Variablen wird dann in einer so genannten Inferenz-Maschine die Regel angewandt (Abb. 5.39).

Nach der Fuzzifizierung des Eingabewertes erfolgt die Fuzzy-Inferenz mittels einer Fuzzy-Engine. Anschließend muss die Fuzzy-Ausgabe-Menge in einen „scharfen“ Ausgabewert defuzzifiziert werden.

Sie erzeugt linguistische Ausgangswerte, die in einem *Defuzzifier* wieder in Zahlenwerte umgewandelt und dann beispielsweise zur Steuerung eines Prozesses verwendet werden können. Die Regeln für den Fuzzy-Prozessor müssen explizit einprogrammiert werden. Insofern ist eine reine Fuzzy Logik nicht in der Lage zu lernen. Erst eine Kombination mit anderen Verfahren kann dies bewerkstelligen.

Neuronale Netze lösen ein Problem mittels Funktionsapproximation, indem sie für gegebene Eingabedaten passende Ausgabedaten erzeugen. Soll ein solches Netz zur Lösung eines Problems herangezogen werden, so muss diesem Netz das Problem anhand hinreichender Beispieldaten präsentiert werden. Kennt man dabei eine Menge von gültigen Ein-/Ausgabepaaren, lässt sich überwachtes Lernen (beispielsweise Multilayer Perceptrons und Backpropagation) verwenden. Kennt man hingegen statt passender Ausgabewerte nur ein Fehlermaß zur Bewertung der Effekte, die eine Eingabe auslöst, lässt sich verstärkendes Lernen einsetzen. Für einige Probleme lässt sich auch unüberwachtes Lernen in selbstorganisierenden Karten verwenden. Generell lässt sich testieren, dass artifizielle neuronale Netze immer dann verwendet werden können, wenn Trainingsdaten zur Verfügung stehen. In solchen Fällen wird auch kein mathematisches Modell des zu lösenden Problems vorausgesetzt und es wird auch kein Vorwissen über die Lösung benötigt. Andererseits lässt sich die Lösung durch das Neuronale Netz auch nicht interpretieren und entzieht sich damit einer gewissen Transparenz. Insofern hat es durchaus seine Berechtigung, wenn ein solches Netz als eine „black box“ beschrieben wird. Man kann seine Struktur nicht einfach in Form von verständlichen Regeln ausdrücken. Damit lässt sich in vielen Fällen auch nicht überprüfen, ob die Lösung überhaupt plausibel ist. Eine weitere Eigenart eines neuronalen Netzes besteht im Umstand, dass beim Lernen immer mit „Tabula rasa“, d. h. mit einem „leeren“ Netz begonnen wird. Es ist nicht möglich, relevante Netzparameter anders als heuristisch zu bestimmen. Eine ungeeignete Wahl kann den Lernerfolg erheblich hinauszögern oder sogar schlimmstenfalls verhindern. Selbst mit geeigneten Parametern kann der Lernprozess zum einen sehr lange dauern. Zum anderen kann unabhängig dieser Lerndauer keine Erfolgsgarantie gegeben werden. Ein weiterer Umstand neuronaler Netze besteht in deren Fehlertoleranz gegenüber Abweichungen in den Eingaben und Änderungen ihrer Struktur, was beispielsweise bei einem Ausfall von neuronalen Einheiten vorkommen kann. Wenn sich die Problemstruktur jedoch zu stark von den ursprünglichen Trainingsdaten entfernt, ist es notwendig, das Netz erneut zu trainieren. Diese Gefahr besteht auch dann, wenn die Lernaufgabe nicht alle denkbaren Systemzustände angemessen berücksichtigt oder das Netzwerk beispielsweise aufgrund zu vieler innerer Einheiten übergeneralisiert erscheint. Erneut kann niemand eine Garantie dafür geben, dass die Wiederaufnahme des Trainings mit neuen Daten zu einer schnellen Anpassung an das veränderte Problem führt. Gegebenenfalls muss der gesamte Lernvorgang wiederholt werden.

Ein Fuzzy-System kann ebenso wie ein neuronales Netz eingesetzt werden, wenn für das zu lösende Problem kein mathematisches Modell bekannt ist. Jedoch wird anstelle von Beispieldaten Wissen über die Problemlösung in Form linguistischer Regeln benötigt. Auch die Ein- und Ausgangsgrößen müssen linguistisch beschrieben werden. Dieses Wissen steht in vielen Fällen zur Verfügung, so dass ein Prototyp des Fuzzy-Systems sehr schnell und einfach implementiert werden kann. Die Interpretation der Regelbasis stellt im Allgemeinen ebenfalls kein Problem dar. Die linguistischen Regeln repräsentieren eine unscharfe, punktweise Definition einer ansonsten unbekannten Funktion und das Fuzzy-System führt bei der Verarbeitung der Eingabewerte eine unscharfe Interpolation durch.

Ohne a-priori Wissen über die Problemlösung in Form von Fuzzy-Regeln ist die Implementierung eines Fuzzy-Systems nicht möglich. Ist das Wissen unvollständig, falsch oder widersprüchlich, vermag das System seine Aufgabe nicht zu erfüllen. In diesem Fall ist eine Nachbearbeitung der Regeln oder eine Nachjustierung der die linguistischen Werte beschreibenden Zugehörigkeitsfunktionen unumgänglich. Für diese Phase der Implementierung existieren jedoch keine formalen Methoden, so dass hier ebenfalls nur heuristisch vorgegangen werden kann. Werden dabei neben dem Hinzufügen, Entfernen oder Ändern linguistischer Regeln oder Zugehörigkeitsfunktionen weitere Verfahren eingesetzt, wie beispielsweise die Gewichtung von Regeln, so wird dadurch die Semantik des Reglers aufgegeben. Da ein Fuzzy-System mit unscharfen Angaben über die Systemvariablen auskommt, ist zu erwarten, dass geringfügige Änderungen in den Zugehörigkeitsfunktionen die Leistung des Systems nicht wesentlich beeinflussen. Allerdings wird diese Erwartung nicht erfüllt und es gibt zahlreiche Beispiele dafür, dass durch die geringe Änderung nur einer Zugehörigkeitsfunktion ein Fuzzy-Regler so beeinflusst wird, dass er seine Regelaufgabe nicht mehr erfüllen kann. In solch einem Fall bleibt einem nichts anders übrig, als das Fuzzy-System manuell an die geänderte Problemstellung anzupassen. Neben der Tatsache, dass kleine Veränderungen eventuell große Auswirkungen auf das Systemverhalten haben, kann eine Optimierung zudem mehr als einem Kriterium gleichzeitig unterworfen sein, wie beispielsweise Geschwindigkeitsaspekte, minimaler Energieaufwand usw. Es ist daher wünschenswert, einen automatischen Adoptionsprozess zur Verfügung zu haben, der dem Lernverfahren neuronaler Netze gleicht und die Optimierung von Fuzzy-Systemen unterstützt (Tab. 5.2).

Nach der vergleichenden Gegenüberstellung der Vor- und Nachteile neuronaler Netze und Fuzzy-Systeme wird deutlich, dass es durch eine geeignete Kombination beider Ansätze möglich ist, Vorteile zu vereinen und Nachteile auszuschließen. Das wichtigste Argument für eine Kombination von Fuzzy-Systemen mit neuronalen Netzen ist die Lernfähigkeit letzterer. Ein entsprechendes System sollte in der Lage sein, linguistische Regeln und/oder Zugehörigkeitsfunktionen zu „erlernen“ oder bereits bestehende zu optimieren. „Erlernen“ bedeutet in diesem Zusammenhang die vollständige Erzeugung einer Regelbasis bzw. von Zugehörigkeitsfunktionen, die die entsprechenden linguistischen Terme modellieren, auf der Grundlage von Beispieldaten. Die Erzeugung einer Regelbasis setzt eine zumindest vorläufige Definition von Zugehörigkeitsfunktionen voraus und ist auf drei verschiedene Arten möglich:

- Das System beginnt ohne Regeln und bildet solange neue Regeln, bis die Lernaufgabe erfüllt ist. Die Hinzunahme einer neuen Regel wird dabei durch ein Musterpaar ausgelöst, das durch die bisherige Regelbasis überhaupt noch nicht oder nicht ausreichend erfasst wird. Diese Vorgehensweise kann zu großen Regelbasen führen wenn die verwendeten Zugehörigkeitsfunktionen ungünstig gewählt sind, d. h. sich nur wenig überdecken. Dies ist vergleichbar mit einer schlechten Generalisierungsleistung eines artifiziellen neuronalen Netzes. Weiterhin ist es in Abhängigkeit von der Lernaufgabe

**Tab. 5.2** Vergleich neuronaler Netze mit Fuzzy Systeme

Vorteile	Neuronales Netz	Fuzzy System
	Kein mathematisches Modell erforderlich	Kein mathematisches Modell erforderlich
	Kein Regelwissen erforderlich	A-priori-Wissen anwendbar
	Umfangreiche Lernalgorithmen vorhanden	Einfache Interpretation und Implementierung
Nachteile	Black-Box-Verhalten	Regelwissen muss vorhanden und verfügbar sein
	Kein Regelwissen extrahierbar	Nicht lernfähig
	Heuristische Wahl der Netzparameter	Keine formalen Methoden für das Justieren oder Tuning
	Schwierige Anpassung an veränderte Parameter und erforderliches Wiederaustossen des Lernvorgangs	Semantische Probleme bei der Interpretation der Systeme
	Kein a-priori-Wissen verwendbar	Anpassung an veränderte Parameter unter Umständen nur sehr schwierig möglich
	Bezüglich des Konvergierens des Lernvorgangs kann eine Garantie gegeben werden	Bezüglich des Tunings kann keine Garantie gegeben werden

möglich, dass eine inkonsistente Regelbasis entsteht. Das System muss daher nach Abschluss der Regelgenerierung gegebenenfalls mühevoll erlernte Regeln wieder löschen.

- Das System beginnt mit allen Regeln, die aufgrund der Partitionierung der beteiligten Variablen gebildet werden können und entfernt ungeeignete Regeln aus der Regelbasis. Für dieses Verfahren ist ein Bewertungsschema notwendig, das die Leistungsfähigkeit der Regeln ermittelt. Das Verfahren kann bei der Anwendung auf physikalische Systeme mit vielen Variablen und feiner Partitionierung zu Komplexitätsproblemen in Hinsicht auf Laufzeit und Speicherbedarf führen. Dieser Ansatz vermeidet jedoch inkonsistente Regelbasen, da die entsprechende Überprüfung in das Bewertungsschema integriert ist. Dieses Verfahren kann im Gegensatz zum ersten Ansatz zu Regelbasen mit zu wenigen Regeln führen.
- Das System beginnt mit einer eventuell zufällig gewählten Regelbasis, die aus einer festen Anzahl von Regeln besteht. Im Laufe des Lernvorgangs werden Regeln ausgetauscht. Dabei muss bei jedem Austauschvorgang die Konsistenz der Regelbasis neu überprüft werden. Der Nachteil dieser Vorgehensweise liegt in der festen Anzahl von Regeln. Weiterhin müssen ein Bewertungsschema zur Entfernung von Regeln und eine Datenanalyse zur Hinzunahme von Regeln implementiert sein. Ist dies nicht der Fall, entspricht der Lernvorgang einer stochastischen Suche. Bei einer Verschlechterung der Leistung müssen dann gegebenenfalls Austauschvorgänge rückgängig gemacht werden.

Die Optimierung einer Regelbasis entspricht einem teilweisen Erlernen. Das bedeutet, dass ein Teil der benötigten linguistischen Regeln bereits bekannt ist, während die restlichen noch erzeugt bzw. überzählige Regeln entfernt werden müssen. In diesem Sinne stellt der dritte oben genannte Ansatz eher eine Optimierung als ein Erlernen einer Regelbasis dar. Es sei denn, die Initialisierung des Systems geschieht rein stochastisch und repräsentiert kein a-priori Wissen. Das Erlernen oder Optimieren von Zugehörigkeitsfunktionen ist weniger komplex als die Anpassung einer Regelbasis. Die Zugehörigkeitsfunktionen können leicht durch Parameter beschrieben werden, die dann in Hinblick auf ein globales Fehlermaß optimiert werden. Durch geeignete *Constraints* lassen sich Randbedingungen erfüllen, beispielsweise so, dass die Menge der Punkte, bei denen der Zugehörigkeitsgrad größer Null ist, von Fuzzy-Mengen, die benachbarte linguistische Werte modellieren, nicht disjunkt sein dürfen. Die Adaption von Parametern ist eine Standardaufgabe für neuronale Netze, so dass Kombinationen mit Fuzzy-Systemen in diesem Bereich recht zahlreich sind. Es lassen sich zwei Ansätze des Erlernens bzw. Optimierens von Zugehörigkeitsfunktionen unterscheiden:

- Für die Zugehörigkeitsfunktionen werden parametrisierte Formen angenommen, deren Parameter in einem Lernvorgang optimiert werden.
- Anhand von Beispieldaten lernt ein neuronales Netz, für eine Eingabe einen Zugehörigkeitswert zu erzeugen.

Der zweite Ansatz hat den Nachteil, dass die Zugehörigkeitsfunktionen nicht explizit bekannt sind, weshalb meist das erste Verfahren gewählt wird.

Während die Lernfähigkeit einen Vorteil aus der Sicht der Fuzzy-Systeme darstellt, ergeben sich aus der Sicht neuronaler Netze weitere Vorteile für ein kombiniertes System. Weil ein Neuro-Fuzzy-Modell auf linguistischen Regeln basiert, lässt sich ohne weiteres a-priori Wissen in das Modell integrieren. Bereits bekannte Regeln und Zugehörigkeitsfunktionen können für eine Initialisierung des Systems genutzt werden, so dass der Lernvorgang erheblich verkürzt wird. Da der Adoptionsprozess mit einer Anpassung der Regelbasis und/oder der Zugehörigkeitsfunktionen endet, ist das Lernergebnis in der Regel weiterhin als Standard-Fuzzy-System interpretierbar, beispielsweise als Fuzzy-Regler oder Fuzzy-Klassifikator. Das Black-Box-Verhalten eines Neuronalen Netzes wird somit vermieden. Auf diese Weise kann sogar neues Wissen aus dem System extrahiert werden.

Die Architektur des Neuro-Fuzzy-Modells wird meist durch die Regeln und die Fuzzy-Mengen bestimmt, die dem zu lösenden Problem zugrunde liegen, so dass die Bestimmung von Netzparametern, beispielsweise die Anzahl der Zwischenschichten und dort die Anzahl innerer Einheiten, entfällt. Die in der Tabelle des letzten Abschnitts genannten Vorteile beider Methoden bleiben in vollem Umfang erhalten. Die Nachteile, die sich auch bei einer Kombination nicht umgehen lassen, bestehen darin, dass der Erfolg des Lernvorgangs nicht garantiert ist, bzw. gleichbedeutend damit, dass das Tuning keine Verbesserung des Regelverhaltens erbringen muss.

## **Rekursive Algorithmen**

*Rekursive Algorithmen* werden häufig zur Optimierung eines komplexen Systems eingesetzt. Dabei geht man von einem komplexen System aus, das über eine Reihe von veränderbaren Parametern verfügt. Weiterhin wird ein Optimierungsziel für die Funktion des Systems definiert. Nunmehr verändert man die Parameter geringfügig, berechnet damit die entsprechend veränderte Funktion und überprüft, ob die Funktion dem Optimierungsziel näher gekommen ist oder nicht. Ist sie dem Ziel näher gekommen, werden die neuen Parameter übernommen. Im anderen Fall werden die Parameter verworfen. Dieser Kreislauf wiederholt sich so lange, bis entweder das Optimierungsziel voll erreicht ist oder sich keine Verbesserungen mehr ergeben. Für die Art und Weise, mit der man die Parameter variieren kann, gibt es eine Reihe von verschiedenen, erprobten Verfahren.

*Evolutionäre Algorithmen* gehören zur Gruppe der rekursiven Algorithmen im Allgemeinen und zu den stochastischen Such- bzw. Optimierungsmethoden im Speziellen. Konzeptionell orientieren sie sich an den Mechanismen des natürlichen Evolutionsprozesses. Man unterscheidet darin vier Hauptströmungen: Genetische Algorithmen, genetische Programmierung, Evolutionsstrategien und Evolutionäre Programmierung.

- *Genetische Algorithmen* verwenden häufig eine binäre Lösungspräsentation. Der wesentliche Selektionsschritt ist die Selektion zur Fortpflanzung. Diese erfolgt stochastisch, so dass auch schlechten Individuen eine Chance zur Reproduktion gegeben wird. Als Hauptsuchoperator dient das Crossover, während die Mutation nur mit geringer Wahrscheinlichkeit auftritt.
- *Genetische Programmierung* als ein weiterentwickelter Ansatz der genetischen Algorithmen greift die Problemstellung auf, inwieweit Computer lernen können ein Problem zu lösen, ohne dafür explizit programmiert zu werden. Genetische Programmierung erzeugt automatisch Problemlösungen, indem evolutionäre Operatoren auf passend repräsentierte Computerprogramme aus Klassen, Methoden, Variablen und Konstanten angewendet werden. Dabei beginnt man mit stochastisch generierten Programmen.
- Bei der *Evolutionsstrategie* verwendet man im Allgemeinen einen Vektor reeller Zahlen zur Lösungsrepräsentation. Der wesentliche Selektionsschritt ist die Selektion zum Überleben. Sie erfolgt deterministisch, so dass nur die besten Individuen überleben. Die Mutation auf Basis normalverteilter Zufallsgrößen ist als Suchoperator besonders wichtig, doch spielt auch die Rekombination eine große Rolle.
- *Evolutionäre Programmierung* verwendet häufig einen Vektor reeller Zahlen zur Lösungsdarstellung. Es wird jedoch mit einer stochastischen Form der Selektion gearbeitet und die Mutation bildet den einzigen Suchoperator.

Auf allgemeiner Ebene unterscheiden sich die evolutionären Algorithmen von konventionellen Such- und Optimierungsverfahren zunächst durch die aus der Evolutionstheorie entlehnte Terminologie (Tab. 5.3):

**Tab. 5.3** Terminologie der evolutionären Algorithmen

Ausdruck	Bedeutung
Chromosomen (besteht aus Genen)	Grundsätzlich identisch mit Individuen, gelegentlich kann ein Individuum auch aus mehreren Chromosomen zusammensetzen, übliche Form: String
Gen	Bit (binäre Codierung unterstellt)
Allel	Genausprägung (binär: 0 oder 1)
Genotyp	Codierte Lösung
Phänotyp	Decodierte Lösung

Ausdruck	Bedeutung für EA
Individuum	Struktur (enthält die in geeigneter Weise repräsentierten Elemente einer Lösung)
Population (von Individuen)	Menge von Strukturen (Lösungen)
Eltern	Zur Reproduktion ausgewählte Lösungen
Kinder, Nachkommen	Aus den Eltern erzeugte Lösungen
Crossover	Suchoperator, der Elemente mehrerer Individuen vermischt
Mutation	Suchoperator, der ein Individuum modifiziert
Fitness	Lösungsgüte bezogen auf die Ziele
Generation	Verfahrensiteration

Die Grundidee *evolutionärer Algorithmen* liegt nun in der Übersetzung der evolutionären Variationsoperatoren Mutation und Rekombination sowie der Selektion in einen Algorithmus. Den Rahmen des evolutionären Algorithmus bildet ein iterativer Prozess, bei dem eine zufällige Variation der Lösungskandidaten eine entscheidende Rolle spielt. Der erste Schritt ist die Erzeugung einer Reihe von verschiedenen potenziellen Lösungen eines Problems. Danach folgt eine Selektion der Lösungen nach ihrer Brauchbarkeit, ihrer sogenannten „Fitness“. In der Regel wird diese Fitness durch die Abweichung einer Lösung zur idealen Lösung angegeben. Diese Abweichung gilt es zu minimieren. Im nächsten Schritt werden die selektierten Lösungen einer *Mutation* unterzogen. Hierbei werden mit Hilfe eines Zufallsgenerators kleine, zufällige Änderungen an den Lösungen vorgenommen. Packt man die Anzahl der möglichen Lösungen in einen Suchraum, so besteht die Aufgabe der Mutation in der Exploration dieses Suchraumes und stellt somit die Hauptquelle für genetische Variation dar. Mutationen sollen mit hoher Wahrscheinlichkeit eher kleine Änderungen an der Lösung erzeugen. Ein Mutationsoperator muss dabei drei Anforderungen erfüllen:

- Ausgehend von einem gegebenen Punkt im Suchraum muss jeder andere Punkt erreichbar sein. Andernfalls wäre es möglich, dass das Optimum nie gefunden werden kann.

- Des Weiteren sollte die durch Mutation verursachte genetische Variation keinen Drift aufweisen, sondern sich in alle Richtungen des Suchraumes mit gleicher Wahrscheinlichkeit bewegen. Erst die Selektion drängt durch die Fitnesswerte der Nachkommen den Suchprozess in eine bestimmte Richtung. Insofern erkundet der Algorithmus mit der Mutation den Suchraum, wohingegen mit der Selektion die gewonnenen Informationen verwendet werden, die in der aktuellen Population stecken.
- Schließlich soll die Stärke der Mutation einstellbar sein, um erfolgreiche Schritte in der Fitnesslandschaft zu ermöglichen.

In der Natur wird bei der *Rekombination*, die häufig auch Crossover genannt wird, das genetische Material zweier Eltern kombiniert. Genauso liegt bei evolutionären Algorithmen die Idee der Rekombination in der Kombination zweier Lösungen. Dabei lassen sich zwei Hypothesen unterscheiden.

- Die *Building Block Hypothese* geht davon aus, dass sich gute Teil-Strings, die sogenannten Building Blocks, von verschiedenen Eltern durch Rekombination kombinieren und im Laufe der Generationen vermehren. Diese guten Gene verteilen sich demnach im Laufe der Generationen in der Population.
- Demgegenüber unterstellt der *Genetic Repair-Effekt* der Rekombination die Wirkung, dass sich nicht die unterschiedlichen Merkmale an die Nachkommen vererben, sondern die gemeinsamen. Die Nachkommen erhalten bei diesem Erklärungsmodell mit Sicherheit die Gene, die beide Eltern gemeinsam haben.

Der *Selektion* kommt als Gegenspieler der Variationsoperatoren Mutation und Rekombination ein großer Stellenwert zu, da erst sie dem Optimierungsprozess eine Richtung verleiht. Basierend auf ihrer Fitness wird ein Teil der Population ausgewählt, die übrigen Lösungen werden verworfen. Die Selektion zur Paarung wählt die an der Rekombination beteiligten Lösungen aus. Währenddessen bestimmt die Überlebensselektion, welche Lösungen überleben und in die nächste Generation übernommen werden. Diese durch dieses Selektionsverfahren gewonnene, neue Generation von Lösungen wird dann wieder selektiert usw. Dieser Kreislauf geht so lange weiter, bis eine der Stopp-Bedingungen erreicht ist.

Dies ergibt folgendes Ablaufschema: Man beginnt mit einer häufig stochastisch generierten Startmenge (Ausgangspopulation) von Lösungsalternativen (Individuen). Für diese werden Fitnesswerte (Gütekriterien) bestimmt. Es folgt ein iterativer Zyklus, indem immer wieder aus den alten Lösungen neue, modifizierte Lösungsvorschläge erzeugt werden. In diesem Generationszyklus werden Individuen aus der Population durch Selektion zur Fortpflanzung bestimmt. Im Rahmen der Erzeugung von Nachkommen wird die in den Eltern enthaltene Lösungsinformation kopiert (Replikation) und durch Anwendung eines oder mehrerer Variationsoperatoren, wie z. B. Mutation oder Crossover, verändert. Die sich ergebenden Nachkommen bewertet man und wählt dann aus, welche Individuen in die neue Population übernommen werden (Selektion zum Überleben). Dabei können je nach EA-Variante auch Eltern mit ihren Nachkommen ums Überleben konkurrieren. Das Wechselspiel aus unge-

richteter Veränderung von Lösungen durch die Variationsoperatoren und Bevorzugung der besten Lösungen im Selektionsprozess führt im Verlaufe vieler Generationszyklen zu sukzessiv besseren Lösungsvorschlägen. Dieser Prozess wird solange fortgesetzt, bis ein Abbruchkriterium greift.

*Genetische Algorithmen* werden in Verbindung mit evolutionären Algorithmen eingesetzt. Sie beschreiben eine sexuelle Fortpflanzung. Sexualität war in der biologischen Evolution ein ganz entscheidender Schritt zur Beschleunigung der Entwicklung. Dieser Algorithmus kombiniert statistisch die Eigenschaften verschiedener Wesen. Aus diesen Ansätzen lassen sich die folgende Merkmale ableiten:

- Die Entscheidungsvariablen des gegebenen Anwendungsprogramms werden im Allgemeinen als String (Zeichenkette) bzw. in Vektorform dargestellt und ihre Ausprägungen in dieser Form vom evolutionären Algorithmus optimiert.
- Ein evolutionärer Algorithmus sucht von verschiedenen Punkten aus gleichzeitig nach guten Lösungen (populationsbasierte statt punktbasierte Suche).
- Für eine zielgerichtete Suche sind nur Angaben zur Güte (Fitness) der betrachteten Lösungen im Hinblick auf die gegebene Aufgabenstellung nötig. Diese Fitnesswerte werden i. d. R. aus Zielfunktionswerten berechnet. Sie können aber auch beispielsweise mit Hilfe von Simulationen oder durch praktische Experimente ermittelt werden. Ableitungen werden nicht benötigt.
- Die Grundoperationen von evolutionären Algorithmen imitieren auf abstrakter Ebene die Phänomene Replikation, Variation und Selektion als treibende Kräfte evolutionärer Prozesse.
- Stochastische Verfahrenselemente werden bewusst eingesetzt. Daraus entsteht jedoch keine reine Zufallssuche, sondern ein intelligenter Suchprozess. Es werden immer wieder neue Lösungsstrukturen generiert (Exploration) und hinsichtlich ihrer Güte bewertet. Dadurch sammelt das Verfahren Informationen über die Struktur des Suchraumes, die implizit in den Individuen der Populationen gespeichert werden. Diese Informationen dienen anschließend dazu, den weiteren Suchprozess einzuschränken. Lösungsalternativen werden verstärkt in solchen Regionen des Suchraumes generiert, die erfolgversprechend sind (Exploitation). Wesentlich für den Optimierungserfolg eines evolutionären Algorithmus ist es, die richtige Balance von Exploration und Exploitation zu finden.

Diese typischen Merkmale eines evolutionären Algorithmus bestimmen auch den allgemeinen Ablauf in einem Computerprogramm, der sich folgendermaßen gestaltet. Die folgende Tabelle zeigt die vielfältigen Einsatzmöglichkeiten evolutionärer Systeme (Tab. 5.4).

Jede Lösung kann durch einen im Umfang begrenzten Digitalcode, bestehend aus Nullen und Einsen, dargestellt werden. Der genetische Algorithmus erzeugt neue Lösungen, indem er Sequenzen des Digitalcodes aus einer Lösung durch die entsprechenden Sequenzen aus einer anderen Lösung ersetzt. Die Länge der ersetzen Sequenzen und die Position im Digitalcode werden durch einen Zufallsgenerator bestimmt. Da hier bereits erprobte

**Tab. 5.4** Einsatzmöglichkeiten evolutionärer Algorithmen

Einsatzgebiet	Problemstellung
Technisch- naturwissenschaftlich	VLSI-Routing
	Pfadplanung für mobile Roboter
	Struktur- und Parameteroptimierung künstlicher Neuronaler Netze
	Mikroprozessorentwurf
	Information Retrieval
Betriebswirtschaft	Produktionsplanung
	Standortplanung
	Kundenklassifizierung
	Kraftwerkseinsatzplanung
	Portfoliooptimierung

**Tab. 5.5** Pseudo-Code des genetischen Algorithmus

Schritt	Inhalt
1.	Initialisiere die Population
2.	Evaluiere die neu erzeugten Verarbeitungseinheiten und berechne ihre Fitnesswerte
3.	Wiederhole, bis Abbruch der Evolution
3.1.	Selektiere Verarbeitungseinheiten zur Reproduktion gemäß der Selektionsmethode
3.2	Erzeuge daraus neue Verarbeitungseinheiten durch Anwendung der genetischen Operatoren
3.3	Evaluiere die neu erzeugten Verarbeitungseinheiten und berechne ihre Fitnesswerte
3.4	Füge die neuen Verarbeitungseinheiten gemäß der Ersetzungsstrategie in die Population ein

und damit erfolgreiche Eigenschaften verwendet werden, führt dies in Verbindung mit einem evolutionären Algorithmus erheblich schneller zu optimierten Lösungen. Genetische Algorithmen werden schon jetzt bei der Lösung von Problemen eingesetzt, bei denen eine Vielzahl von Parametern das Ergebnis beeinflusst.

Der Ablauf des *genetischen Algorithmus* kann durch folgenden Pseudo-Code beschrieben werden (Tab. 5.5):

Der genetische Algorithmus besteht in seiner Grundform aus folgenden Komponenten:

- Einer *Population* von Verarbeitungseinheiten, die durch ihren genetischen Code und ihren Fitnesswert repräsentiert sind.
- Einer *Initialisierungsmethode*, nach der eine Ausgangspopulation erzeugt wird.

- Einer *Selektionsstrategie*, gemäß der aus der Population Individuen zur Reproduktion ausgewählt werden.
- *Genetischen Operatoren*, wie *Mutation* und *Crossover*, um aus den alten Verarbeitungseinheiten neue, veränderte Verarbeitungseinheiten zu erzeugen.
- Einer *Bewertungsfunktion*, die das Verhalten der Verarbeitungseinheit in seiner „Umwelt“ beschreibt.
- Einer *Fitnessfunktion*, um die Resultate der Bewertungsfunktion in einen Fitnesswert für die Verarbeitungseinheit zu transformieren.
- Einer *Ersetzungsstrategie*, gemäß der die neuen Verarbeitungseinheiten in die Population übernommen werden und dabei die alten Verarbeitungseinheiten verdrängen.

Damit ist die Grundform des genetischen Algorithmus gegeben, die natürlich in vielfältiger Weise variiert werden kann. Wesentlich sind die verschiedenen Komponenten, aus denen der GA zusammengesetzt ist, und die im Folgenden genauer dargestellt werden sollen.

Der *genetische Code* speichert die Eigenschaften der Verarbeitungseinheit. Wie beim natürlichen Vorbild kann eine Aufteilung in mehrere Chromosomen vorgenommen werden, jedoch ist dies in der Regel nicht unbedingt erforderlich. Auf dem Chromosom sind die einzelnen Gene lokalisiert, wobei in vereinfachender Weise eine Eins-zu-Eins Korrespondenz zwischen Genen und Eigenschaften (Phänen) gesetzt wird. Eine Eigenschaft kann verschiedene Ausprägungen haben, die durch verschiedene Werte (Allele) eines Gens kodiert werden. So wie der natürliche genetische Code aus Basenpaaren aufgebaut ist, so muss auch für den simulierten genetischen Code eine Repräsentation – gewissermaßen ein Alphabet – gewählt werden. Im klassischen genetischen Algorithmus ist dies ein Bit-String, also einfach eine Kette, bestehend aus Nullen und Einsen. In den evolutionären Strategien wird hingegen mit reellen Zahlen gearbeitet, in der genetischen Programmierung lassen sich auch komplexe Regeln oder Regelbäume verwenden. Im Prinzip sind beliebige Datenstrukturen als genetischer Code denkbar. Wichtig ist nur, dass die genetischen Operatoren auf diese Datenstruktur zugeschnitten sind, und dass die Bewertungsfunktion diesen genetischen Code auslesen kann. Der Vorteil einer komplexen Kodierung liegt darin, dass die Struktur des Problems direkter und damit besser abgebildet werden kann, als würde man eine Transformation auf eine einfache Repräsentation, wie beispielsweise einen Bitstring, durchführen. Andererseits hat die traditionelle Bit-String-Kodierung den Vorteil, dass keine speziellen Crossover- und Mutationsoperatoren entwickelt werden müssen, der genetische Algorithmus somit allgemeiner verwendet werden kann.<sup>55</sup>

Die *Selektionsstrategie* legt fest, wie die Verarbeitungseinheiten einer Population ausgewählt werden, um daraus neue Verarbeitungseinheiten für die nächste Generation zu erzeugen. Erstrebenswert ist, dass Verarbeitungseinheiten mit höheren Fitnesswerten sich stärker fortpflanzen, als Verarbeitungseinheiten mit niedrigen Fitnesswerten. Auf der anderen Seite sollte keine Verarbeitungseinheit so dominant sein, dass es zu einer vorzeitigen Konvergenz innerhalb der Gesamtpopulation kommt. Dadurch würde die Suche auf einen

---

<sup>55</sup> Siehe auch Mildenberger 1992.

Teilbereich des Suchraumes beschränkt, was selten zu optimalen Lösungen führen wird. Es soll also einerseits der Suchraum möglichst umfangreich durchmustert werden, andererseits soll bei der Suche die in den guten Verarbeitungseinheiten gespeicherte Information möglichst gut ausgenutzt werden. Diese beiden Ziele stehen in gewissem Gegensatz zueinander, der in der Literatur mit dem Begriffspaar *Exploration* vs. *Exploitation* zum Ausdruck gebracht wird. Die klassische Selektionsstrategie der genetischen Algorithmen ist die der Roulette-Auswahl. Dabei wird jede Verarbeitungseinheit mit einer Wahrscheinlichkeit proportional zu seiner Fitness ausgewählt. Die Bezeichnung Roulette-Auswahl röhrt daher, dass man sich die Fitnesswerte aller Verarbeitungseinheiten auf einem Kreis aufgetragen denken kann. Die Summe aller Fitnesswerte ist gleich dem Umfang des Kreises. Wählt man jetzt zufällig eine Zahl zwischen 0 und der Gesamtfitness (=Einbringen der Roulett-Kugel), so besitzt jede Verarbeitungseinheit eine Trefferwahrscheinlichkeit, die proportional zu seinem Fitnesswert ist. Bei der Roulette- Auswahl kann eine Verarbeitungseinheit mit überdurchschnittlich hohem Fitnesswert so stark in die Reproduktion einfließen, dass die Population sehr schnell konvergiert. Es besteht dann keine Möglichkeit, den Suchraum genügend breit zu durchsuchen. Um die Dominanz einer solchen Super-Verarbeitungseinheit zu verhindern, kann das *Ranking* zur Auswahl benutzt werden. Dabei werden die Individuen nach ihrer Fitness sortiert. Die Auswahl erfolgt dann gemäß der Position innerhalb dieser Liste, d. h. die Selektions-Wahrscheinlichkeit ist im einfachsten Fall eine lineare Funktion der Rangfolge.

Eine wichtige Unterscheidung für die Reproduktion besteht darin, ob aus der ganzen Population ausgewählt wird, oder nur aus der lokalen Umgebung einer Verarbeitungseinheit. Die Umgebung kann dabei zuerst einmal als räumliche Nähe gedacht werden, d. h. jede Verarbeitungseinheit besitzt eine Position und sucht sich seinen Partner in der Nähe aus. Durch diesen Mechanismus können sich lokal Subpopulationen mit besonderen Eigenschaften ausprägen, was eine vorzeitige Konvergenz verhindern kann. Eine andere Nachbarschaftsbeziehung kann darin bestehen, dass zwei Verarbeitungseinheiten zueinander ähnlich sein müssen, damit sie miteinander gekreuzt werden können. Auch hierdurch wird Diversität länger aufrechterhalten. Diese Unterscheidung in lokale und globale Selektion ist auch aus der Sicht der Implementierung von Bedeutung.

Auch bei den *Ersetzungsstrategien* gibt es vielfältige Variationen, die teilweise miteinander kombiniert werden können. Die klassische Methode für genetische Algorithmen ist der *Generationenwechsel*: die alten Verarbeitungseinheiten in der Population werden immer komplett durch die neuen Verarbeitungseinheiten ersetzt. Dabei können natürlich neu erzeugte Verarbeitungseinheiten identisch zu alten sein. Es ist aber nicht garantiert, dass die besten Verarbeitungseinheiten der Population beim Generationenwechsel erhalten bleiben. Somit kann der Fitnesswert der jeweils besten Verarbeitungseinheit auch wieder absinken. Um den Verlust der besten Verarbeitungseinheiten beim Generationenwechsel zu verhindern, kann mit einer sogenannten *Elite* gearbeitet werden. Die besten n Verarbeitungseinheiten, wobei n ein einstellbarer Parameter ist, werden automatisch in die nächste Generation übernommen. Somit wird garantiert, dass der Fitnesswert der

jeweils besten Verarbeitungseinheiten eine monoton steigende Funktion ist. Eine Alternative zum Generationenwechsel bietet der *kontinuierliche genetische Algorithmus*. Dieser besteht darin, für jede neu erzeugte Verarbeitungseinheit sofort zu überprüfen, ob es in die Population aufzunehmen ist. Dies kann an Bedingungen geknüpft sein, wie beispielsweise, wenn die neue Verarbeitungseinheit besser ist als mindestens eine Verarbeitungseinheit in der Population oder wenn es nicht schon eine gleiche Verarbeitungseinheit gibt. Freiheit besteht auch in der Auswahl der zu entfernenden Verarbeitungseinheit. So kann beispielsweise die Verarbeitungseinheit mit der geringsten Fitness oder eine eher zufällige Verarbeitungseinheit gelöscht werden.

Die *Mutation* führt eine Änderung auf einem einzelnen Chromosom durch. Dabei wird für jedes Gen ermittelt, ob gemäß der Mutationswahrscheinlichkeit eine Veränderung durchgeführt werden soll. Wenn ja, dann erfolgt die Änderung gemäß der Mutationsvorschrift. Bei der traditionellen Bitstring-Kodierung ist die Mutation einfach ein Bitflip. Bei vielen Anwendungen werden jedoch spezielle Mutationsoperatoren definiert, bei denen Änderungen nicht an allen Positionen mit der gleichen Wahrscheinlichkeit durchgeführt werden. Damit kann dem Wissen, das man über den Anwendungsbereich hat, Rechnung getragen werden. Der Mutationsoperator bewirkt eine Wanderung einer Verarbeitungseinheit durch den Suchraum. Damit ist sichergestellt, dass – zumindest theoretisch – der gesamte Suchraum durchlaufen und damit die optimale Lösung gefunden werden kann. In der Praxis steht dem jedoch oft die zu lange Laufzeit entgegen. Beim Crossover-Operator wird die genetische Information von mehreren Verarbeitungseinheiten gemischt und daraus neue Verarbeitungseinheiten erzeugt. Normalerweise sind die Crossover-Operatoren so formuliert, dass aus zwei alten Verarbeitungseinheiten zwei neue entstehen. An Crossover-Operatoren sind gebräuchlich:

- *One-Point-Crossover*: An einem zufällig gewählten Punkt werden die beiden Chromosomen gekreuzt.
- *Two-Point-Crossover*: An zwei zufällig gewählten Punkten werden die beiden Chromosomen gekreuzt.
- *Uniform-Crossover*: Für jedes Gen wird mit einer gewissen Wahrscheinlichkeit ein Austausch durchgeführt.

Ob ein Crossover tatsächlich durchgeführt wird, hängt von der einzustellenden Crossover-Wahrscheinlichkeit ab. Und auch hier gilt wieder, dass spezifische Crossover-Operatoren definiert werden können, um der gewählten Kodierung und dem Anwendungsfall angemessen zu sein.

Neben Mutation und Crossover sind andere genetische Operatoren beschrieben worden, die jedoch weit weniger Bedeutung besitzen. Bei der *Inversion* wird ein Stück des Chromosoms ausgeschnitten und umgekehrt wieder eingesetzt. Bei der *Transiation* wird ein Stück des Chromosoms ausgeschnitten, und an anderer Stelle eingefügt. Diese Operatoren arbeiten nur auf einem Chromosom, und können in gewisser Weise als verallgemeinerte Mutationen angesehen werden. Falls es nicht gelingt, geeignete Operatoren zu finden, die unzulässige Chromosomen in zulässige transformieren, so müssen unter Um-

ständen eine Vielzahl von Verarbeitungseinheiten erzeugt und verworfen werden, bis eine zulässige Verarbeitungseinheit entstanden ist. Damit der genetische Algorithmus funktioniert, muss jede Verarbeitungseinheit mit einem Fitnesswert versehen werden, der die Selektion beeinflusst. Der Fitnesswert soll Auskunft darüber geben, wie gut die Verarbeitungseinheit an seine „Umwelt“ angepasst ist. Zu diesem Zweck wird jede Verarbeitungseinheit einer Bewertung unterworfen. In dieser Bewertungsfunktion wird der genetische Code interpretiert und dadurch in die Sprache des Problembereichs rücktransformiert. In der Bewertungsfunktion steckt das Wissen über den Anwendungsbereich und ist deshalb in jedem Fall anwendungsspezifisch. Die Bewertungsfunktion muss einen Zahlenwert liefern, der ein Maß für die Leistungsfähigkeit der Verarbeitungseinheit bezüglich der Optimierungsaufgabe darstellt. In vielen Fällen kann der so ermittelte Zahlenwert nicht direkt als Fitnesswert genommen werden, sondern muss durch eine Fitness-Funktion in einen geeigneten Fitnesswert umgewandelt werden. Insbesondere kann durch die Fitness-Funktion eine Skalierung durchgeführt werden.

Bei den bisher behandelten Optimierungen wurde das konnektionistische Modell als Ganzes, d. h. die Anzahl der Verarbeitungseinheiten, die Verbindungsmaatrix, die einzelnen Gewichte nicht berührt. Um ein solches Modell durch einen genetischen Algorithmus ausführen zu können, muss es in irgendeiner Weise kodiert werden. Entsprechend dieser Codierung sind dann die Mutations- und Crossover-Operatoren zu formulieren. Im klassischen genetischen Algorithmus ist der genetische Code ein Bitstring, also ein eindimensionales Konstrukt. Ein konnektionistisches Modell ist aber eine zwei oder dreidimensionale Struktur. Insofern gilt es, das Problem zu lösen, wie die zweidimensionale Graphstruktur sich so auf eine lineare Repräsentation abbilden lässt, dass sich sinnvolle genetische Operatoren finden lassen. In Anlehnung an die Literatur und die Erfahrung der letzten Jahre kommen folgende Kodierungsverfahren in Betracht:

- *Blueprint-Kodierung*: Das konnektionistische Modell wird komplett kodiert, d. h. jeder Knoten und jede Verbindung wird durch ein Segment des genetischen Codes repräsentiert. Diese Kodierung wird am häufigsten angewendet.<sup>56</sup>
- *Parameter-Kodierung*: Ein Satz von Parametern beschreibt das Aussehen des konnektionistischen Modells. Es werden Eigenschaften einer Struktur, nicht die Struktur selbst, kodiert. Diese Kodierungsart wird insbesondere zur Beschreibung modularer Modelle eingesetzt.<sup>57</sup>
- *Regel-Kodierung*: Durch eine Reihe von Regeln wird beschrieben, wie das konnektionistische Modell aufgebaut wird. Dieser Ansatz erlaubt durch den Einsatz rekursiver Regeln insbesondere auch Modelle, die gleiche Substrukturen aufweisen. Dieser Ansatz erlaubt die kompakteste Kodierung.<sup>58</sup>

<sup>56</sup> Vgl. Cliff et al. 1993.

<sup>57</sup> Vgl. Martin Mandischer. *Representation and Evolution of Neuronal Networks*. Tech. Rep. Department of Computer Science VI. University of Dortmund. 1993.

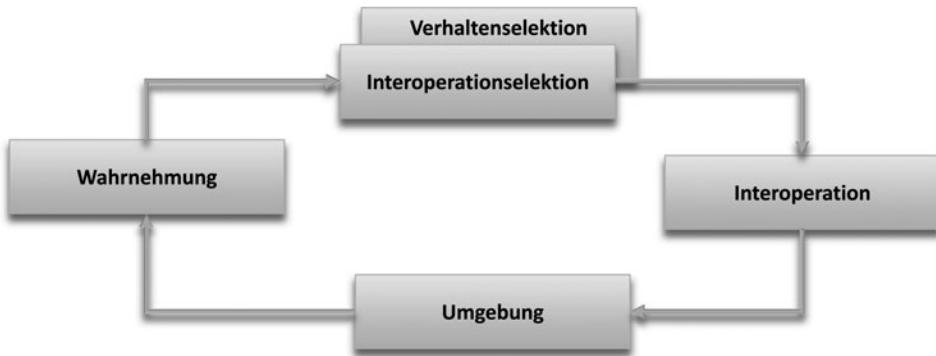
<sup>58</sup> Vgl. Frederic Gruau. *Genetic Synthesis of Modular Neuronal Networks*. In: ICCGA93, S. 318–325. 1993.

Die meisten Anwendungen genetischer Algorithmen verwenden einen genetischen Code, bei dem die Bedeutung des Gens durch seine absolute Position auf dem Chromosom gegeben ist. Mehr Flexibilität erhält man, wenn man für jedes Gen dessen Position mitschreibt. Durch Inversion kann dann die Reihenfolge der Gene verändert werden. Dies kann Vorteile im Fall von korrelierten Genen bringen, weil durch den Crossover-Operator nahe zusammenliegende Gene seltener auseinandergerissen werden. Eine dritte Variante der Kodierung ist durch das biologische Vorbild gegeben. Start- und Stoppgene markieren sinnvolle Abschnitte auf dem Chromosom. Beim Umsetzen des Geno- in den Phentotyp muss dann das Chromosom sequentiell gelesen und interpretiert werden.

Die einfachste Vorgehensweise zur Optimierung der Topologie des konnektionistischen Modells besteht darin, die maximale Netzgröße vorzugeben und in einem evolutionären Prozess die Menge der Verbindungen zu suchen, die die Fehlerrate für das Netz minimiert. Durch das Streichen einzelner Verbindungen können einzelne Knoten isoliert werden, indem sie keine Ein- und oder Ausgabe mehr besitzen. Knoten ohne Ausgabe können den Output des Netzes nicht mehr beeinflussen und können somit auf jeden Fall gestrichen werden. Knoten ohne Eingabe können als Generator eines konstanten oder variablen Signals dienen und somit als Threshold- oder Noisegenerator dienen. Die Gesamtarchitektur kann noch zusätzlichen Restriktionen unterworfen werden, wie sie beispielsweise durch die verwendete Lernfunktion gegeben sind. Soll mit Backpropagation trainiert werden, ist eine Beschränkung auf vorwärts gerichtete Netze erforderlich. Es hat sich wiederholt gezeigt, dass in dieser Weise optimierte Netze bei geringerer Komplexität, und daraus resultieren insbesondere kürzere Trainingszeiten, vergleichbare oder sogar bessere Leistungen erbringen als vollverknüpfte Netze mit gleicher Knotenanzahl.

Eine relativ einfache Optimierung besteht darin, die günstige Anzahl und Anordnung der versteckten Knoten zu ermitteln. Bei nur einer versteckten Schicht braucht man dafür keinen genetischen Algorithmus zu bemühen. Bei mehrschichtigen Netzen wird ein manuelles Austesten jedoch aufwendig, die Anzahl der Möglichkeiten wächst sehr schnell in eine Größenordnung, in der vollständige Enumeration nicht mehr möglich ist. Große konnektionistische Modelle oder neuronale Netze zeichnen sich häufig durch eine modulare Struktur aus. Darunter ist zuerst einmal zu verstehen, dass sich durch eine heterogene Verknüpfungsstruktur die Verarbeitungseinheiten in Cluster einteilen lassen. Dies kann auf zweierlei Weise geschehen. Zum einen sind die Layers von Verarbeitungseinheiten stark miteinander verbunden. Die Verarbeitungseinheiten innerhalb eines Layers sind nicht oder nur schwach vernetzt. Dies entspricht in etwa einer typischen Multilayer-Verarbeitungseinheiten-Architektur. Zum anderen weisen die Verarbeitungseinheiten innerhalb einer Gruppe einen höheren Verbindungsgrad auf als die Gruppen untereinander.

Die zweite Form der Modularisierung wird insbesondere dann erforderlich, wenn ein konnektionistisches Modell nicht zur einfachen Mustererkennung, sondern zur Simulation komplexer kognitiver Prozesse eingesetzt werden soll. In Anlehnung an die Literatur zur Optimierung der Topologie modularer Netze besteht ein Netz aus einer beliebigen Anzahl von Gebieten, die miteinander verbunden sind. Jedes Gebiet wird durch eine Anzahl von Parametern kodiert, ebenso die Verbindungen zwischen den einzelnen Gebieten, die als *Projektionen* bezeichnet werden. Ein Gebiet wird durch eine eindeutige Identifikationsnum-



**Abb. 5.40** Autonome Agenten

mer, die Gesamtgröße und Angaben zur Ausdehnung in x, y und z Richtung beschrieben. Eine Projektion wird beschrieben durch Angabe des Zielgebietes, der Auffächerung der Verbindungen sowie der Verbindungsichte, d. h., wie viel Prozent der möglichen Verbindungen tatsächlich ausgebildet werden. Zusätzlich werden für jede Projektion die Lernrate und der Abfall der Lernrate für Backpropagation gespeichert. Da die Anzahl der Projektionen je Gebiet nicht festgelegt ist, ergeben sich Chromosomen unterschiedlicher Länge. Dies erfordert angepasste Crossover Operatoren, die vergleichbare Stellen innerhalb zweier Chromosomen aufzufinden. Das heißt, dass nicht die absolute Position sondern die relative Position innerhalb der Beschreibung eines Gebietes oder einer Projektion gewählt werden muss. Damit dies möglich ist, wurde jedes Segment des genetischen Codes mit speziellen Marker-Genen versehen, die durch den genetischen Algorithmus selbst nicht verändert werden.

Zu guter Letzt soll noch einmal auf die biologischen Grundlagen hingewiesen werden. Diese ergaben, dass das menschliche Gehirn etwa aus  $10^{11}$  Neuronen und etwa  $10^{14}$  Synapsen besteht.<sup>59</sup> Auf der DNA steht bei weitem nicht genügend Platz für eine Eins-zu-Eins Kodierung zur Verfügung. Daraus folgt, dass nicht die endgültige Struktur des Gehirns, die zudem durch Lernprozesse modifiziert wird, in der DNA kodiert wird, sondern vielmehr das Wachstum des Gehirns. Für die Optimierung konnektionistischer Modelle ist dies ein interessanter Gesichtspunkt, indem die Möglichkeit, nicht die Struktur, sondern Wachstumsprozesse zu kodieren, in Zukunft in Betracht gezogen werden muss.

### Interoperative Agenten

Der Begriff des Softwareagenten im Allgemeinen und dort der des autonomen Agenten im Speziellen hat in den letzten Jahren über die Grenzen der Forschungslabors hinaus zunehmend an praktischer Relevanz gewonnen. Solche autonomen Agenten sind vor allem dadurch charakterisiert, dass sie in einer Umgebung agieren und dadurch selbst zum aktiven Bestandteil der Umgebung werden. Als solcher partizipierender Bestandteil können diese Agenten ihre Umgebung wahrnehmen und durch ihre Interoperationen auf die Umgebung einwirken (Abb. 5.40).

<sup>59</sup> Siehe auch Thompson 1990.

Gerade der im Gegensatz zu normalen Interaktionen erweiterte Einwirkungsumfang der Interoperationen impliziert, dass sich Umwelt und Agenten gegen- und wechselseitig bedingen und beeinflussen. Zurecht wird im Rahmen dieses Buches der klassische Begriff der Handlung als Form der Interaktion erweitert und als *Interoperation* ausgedrückt, die für zielgerichtetes, notwendigerweise bewusstes Verhalten und Einwirken auf die Umgebung steht.

Spätestens bei der Ausgestaltung des Agentenbegriffes scheint es nochmals angebracht, die Begriffe Handlung, Verhalten, Aktion und Interoperation voneinander abzugrenzen, insbesondere deshalb, weil deren Verwendung in diesem Buch von der etwa in der Psychologie und in der künstlichen Intelligenz abweicht. In der Psychologie wird ein menschliches Verhalten als Handlung bezeichnet, wenn es aus Sicht des Handelnden zielgerichtet, intentional und damit sinnbehaftet ist. Ein Beobachter wird ein Verhalten dann als Handlung bezeichnen, wenn er diesen Sinn erkennen, nachvollziehen oder verstehen kann. Als Verhalten werden im behavioristischen Sinn alle beobachtbaren Reaktionen eines Organismus auf einen Reiz bezeichnet. Im Unterschied zu Handlungen kann man Verhalten zwar Ursachen, aber unter Umständen keinen Sinn zuweisen. So kann dasselbe Verhalten unterschiedlichen Handlungen zugrunde liegen. In der KI werden oftmals Interaktionen eines Agenten als *Aktionen* (actions) bezeichnet. Ausgehend von diskreten und statischen Umgebungen ist eine Aktion formal die von einem Agenten verursachte Überführung eines Zustands in einen Folgezustand. So werden für ein zielgerichtetes Handeln einzelne Aktionen zu komplexen Handlungsplänen zusammengefasst.

Als *Verhalten* werden weiterhin zeitlich ausgedehnte, situationsgesteuerte Interoperativen als Sequenzen bezeichnet. Verhalten wird also durch die Ausführung einzelner Interoperationen realisiert. Als aktiver Bestandteil ist ein solcher Agent bei der Ausführung seiner Funktion auch den Problemen und damit den daraus resultierenden Anforderungen seiner problembehafteten Umgebung ausgesetzt. Insofern kann man davon sprechen, dass der Agent in einer *Problemdomäne* interoperiert und damit in diese eingebettet ist. Der hier verwendete Begriff der Problemdomäne wird somit synonym zum Begriff der Umwelt oder der Umweltumgebung verwendet. Über die Wahrnehmungssysteme erhält der Agent Daten und Information über den Zustand der Problemdomäne und über die reflektionschen Systeme auch Daten und Informationen über sich selbst. Durch die Interoperativen und das dadurch bedingte Einwirken lässt sich der Zustand der Problemdomäne als auch der Zustand des Agenten verändern.

In der Fachliteratur werden die Problemdomänen entsprechend verschiedener Eigenschaften kategorisiert, wobei in der Regel die Klassifikation aus der Perspektive des interoperierenden Agenten erfolgt:<sup>60</sup>

- *Deterministisch vs. nicht-deterministisch*: Eine Problemdomäne ist deterministisch, wenn sich aus dem aktuellen Zustand und der Interoperation des Agenten der zukünf-

---

<sup>60</sup> Vgl. Russel und Norvig 1995.

tige Zustand der Problemdomäne und des Agenten berechnen lässt. Dies bringt den praktischen Umstand mit sich, dass alle Interoperationen eines Agenten in einer deterministischen Umgebung immer die erwarteten Resultate bzw. Effekte erzielen. Dadurch sind solche Umgebungen einfacher als nichtdeterministische, in denen ein Agent lediglich statistische Aussagen über das Eintreffen von Ereignissen und dem Erzielen von Effekten treffen kann.

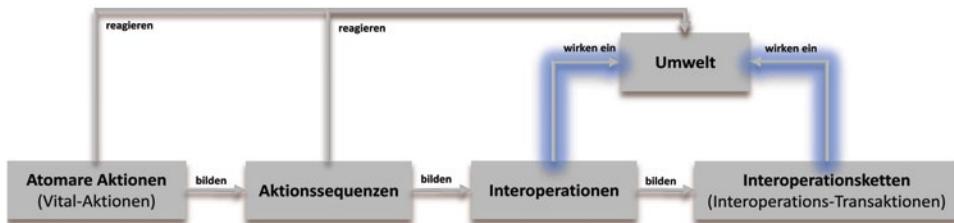
- *Statisch vs. dynamisch*: Als statisch wird eine Problemdomäne bezeichnet, wenn der Zustand der Problemdomäne ausschließlich über die Interoperationen des Agenten den Zustand der Domäne beeinflusst werden kann. Ändert sich der Zustand ohne Eingriff des Agenten, was insbesondere auch in Multiagentensystemen der Fall ist, spricht man hingegen von einer dynamischen Umgebung. Statische Umgebungen sind damit in Bezug auf die Handlungsplanung, -durchführung und -kontrolle einfacher als dynamische, da der Agent in einer statischen Umgebung keinem Zeitdruck ausgesetzt ist und ihm die notwendige Ressource für Planung und Ausführung der notwendigen Interoperationen zur Verfügung steht. Außerdem spielt in einer statischen Umgebung nur die Reihenfolge der Interoperationen eine Rolle, während in dynamischen Umgebungen auch der Zeitpunkt einer Interoperation von Bedeutung ist.
- *Diskret vs. kontinuierlich*: Diese Klassifikation bezieht sich auf die Mächtigkeit der Zustände, durch die eine Problemdomäne charakterisiert ist. Sind in einer Problemdomäne nur eine wohl definierte und abzählbare Anzahl von Zuständen vorhanden, so wird diese als diskret bezeichnet. Zeichnen sich hingegen die Zustände durch kontinuierliche Größen aus, wie beispielsweise Länge, Gewicht oder Geschwindigkeit, spricht man von einer kontinuierlichen Umgebung.
- *Vollständig vs. nicht vollständig*: Kann der Agent den gesamten momentanen Zustand der Umgebung wahrnehmen und verfügt damit über ein vollständiges Weltwissen, wird die Problemdomäne als vollständig bezeichnet. Hat der Agent nur partiellen Einblick in den Zustand der Umgebung, wird diese als nicht vollständig, oder partiell bezeichnet. Vollständige Umgebungen vereinfachen die Handlungsplanung, -ausführung und -kontrolle, da der Agent über das vollständige Wissen verfügen bzw. sich erschließen kann.

Die Charakteristika der Problemdomänen lassen sich mit Hilfe einer Gegenüberstellung zwischen dem Schach und dem Fußballspiel noch einmal verdeutlichen (Tab. 5.6).

Ein zentrales Problem bei der Entwicklung besteht darin, dass ein Agent aufgrund der wahrgenommenen Situation der Gegenwart, seinem Wissen über die Vergangenheit und seinen Zielen, für die Zukunft entscheiden muss, welche der zur Auswahl stehenden Interoperationen konkret zur Ausführung kommen sollen. Die Planung zur Ausführung bedingt eine Entscheidung, die sowohl den langfristigen Nutzen des Verhaltens eines Agenten optimieren als auch die Angemessenheit an die aktuelle Situation sicherstellen soll.

**Tab. 5.6** Charakteristika der Domäne

	Computer Schach	Roboter Fußball
Umgebung	Statisch	Dynamisch
Zustandsübergang	Zug-orientiert	Echtzeit
Informationszugänglichkeit	Vollständig	Unvollständig
Informationsverarbeitung	Symbolisch	Nicht-symbolisch
Kontrolle/Steuerung	Zentral	Verteilt/dezentral

**Abb. 5.41** Interoperationen

Um erfolgreich sich in einer problembehafteten Situation verhalten und damit interoperieren zu können, muss der Agent einige grundlegende Kriterien erfüllen, die aus der dynamischen, teilweise zugänglichen und nicht-deterministischen Umgebung, resultieren. So kann es beispielsweise zu Zielkonflikten kommen, wenn mehrere Ziele gleichberechtigt und gleichzeitig bestehen. Aber der Agent wird auch mit Situationen konfrontiert, in denen Ziele eher unterschiedlich bewertet werden müssen, weil sie je nach Situation unterschiedlich relevant zu sein scheinen. Ein Agent muss also über die Fähigkeit eines Interoperationsmanagements verfügen, d. h., er sollte zum einen in der Lage sein, eine Zielmanagement zu betreiben, um darin mehrere Ziele zu berücksichtigen und gegebenenfalls darin Zielkonflikte aufzulösen. Zum anderen muss dieses Management auch alternative Zielerreichungen erkennen können, wenn sich beispielsweise herausstellt, dass bei der Verfolgung eines Ziels sich günstigere Gelegenheiten zum Erreichen ergeben. Die Angemessenheit einer Interoperation bzw. des Verhaltens wird auch dahingehend bewertet werden müssen, inwiefern der Agent in der Lage ist, auf nicht vorhersehbare Änderungen in der Umgebung, wie sie beispielsweise in Gefahrensituation auftreten, ausreichend schnell zu reagieren. Dazu ist es notwendig, dass sowohl die Planung, die Entscheidung für ein Verhalten als auch dessen Ausführung in relativ kurzer Zeit erfolgt. Kommt es dann zu einem Verhalten durch Interoperationen, so kann es sich als Vorteil erweisen oder sogar auch zwingend notwendig sein, mehrere solcher Interoperationen gleichzeitig auszuführen (Abb. 5.41).

Man kann also davon ausgehen, dass ein Interoperationsmanagement, das es erlaubt, mehrere Verhalten gleichzeitig auszuführen, sich gerade in dynamischen und damit zeitkritischen Umgebungen bewähren wird. Durch das Parallelisieren mehrerer Interoperationen lässt sich unter Umständen ein Ziel schneller erreichen, bzw. mehrere Ziele gleichzeitig

verfolgen. Trotz dieser Parallelisierung sollte das Gesamtverhalten des Agenten robust sein. Robustheit wird auch dann gefordert, wenn Umgebungen keine perfekte Wahrnehmung zulassen, d. h., wenn einerseits nicht alle relevanten Informationen über die Umgebung vorliegen, verrauscht sind oder gar sich als falsch erweisen. In diesem Zusammenhang gilt es auch zu berücksichtigen, dass die Ausführung von Interoperationen nicht-deterministischer Natur ist, d. h. dass die Effekte einer Interoperation allenfalls mit einer gewissen Wahrscheinlichkeit versehen sind. Letzteres kann sowohl durch die Ungenauigkeiten der Aktorik des Agenten, als auch durch den Nichtdeterminismus der Umgebung bedingt sein. Insofern lässt sich der Aspekt der Robustheit dadurch beschreiben, dass diese vorliegt, wenn das Interoperationsmanagement in solchen Fällen bzw. in solchen Situationen nicht versagt, sondern mit adäquaten, d. h. mit an diese Situation angepassten Lösungen in Form von Interoperationen zu reagieren in der Lage ist. Leistungsfähige und intelligente Agenten sollten auch so prozessual und funktional ausgestaltet sein, dass sich die Qualität des Verhaltens mit der Zeit verbessert. Dies inkludiert die Anforderung, nicht nur kurz-, mittel- und langfristige Änderungen in der Umgebung angemessen zu reagieren, sondern eine gewisse Adaptivität bereitzustellen. Adaptiv zu sein bedeutet in diesem Zusammenhang, aus Fehlern und Erfolgen der Vergangenheit bzw. Gegenwart für die Zukunft zu lernen. Es beinhaltet aber auch die Anforderung, in Fällen, in denen auf ein solches Wissen nicht zurückgegriffen werden kann, sich ein solches Domänenwissen durch Exploration in und mit der Umgebung zu erschließen. Insgesamt bedingt dies Mechanismen, um Domänenwissen vorgeben, erweitern, ändern, löschen und damit verwalten zu können. Schließlich ist es in notwendig, das eigene Verhalten auf die Anforderungen abzustimmen und mit dem Verhalten anderer Agenten zu synchronisieren. Idealerweise handeln die einzelnen Agenten nach einem gemeinsamen Plan, sozusagen als Team und werden damit erfolgreicher sein, als es ein einzelner Agent jemals sein kann. Diesen Anforderungen kann ein rein reaktiver Agent nicht gerecht werden, da er ausschließlich den momentanen Zustand für eine schnelle Entscheidung und Interoperationsausführung heranzieht. Ein kognitiver und damit interoperativer Agent hingegen optimiert sein Verhalten, lernt und berücksichtigt seine und die Ziele anderer Agenten.

Die Basis interoperationaler Agenten und deren Fähigkeit zur Planung und Ausführung solcher Interoperationen bildet deren Fähigkeit, rationale Entscheidungen treffen zu können. Eine Entscheidung wird dabei in diesem Zusammenhang dann als rational bezeichnet, wenn sie auf die Erreichung von Zielen ausgerichtet und auf Wissen fundiert ist. Als Grundlage der späteren Realisierung solcher rationaler Entscheidungen in nicht-deterministischen Umgebungen dient das Erwartungs-mal-Wert Prinzip der Entscheidungstheorie. Dieses Prinzip formuliert allgemein, dass eine Entscheidung sowohl den Wert als auch die Unsicherheit der Konsequenzen berücksichtigen muss.

Die Entscheidungstheorie beschäftigt sich mit Situationen, in denen ein Entscheider sich selbst durch eigene Aktion in einen von mindestens zwei unterscheidbaren Zuständen versetzen kann. Als verhaltensorientierte Entscheidungstheorie versucht sie die resultierenden Aktion vorherzusagen und dementsprechend bei der Planung zu berücksichtigen. Als nor-

mative Theorie deklariert sie bestimmte Aktionen als „rationale“ Entscheidungen und untersucht die Konsequenzen verschiedener Rationalitätskonzepte. Eher als präskriptive Theorie entwickelt sie Handlungs- bzw. Verhaltensanweisungen mit entsprechenden Hilfssystemen, die bei gegebener Zielsetzung und Akzeptanz eines bestimmten Rationalitätsprinzips die Auswahl einer entsprechenden Handlungsalternative ermöglichen. Insgesamt betrachtet die Entscheidungstheorie die Auswahl einer Handlungs- bzw. Verhaltensalternative in ihrer Abhängigkeit von den erwarteten Konsequenzen und den jeweiligen aktuellen und individuellen Präferenzen des Systems auf der Menge möglicher Konsequenzen.<sup>61</sup>

In Anlehnung an die Erfahrungen aus dem Forschungsbereich der Künstlichen Intelligenz und dort speziell bei den Expertensystemen arbeitet der Ansatz der interoperativen Agenten mit expliziten Repräsentationen von Zustandsräumen, um in diesen Schlussfolgerungen zu ziehen. Mittels Erfahrungen und Wissen über vergangene Interoperationen werden zukünftige Situationen antizipiert, um dadurch erkennen zu können, welche Interoperationen in der gegebenen Situationen zu Lösungen führen, durch die sich dann letztlich die Ziele des Agenten erfüllen lassen.

Expertensysteme sind produktionsregelbasierte Programme, mit denen das Spezialwissen und die Schlussfolgerungsfähigkeit qualifizierter Fachleute auf eng begrenzten Aufgabengebieten rekonstruiert werden soll. Expertensysteme benötigen daher detaillierte Einzelkenntnisse über das Aufgabengebiet und Strategien, wie dieses Wissen zur Problemlösung benutzt werden soll. Um ein Expertensystem zu bauen, muss das Wissen also formalisiert, im Computer repräsentiert und mit Hilfe einer Problemlösungsmethode manipuliert werden. Expertensysteme unterscheiden sich von wissensbasierten Systemen dadurch, dass ihr Wissen ausschließlich von Experten stammt. Eine häufig verwendete Wissensrepräsentation, die von der Trennung zwischen Wissen und Problemlösungsmethode ausgeht, sind Regeln der Form *Wenn X dann Y*. Diese funktionale Trennung zwischen dem Expertenwissen und den Problemlösungsstrategien spiegelt sich bei der Architektur in den beiden Hauptmodulen Steuersystem und Wissensbasis wider. Das Steuersystem enthält Programmcode für die Problemlösungsstrategie und für die Benutzerschnittstelle, wobei sich letztere in die drei relativ eigenständigen Teile für den Benutzerdialog, für die Generierung von Erklärungen und für den Wissenserwerb aufteilen lässt. Auch die Wissensbasis besteht aus verschiedenen Teilen. Eine Unterteilung orientiert sich am Gebrauch des Wissens. Die daraus resultierenden Wissensarten sind Faktenwissen, Ableitungswissen und Steuerungs- oder Kontrollwissen. Die wichtigsten Wissensrepräsentationen in Expertensystemen, die immer mit Prozeduren zu ihrer Verarbeitung gekoppelt sind, umfassen Regeln mit Vorwärts- und Rückwärtsverkettung, Frames (Objekte) mit Vererbung, Defaults (Standardannahme) und zugeordneten Prozeduren, Constraints mit Propagierungs- und Relaxierungstechniken sowie Darstellungen und Inferenztechniken (Inferenzverfahren) zum Umgang mit sicheren bzw. unsicherem, vollständigen bzw. unvollständigen, räumlichen und zeitlichem Wissen.

Bildlich kann man sich das zielorientierte Verhalten eines Agenten dadurch vorstellen, dass unter Anwendung von Wissen über die Effekte Interoperationen entwickelt werden, die, ausgehend vom angegebenen Startzustand, die Erreichung der Ziele als Endzustand gewährleisten. Dabei sind entgegen der klassischen bzw. theoretischen Handlungsplanung

---

<sup>61</sup> Siehe auch Poddig 1992.

bei der technischen Umsetzung dann zusätzlich die allgemeinen Kriterien der Korrektheit, Vollständigkeit, Optimalität der verwendeten Algorithmen und die Forderung der Ausdrucksfähigkeit zu beachten sowie Komplexitätstheoretischen Aspekte zu berücksichtigen.

Annahmen der klassischen Handlungsplanung sind, dass die betrachtete Umgebung statisch, deterministisch und vollständig zugänglich ist. Die Ziele gelten als konsistent und explizit vorgegeben, und für die Erstellung des Plans ist ausreichend viel Zeit vorhanden. Moderne Planungsverfahren versuchen, diese nicht realistischen Einschränkungen aufzuheben. So erstellt hierarchisches Planen zunächst Pläne unter Verwendung abstrakter Operatoren und versucht dann diese zu verfeinern, wodurch der ursprüngliche Suchraum stark eingeschränkt werden kann. Schnellere Planungsverfahren werden entwickelt, um größere oder zeitbeschränkte Planungsaufgaben lösen zu können. Probabilistisches Planen erweitert den Anwendungsbereich auf nicht-deterministische Umgebungen. Konditionales Planen erweitert den Anwendungsbereich auch auf nicht vollständig zugängliche Umgebungen.

Diese Kriterien bzw. Aspekte finden in der Tatsache ihre Berechtigung, dass in dynamischen Umgebungen durch die schnellen und unvorhergesehenen Änderungen der Situation Pläne schnell ihre Validität einbüßen bzw. ihre Gültigkeit verlieren. Hier gilt der Grundsatz, dass je dynamischer eine Umgebung sich gestaltet, desto weniger lohnt sich der Aufwand, langfristig zu planen. Um diese Vernachlässigung von Planung auszugleichen, lassen sich Motivationskonzepte heranziehen. So werden beispielsweise motivationale Aspekte im Zusammenhang mit rationaler Entscheidungsfindung unter anderem im Rahmen von sogenannten BDI-Architekturen behandelt (*Belief*: informationaler Zustand, Annahme; *Desire*: motivationaler Zustand, Ziel; *Intention*: deliberativer Zustand des Agenten, intendiertes Ziel). Ein Schwerpunkt stellt hier sicherlich die hierzu unabdingbare Informationsverarbeitung dar, wobei inzwischen das Paradigma der Symbolverarbeitung mit subsymbolischen Ansätzen angereichert wird. Neben der Erforschung und Modellierung vieler Einzelphänomene wird damit versucht, umfassende kognitive Architekturen zu modellieren, um damit wiederum ein breites Spektrum menschlicher und artifizieller Kognition zu realisieren.

Bei einem BDI-Ansatz wird die Umgebung als Zustandsbaum repräsentiert mit Verzweigungen für mögliche Zustände in der Zukunft und einer Vergangenheitslinie. Annahmen des Agenten sind dabei die für möglich gehaltenen Zustände, Ziele sind wünschenswerte Zustände, und Intentionen sind gerade angestrebte wünschenswerte Zustände der Umgebung. Verzweigungspunkte des Zustandsbaums repräsentieren Wahlmöglichkeiten des Agenten (Wahlknoten, choice nodes) sowie Nichtdeterminismus von Aktionen des Agenten (chance nodes). BDI-Architekturen erlauben die Berücksichtigung motivationaler Zustände des Agenten. Ein Agent kann mehrere, unterschiedlich wichtige Ziele besitzen, von denen er zu einem Zeitpunkt nicht alle anstrebt. Ein weiterer Vorteil ist, dass Nichtdeterminismus der Aktionen des Agenten explizit repräsentiert werden kann. Problematisch erscheint jedoch, dass die Interoperationskontrolle des Agenten ausschließlich zielgerichtet ist. Reaktionen auf unerwartete Situationen sind nur durch die Bildung entsprechender neuer Ziele und Intentionen, also durch einen kompletten Deliberationsprozess möglich. Das kann in dynamischen Umgebungen zu lange dauern, wenn beispielsweise auf eine Gefahrensituation reagiert werden soll.

Im Gegensatz zur rationalen Interoperation versucht die reaktive Interoperation ohne explizite Repräsentation zur Ausführung zu kommen. Vielmehr wird die Umgebung selbst als Repräsentation herangezogen. Dabei stehen weniger die Optimierung und Vollständigkeit, sondern eher Robustheit und Schnelligkeit im Vordergrund der Algorithmisierung. Dieser Ansatz kommt vor allen in solchen dynamischen Umgebungen zum Tragen, wo Informationen unvollständig und falsch sein können. Er findet sich bei höheren Organismen, die sich durch ihre Lernfähigkeit auszeichnen, obwohl Sensorik und Aktorik nur indirekt miteinander verbunden sind. Die spätere Ausgestaltung der interoperativen Agenten lehnt sich dabei an die Forschungen der künstlichen Intelligenz an, aus denen inzwischen verschiedene Architekturen hervorgegangen sind, um möglichst einfach robustes Verhalten zu gewährleisten. Daneben wird auch auf Arbeiten zurückgegriffen, in denen reaktives und zielgerichtetes Handeln kombiniert werden, da es sich je nach Problemdomäne und Umgebung bei rein reaktiven Ansätzen als schwierig erweist, ein gewünschtes zielgerichtetes Verhalten auch tatsächlich zu realisieren. Exemplarisch und damit für weitere Arbeiten stellvertretend sei auf die Architektur von Brooks hingewiesen, deren wesentliches Merkmal durch einen horizontalen Schichtenaufbau gegeben ist. In Anlehnung an Brooks lassen sich für die verschiedenen Verhaltensschichten in einer Subsumptionsarchitektur die folgenden Kompetenzstufen vorstellen (Abb. 5.42).<sup>62</sup>

Mit Hilfe der Subsumptionsarchitektur gelang es, echte Roboter relativ schnell mit einfacherem aber robustem Verhalten sowie der Hindernisvermeidung oder Wander- und explorativen Fähigkeiten auszustatten. Auch hier soll ein Auszug der Leistungsmerkmale genügen:

- Kollisionen mit anderen mobilen oder stationären Objekten vermeiden.
- Ziellos umherwandern, ohne irgendwo anzustoßen.
- Die Umwelt erkunden, indem man entfernte Objekte wahrnimmt und auf sie zusteuert.
- Eine Karte der Umgebung erstellen und Routen zwischen einzelnen Orten planen.
- Veränderungen in der statischen Umgebung bemerken.
- Logische Schlüsse über die Umwelt durch identifizierbare Objekte ziehen und Aufgaben erfüllen, die sich auf bestimmte Objekte beziehen.
- Pläne formulieren, die den Zustand der Umwelt in erwünschter Weise verändern, und diese Pläne ausführen.
- Schlüsse über das Verhalten von anderen Objekten in der Umwelt ziehen und die Pläne dementsprechend anpassen.

Das Ziel, auch höhere kognitive Fähigkeiten zu modellieren, lässt sich jedoch nicht erreichen. Insbesondere ist es damit auch nicht gelungen, Ziele explizit zu repräsentieren und damit ein zielgerichtetes Verhalten der Agenten zu erreichen. Es liegt also nahe, reaktive und rationale Interoperationen zu kombinieren, indem jeweils eine reaktive und eine rationale Komponente in einer gesamten Architektur vereinigt werden. Dabei stellt sich die sogenannte Kohärenzproblematik, indem zunächst geklärt werden muss, wie diese beiden

---

<sup>62</sup> Vgl. Brooks 1986, S. 15–36.



**Abb. 5.42** Verhaltensbasierte Aufteilung von Systemfunktionen

Komponenten zusammenarbeiten. Ein Ansatz dazu ist, je nach Situation zwischen reaktiver und rationaler Interoperation zu wechseln.<sup>63</sup> Dabei gilt der Grundsatz, dass, solange rationale Interoperationen verfolgt werden, solange nichts Unvorhergesehenes auftritt. Tritt aber eine unerwartete Situation ein, wird auf reaktive Interoperationen umgeschaltet und zwar solange, bis ein neuer, der Situation angemessener, modifizierter Plan entwickelt ist.

Eine weitere Möglichkeit die Kohärenzproblematik zu lösen, besteht darin, dass eine direkte Kommunikation zwischen rationalen und reaktiven Interoperationen zur Koordination erfolgt. Der Vorteil einer solchen *hybriden Architektur* ist, dass der damit mögliche modulare Aufbau die Konzeptionalisierung und die Implementierung des Agenten unter Umständen vereinfacht.

Von Modularität spricht man hier in enger Anlehnung an die Modularisierung von klassischen Computerprogrammen immer dann, wenn eine Aufteilung in relativ selbständige Teile erfolgt, die nur durch klar definierte Schnittstellen Daten austauschen.

<sup>63</sup> Vgl. Jensen und Veloso 1998.

Die Module können idealerweise getrennt voneinander modelliert und realisiert und zum Teil dann auch getrennt getestet werden. Durch eine solche Trennung ist es auch möglich, reaktive und rationale Interoperationen parallel ausführen zu lassen. Neben dem Versuch, zielgerichtete und rationale Interoperationen durch einen hybriden Architekturansatz zu realisieren, wurden auch integrierte monolithische Ansätze erarbeitet. Diese zeichnen sich dadurch aus, dass sie im Gegensatz zu hybriden Architekturen keine getrennten Module für rationale und reaktive Interoperationen vorsehen, sondern mit Hilfe eines einzigen Mechanismus zielgerichtetes, rationales und reaktives Verhalten zu erzielen.

Des Weiteren kann der Ansatz verfolgt werden, die Spezifikation eines Agenten in Form von Zielen und *Zielreduktionsregeln* durch ein zelluläres Automatensystem abzubilden. Dieses Automatensystem steuert den Agenten, indem es die wahrgenommene Situation als auch den internen Zustand des Agenten direkt auf Interoperationen durch Effektoren abbildet. Diese Abbildung muss dabei nicht vollständig sein, sondern kann Lücken aufweisen, indem Situationen, von denen aus die Erreichung des Ziels nicht möglich ist, im Programm nicht vorgesehen werden. Die Abbildung ist außerdem in der Regel nicht bijektiv, d. h. führen mehrere Interoperationen in einer Situation zur Zielerreichung, wird eine davon per Zufallsverfahren ausgewählt. Die Bestimmung der Interoperationen erfolgt durch Zielreduktion des obersten Ziels mit Hilfe vorgegebener Reduktionsregeln. Ziele können dabei entweder Ausführungsziele, Erhaltungsziele, Erreichungsziele sowie beliebige bool'sche Verknüpfungen dieser einzelnen Ziele sein. Damit wird dem Umstand Rechnung getragen, dass wichtige Ziele nicht in jeder Situation zur Verhaltensbestimmung relevant sind. Daher lassen sich sogenannte Relevanzkriterien von Zielen einführen, durch die festgelegt wird, in welchen Situationen ein Ziel handlungs- bzw. entscheidungsrelevant ist. Zusammen mit den Mechanismen der Aktivierung und Hemmung von Interoperationen ermöglichen Relevanzkriterien die Modellierung unterschiedlich ausgeprägter Zielkonstellationen. So lassen sich dann die Erreichungsziele als Ziele definieren, die einen momentan nicht gegebenen Zustand erreichen wollen. Sie sind in der Regel umso relevanter, je näher der Agent das angestrebte Ziel vor sich hat. Erhaltungsziele lassen sich dann als Ziele auffassen, die einen gegebenen Zustand unbedingt erhalten wollten. Hier greift dann in der Regel der Mechanismus der Hemmung ein. Erhaltungsziele sind in der Regel umso relevanter, je weniger erfüllt sie sind. Generell gilt, dass sowohl Erreichungs- als auch Verhaltensziele einen stetigen Einfluss auf die Verhaltens- und damit Interoperationsauswahl haben. Reduktionsregeln ermöglichen es, komplexe Ziele auf mehrere einfache Ziele zu verteilen und damit die Komplexität an sich zu reduzieren. Sind mehrere Reduktionen möglich, können diese priorisiert werden, so dass beim Fehlschlagen einer Reduktion die nächste erfolgversprechende Reduktion zur Ausführung kommt. Solche Agenten zeichnen sich dadurch aus, dass sie hoch performant arbeiten, da Wahrnehmungen mittels vorwärts gerichteter Schlussfolgerungsmechanismen direkt auf die Effektoren des Agenten abgebildet werden. Auch erreicht man damit ein zielorientiertes Verhalten, jedoch scheinen sich automaten-basierte Agenten bei der dynamischen Zielerreichung schwer zu tun, da sie nicht zu priorisieren und nicht zur Berücksichtigung von Erfolgswahrscheinlichkeiten von Interoperationen in der Lage sind.

Mit Hilfe von entscheidungstheoretischem Planen können auch in nicht-deterministischen Umgebungen optimale Interoperationsstrategien berechnet und im Rahmen von Entscheidungsprozessen in konkrete Interoperationen überführt werden. Ein solcher Entscheidungsprozess gestaltet sich bekanntlich durch eine endliche Menge von diskreten Zuständen, eine endliche Menge von Aktionen, eine Zustandsübergangsfunktion, die für jeden Zustand und für jede Aktion eine Wahrscheinlichkeitsverteilung über alle Zustände angibt und letztlich eine Belohnungsfunktion sowie eine Kostenfunktion. Dem allem liegt die Annahme zugrunde, dass die Informationen über die aktuelle Situation dahingehend ausreichend sind, um Vorhersagen über die Zukunft zu machen. Auf Basis dieser Informationen lassen sich dann optimale Verhaltensstrategien ableiten. Entgegen der klassischen Verhaltensplanung lassen sich diese Strategien allerdings nicht einfach als Wirkungen von Interoperationen beschreiben. Zum einen werden die Effekte von Interoperationen als nicht-deterministisch angenommen und zum anderen muss man berücksichtigen, dass der Agent während der Ausführung der Interoperation oder ganzer Interoperationsketten neue Information über die von ihm und den anderen Agenten beeinflusste Umgebung erhält, die dann wiederum die Interoperationsauswahl beeinflussen wird. Der Agent entscheidet sich je nach aktuellem Zustand für die Interoperation, die den höchsten erwarteten Nutzen bringt. Problematisch ist an diesem Ansatz jedoch, dass die hierfür notwendigen Algorithmen alle Zustände explizit berechnen müssen, wobei die Anzahl der Zustände exponentiell mit der Anzahl der Eigenschaften wächst. Daher wird versucht, Algorithmen zur Berechnung einer optimalen Verhaltensstrategie zu entwickeln, die keine explizite Aufzählung aller Zustände benötigen. Solche Algorithmen verfolgen das Prinzip zur Lücke, indem sie realen Problemen und Domänen eine Struktur unterstellen, machen sich zunutze, dass viele reale Probleme eine Struktur besitzen. Diese Unterstellung führt dann beispielsweise zu der Annahme, dass Interoperationen in vielen Zuständen dieselben Effekte erzielen, eine Belohnungsfunktion gleich für viele Zustände gewählt werden kann und die Kostenfunktion oft nicht vom aktuellen Zustand, sondern nur von der jeweiligen Interoperation abhängt.

Für deterministische als auch für nicht-deterministische Domänen lassen sich optimale Verhaltensstrategien aus den Problemrepräsentationen ableiten und in Entscheidungsäbäumen relativ effizient darstellen, speichern und verarbeiten.

Ein Entscheidungsbaum ist eine Datenstruktur, die zur Steuerung der Suche in einem Problemlöseverfahren dient. Sie lassen sich dann gut einsetzen, wenn die Suche in einer Folge diskreter Einzelentscheidungen besteht, wobei in jedem Schritt eine Möglichkeit aus einer kleinen Menge von Alternativen ausgewählt werden kann.

Solche Strategien haben den Nachteil, dass für deren Erstellung die Ziele bekannt und konsistent sein müssen. Konflikte zwischen Zielen sowie unterschiedlich wichtige Ziele können in der Regel nicht berücksichtigt werden.

Zur Repräsentation nicht-deterministischer Aktionen wurden unter anderem auch Bayesische Netzwerke eingesetzt. Die bedingten Wahrscheinlichkeiten der Bayes Netzwerke können

dabei als Entscheidungsbäume kompakt repräsentiert werden. Auch die Belohnungs- und Kostenfunktion sowie Strategie- und Wertfunktion können als Entscheidungsbäume übersichtlich und effizient repräsentiert werden.

### 5.3.5 Subsymbolische Verfahren

Im Rahmen der Implementierung werden unter dem Begriff des Subsymbolismus im Allgemeinen und dem Konnektionismus im Speziellen informationsverarbeitende Systeme behandelt, die aus sehr vielen einfachen, untereinander verbundenen und Informationen austauschenden Verarbeitungselementen bestehen. In konnektionistischen Systemen steckt daher das „Wissen“ in der Verbindungsstruktur, den Gewichten der einzelnen Verbindungen sowie auch in den Eigenschaften der einfachen Verarbeitungselemente.

#### Konnektionistische Systeme

Als *Konnektionismus* bezeichnet man den interdisziplinären Forschungszweig innerhalb der Kognitionswissenschaften, der sich mit der Analyse und Konstruktion solcher Systeme beschäftigt, die wesentliche Eigenschaften und Verarbeitungsprozesse neuronaler Netzwerke nachbilden.<sup>64</sup> Hierbei sind im wesentlichen Informatiker, Physiker, Mathematiker, Psychologen, Neurophysiologen und Philosophen beteiligt. Das von diesen Forschern propagierte Prinzip neuronaler Netze besteht darin, dass durch das Überschreiten von Schwellenwerten (elektrische Impulse) eine Einheit („unit“) mit Hilfe von Eingabemustern aktiviert wird, bei Nichtüberschreitung des Schwellenwertes hingegen das künstliche Neuron passiv bleibt, so dass Informationen als stetige Zustandsveränderung in die Struktur eines Netzes eingehen und nicht als lineare Symbolketten abgearbeitet und diskret gespeichert werden. Der Fokus des Konnektionismus, wie er im Zusammenhang mit der Robotik auch zur Anwendung kommt, liegt in der Konstruktion informationsverarbeitender Systeme, die sich aus vielen primitiven, uniformen Einheiten zusammensetzen und deren wesentliches Verarbeitungsprinzip die Kommunikation zwischen diesen Einheiten ist. Die Kommunikation erfolgt dabei in Form der Übertragung von Nachrichten oder Signalen. Ein weiteres Charakteristikum dieser Systeme ist die parallele Verarbeitung von Information innerhalb des Systems durch eine gleichzeitige Aktivität vieler Einheiten. Die generelle Zielsetzung dieser Fokussierung ist die Modellierung kognitiver Prozesse, unter anderem durch Integration derartiger „klassischer“ konnektionistischer Modelle. Der Konnektionismus wurde initiiert durch das Interesse daran, wie das menschliche Gehirn komplexe kognitive Leistungen vollbringen kann. Grundlegende Forschungsergebnisse der Neurophysiologie führten zur Basisannahme des klassischen Konnektionismus, dass die Informationsverarbeitung auf der Interaktion einer großen Anzahl einfacher Verarbeitungselemente basiert und in hohem Maß parallel erfolgen soll<sup>65</sup>. Diese massiv parallele Verarbeitung, (Par-

---

<sup>64</sup> Siehe auch Strube 1996.

<sup>65</sup> Vgl. McClelland et al. 1986.

allel Distributed Processing) wird dabei als notwendige Voraussetzung für die Realisierung komplexer kognitiver Prozesse postuliert.<sup>66</sup> Diese Grundzüge der heutigen konnektionistischen Modelle sind zurückzuführen auf Forschungsarbeiten in der Neurophysiologie, die in den letzten Jahrzehnten des 20. Jahrhunderts als bedeutendes Ergebnis die These formuliert hat, dass die Informationsverarbeitung im Nervensystem im Wesentlichen auf der Übertragung von „Erregung“ zwischen Neuronen basiert. Innerhalb des Konnektionismus zeichnen sich auch bis heute noch zwei Richtungen ab, in denen konnektionistische Systeme untersucht werden. In der einen Richtung wird von den abstrakten, neuronal-orientierten Modellen ausgegangen, um dann mit formalen Methoden Klassen von Modellen bezüglich ihrer Eigenschaften und Funktionsprinzipien zu untersuchen. Die andere Richtung ist eher anwendungsorientiert, indem sie sich mit der konkreten Realisierung bestimmter Modelle kognitiver Fähigkeiten beschäftigt (Spracherwerb, Sprachverständigen etc.). Im Zusammenhang mit Konnektionismus wird auch oft von „Neuronalen Netzen“ oder „neuronalen Netzwerken“ gesprochen. Die Bezeichnung „Neuronale Netzwerke“ ist zeitlich früher entstanden als der Begriff des „Konnektionismus“ und bezeichnete keine eigene Schule, auch keine grundlegende Neuorientierung, sondern generell konkrete Modelle, die auf dem Transfer grundlegender neuronaler Funktionsprinzipien beruhen, also im wesentlichen einfache Neuronen-Modelle sind und dem Zweck dienen, gewisse (postulierte) Eigenschaften neuronaler Verbände zu demonstrieren.

Die Wahl des neuronalen Netzwerkes als Metapher für die Modellbildung in der Kognitionspychologie und Forschung zur KI ist indirekt (und damit nicht nur, Anmerkung des Autors) das Resultat mancher Fehlschläge der KI-Forschung, Computerprogramme zu entwickeln, die menschliches intelligentes Verhalten nicht nur bezüglich des Leistungsergebnisses simulieren, sondern auch bezüglich der Funktionsweise der zugrundeliegenden Prozesse.<sup>67</sup>

In diesem Buch wird daher der Begriff des neuronalen Netzwerkes immer dann verwendet, wenn grundlegende Eigenschaften oder Verarbeitungsprinzipien von Klassen solcher Neuronaler Netzwerke untersucht und implementiert werden, wohingegen von konnektionistischen Modellen immer dann gesprochen wird, wenn solche Netzwerke in einer bestimmten Interpretation, beispielsweise zur Konzeptionalisierung bzw. Modellierung eines kognitiven Prozesses, eingesetzt werden. Dies führt zu einer weiteren begrifflichen Festlegung, die in diesem Buch konsequent verfolgt wird. Konnektionistische Repräsentation und Modelle werden in der Literatur oft als „sub-symbolische“ Modelle bezeichnet, um sie von klassischen „symbolischen“ Modellen zu unterscheiden. Diese Bezeichnung wird unter anderem auch damit legitimiert, dass verteilte Repräsentationen kraft ihrer feineren Auflösung sozusagen unterhalb der symbolischen bzw. begrifflichen Ebene liegen, letztere in ihnen eingebettet sind oder als emergentes Phänomen aus ihnen hervorgehen.<sup>68</sup> Im

<sup>66</sup> Vgl. Coulouris et al. 2002.

<sup>67</sup> Vgl. Stoffer 2002, S. 278.

<sup>68</sup> Smolensky 1988.

Rahmen dieses Buches wird dieser Bezeichnungskonflikt dadurch gelöst, dass die Symbole so aufgefasst werden, wie sie in der Semiotik, als die Lehre der Zeichen, definiert sind.<sup>69</sup> Symbole sind Zeichen, die diskret sind, die ihrer Form nach nichts über ihre Bedeutung etwas aussagen, damit arbiträr sind, und sich auf etwas beziehen. Erstere beider Eigenschaften unterscheiden sie von anderen Zeichen, Ikonen oder Indexen, letztere Eigenschaft zeichnet sie als Zeichen überhaupt aus.

So lässt sich sowohl das Wort „Apple“, „Apfel“ oder gar „Schrzl“ verwenden, um den Begriff Apfel zu repräsentieren, was der eben erwähnten Formunabhängigkeit („Arbitrarität“) entspricht.

Konnektionistische Modelle haben nun das Potential, auch nicht-diskrete (d. h. kontinuierliche Übergänge enthaltende) arbiträre (d. h. durch ihre Form die Relation zu anderen Repräsentationen ausdrückende) Repräsentationen zu realisieren, die sich nicht im Sinne eines Zeichens auf etwas beziehen müssen (d. h. wieder abbildungsfrei in obigem Sinn sind). Symbole kommen in einem plausiblen Modell erst dann ins Spiel, wenn das System selbst, etwa in der Sprache, sich auf etwas bezieht. Daher sind solche Modelle besser als „sub-konzeptuell“ (unter-begrifflich) oder „nicht symbolisch“ oder aber wie in diesem Buch verfolgt, als „konnektionistisch“ zu bezeichnen.

Unter einem *konnektionistischen System* im engeren Sinn kann man daher ein Netzwerk verstehen, das aus einer Anzahl etwa gleichmächtiger Verarbeitungseinheiten (Units) besteht, die entlang der Verbindungsstruktur des Netzwerks miteinander kommunizieren, und dessen Informationsverarbeitung wesentlich auf der Kommunikation zwischen diesen elementaren Verarbeitungseinheiten beruht. Dabei wird vorausgesetzt, dass die Verarbeitungseinheiten einen eindimensionalen geordneten Zustandsraum haben, der durch die Menge der reellen Zahlen  $R$  oder durch die Menge der ganzen Zahlen  $Z$  beziehungsweise ein Intervall daraus oder durch die Menge  $\{0,1\}$  repräsentiert wird. Ein Zustand wird als *Aktivierungszustand* oder einfach *Aktivierung* und der entsprechende Wert als *Aktivierungsgrad* bezeichnet. Wesentlich ist außerdem, dass die (gerichteten) Verbindungen zwischen den einzelnen Verarbeitungseinheiten gewichtet sind, im Allgemeinen mit ganzzahligen Werten, den sogenannten *Verbindungsstärken* oder *Gewichten*. Die Verarbeitungseinheiten können sich entlang dieser Verbindungen beeinflussen, wobei der Grad der Beeinflussung vom aktuellen Aktivierungsgrad der vorgeschalteten Verarbeitungseinheit und dem Gewicht der Verbindung abhängt. Ist beispielsweise die Verbindung positiv gewichtet, wirkt sie aktivitätserhöhend auf die nachgeschaltete Verarbeitungseinheit Element, ist sie negativ gewichtet, wirkt sie in Richtung einer Aktivitätssenkung. Der Struktur dieses Netzwerks von Kommunikationsverbindungen und den zugeordneten Verbindungsstärken werden die wesentlichen Eigenschaften und Funktionen eines konkreten konnektionistischen Systems zugeschrieben. Insofern ist in dieser Verbindungsstruktur

---

<sup>69</sup> Vgl. Eco 1972.

das gesamte „Wissen“ des Systems „verteilt“ gespeichert. Dem Phänomen des Lernens, das auf einer Modifizierung der Gewichtung der Verbindungen beruht, kommt in konnektionistischen Modellen eine besondere Bedeutung zu, da komplexe konnektionistische Systeme oft nicht mehr explizit konstruierbar sind. Ein weiterer unterscheidender Punkt gegenüber der Symbolverarbeitung ist, dass sich die Art der Wissensrepräsentation in konnektionistischen Modellen vollkommen anders gestaltet. Das „Wissen“ in konnektionistischen Modellen ist im ganzen System verteilt, es ist gespeichert in der Struktur und der Gewichtung des Netzwerks. Konzeptuelle Entitäten werden nicht notwendigerweise „lokal“ durch einzelne Elemente repräsentiert, sondern „verteilt“ durch ein bestimmtes Aktivitätsmuster, an dem mehrere Verarbeitungseinheiten beteiligt sind. Man spricht dann auch von einer „verteilten Repräsentation“ von Konzepten im Gegensatz zu einer „lokalen Repräsentation“, bei der jedem Konzept der modellierten Domäne eine Verarbeitungseinheit zugeordnet wird. Diese verteilte Repräsentation scheint vollkommen neue Möglichkeiten zu bieten, die im Rahmen dieses Buches dem Paradigma der Symbolverarbeitung zur Seite gestellt wird. Konnektionistische Modelle bieten gerade unter den Aspekten „Wissensrepräsentation und -verarbeitung“, „Lernen“, „Selbst-Organisation“ und „Fehler-toleranz“ neue, erfolgversprechende Möglichkeiten.

Im Rahmen der Implementierung von konnektionistischen Systemen wird angenommen, dass die Leistungen kognitiver Systeme ohne den Rückgriff auf Symbolmanipulation und symbolische Repräsentation erklärt werden können. Diese Annahme ist jedoch keine Absage an den Symbolverarbeitungsansatz und damit auch keine an den dieser Arbeit zugrunde liegenden Funktionalismus. Konnektionistische Systeme werden daher auch nicht als empirisches Potenzial für die Ablehnung mentaler und funktionaler Zustände gesehen. Vielmehr wird die Gegenthese dazu vertreten, dass auch konnektionistische Ansätze funktionale Zustände postulieren, weil sie diese implizit voraussetzen. So ist evident, dass jedes neuronale Netzwerk eine Maschine mit endlichen Zuständen darstellt. Theoretisch ist zu jedem Zeitpunkt der Gesamtzustand des Netzwerkes durch die Zustände der Neuronen gegeben, wenngleich diese in praktischer Hinsicht sicherlich eine Herausforderung darstellen. Wesentlich aber ist, dass sich symbolverarbeitende und konnektionistische Systeme nicht nur wechselseitig ersetzen, sondern vielmehr sinnvoll miteinander kombiniert werden können. Insofern sind gemäß dieser Sichtweise auch konnektionistische Systeme zumindest auf der Ebene der Implementierung symbolisch strukturiert.

Der Ansatz der Symbolverarbeitung, der oft zu nur unzureichenden Lösungen beispielsweise bei der Behandlung von Ausnahmen, vagen Werten und unvollständigem Wissen führt, wird in konnektionistischen Modellen ersetzt durch ein inexaktes, „evidentialles“ Schließen, das auf einer verteilten Repräsentation von Entitäten der realen und abstrakten Welt, also Objekten, Fakten, Ereignissen usw., basiert. Assoziative Beziehungen zwischen Repräsentationen von Entitäten der realen und abstrakten Welt, die sich im Eingabe-/Ausgabeverhalten der Systeme widerspiegeln, entsprechen in konnektionistischen Modellen eher einer statistischen Korrelation als einer exakten „Wenn-dann“ – Beziehung und ge-

nauen 1:1-Abbildung. So werden beispielsweise ähnliche Eingabemuster auch ähnliche Ausgabemuster erzeugen und ein Eingabemuster, das nur leicht von einem bekannten Standardmuster abweicht, wird die gleiche, dem Standardmuster zugeordnete Ausgabe erzeugen. Da an einem Entscheidungsprozess in einem konnektionistischen Modell eine Vielzahl von Verarbeitungseinheiten beteiligt sind, erfordert das Erkennen einer gegebenen Situation nicht einen genau spezifizierten Zustand des Systems, sondern eine hinreichende Übereinstimmung zwischen dem gespeicherten Muster und der vorliegenden Situation. Dies ermöglicht auch eine Behandlung von Defaults, Grenzfällen und Abweichungen, was in Systemen mit ausschließlich exaktem logischem Schließen im Rahmen einer Symbolverarbeitung nicht ohne weiteres realisiert werden kann. Da nicht für alle Entscheidungen und Fähigkeiten ein klar bestimmter Satz logischer Regeln angegeben werden kann, sind konnektionistische Modelle in diesen Fällen logik-basierten Systemen überlegen. Ein weiterer Vorteil, der sich aus der Tatsache des verteilten Wissens ergibt, ist die Fehlertoleranz konnektionistischer Modelle. Da viele Verarbeitungseinheiten an einem Verarbeitungsprozess beteiligt sind, spielt die Funktion einer Einheit keine ausschlaggebende Rolle, so dass der Ausfall eines Elementes keine ernsthaften Funktionsstörungen im Gesamtsystem zur Folge haben dürfte.

Lernen kommt in konnektionistischen Modellen eine große Bedeutung zu, da konnektionistische Systeme aufgrund ihrer Komplexität, die aus der parallelen, interagierenden Arbeitsweise der Verarbeitungseinheiten resultiert, im Allgemeinen nicht vollständig konstruierbar sind. Erschwerend kommt hinzu, dass bisher keine generellen Methoden für den Entwurf konnektionistischer Systeme entwickelt worden sind. Die Art des Lernens, wie sie in konnektionistischen Modellen stattfindet, bietet allerdings auch eine große Chance, nämlich statt der expliziten Erfassung jeglichen Wissens die Möglichkeit einer selbständigen Entwicklung der Systeme zu nutzen. Konnektionistische Modelle scheinen sich hierfür besonders zu eignen. Es wurden bisher grundlegende Lernmechanismen entwickelt und diese werden im Rahmen der Konzeptionalisierung bzw. Implementierung auch eingesetzt. Aufgrund der speziellen Struktur konnektionistischer Modelle ist in diesen Systemen im Gegensatz zu herkömmlichen logik-orientierten Systemen eine selbständige Kategorien- oder Konzeptbildung möglich.

Eine brisante Streitfrage ist, ob der Konnektionismus vereinbar mit der „Physical Symbol Systems Hypothesis“ ist, auf der die heutige KI-Forschung basiert.

Diese Hypothese fasst kognitive Prozesse als Transformationen von Symbolstrukturen auf. Symbolstrukturen wiederum sind aus elementaren Symbolen als den bedeutungstragenden Einheiten mittels syntaktischer Regeln in der Form zusammengesetzt, dass sie für etwas in der Welt stehen. Dies lässt die weitere These zu, dass ein symbolverarbeitendes System, gleich auf welche Materie es zur Darstellung seiner Symbole zurückgreift, die notwenige und hinreichende Voraussetzung für intelligentes Verhalten ist.

Diese Frage stellt sich allerdings nur, wenn eine verteilte Repräsentation von Konzepten bzw. Symbolen (*distributed representation*) angenommen wird. Wird der Ansatz gewählt,

dass jedem Konzept genau eine Verarbeitungseinheit entspricht (*local representation*), was allerdings den meisten Neuronale-Netzwerke-Forschern widerstrebt, scheint der konnektionistische Ansatz mit der Symbolverarbeitungshypothese durchaus vereinbar.

In diesem Fall ist Konnektivität eines Systems nicht mehr von der Geschichte seiner Veränderungen zu trennen, sondern ist vielmehr von der Art der Aufgabe abhängig, die dem System gestellt ist. Information ist dann weiterhin nicht nur Information „für das System“, sondern auch nur zu einem bestimmten Zeitpunkt (Lebenszyklus des Systems, Zeitlichkeit, etc.) und die Veränderungen beziehen sich dann nicht nur auf die Verhaltensebene (Ausgabe), sondern werden in der Struktur der Neuronen repräsentiert. Ein solches neuronales Netz mit seiner strukturellen Plastizität ist somit ein Produkt sowohl seines eigenen Lebenszyklus, das wesentlich von der Trainings- bzw. Lernphase vorgeprägt wird, als auch von dem sich darin manifestierenden Umweltbezug.

Konnektionistische Modelle können als formale Strukturen anhand einiger Elemente charakterisiert werden, die ihre Verarbeitungs- und Lernmechanismen beschreiben. Ein konnektionistisches Modell besteht im Wesentlichen aus:

- einer Menge von *Verarbeitungseinheiten* (processing units), die man sich als Systeme mit Gedächtnis und einer inhärenten Funktion vorstellen kann, die in Abhängigkeit von ihrem aktuellen Aktivierungszustand und der momentanen Eingabe ihren neuen Zustand bestimmen und eine Ausgabe produzieren,
- einer *Netzwerkstruktur*, die meistens in Matrixform oder als gerichteter bewerteter Graph dargestellt wird, wobei die Knoten den Verarbeitungseinheiten zugeordnet werden und die bewerteten Kanten die gewichteten Kommunikationsverbindungen repräsentieren.

Die Verarbeitungseinheiten eines konnektionistischen Modells sind spezifiziert durch

- eine wohldefinierte *Menge von Aktivierungszuständen* als Zustandsmenge der einzelnen Verarbeitungseinheiten, die für alle Verarbeitungseinheiten identisch ist,
- eine *Eingabemenge* und eine *Ausgabemenge*, die zulässige Eingaben bzw. Ausgaben für jeweils eine Eingangs- bzw. Ausgangsverbindung festlegen. Im Allgemeinen wird nur eine, für alle Verbindungen des Netzwerks identische Eingabe-/Ausgabemenge verwendet.

Folgende Aspekte entscheiden über die Dynamik eines konnektionistischen Modells:

- eine *Ausgabefunktion* für jede Verarbeitungseinheit zur Bestimmung der aktuellen Ausgabe einer Verarbeitungseinheit in Abhängigkeit vom aktuellen Aktivierungszustand,
- eine *Propagierungs- oder Übertragungsfunktion* zur Berechnung der aktuellen Eingabe in interne Verarbeitungseinheiten anhand der Ausgabe der vorgeschalteten Verarbeitungseinheiten und der Gewichtung der Verbindungen,

eine *Aktivierungsfunktion* für jede Verarbeitungseinheit zur Bestimmung des neuen Aktivierungszustandes einer Verarbeitungseinheit in Abhängigkeit vom aktuellen Aktivierungszustand und der übertragenen Aktivierung.

In den klassischen Modellen werden die Ausgabe- und die Aktivierungsfunktion einheitlich für alle Verarbeitungseinheiten des Netzwerks festgelegt. Unterschiede in den Aktivierungsfunktionen oder in den Ausgabefunktionen der einzelnen Verarbeitungseinheiten werden in einigen Modellen verwendet, um Komponenten mit unterschiedlichen Aufgaben zu realisieren. Anhand verschiedener Typen von Ausgabe-, Propagierungs- und Aktivierungsfunktionen kann eine Klassifizierung der konnektionistischen Modelle vorgenommen werden. Innerhalb dieser Modellklassen kann das Verhalten des konnektionistischen Systems als Ganzes bzw. der einzelnen Elemente durch eine Parametrisierung dieser Funktionen, beispielsweise der Verwendung eines Schwellwertes in der Aktivierungsfunktion, variiert werden. Ein konnektionistisches Modell wird durch Angabe der genannten Elemente und deren Parameter vollständig beschrieben. Bezuglich der Beschreibung dieser Struktur kann eine endliche Menge erreichbarer Aktivierungszustände vorausgesetzt, ein konnektionistisches System als Netzwerk endlicher, ggf. stochastischer Automaten, aufgefasst werden, wobei die Automaten den Verarbeitungseinheiten entsprechen und die Netzwerkstruktur durch einen Digraphen dargestellt wird. Dabei kann die Gewichtung des Netzwerks in die Automaten als Gewichtung der jeweiligen Ausgaben integriert werden. Die Zustandsüberführungs- und die Ausgabefunktion werden als entsprechende Funktionen der Automaten behandelt. Die Propagierungsfunktion und die Umgebungsfunktion werden zusammengefasst in eine Eingabefunktion, die die aktuelle Eingabe für jeden Automaten bestimmt. Die Propagierungsfunktion wird dazu reduziert auf eine direkte Übertragung der gewichteten Ausgaben der einzelnen Elemente. Integrative Verrechnungen mehrerer Ausgaben/Eingaben, die eventuell durch die Propagierungsfunktion vorgenommen werden, können als Bestandteil der Zustandsüberführungsfunktion spezifiziert werden. Modelle mit einer regelmäßigen Verbindungsstruktur können auch als Zellulärautomaten dargestellt werden. Im Rahmen der Implementierung wird auf diese unterschiedliche Beschreibungsmöglichkeiten zurückgegriffen. Weiterhin werden im Rahmen der Implementierung Begriffe und Methoden aus der Linearen Algebra verwendet, die auf einer Darstellung der Aktivitätsübertragung zwischen Elementen durch lineare Abbildungen in einem Vektorraum, dargestellt durch Matrizen, basieren.

## Neuronale Netze

Viele Tätigkeiten können Lebewesen sehr schnell und erfolgreich ohne langes Nachdenken erledigen. So kann das Gleichgewicht beim Gehen oder Fahrradfahren gehalten, Gesichter erkannt und eingeordnet werden, man redet, hört zu, versteht oder man schreibt bzw. liest. Die Liste solcher Tätigkeiten, die ausgeführt werden, ohne dass explizit über die ablaufenden, internen Teilschritte nachgedacht wird, lässt sich beliebig fortsetzen. Alle diese Fähigkeiten lassen sich nicht ausschließlich mittels logischer Schlussregeln erklären.

Der Konnektionismus ist ein Problemlösungsansatz, der auf die Wechselwirkungen vieler vernetzter Einheiten baut.<sup>70</sup> Wesentlich dabei ist, dass diese Einheiten recht einfach gehalten sind, gemäss dem Prinzip: „Keep it simple“. Insofern zeigt sich das Verhalten eines konnektionistischen Systems durch eine große Anzahl einfacher Einheiten, die in Form eines Netzwerkes miteinander verbunden sind. Dabei arbeiten diese Einheiten lokal und kommunizieren via Verbindungen miteinander.

Künstliche neuronale Netze sind konnektionistische Modelle, die ein biologisches neuronales Netz und damit Nervensysteme in Struktur- und Funktionsweise nachzubilden versuchen. Solche Nervensysteme bestehen aus vielen Tausenden oder Millionen von miteinander vernetzten Nervenzellen, die Signale empfangen und neue, von ihnen erzeugte Signale, weitergeben. Die Fortsätze des Neurons, die die Eingangsinformationen sammeln, werden *Dendriten* genannt. Sie übernehmen Signale aus anderen Nervenzellen an spezifischen Kontaktstellen, den *Synapsen*. Der Zellkörper reagiert auf diese Stimuli und fängt an, Signale zu übertragen. Die Ausgabesignale eines Neurons werden durch das *Axon* an andere Neuronen weitergereicht. Diese vier Elemente (Dendriten, Synapsen, Nervenkörper und Axon) bilden die minimale Struktur, die aus biologischen Modellen für Modellierungszwecke übernommen werden, so dass auch künstliche Neuronen als informationsverarbeitende Elemente *gerichtete, gewichtete Eingabeleitungen*, einen *Berechnungskörper* und eine *Ausgabeleitung* besitzen werden.<sup>71</sup>

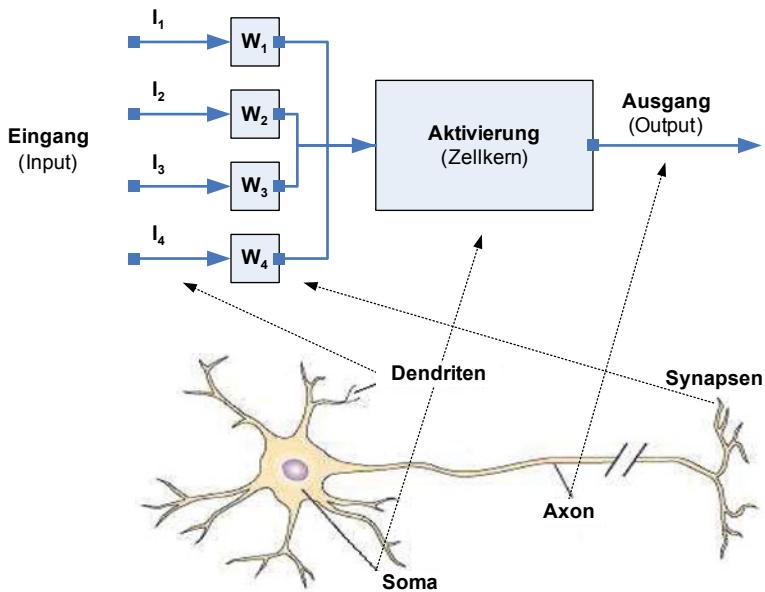
Ein künstliches neuronales Netz besteht aus einigen hundert oder gar tausenden formaler Verarbeitungseinheiten (*units*), die auf verschiedene Arten miteinander verbunden sind. Die chemischen, elektrischen und thermischen Vorgänge, die in natürlichen neuronalen Netzen zu beobachten sind, werden bei den formalen Neuronen durch mathematische Funktionen und Verfahren beschrieben.

Der Autor verwendet den englischen Begriff „Unit“ synonym zu dem des eher neutralen Begriffs der Verarbeitungseinheit im neuronalen Netzwerk und damit auch synonym zum Begriff des biologischen Vorbildes „Neuron“. Die Verwendung eines neutralen Begriffs erleichtert den späteren Übergang von der Theorie in die Praxis, wenn es im Rahmen der Implementierung um die programmiertechnische Ausgestaltung der bis dahin theoretisierten Units in funktionale Verarbeitungseinheiten geht.

Das Modell eines künstlichen neuronalen Netzes setzt sich aus den gleichen Grundbausteinen wie ihre Vorbilder aus der Biologie zusammen. So sehen diese Modelle gleichfalls Dendriten, Soma und ein Axon als Modellelemente vor. Die Funktionen, die sie in einem solchen neuronalen Netz übernehmen, sind mit denen der Vorbilder identisch geblieben. So nehmen die Dendriten Informationen von vorgeschalteten Neuronen auf und leiten diese ans Soma. Das Soma summiert alle eingehenden Signale auf und bestimmt die Größe des ausgehenden Signals, das dann über das Axon und dessen Synapsen an nachgeschaltete Neuronen weitergeleitet wird (Abb. 5.43).

<sup>70</sup> Vgl. Dorffner 1991.

<sup>71</sup> Vgl. Rojas 1992.



**Abb. 5.43** Natürliches und formales Neuron

Typischerweise sollen die Verarbeitungseinheiten einfach sein und es soll keine Informationsverarbeitung, außer durch diese Verarbeitungseinheiten stattfinden, d. h. es gibt keinen externen Kontroll- oder Steuerungsmechanismus. Innerhalb eines Netzwerks kann zwischen *Eingabe-, Ausgabe- und internen Verarbeitungseinheiten (hidden units)* unterschieden werden. Eingabe- bzw. Ausgabeeinheiten haben eine Schnittstelle zur Umwelt, sie erhalten externe Eingaben bzw. produzieren externe Ausgaben. Ein Neuron kann als eine Art Addierer von über Dendriten aufgenommener Impulse aufgefasst werden. Diese Impulse werden mit einer bestimmten Gewichtung im Soma summiert und, sofern die Summe einen bestimmten Schwellwert übersteigt, wird diese Information über das Axon weitergeleitet. Der Kontakt zu anderen Neuronen erfolgt über sogenannte Synapsen. Diese können die kontaktierten Neurone entweder hemmen oder erregen. In diesem Sinne werden Informationen übertragen und verarbeitet.

In der folgenden Abbildung sind zwei künstliche Neuronen modelliert. Die Stärke der Verbindungen, die diese Neuronen über ihre Synapsen mit dem Soma eingehen, ist mit  $w_{ij}$  bezeichnet, wobei die erste Stelle des Indexes die vorgeschaltete, die zweite Stelle die Nervenzelle, auf die sie einwirkt, indiziert. Besteht zwischen dem Neuron  $i$  und dem Neuron  $j$  keine Verbindung, ist der Wert ihrer Verbindung  $w_{ij}$  gleich Null (Abb. 5.44).

Links der Neuronen wirken die Informationen vorgeschalteter Neuronen als Netzeingaben auf das Netz  $\text{Net}_{ij}$  ein. Der Aktivierungszustand der Neuronen ist in der Abbildung mit  $A_i$  und  $A_j$  bezeichnet, die Ausgabe mit  $O_i$  und  $O_j$ . Alle diese Werte sind durch Funktio-

nen untereinander verknüpft und von ihnen abhängig. Die Ausgabe eines Neurons wird durch eine Ausgabefunktion  $f_{\text{out}}$  bestimmt, die sich aus dem Aktivierungszustand eines Neurons ergibt. Es gilt daher folgender formelhafter Zusammenhang:

$$O_i = f_{\text{out}}(A_i)$$

Mit den so errechneten Werten für die Ausgabe vorgeschalteter Neuronen und den Werten der Verbindungsgewichte lässt sich mit der Propagierungsfunktion die Netzeingabe Net berechnen:

$$\text{Net}_j(t) = \sum O_i(t) w_{ij}$$

Der neue Aktivierungszustand eines Neurons ist abhängig von dem alten Aktivierungszustand  $A_j(t)$ , der Netzeingabe  $\text{Net}_j$  und dem Schwellenwert. Der Schwellenwert des Neurons wird mit  $\Phi_j$  bezeichnet. Die Aktivierungsfunktion  $f_{\text{act}}$  berechnet mit diesen Parametern den neuen Wert der Aktivierung  $A_j(t+1)$ :

$$A_j(t+1) = f_{\text{act}}(A_j(t), \text{Net}_j(t), \Phi_j)$$

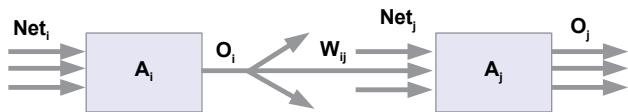
Der Aktivierungszustand, die Ausgabe-, Propagierungs- und die Aktivierungsfunktionen können dabei je nach Anwendungsfall unterschiedliche Formen annehmen.

Die verschiedenen Werte, die der Aktivierungszustand  $A_i(t)$  eines Neurons annehmen kann, lassen sich in diskrete und (quasi-)kontinuierliche Wertebereiche trennen. Oft werden Intervalle in R oder binäre Werte gewählt. Bei kontinuierlichen Wertebereichen beschränken die meisten Modelle den Aktivierungszustand auf ein definiertes Intervall. Das liegt daran, dass die meisten Netzmodelle nichtlineare oder sigmoide Aktivierungsfunktionen und die Identität als Ausgabefunktion verwenden. Ebenfalls zu den kontinuierlichen Wertebereichen gehört die Menge aller reellen Zahlen (Tab. 5.7).

Der Aktivierungszustand eines Netzwerkes zu einem Zeitpunkt t wird dargestellt durch einen n-dimensionalen Vektor  $A(t)$ , wobei das i-te Element des Vektors der Aktivierungszustand  $A_i(t)$  der Verarbeitungseinheit  $u_i$  ist. Bei den biologischen Nervenzellen kann man ebenfalls zwischen Wertebereichen unterscheiden. Das einfachste Beispiel ist ein binärer Wertebereich, der angibt, ob sich ein Neuron im nicht erregten Zustand befindet oder ob gerade ein Aktionspotential ausgeführt wird. Auch die Natriumionen-Konzentration gibt Aufschluss über den Aktivierungszustand. Misst man den relativen Anteil, können die Werte in einem Intervall von 0 bis 100 auftreten.

Die Verarbeitungseinheiten in Form der Neuronen interagieren durch Übertragung von Signalen. Mit Hilfe der *Ausgabefunktion* lässt sich der Wert berechnen, den ein Neuron an ein nachgeschaltetes Neuron weitergibt und damit ins neuronale Netz eingibt. Es verwendet dabei den aktuellen Aktivierungszustand des Neurons. Somit gibt die Ausgabefunktion an, wie stark ein einzelnes Neuron feuert. Die Ausgabefunktion kann verschiedene For-

**Abb. 5.44** Modell der Neuronen eines neuronalen Netzes



**Tab. 5.7** Wertebereiche

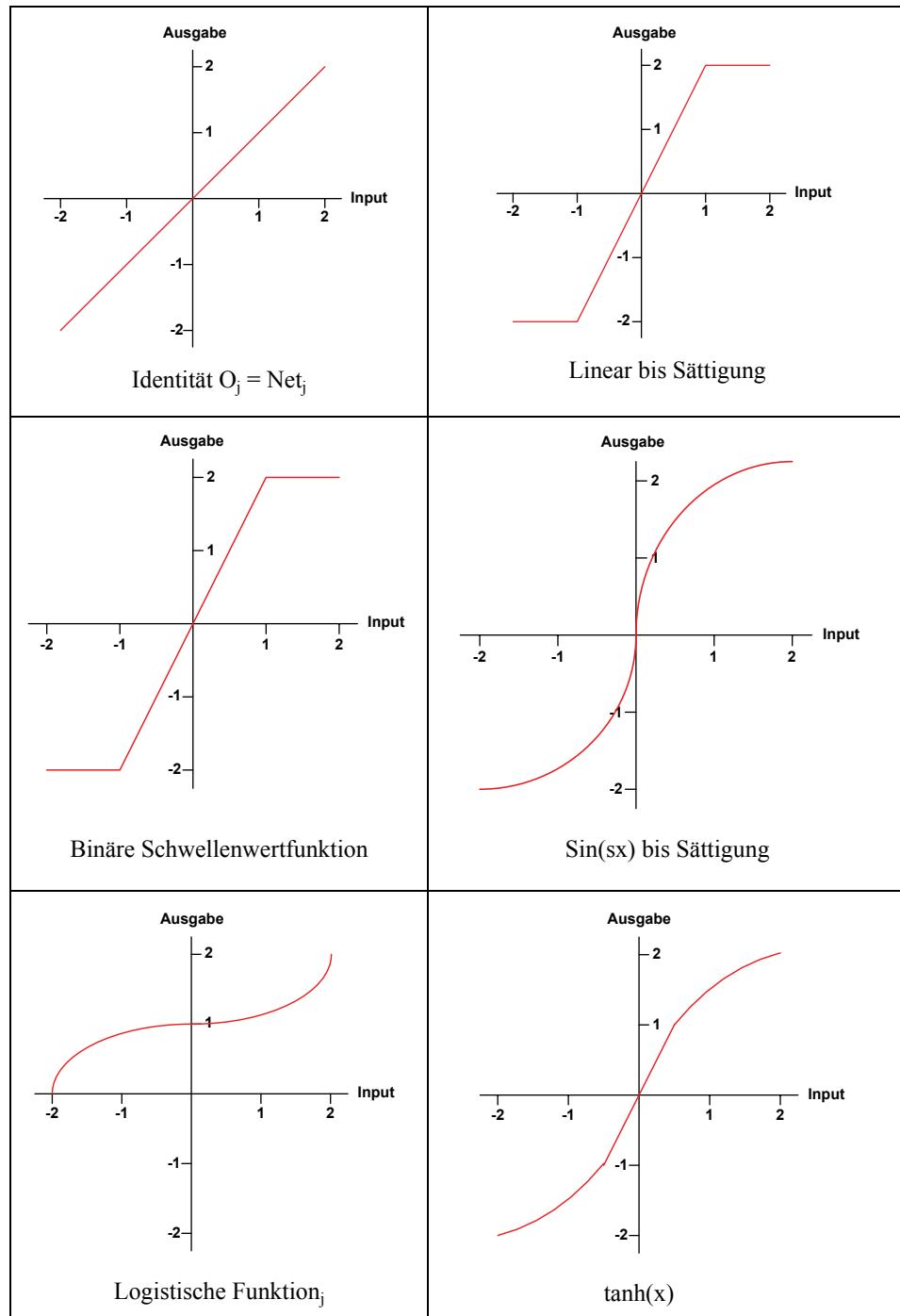
Kontinuierlich	Diskret
Unbeschränkt	Binär
Realer	{0, 1}
Zahlenraum R	{-1, 1}
	{-, +}
	Mehrwertig
	{-100, ..., 100}
	{-1000, ..., 1000}
	Ganzzahlig Zahlenraum Z

men haben. Eine dieser Funktionen ist beispielsweise die Identitätsfunktion. Das bedeutet, dass die Ausgabe identisch mit dem Aktivierungszustand der Zelle ist. Der Wertebereich der Aktivierungsfunktion gibt dadurch den Wertebereich des Aktivierungszustandes an (Abb. 5.45).

Die binäre Schwellenwertfunktion kommt dem biologischen Original sehr nahe, da ein Axon erst beim Erreichen eines Schwellenwertes feuert. Jedoch wird dabei vernachlässigt, dass das Neuron in unterschiedlicher Stärke (Impulsfrequenz) feuern kann. Daher werden lineare Ausgabefunktionen verwendet. Die Abstände, in denen ein Neuron feuern kann, sind durch die Refraktärzeit nach unten hin beschränkt, daher bietet es sich an, eine Mischform aus einer linearen Funktion und einer mit Schwellenwert zu verwenden. In der Abbildung ist ein Beispiel einer solchen semilinearen Funktion dargestellt (linear bis Sättigung). Einige Funktionen stellen eine Glättung dieser semilinearen Funktion dar,  $\sin(x)$ , logistische Funktion,  $\tanh(x)$ . Da sie S-förmig sind, werden sie auch sigmoiden Funktionen genannt. Diese nichtlinearen Funktionen sind die am häufigsten verwendeten.

Die *Propagierungsfunktion*  $Net$  errechnet aus der Summe aller Ausgaben vorgeschalteter Zellen mit ihren jeweiligen Gewichten die Netzeingabe zu einem Zeitpunkt  $t$ , die auf ein Neuron wirkt. Das Vorbild in der Biologie ist ein Neuron, das mehrere Impulse von vorgesetzten Neuronen erhält. Die Impulse können sowohl erregender wie auch hemmender Natur sein. Für das Auslösen eines Aktionspotentials ist die Summe aller Impulse, die zu einem Zeitpunkt ankommen, ausschlaggebend, also die räumliche Summation. Bei den Modellen lassen sich die Neuronen ebenfalls in erregende und hemmende Neuronen separieren. Dies spiegelt sich in den Werten ihrer Gewichte wieder. Sie sind positiv, wenn das Neuron erregend wirkt, negativ bei hemmenden Neuronen. Die Netzeingabe die auf ein Neuron  $j$  zur Zeit  $t$  einwirkt, errechnet sich also aus der Summe aller Netzeingaben, die erregend wirken, abzüglich der Summe aller Netzeingaben, die hemmend sind.

$$Net_j(t) = Net_{j, \text{erregend}}(t) - Net_{j, \text{hemmend}}(t)$$



**Abb. 5.45** Beispiele für Ausgabe- bzw. Aktivierungsfunktionen

Die Gewichte der einzelnen Neuronenverbindungen werden in einer sogenannten (quadratischen) *Konnektionsmatrix* zusammengefasst. Die Anzahl ihrer Spalten, bzw. Reihen ist gleich der Anzahl der Neuronen im Netzwerk. Für ein Netzwerk mit  $n$  Neuronen ergibt sich also eine  $n^2$ -große Matrix.

Der *Aktivierungszustand* eines Neurons muss nicht zwangsläufig immer gleich sein. Nach einem ausgebildeten Aktionspotential kann die Zellmembran noch so stark depolarisiert sein, dass ein schwacher Impuls von einem vorgeschalteten Neuron ausreicht, um das Neuron zum Ausbilden eines neuen Aktionspotentials anzuregen. Bei den künstlichen neuronalen Netzen errechnet die *Aktivierungsfunktion* den neuen Aktivierungszustand einer Zelle, der ebenfalls von seinem alten Aktivierungszustand und der Netzeingabe abhängig ist. Die Aktivierungsfunktion kann wie die Ausgabefunktion linear oder nichtlinear sein, häufig sind auch beide gleich. Es gibt zwei Klassen von Aktivierungsfunktionen. Bei deterministischen Funktionen, wie beispielsweise der Schwellwertfunktion, ist der ausgegebene Wert eindeutig durch die Eingabe bestimmt. Hingegen ist bei stochastischen Funktionen der ausgegebene Wert durch eine Zufallsverteilung von der Eingabe abhängig. Das Aktualisieren des Aktivierungsgrades erfolgt in den meisten Modellen synchron.

Werden nun mehrere solcher Neuronen miteinander verknüpft, entsteht ein Netz von Neuronen. Ein solches Netz besteht aus einer Menge von einzelnen Neuronen, die durch gerichtete und gewichtete Verbindungen miteinander verknüpft sind. Durch die Vernetzungsmuster und der Lernalgorithmen werden unterschiedliche Architekturen definiert. Die Konnektions- bzw. Netzwerkstruktur, über die die Verarbeitungseinheiten verbunden sind und über die sie durch das Schicken von Signalen miteinander kommunizieren, bestimmt wesentlich das Verhalten des konnektionistischen Systems. Die Netzwerkstruktur wird dargestellt durch einen bewerteten Digraphen oder eine *Konnektionsmatrix*  $W$ , bei der ein Eintrag  $w_{ij}$  die Stärke oder das *Gewicht* der Verbindung zwischen der Verarbeitungseinheit  $u_i$  und der Verarbeitungseinheit  $u_j$ , das durch das Element  $u_i$  beeinflusst wird, angibt. Es ist dabei üblich, als Menge der Gewichte  $G$  die ganzen Zahlen zu wählen, wobei negative Gewichte als inhibitorische, hemmende Verbindungen interpretiert werden und positive Gewichte als exzitatorische, aktivitätssteigernde Verbindungen. In einigen Modellen werden verschiedene Verbindungstypen, beispielsweise für exzitatorische und inhibitorische Verbindungen, verwendet. In diesen Fällen wird für jeden Typ eine eigene Konnektionsmatrix aufgestellt, also  $W_1$  für Verbindungen vom Typ 1,  $W_2$  für Typ 2 usw. Werden nur exzitatorische und inhibitorische Verbindungen betrachtet und ergibt sich die Eingabe in ein Netzwerkelement aus einer einfachen Summierung der gewichteten Ausgaben der vorgeschalteten Elemente, ist eine Konnektionsmatrix mit entsprechenden positiven und negativen Werten ausreichend. Hierarchische Netzwerkstrukturen, also solche, deren Elemente in Schichten angeordnet sind, können eingeteilt werden in bottom-up, top-down und interaktiv arbeitende Systeme. In bottom-up arbeitenden Netzwerken kann jedes Element nur von Elementen einer niedrigeren Ebene beeinflusst werden. Die Konnektionsmatrix ist demnach eine obere Dreiecksmatrix. Die Arbeitsweise dieser Netzwerke, die Aktivierungen nur von der Eingabeseite in Richtung der Ausgabeseite propagieren, ist typisch für rein sequentielle Wahrnehmungsprozesse. In top-down arbeitenden Sys-

men können Elemente nur von höheren Ebenen beeinflusst werden, die Konnektionsmatrix ist eine untere Dreiecksmatrix. Derartige Systeme können die Beeinflussung niedriger Verarbeitungsprozesse durch die Aktivität höherer Elemente darstellen, also beispielsweise erwartungsgesteuerte Wahrnehmung. Interaktive Systeme erlauben beide Verarbeitungsmodi, wobei in den meisten interaktiven Modellen jedoch nur ein Signalfluss zwischen benachbarten Ebenen stattfindet. Diese Systeme können eine beschränkte Rückwirkung höherer Verarbeitungsprozesse auf niedrigere darstellen. Die häufigste Systemarchitektur sind Bottom-Up-Netzwerke, die wegen des uni-direkionalen, vom Systemeingang zum Systemausgang gerichteten Signalflusses auch als *feed-forward-Netzwerke* bezeichnet werden. In vielen Modellen wird eine vollständige Verbindungsstruktur zwischen Verarbeitungseinheiten angrenzender Schichten angenommen, d. h. jede Verarbeitungseinheit einer Schicht ist mit allen Verarbeitungseinheiten der darunter liegenden bzw. darüber liegenden Schicht verbunden. Ebenfalls häufig sind Modelle mit symmetrischen Verbindungen, wozu beispielsweise die thermodynamischen Modelle gehören. Wesentlich für die Arbeitsweise vieler Netzwerkmodelle neben dieser topologischen Struktur sind die Wahl der Aktivierungsform und die Änderung der Verbindungsgewichtung als Ergebnis eines Lernprozesses.

Eine Verknüpfung zwischen zwei Neuronen  $i$  und  $j$  wird durch ein Verbindungsge wicht  $w_{ij} \neq 0$  angezeigt. Die Definition eines Netzes kann damit über die Festlegung der gerichteten Verbindungen und deren Verbindungsge wichte erfolgen. Insofern stellen diese Verbindungsge wichte die wichtigste Grundlage aller Netzwerktypen dar, aus der sich die wesentlichen Informationen über ein bestimmtes Netzwerk ableiten lassen. Durch die Menge der Verbindungen bzw. die unterschiedlichen Verknüpfungen von Neuronen untereinander, entstehen verschiedene *intrinsische topologische Architekturen*, wobei für jede dieser Architekturen das singuläre Neuron als Ausgangspunkt dient.

So besteht das einfachste neuronale Netz aus einem einzigen Neuron. Eine derart simple neuronale Einheit bezeichnet man als Linear Treshold Unit (LTU) oder als adaptives lineares Element (adaptive lineare element, ADALINE). Trotz dieser Einfachheit lassen sich mit Hilfe solcher Neuronen einige basale Funktionen realisieren. Um beispielsweise die logische UND-Funktion zu realisieren, genügt ein solches Neuron, das zwei Eingabewerte entgegennehmen kann. Dann genügt die allgemeine Summenfunktion als eine Schwellenwertfunktion als Aktivierungsfunktion und die Identitätsfunktion als Ausgabefunktion. Mit den angegebenen Gewichten  $w_1 = 1$  und  $w_2 = 1$  und dem Schwellenwert  $\Phi = 1.5$  lässt sich das gewünschte Verhalten erzielen.

Die einzelnen Verbindungsge wichte lassen sich in einer Matrix darstellen. Zum Vergleich der einzelnen Architekturen lässt sich neben dem Matrix-Muster der Verbindungen auch die Darstellung des Netzes als gerichteter Graph angeben.

Als gerichteter Graph bezeichnet man in der Graphentheorie einen Graphen, dessen Kantenmenge eine zweistellige Relation über die Knoten ist. Gerichtete Graphen können dabei von azyklischer oder zyklischer Natur sein. Azyklische Graphen wiederum kann man topologisch

sortieren, sie können zusammenhängend oder unzusammenhängend sein. Darüber hinaus können sie endlich oder aber unendlich viele Knoten besitzen.<sup>72</sup>

Neben dieser intrinsischen Topologie entscheidet die *extrinsische Topologie* darüber, welche Neuronen auf die externe Umwelt direkt reagieren (Eingabeschicht), welche nur indirekt über andere Neuronen (Zwischenschichten) und welche Neuronen das Ergebnis (Output) nach außen repräsentieren bzw. weitergeben (Ausgabeschicht). Insofern hat jedes Netzwerk aus Neuronen einen Ein- und einen Ausgang. Die Eingabeneuronen (input units) in einem künstlichen neuronalen Netz entsprechen den Neuronen des biologischen Vorbildes und bilden gemeinsam die Eingabeschicht (input layer). Die Ausgabeneuronen (output units) sind die Neuronen, die die verarbeiteten Informationen wieder aus dem Netzwerk ausgeben. Sie bilden zusammengenommen die Ausgabeschicht eines Netzwerkes (output layer). Sie entsprechen den Neuronen im biologischen Vorbild, die beispielsweise ihre Impulse an die Muskeln weitergeben, die diese Impulse dann wiederum in motorische Impulse umwandeln. Die Schichten von Neuronen, die zwischen der Eingabe- und der Ausgabeschicht liegen, werden *verdeckte Schichten* (hidden layers) genannt. Sie tragen diesen Namen, weil nach einer Eingabe in das Netz die Ausgabe erfolgt, ohne dass der Betrachter immer weiß, wie viele Neuronenschichten an der Informationsverarbeitung beteiligt und wie sie untereinander verschaltet sind.

Neuronale Netze lassen sich auch bezüglich der Richtung der Aktivierungsprozesse entsprechend der Gliederung in Schichten unterscheiden. Man spricht dabei von einem *Feedforward-Netz*, wenn ein solches Netz einen externen Input enthält, diesen verarbeitet und damit eine Aktivierung in der Ausgabeschicht bedingt. Dieser Aktivierungswert wird gegebenenfalls als neuer Input an die Eingabeschicht eingereicht, was einen weiteren Verarbeitungsprozess in Richtung Ausgabeschicht indiziert. Dieser Prozess wird solange fortgesetzt, bis eine Lösung gefunden ist.<sup>73</sup>

Beim *Feedback-Netz* senden hingegen die Neurone der Ausgabeschicht ihre Aktivierungen über die Neuronen der Zwischenschichten zur Eingabeschicht, die daraufhin mit einer einschlägigen Veränderung ihrer Zustandswerte reagieren, um die Aktivierungen über die Neurone der Zwischenschichten an die Ausgabeschicht zu leiten. Insofern handelt es sich hier um einen speziellen Fall einer „Rückkopplung“, nämlich um die Umkehrung der Aktivierungsausbreitung.

Ein *vorwärts verkettetes Netz* (Feedforward-Netz) ohne eine solche Rückkopplung, das auch als mehrstufiges *Perzeptron* bezeichnet wird, zeigt folgendes Bild (Abb. 5.46):

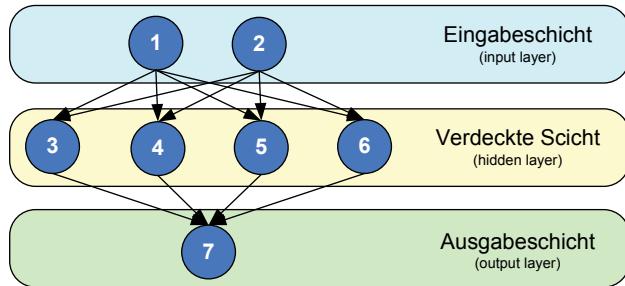
Das Bild zeigt auch einige wichtige Strukturelemente eines solchen Netzes:

- Als *Eingabe-Neuron* (input neuron) wird ein Neuron  $i$  bezeichnet, falls keine gerichtete Verbindung  $w_{ij}$  zu diesem Neuron existiert. Die Eingabeschicht (input layer) ist die Menge aller Eingabe-Neuronen eines Netzes.

<sup>72</sup> Siehe auch Clark und Holton 1994.

<sup>73</sup> Vgl. Kinnebrock 1992.

**Abb. 5.46** Feedforwarded Netz



- Die *Ausgabe-Neuronen* (output neuron) stellen mit ihren Ausgabewerten das Ergebnis der Verarbeitung dar. In vorwärts verketteten Netzen ist für die Ausgabeneuronen charakteristisch, dass sie keine weiterführenden Verbindungen besitzen. Die einzelnen Ausgabe-Neuronen werden in der Ausgabeschicht (output layer) zusammengefasst.
- Solche Neuronen, die weder zur Eingabe- noch zur Ausgabeschicht gehören, werden als *verdeckte Neuronen* (hidden neurons) bezeichnet, die dann wiederum auf mehrere verdeckte Schichten (hidden layer) verteilt sein können (Abb. 5.47).

Die Neuronen dieses Netzwerkes sind schichtweise miteinander verknüpft, wobei jedes Neuron einer Schicht mit allen Neuronen der nächsthöheren Schicht verbunden ist. Im obigen Beispiel gibt es eine verdeckte Schicht. Zusammen mit der Eingabe- und der Ausgabeschicht bilden sie drei Zellschichten, die durch insgesamt zwei Schichten von Verbindungen voneinander getrennt sind. Diese Anzahl gibt dem Netz den Namen 2-stufiges Netz. Ein n-stufiges feedforward-Netz besteht somit aus  $n + 1$  Schichten, von denen  $n - 1$  Schichten hidden layers sind. Systemtechnisch betrachtet werden bei einem solchen Feed-forward-Netz nur die Gewichtswerte der Verbindungen von „oben nach unten“ berücksichtigt.

Neben diesen Feedforward-Netzen, die schichtweise verbunden sind, gibt es noch solche, die sogenannte *Kurzschluss-Verbindungen* (shortcut connections) haben. Diese Verbindungen überspringen eine oder mehrere Schichten des Netzwerkes (Abb. 5.48).

Ein *Feedback-Netz* zeichnet sich dadurch aus, dass Feedbackschleifen von der Ausgabe zur inneren Schicht über spezielle Neuronen vorgesehen sind. Diese Netze mit Rückkopplung lassen sich nach Art ihrer Rückkopplung unterteilen.

Netze mit direkter Rückkopplung (direct feedback) geben ihr Ausgangssignal als Eingangssignal an sich selbst weiter. Dadurch wird je nach Gewicht seine Aktivierung verstärkt oder geschwächt (Abb. 5.49).

Bei Netzen mit indirekter Kopplung (indirect feedback) besteht die Rückkopplung zwischen Neuronen einer Schicht mit Neuronen untergeordneter Schichten (Abb. 5.50).

Bei Netzen mit Rückkopplung innerhalb einer Schicht (lateral feedback) wirkt die Ausgabe eines Neurons einer Schicht als Eingabe eines Neurons der gleichen Schicht. Eines der bekanntesten lateralfeedback-Netzwerke ist das Winner-takes-all-Netzwerk. Bei diesem Netzwerk gibt es zusätzlich noch direkte Rückkopplungen. Das Neuron einer Schicht, das die größte Aktivierung hat (der Gewinner), hemmt dann über die laterale Rückkopplung

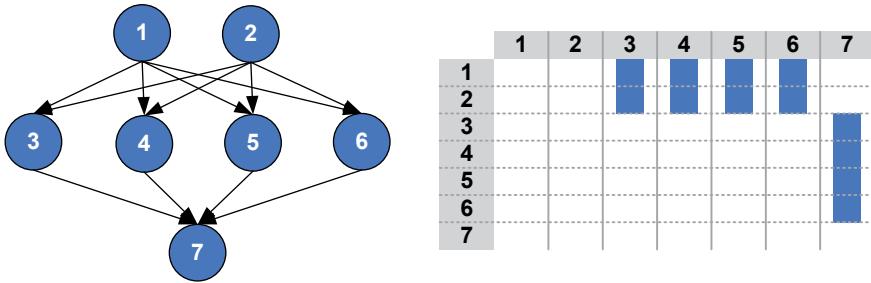


Abb. 5.47 Feedforwarded Netz und Verbindungsmarix

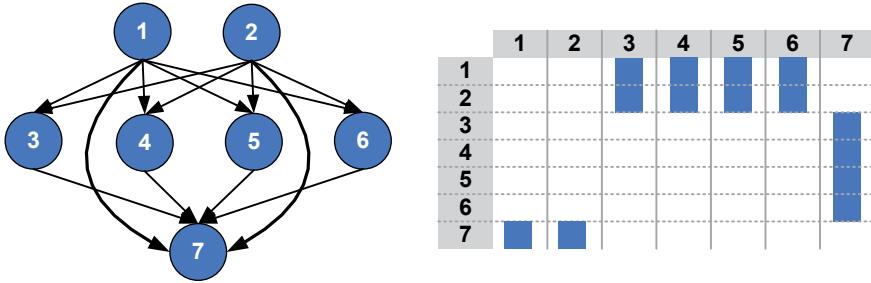


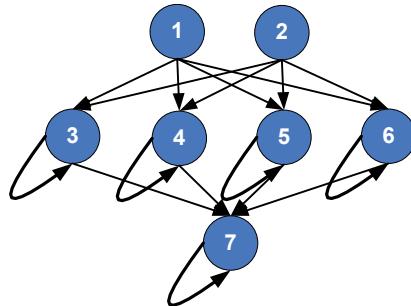
Abb. 5.48 Netz mit Kurzschluss-Verbindungen

die anderen Neuronen in seiner Schicht und verstrtt ber die direkte Rckkopplung die eigene Aktivierung (Abb. 5.51).

Bei *vollstndig verbundenen Netzen* gehen die Neuronen Verbindungen mit allen anderen Neuronen ein, unabhangig davon, in welcher Schicht sie sich befinden (Abb. 5.52).

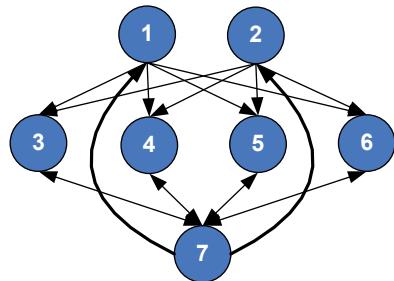
In einem knstlichen Netz aus Neuronen hat die Reihenfolge, in der die Neuronen ihre Werte berechnen, unter Umstnden Einfluss auf die Ergebnisse der jeweiligen Netze. Man unterscheidet dabei *synchrone* und *asynchrone* Aktivierung. Bei der *synchrone Aktivierung* werden die Aktivierungszustnde aller Neuronen des Netzes zur gleichen Zeit berechnet. In einem weiteren Schritt werden ebenfalls gleichzeitig ihre Ausgaben berechnet. Diese Form der Aktivierung ist bei rekurrenten Netzen von Vorteil, da die Berechnung der Aktivierungszustnde einzelner Neuronen nachvollziehbar bleibt und kein Datenchaos entsteht. Bei feedforward-Netzen ist die *synchrone Aktivierung* allerdings nachteilig, da sie im Vergleich zu anderen Aktivierungsarten langsam arbeitet. Bei der *asynchronen Aktivierung* werden die Werte einzelner Zellen zu unterschiedlichen Zeitpunkten berechnet. Damit ist die *asynchrone Aktivierung* dem biologischen Vorbild nher als die *synchrone*. Je nach Reihenfolge, in der die Neuronen angesprochen werden, lassen sie sich in vier Modi aufteilen:

- *Feste Ordnung* (fixed order): Bei einer festen Ordnung errechnen die Zellen in einer vorher festgelegten Reihenfolge ihre Werte. Jede Zelle wird dabei einmal angesprochen.



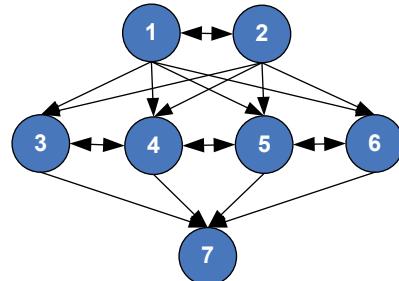
	1	2	3	4	5	6	7
1	1						
2	2						
3		3					
4			4				
5				5			
6					6		
7						7	

Abb. 5.49 Netz mit direkter Rückkopplung



	1	2	3	4	5	6	7
1	1						
2	2						
3		3					
4			4				
5				5			
6					6		
7						7	

Abb. 5.50 Netz mit indirekter Rückkopplung

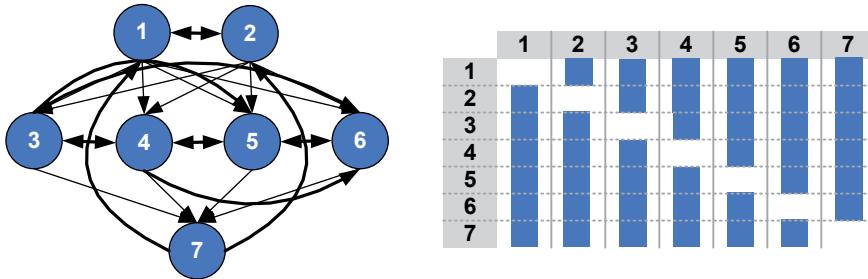


	1	2	3	4	5	6	7
1	1						
2		2					
3			3				
4				4			
5					5		
6						6	
7							7

Abb. 5.51 Netz mit lateraler Rückkopplung

Entspricht die Reihenfolge der Netzwerktopologie, ist dieses Verfahren für feedforward-Netze das schnellste Verfahren.

- *Zufällige Ordnung* (random order): Die Reihenfolge der angesprochenen Neuronen unterliegt bei der zufälligen Ordnung dem Zufall. Es errechnet dann sowohl seinen Aktivierungszustand, wie auch seine Ausgabe. Ein Lebenszyklus für ein Netz mit n Neuronen ist dann beendet, wenn n Neuronen angesprochen wurden. Dabei kann es sein,



**Abb. 5.52** Vollständig verbundenes Netz ohne direkte Rückkopplung

dass eine Zelle mehrfach, andere gar nicht angesprochen werden. Sie findet daher nur selten Verwendung bei der Modellierung künstlicher neuronaler Netze.

- *Zufällige Permutation* (random permutation): Dieser Modus funktioniert ähnlich wie der Zufallsmodus. Die Reihenfolge der angesprochenen Neuronen ändert sich jedoch von Zyklus zu Zyklus. Außerdem wird sichergestellt, dass jedes Neuron genau einmal angesprochen wird. Die Berechnung der zufälligen Permutation ist allerdings zeitaufwendig.
- *Topologische Ordnung* (topological order): Diese Ordnung ist bestimmt durch die Topologie eines Netzes. Das bedeutet, dass zuerst alle Werte der Neuronen, die in der Eingabeschicht vorkommen, berechnet werden, dann die der versteckten Schichten. Der Zyklus ist abgeschlossen, nachdem zuletzt die Werte der Zellen der Ausgabeschicht berechnet wurden. Dieser Modus ist für feedforward-Netze besonders günstig.

Neuronale Netze zeichnen sich dadurch aus, dass sie lernfähig sind. Gemäß einer Klassifizierung nach der Lernstrategie lassen sich auf einen ersten Blick Unterscheidungen zwischen einem überwachten, – nicht-überwachten und einem bestärkenden Lernen treffen (Abb. 5.53).

Zum Lernen bedarf es unter Umständen einer Kontrollinstanz, die die Netzausgabe mit der gewünschten Ausgabe vergleicht. Je nachdem, wo diese Kontrollinstanz in den Ablauf der Berechnung eingreift, lässt sich das Lernen klassifizieren. Es gibt drei verschiedene Arten des systemischen Lernens durch neuronale Netze.

- *Überwachtes Lernen* (supervised learning): Beim überwachten Lernen wird dem Netz neben dem Eingabemuster auch die erwünschte Ausgabe angegeben. Das Netz vergleicht sie mit der errechneten Ausgabe und modifiziert die verwendeten Rechenkomponenten (meist die Verbindungsgewichte), um die Abweichung zu verringern. Ziel beim überwachten Lernen ist, dass dem Netz nach mehreren Rechengängen mit unterschiedlichen Ein- und Ausgaben die Fähigkeit antrainiert wird, Assoziationen herzustellen. Das bedeutet, dass die Abweichungen erstmaliger Präsentationen neuer Muster von den gewünschten Ausgabemustern immer geringer werden. Diese Art des Lernens ist die schnellste.



**Abb. 5.53** Lernmethoden neuronaler Netze

- *Bestärkendes Lernen* (reinforcement learning): Beim bestärkenden Lernen gibt die Kontrollinstanz nach einer Ausgabe vom Netz die Information, ob die Ausgabe falsch oder richtig war, eventuell auch noch den Grad der Richtigkeit. Liegt die gewünschte Ausgabe vor, ist diese Art des Lernens langsamer als überwachtes Lernen. Sie ist aber ihrem Original in der Biologie wesentlich näher, da das motivierte Lernen (durch Belohnung oder Bestrafung) die häufigste Lernform beim Menschen ist.
- *Unüberwachtes Lernen* (unsupervised learning): Das unüberwachte Lernen verzichtet vollständig auf den Eingriff einer netzexternen Kontrollinstanz. Das bedeutet, dass dem Netz weder ein gewünschtes Ausgabemuster vorliegt, noch ihm mitgeteilt wird, ob sein errechnetes Ergebnis korrekt ist oder nicht. Das Netz erstellt selbstständig Klassifikatoren, nach denen es die Eingabemuster einteilt. Diese Methode ist die langsamste der bisher vorgestellten. Allerdings ist sie dem menschlichen Lernen am ähnlichsten, da das autodidaktische Lernen über dem didaktischen steht.

Es gibt sehr viele Möglichkeiten, einzelne Komponenten des Netzes nach einem Lernschritt so zu modifizieren, dass es dem gewünschten Ergebnis am nächsten kommt. Erlernen ist möglich durch eine oder mehrere der folgenden Möglichkeiten:

- Entwicklung neuer Verbindungen,
- Entwicklung neuer Zellen,

- Löschen existierender Verbindungen,
- Löschen existierender Zellen,
- Modifikation der Gewichtsmatrix,
- Modifikation der Schwellenwerte
- und/oder Modifikation der Rechenfunktionen (Aktivierungs-, Propagierungs- oder Ausgabefunktion).

Diese Methoden sind nicht vollkommen voneinander getrennt. So erreicht man mit Hilfe der Modifikation der Verbindungsgewichte, dass Verbindungen gelöscht (nicht angesprochen) oder dem Netz hinzugefügt werden (das entsprechende Verbindungsgewicht wird von Null auf einen Wert erhöht oder erniedrigt). Einige Modelle künstlicher neuronaler Netze eignen sich jedoch nicht für diese Methode, da ihre Gewichtsmatrix nicht für alle Neuronen Verbindungen vorsieht und die entsprechenden Werte nicht mit in die Gewichtsmatrix aufnehmen, anstatt sie in ihr zu belassen und auf null zu setzen. Das Hinzufügen oder Löschen von Zellen tritt bei neueren Modellen häufiger auf. Diese Methode ist jedoch nur bedingt plausibel, da das Wachstum biologischer Neuronen bereits im Kindesalter nahezu abgeschlossen ist, das Zellsterben aber zu jeder Zeit stattfinden kann. Diese Modelle bieten jedoch den Vorteil, dass sie durch das Absterben selten genutzter Zellen und die Entwicklung neuer benötigter Zellen ihre Netzwerk-Topologie selbst zum Optimum weiterentwickeln. Wenig verbreitet ist die Modifikation der Rechenfunktionen. Die meisten Lernmethoden verwenden die Modifikationsform der Gewichtsmatrix, damit das Netz lernen kann. Gewöhnlich bestehen Lernregeln jedoch darin, dass sie die Gewichtsmatrix in Abhängigkeit vom Lernerfolg modifizieren, bis der gewünschte Erfolg (Problemlösung, Optimum, Attraktor im Lösungsraum etc.) erreicht ist.

Die *Hebb'sche Lernregel* basiert auf dem Prinzip, Verbindungen von Neuronen untereinander zu verstärken, wenn sie zur gleichen Zeit stark aktiviert sind. Sie wurde 1949 von Donald O. Hebb formuliert und dient häufig als Grundlage komplizierterer Lernregeln. Die mathematische Formel der Hebb'schen Regel lautet wie folgt:

$$\Delta w_{ij} = \eta O_i A_j$$

Diese Formel bringt mathematisch zum Ausdruck, dass eine Verbindung zwischen zwei Neuronen immer dann gestärkt bzw. geschwächt wird, wenn beide Neurone annähernd zeitgleich aktiv respektive inaktiv sind. Aus physiologischer Perspektive ändern sich die synaptische Eigenschaften (exzitorisch oder inhibitorisch) proportional zur Summe der prä- und postsynaptischen Aktivitäten der beteiligten Neuronen, d. h. gleichzeitig feuernde Neuronen stärken ihre synaptische Verbindung und hemmen gleichzeitig die Aktivitäten der nicht-feuernden Neuronen. Aus Sicht der Lerntheorie beschreibt die Formel der synaptischen Verstärkung bzw. Abschwächung sogar ein universelles Muster, indem nämlich Lernen durch Wiederholung und Wissen durch Bestätigung von sich bewährenden Mustern erfolgt.

Wobei  $\Delta w_{ij}$  die Änderung des Gewichtes zwischen dem vorgeschalteten Neuron  $i$  und dem Neuron  $j$ , auf das jenes einwirkt, darstellt. Die Ausgabe der vorgeschalteten Zelle

$i$  wird mit  $O_i$ , die Aktivierung der darauf folgenden Zelle  $j$  mit  $A_j$  bezeichnet.  $\eta$  ist die sogenannte Lernrate. Unter der Lernrate als numerischer Wert versteht man den Faktor, der die individuellen Lernprozesse je nach Fähigkeit oder Begabung steuert. Die Hebb'sche Lernregel wird häufig im Zusammenhang mit binären Aktivierungswerten verwendet. Sind diese Werte 1 und 0, ist zu beachten, dass eine Verbindung zwar gestärkt, aber nicht geschwächt, sondern nur inaktiv gesetzt werden kann. Aus diesem Grund werden als Binärwerte bevorzugt 1 und  $-1$  verwendet. Da die Verbindungsgewichte die Synapsenart (hemmend bei negativen oder erregend bei positiven Werten) darstellen, ist es jedoch nicht plausibel, das Gewicht vom ursprünglichen positiven Wert auf einen negativen Wert herabzusetzen, da aus einem hemmenden Neuron kein erregendes Neuron werden kann.

Bei der allgemeinen mathematischen Form der Hebb'schen Regel, spielt noch ein weiterer Parameter eine Rolle: das teaching input  $t_j$ , das die erwartete Aktivierung darstellt. Die allgemeine Form lautet:

$$\Delta w_{ij} = \eta h(O_i, w_{ij}) g(A_j, t_j)$$

Hierbei werden zwei Funktionen  $h$  und  $g$  als weitere Faktoren neben der Lernrate  $\eta$  verwendet, von denen eine vom teaching input  $t_j$  abhängig ist.

Die *Delta-Regel* gehört zu den Derivaten der Hebb'schen Lernregel. Auch bei dieser Lernregel wird das Verbindungsgewicht verändert. Der Grad der Veränderung steigt dabei proportional zur Differenz der aktuellen Aktivierung und der erwarteten Aktivierung eines Neurons. Die Delta-Regel tritt in zwei Formen auf:

$$\Delta w_{ij} = \eta O_i(t_j - A_j) = \eta O_i \delta_j$$

und

$$\Delta w_{ij} = \eta O_i(t_j - O_j) = \eta O_i \delta_j$$

Bei beiden Formen entspricht das teaching input  $t_j$  der erwarteten Ausgabe. Die Delta-Regel ist ein Spezialfall, denn sie wird nur bei linearen Aktivierungsfunktionen von Netzen mit nur einer Schicht trainierbarer Gewichte verwendet.

Die *Backpropagation-Regel* ist der Delta-Regel sehr ähnlich. Sie wird bei Netzen mit mehr als einer Schicht trainierbarer Gewichte und für Neuronen mit nichtlinearer Aktivierungsfunktion verwendet. Die Aktivierungsfunktion muss bei dieser Lernregel semilinear sein. Das bedeutet, dass sie monoton und differenzierbar sein muss. Die Backpropagation-Regel lautet:

$$\Delta w_{ij} = \eta O_i \delta_j$$

Soweit ist sie analog zur Delta-Regel. Der Unterschied zu ihr liegt in der Berechnung von  $\delta_j$ . Falls  $j$  eine Ausgabezelle ist, berechnet sich  $\delta_j$  aus

$$\delta_j = f'(\text{Net}_j)(t_j - O_j)$$

Falls  $j$  eine verdeckte Zelle betrifft, berechnet sich  $\delta_j$  hingegen aus

$$\delta_j = f'(\text{Net}_j) \sum_k (\delta_k \Delta w_{jk})$$

Dabei ist  $f'$  die Aktivierungsfunktion,  $\text{Net}$  die Propagierungsfunktion und  $t$  die erwartete Ausgabe. Bei der zweiten Fallunterscheidung ist der Summationsindex  $k$  zu beachten. Das Verbindungsgewicht  $w_{jk}$  zeigt an, dass damit alle nachfolgenden Zellen gemeint sind.

Die bisher eingeführten generischen Modelle konnektionistischer Systeme beinhalten eine Vielzahl von Parametern und lässt somit eine große Variabilität innerhalb der Klasse konnektionistischer Modelle zu.

Diese Variabilität führt dazu, dass vor der Entwicklung eines künstlichen neuronalen Netz vor allem drei grundlegende Fragen geklärt werden müssen. *Was soll das neuronale Netz lernen?* Damit einher geht die Frage, welche spezifische Aufgabe das neuronale Netz erfüllen soll (beispielsweise Mustervervollständigung, Mustererkennung etc.). Die Antwort auf diese Frage bestimmt den Netztypus. Wie soll es lernen? Dies betrifft die Frage nach dem adaptiven Verfahren, mit der die gestellte Aufgabe bewältigt werden soll (beispielsweise Fehlerminimierung, etc.). Die Antwort auf diese Frage betrifft die Auswahl der spezifischen Lernalgorithmen. *In welchem Rahmen soll es lernen?* Diese Frage bezieht sich auf die Netzwerk-Architektur (Eingabe- und Ausgabeschichten, verdeckte Einheiten). Die Antwort liefert einen ersten Eindruck auf die durch die Aufgabenspezifikation erforderliche Komplexität des spezifischen Netzwerktypus.

Anhand verschiedener Einschränkungen dieser Parameter, insbesondere der Ausgabe-, der Propagierungs- und der Aktivierungsfunktion, können unterschiedlich komplexe Modellklassen gebildet werden. Eine weitere Unterscheidung kann anhand des Verarbeitungsmodus getroffen werden, in dem die Modelle eingesetzt werden. Die meisten Modelle werden als feed-forward oder interaktiv arbeitende Netzwerke verwendet, wobei die Grundidee hierbei ist, dass eine externe Eingabe durch Vorwärtspropagierung im Netzwerk verarbeitet wird und abschließend eine externe Ausgabe erzeugt. In anderen Modellen hingegen, insbesondere den thermodynamischen Modellen, besteht die Verarbeitung einer Eingabe darin, dass das Netzwerk sich ausgehend von einem Initialzustand, der durch eine externe Eingabe bestimmt wird, in einen Gleichgewichtszustand einschwingt. Aus diesem Endzustand kann dann die externe Ausgabe abgelesen werden. Solche Systeme werden auch als *Relaxationsnetzwerke* bezeichnet. Außerdem kann zwischen *synchron* arbeitenden Netzwerken, bei denen alle Elemente gleichzeitig ihren Zustand ändern und ggf. eine Ausgabe erzeugen, und *asynchronen* unterschieden werden. In asynchron arbeitenden Netzen wird

der Zeitpunkt der Zustandsänderung durch eine probabilistische Funktion bestimmt oder (pseudo-) zufällig festgelegt. Die Elemente selbst arbeiten in den meisten Modellen deterministisch, manchmal auch probabilistisch. Weiterhin kann man unterscheiden zwischen Netzwerken, die *vollständig* oder *zufällig verbunden* sind, deren Elemente in *Schichten* angeordnet sind, wobei Verbindungen nur in eine Richtung erlaubt sein können (*bottum-up*, *top-down*) oder in beide (*interaktiv*), und zwischen *symmetrischen* Netzen.

Dies signalisiert, dass man neuronale Netze durchaus als struktur determinierte Systeme auffassen kann, da natürlich die Komplexität und Größe der Netzwerkarchitektur eine ausschlaggebende Rolle für die Problemlösung durch ein so konzipiertes System darstellt. Auch das Lernen zeigt sich unter dieser Perspektive als Strukturveränderung in einem dynamischen System, ohne die generelle Rahmensetzung (Architektur) des Systems überschreiten zu können.

Um eine ausreichende Funktionsfähigkeit der neuronalen Netze zu erreichen, ist die Auswahl des Netzmodells und dessen Konfiguration von grundlegender Bedeutung. Insbesondere sind hier mehrere Schritte zu durchlaufen:

- Eingabe und Ausgabeparameter definieren,
- Netzarchitektur festlegen,
- einen Lernalgorithmus auswählen und die verschiedenen Parameter geeignet setzen,
- Trainingsdaten auswählen und aufbereiten
- und Netz trainieren.

Hier sind eine ganze Reihe von Entscheidungen zu treffen, die jeweils einen starken Einfluss auf die Leistungsfähigkeit des resultierenden Netzes haben können. Die normale Methode besteht zu einem großen Teil darin, in einer Reihe von „Trial and Error“ Schritten zu immer besseren Ergebnissen zu gelangen. Diese Vorgehensweise wird durch heuristische, aus der Erfahrung gewonnene Regeln unterstützt, eine verlässliche theoretische Basis ist jedoch nicht vorhanden. Anstelle einer konkreten Handlungsempfehlung lässt sich jedoch in der folgenden Klassifizierung aufzeigen, dass sich der Einsatz bestimmter Modelle auf bestimmte Problem- und Anwendungsbereiche in der Praxis bewährt haben (Tab. 5.8).

In der Praxis kommen derzeit mehr als 30 verschiedene Typen artifizieller neuronaler Netze zum Einsatz, unter anderem: *ADALINE* (Adaptive Linear Neural Element), *ART* (Adaptive Resonant Theory), *AM* (Associative Memories, Assoziativspeicher), *BAM* (Bidirectional Associative Memory, Bidirektionaler Assoziativspeicher), *Boltzmann-Maschine*, *BSB* (Brain-State-in-a-Box), *CCN* (Cascade Correlation), *Cauchy-Maschine*, *CPN* (Counter Propagation), *GRNN* (Generalized Regression Neural Network), *Hamming*, *Hopfield*, *LVQ* (Learning Vector Quantization, Lernvektor-Quantisierung), *MADALINE*, *MLPT* mit *BP* (Mehrschichtige Feedforward Backpropagation), *Neokognitron*, *NLN* (Neurologische

**Tab. 5.8** Übersicht über die Modelle neuronaler Netze

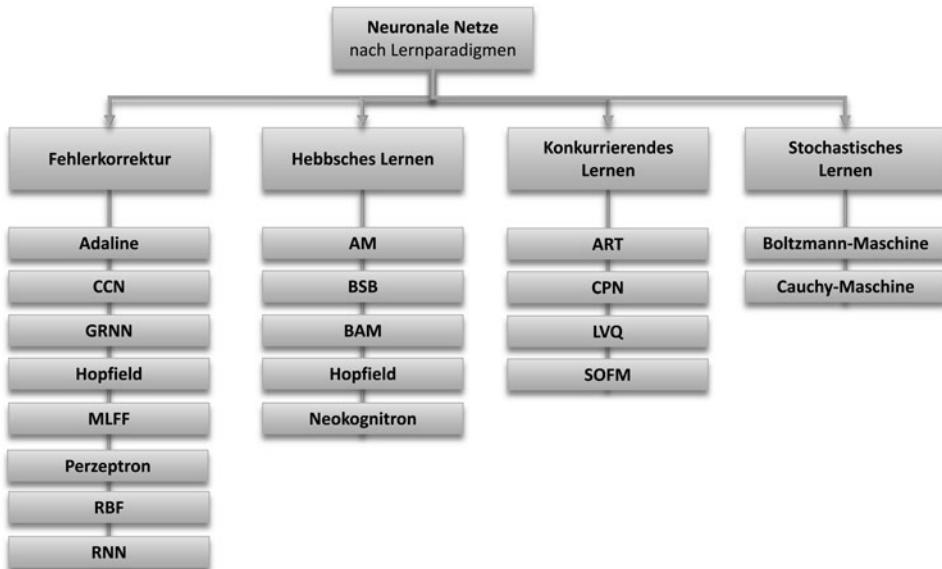
Klassifizierungsmerkmal		Geeignete Modelle
<i>Anwendungsgebiet</i>	Musterassoziation	Heteroassoziative Speicher
	Mustererkennung	Hopfield
	Optimierung	Hopfield, Selbstorganisierende Karten, Kohonen Karte
	Ordnung implizit vorgegebener Daten	SOM
<i>Lernverhalten</i>	Prognose	Boltzmann-Maschine
	Überwachtes Lernen	Assoziative Netze
	Nicht-Überwachtes Lernen	SOM
<i>Richtung des Informationsflusses</i>	Verstärkendes Lernen	
	Feed forward Netze	Assoziative Speicher, SOM
	Feed back Netze	Hopfield, BAM, Interaktive Netze
<i>Bedingtheit der Entscheidungen</i>	Deterministisch	Hopfield, Selbstorganisierende Karten, Kohonen Karte, SOM, Assoziative Netze,
	Stochastisch	Boltzmann-Maschine
Ackley et al. 1985		

Netzwerke), *Perceptron*<sup>74</sup>, PNN (Probabilistic Neural Network, Neuronales Netzwerk mit Wahrscheinlichkeitsverteilung), RBF (Radial Basis Function), RNN (Recurrent Neural Networks, Rekursive Neuronale Netze), RCE (Reduced Coulomb Energy), SOFM (Self-Organizing Feature Map). Insofern macht es Sinn, diese artifiziellen neuronalen Netztypen je nach Perspektive in verschiedene Taxonomien oder Klassifizierungen einzuteilen. Dabei werden die verschiedenen Perspektiven hinsichtlich der Lernparadigmen, der Netzwerkarchitektur und des allgemeinen Einsatzgebietes betrachtet (Abb. 5.54).

Aus der Vielzahl dieser existenter Netzmodelle sollen im Folgenden exemplarisch das Backpropagation-Netz, das Hopfield-Netz und das Kohonen-Netz beschrieben werden.

Das *Backpropagation-Netz* ist das am häufigsten eingesetzte Netzmodell. Es handelt sich dabei um ein mehrschichtiges Netz mit mindestens einer verborgenen Schicht und einer Feed-forward-Informationsausrichtung, bei der alle Neuronen einer Schicht vollständig mit den Neuronen der nächsten Schicht verbunden sind. Die Informationsverarbeitung in den Input-Neuronen erfolgt über die Summenfunktion (Eingangsfunktion) und eine lineare, unbegrenzte Aktivierungsfunktion. Die Ausgangsfunktion ist als Identitätsfunktion gestaltet, d. h., sie nimmt auf die Signale keinen weiteren Einfluss mehr. Ähnlich ist das Neuronen-Modell der Hidden-Schicht(en) und der Ausgabeschicht aufgebaut, d. h., bei diesen beiden Schichten dient die Summenfunktion wiederum als Eingangsfunktion sowie

<sup>74</sup> Vgl. Minsky und Papert 1969.



**Abb. 5.54** Neuronale Netze nach Lernparadigmen

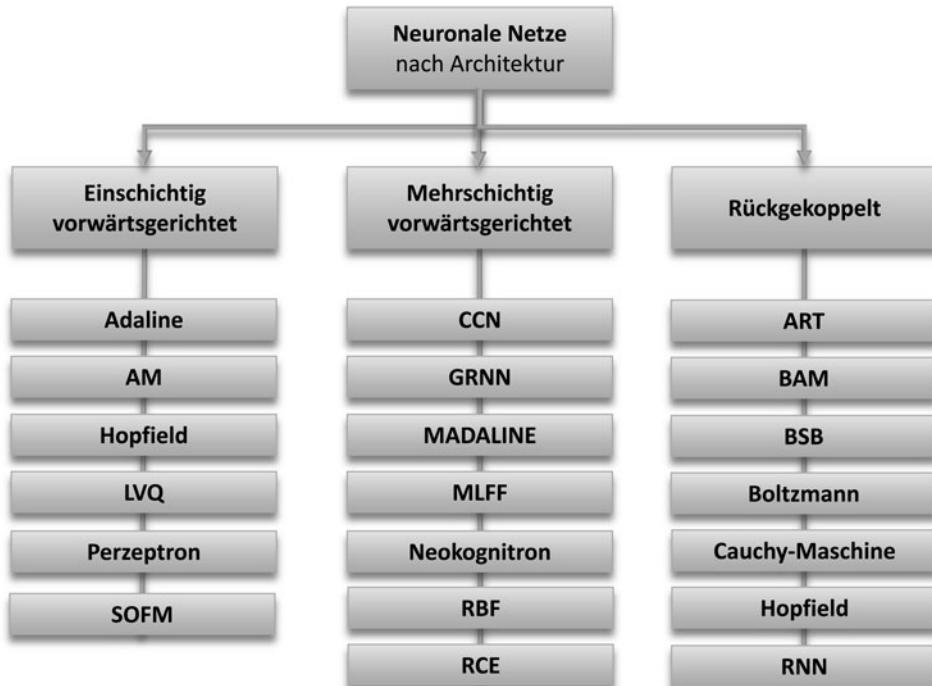
die Identitätsfunktion als Ausgangsfunktion. Lediglich die Aktivierungsfunktion wird durch eine Sigmoidfunktion gebildet. Beim Lernverfahren handelt es sich um überwachtes Lernen, d. h., es werden Trainingsdaten mit Angabe der gewünschten Ausgabe benötigt. Das wesentliche Kennzeichen des Backpropagation-Netzes bildet der spezielle Lernalgorithmus. Hierbei gilt es zu beachten, dass die Delta-Regel den Nachteil hat, dass sie lediglich für zweischichtige Netze angewandt werden kann, da die von ihr zur Gewichtsanpassung benötigte gewünschte Ausgabe nur für die Ausgabe-Neuronen vorliegt. Eine Verallgemeinerung dieser Regel für beliebig viele verborgene Schichten leistet die *generalisierte Delta-Regel* (generalized delta rule), auch Backpropagation-Algorithmus genannt. Die generalisierte Delta-Regel erweitert die herkömmliche Delta-Regel um die Berechnung der Fehlersignale verborgener Neuronen bei Feed-forward-Topologie. Dabei werden zunächst die Eingabedaten dem Netz zugeführt, um die Aktivierungszustände aller Neuronen und damit auch die Ausgabewerte des Netzes zu berechnen. In einem weiteren Schritt werden die Fehlerwerte der Ausgabeschicht und der verborgenen Schicht(en) jeweils schichtweise zurückgeführt und die entsprechenden Gewichtsänderungen berechnet. Diese Prozedur wird für alle Trainingsdatensätze wiederholt, so dass auf diese Weise der vom Netz produzierte Fehler sukzessive verringert wird. Bei einer derartigen Minimierung des Netzfehlers handelt es sich um einen Gradientenabstieg, was anschaulich einer Bewegung auf einem mehrdimensionalen Fehlergebirge hin zu dessen tiefsten Punkt entspricht.

Problematisch bei diesem Vorgehen ist der Umstand, dass der Lernvorgang relativ langsam abläuft und nicht sichergestellt ist, dass das globale Minimum des Fehlergebirges tat-

sächlich gefunden wird. Aus diesem Grund existiert eine Reihe von Modifikationen der generalisierten Delta-Regel, die auf eine Beschleunigung des Lernverfahrens abzielen und die Wahrscheinlichkeit erhöhen sollen, dass das globale Minimum gefunden wird. Am häufigsten wird hierbei der sogenannte *Momentumterm* eingeführt. Dabei fließen in die Berechnung der aktuellen Gewichtsänderung die Gewichtsänderungen der vorherigen Lernschritte ein, was dazu führt, dass die Bewegung auf dem Fehlergebirge „begradiert“ wird und lokale Minima überwunden werden können.

Das von Hopfield vorgestellte Modell stellt einen Sonderfall in der Klasse der überwacht lernenden neuronalen Netze dar. Es handelt sich um ein sogenanntes *autoassoziatives* Netz, bei dem Eingabe- und Ausgabemuster identisch sind, d. h., die Präsentation eines Eingabemusters führt nicht zur Ausgabe eines anderen assoziierten Musters, sondern des eingespeisten Musters selbst. Auf diese Art und Weise kann das neuronale Netz ein verfälschtes Eingabemuster korrigieren und ein Muster ausgeben, das dem ursprünglichen, unverfälschten Eingabemuster entspricht. Diese Eigenschaft, die beispielsweise bei der Spracherkennung genutzt werden kann, wird auch Mustervervollständigung genannt, d. h., das Netz dient als Speicher, der die vollständig abgespeicherte Information mit Hilfe einer Teilinformation, einer Art Schlüssel, wiederfindet. Während das Backpropagation-Netz keine Rückkopplungen zwischen Neuronen beinhaltet, wird die besondere Art der Informationsverarbeitung im Hopfield-Netz über eine vollständige symmetrische ( $w_{ij} = w_{ji}$ ) Rückkopplung aller Neuronen in Form ungerichteter Verbindungen erreicht, was zu einer ungeschichteten Architektur bzw. Topologie führt. Dabei ist zu beachten, dass das Hopfield-Netz über keine direkten Rückkopplungen verfügt, d. h. Rückkopplungen von Neuronen mit sich selbst. Die Informationsverarbeitung innerhalb der Neuronen erfolgt über ein einheitliches Neuronen-Modell, wobei die Eingangsfunktion als Summenfunktion und die Ausgangsfunktion als Identitätsfunktion gestaltet ist, die somit keinen weiteren Einfluss mehr auf die Signale nimmt. Als Aktivierungsfunktion kommt in den meisten Fällen die Treppenfunktion zum Einsatz, die binäre Eingabe- und Ausgabedaten bedingt. Theoretisch ist auch die Verwendung kontinuierlicher Aktivierungsfunktionen mit den entsprechenden kontinuierlichen Ein- und Ausgabedaten denkbar. Der Lernalgorithmus des Hopfield-Netzes ist bestrebt, die sogenannte Energiefunktion, die den Netzzustand beschreibt, zu minimieren. Damit sind Hopfield-Netze grundsätzlich auch für Optimierungsprobleme geeignet, sofern sich die zu optimierende Funktion als derartige Energiefunktion formulieren lässt (Abb. 5.55).

Im Gegensatz zu Backpropagation- und Hopfield-Netzen, bei denen überwachtes Lernen zum Einsatz kommt, verwendet das nach seinem Entwickler benannte *Kohonen-Netz* ein unüberwachtes Lernverfahren, dem beim Lernen ausschließlich Eingabemuster, also keine gewünschten Ausgabemuster, zur Verfügung stehen. Es erkennt selbständig Ähnlichkeiten, Häufigkeiten oder Regelmäßigkeiten in den Eingabemustern und wandelt sie in eine topologische Anordnung um. Das Kohonen-Netz, auch self-organizing map oder Modell selbstorganisierender Karten genannt, hat eine wesentlich stärkere biologische Orientierung als das Backpropagation- und das Hopfield-Netz. Es basiert auf der Erkenntnis neurologischer Forschung, dass die räumliche Anordnung der Neuronen im



**Abb. 5.55** Neuronale Netze nach Architektur

Gehirn oft in Relation zu bestimmten Merkmalen der Eingabemuster steht. Die Neuronen des Kohonen-Netzes sind in zwei Schichten angeordnet, einer Eingabeschicht und einer Kohonen-Schicht, die auch als Wettbewerbs- oder Kartenschicht bezeichnet wird. Alle Eingabe-Neuronen sind dabei über einen gerichteten Informationsfluss vollständig mit den Neuronen der Kohonen-Schicht verbunden, so dass eine Feed-forward-Architektur entsteht. Darüber hinaus bestehen in der Kohonen-Schicht laterale Verbindungen, die eine zweidimensionale Neuronen-Fläche, auch Karte genannt, aufspannen. Die den zwei Schichten zugrunde gelegten Neuronen-Modelle sind sehr verschiedenartig ausgelegt. Die Informationsverarbeitung in den Neuronen der Eingabeschicht erfolgt, wie beim Backpropagation-Netz, über die Summenfunktion als Eingangsfunktion, eine lineare, unbegrenzte Aktivierungsfunktion und die Identitätsfunktion als Ausgangsfunktion. Die Neuronen der Kohonen-Schicht hingegen verwenden als Eingangsfunktion eine Summenfunktion, die, neben den Ausgabewerten der Eingabe-Neuronen, auch die Ausgabewerte der lateral verbundenen Neuronen berücksichtigt. Als Aktivierungsfunktion kommt eine Sigmoidfunktion zum Einsatz. Die Ausgangsfunktion schließlich bezieht die Aktivitätspeglle aller anderen Neuronen in die Berechnung des Ausgabe-Signals ein, wobei nur das Neuron mit dem höchsten Aktivitätspeglle sein Signal weiterleiten darf, ein Vorgehen, das auch als Winner-takes-all-Prinzip oder Wettbewerbslernen bezeichnet wird. Entsprechend dem Kohonen-Lernalgorithmus werden zunächst alle Gewichte mit Zufallswerten initialisiert.

Das Neuron mit der größten Ähnlichkeit des Gewichtsvektors zum Eingabevektor erhält den höchsten Aktivitätspegel und hemmt die anderen Neuronen, d. h., ausschließlich dieses Gewinner-Neuron und dessen Nachbarn erhalten eine Gewichtsänderung. Die Definition des Umfangs der Nachbarschaft eines Neurons ist von großer Bedeutung für die Funktionsfähigkeit des Netzes, wobei es vorteilhaft ist, zu Beginn der Lernphase die Nachbarschaft weit zu fassen und mit zunehmendem Lernfortschritt zu verkleinern. Die Gewichtsänderung zieht, geometrisch betrachtet, den Gewichtsvektor in Richtung des Eingabevektors, so dass letztlich jedem Eingabemuster ein Punkt auf der durch die Kohonen-Schicht gebildeten Karte zugeordnet wird. Ähnliche Eingabemuster werden dabei auf den gleichen Ort abgebildet, so dass eine Gruppierung der Eingabemuster entsteht, die die in den Eingabemustern vorhandenen Ordnungsrelationen widerspiegelt. Damit bildet das Kohonen-Netz einen Ansatz, der für ein breites Spektrum von Klassifizierungsaufgaben geeignet erscheint. Darüber hinaus ist es grundsätzlich auch für Probleme geeignet, bei denen es um die Optimierung von „Nachbarschaften“ geht, wie beispielsweise beim Travelling-Salesman-Problem.

Neuronale Netze haben sich in der Praxis und dort nicht nur in der Robotik bewährt. In Bezug auf Anwendungsdomänen lassen sich die neuronalen Netze in fünf Anwendungsbereiche gruppieren: Assoziativspeicher, Klassifizierung, Mustererkennung, Vorhersage, und Optimierung.

Einige Netztypen vermögen zu lernen, als Speicher zu agieren und Muster aufzunehmen, die wiedergefunden werden, wenn man ihnen ein assoziiertes Muster präsentiert. Wenn das wiedergefundene und das gespeicherte Muster gleich sind, wird der Prozess als autoassoziatives Wiederfinden bezeichnet. Wenn sich die beiden Muster unterscheiden, spricht man von heteroassoziativem Wiederfinden. Neuronale Netze werden in den verschiedensten Bereichen der Diagnostik und Vorhersage eingesetzt: Medizin, Ingenieurswesen oder Fertigungswesen, um nur einige davon zu nennen. Dabei handelt es sich im Wesentlichen um ein Klassifizierungsproblem. Man benötigt eine konkrete Assoziation zwischen Eingabemustern, die irgendein Symptom oder ein nicht regelgerechtes Verhalten beschreiben, und der entsprechenden Krankheit oder einem Hardwarefehler oder einer anderen Art der Fehlfunktion. Auch die Vorhersage ist in vielen Bereichen gebräuchlich (Abb. 5.56).

Neuronale Netze haben sich als geeignete Werkzeuge für Vorhersagen erwiesen: Vorraussagen, dass etwas passiert oder nicht, wann ein Ereignis passiert oder mit welcher Wahrscheinlichkeit. Neuronale Netze werden auch für unterschiedliche Aufgabenstellungen eingesetzt, für die eine optimale oder fast optimale Lösung benötigt wird. Im Allgemeinen sind neuronale Netze gut geeignet, Wahrnehmungsaufgaben zu lösen, wie etwa die Erkennung komplexer Muster: Visuelle Bilder von Objekten, gedruckte oder handschriftliche Zeichen, Spracherkennung oder andere Arten der Mustererkennung.<sup>75</sup>

---

<sup>75</sup> Vgl. Haun 1998.



**Abb. 5.56** Neuronale Netze nach Anwendungsdomäne

## Literatur

- Ackley, D.H., Hinton, G., Sejnowski, T.: A learning algorithm for Boltzmann machines. *Cognitive Science* (1985)
- Alexandrescu, A.: *Modern C++ design: Generic programming and design patterns applied*. Addison-Wesley, Boston (2001)
- Barr, A., Feigenbaum, E.A.: *The handbook of artificial intelligence*. 3 Bde. London (1982)
- Baumgarten, B.: *Petri-Netze*, 2. Aufl. Spektrum Akad. Verlag, Heidelberg (1996)
- Booch, G.: *Object-oriented design with applications*. Benjamin/Cummings, Redwood City (1991)
- Brooks, R.: A robust layered control system for mobile robot. *IEEE. J. Robot. Autom.* (1986)
- Clark, D., Holton, A.: A first look at graph theory. World scientific publishers, New Jersey (1991) (deutsch: *Graphentheorie – Grundlagen und Anwendungen*; Spektrum Akad. Verlag, Heidelberg 1994)
- Clocksin, W.F., Mellish, C.S.: *Programming in Prolog*, 5. Aufl. Springer, New York (2003) (deutsch: *Programmieren in Prolog*. Springer, Berlin 1990)
- Corner, D.E., Stevens, W.R.: *Internetworking with TCP/IP (mehrbandig)*. Prentice Hall, Upper Saddle River (1999/2000)
- Coulouris, G.F., Dollimore, L., Kindberg, T.: *Distributed Systems*, 3. Aufl. Pearson Education, Harlow (2001) (deutsch: *Verteilte Systeme*. Pearson Studium, München 2002)
- Cliff DT, Husbands P, Harvey I: Evolving recurrent dynamical networks for robot control (1993).
- Date, C.J., Darwen, H.: *A Guide to the SQL Standard*. Addison-Wesley, Reading (Mass.), 4. Aufl. (1997) (deutsch: *SQL – Der Standard*. Addison-Wesley-Longman, Bonn 1998)
- Dobzhansky, T.: *Genetics and the origin of species*. Columbia University Press, New York (1937)

- Dörner, D.: Problemlösen als Informationsverarbeitung. Stuttgart (1976)
- Dreyfus, H.L.: Schöpfung des Geistes oder Modellierung des Gehirns. Künstliche Intelligenz am Scheideweg. Klagenfurt (1989)
- Eco, U.: Einführung in die Semiotik. Fink, München (1972)
- Elbow Room: The Varieties of Free Will Worth Wanting, 1984, Cambridge, MA: Bradford Books/MIT Press deutsch: Ellenbogenfreiheit. Die wünschenswerten Formen von freiem Willen, Beltz Athenäum, 2. Aufl. 1994
- Everett, H.R.: Sensors for mobile robots. A K Peters, Ltd., Addison-Wesley, Massachusetts (1995)
- Frank, E.: Künstliche Intelligenz. Eine grundlagentheoretische Diskussion der Einsatzmöglichkeiten und -grenzen. Tübingen (1991)
- Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading (1995). Deutsche Ausgabe: Entwurfsmuster: Elemente wiederverwendbarer objektorientierter Software. Addison-Wesley, Bonn (1996)
- Güsmann, B.: Einführung in die Roboterprogrammierung – Lehr- und Übungsbuch mit Trainingssoftware PRO-Tutor. Vieweg, Braunschweig Wiesbaden (1992)
- Gulbins, K.: UNIX System V. Begriffe, Konzepte, Kommandos, Schnittstellen, 4. Aufl. Springer, Berlin (1995)
- Haun, M.: Simulation Neuronaler Netze. expert, Renningen (1998)
- Haun, M.: Wissensbasierte Systeme. expert, Renningen (2000)
- Haun, M.: Einführung in die rechnerbasierte Simulation Artifiziellen Lebens. expert, Renningen (2004)
- Heinemann, B., Weihrauch, K.: Logik für Informatiker, 2. Aufl. Teubner, Stuttgart (1992)
- Herrmann, D.: Effektiv Programmieren in C und C++; vieweg, Braunschweig, Aufl. (2001)
- Husty, M., Karger, A., Sachs, H., Steinhilper, W.: Kinematik und Robotik. Springer, Berlin (1997)
- Irrgang B., Klawitter, J. (Hrsg.): Künstliche Intelligenz. Stuttgart (1990)
- Jensen, R.M., Veloso, M.: Interleaving deliberative and reactive planning in dynamic multi-agent domains. In: Proceedings of the AAAI-Fall Symposium on Integrated Planning for Autonomous Agent Architectures. AAAI Press (1998)
- Jungnickel, D.: Graphen, Netzwerke und Algorithmen, 3. Aufl. BI Wissenschaftsverlag, Mannheim (1994)
- Kahlbrandt, B.: Software-Engineering. Springer Verlag, Berlin (1998)
- Kernighan, B.W., Ritchie, D.M.: The C programming language, 2. Aufl. Prentice Hall, Englewood Cliffs (1988) (deutsch: Programmieren in C, 2. Aufl. Hanser, München 1990)
- Kinnebrock, W.: Neuronale Netze. Grundlagen, Anwendungen, Beispiele. München (1992)
- Koch, S.: JavaScript – Einführung, Programmierung und Referenz, 3. Aufl. dpunkt, Heidelberg (2001)
- Kovács, P.: Rechnergestützte symbolische Roboterkinematik. Vieweg, Braunschweig Wiesbaden (1993). Fortschritte der Robotik 18
- Kurose, J.F., Ross, K.W.: Computer Networking, 2. Aufl. Addison-Wesley, Boston 2003 (deutsch: Computernetze; Pearson Studium, München 2002)
- Kurzweil, R. (Hrsg.) Das Zeitalter der Künstlichen Intelligenz. München (1993)
- Krämer, S.: Geist – Gehirn – künstliche Intelligenz. de Gruyter, Berlin (1994)
- Leidlmaier, K.: Wozu Künstliche Intelligenz? Wien (1988)
- Lunze, J.: Künstliche Intelligenz für Ingenieure Bd. 2: Technische Anwendungen. R. Oldenbourg Verlag, München (1995)
- Mayer, O.: Programmieren in COMMON LISP, 2. Aufl. Spektrum Akademischer Verlag, Heidelberg (1995)
- McCorduck, P.: Denkmaschinen. Die Geschichte der künstlichen Intelligenz. München 1989
- McClelland, J.L., Rumelhart, D.E., Hinton, G.E.: The appeal of parallel distributed processing. In: Rumelhart, D.E., McClelland, J.A. (Hrsg.) Parallel distributed processing. MIT Press (1986)

- Meyer, B.: Object-oriented software construction. Prentice Hall, Englewood Cliffs 1988. Deutsche Ausgabe: Objektorientierte Softwareentwicklung. Hanser, München (1988)
- Mildenberger, O.: Informationstheorie und Codierung. Vieweg, Braunschweig (1992)
- Mitchie, D., Johnston, R.: Der kreative Computer. Künstliche Intelligenz und menschliches Wissen. Hamburg (1985)
- Minsky, M., Papert, S.: Perceptrons. MIT Press, Cambridge, Mass (1969)
- Misgeld, W.D.: SQL-Einstieg und Anwendung, 4. Aufl. Hanser, München (2001)
- Münch, Dieter (Hrsg.): Kognitionswissenschaft. Frankfurt a. M. (1992)
- Newell, A.: Unified theories of cognition. Harvard University Press, Cambridge, Mass (1990)
- Ottmann, T., Widmayer, P.: Algorithmen und Datenstrukturen. Spektrum Akademischer Verlag (2002).
- Partridge, D: Künstliche Intelligenz. Hamburg (1989)
- Poddig, T.: Künstliche Intelligenz und Entscheidungstheorie. Wiesbaden (1992)
- Rojas, P: Theorie der neuronale Netze. Eine systematische Einführung. Berlin (1992)
- Rose, F.: Ins Herz des Verstandes. Auf der Suche nach der Künstlichen Intelligenz. Reinbeck b. Hbg. (1986)
- Russel, S., Norvig, P.: Artificial Intelligence: A Modern Approach. Prentice Hall. New York (1995)
- Schefe, P.: Künstliche Intelligenz – Überblick und Grundlagen. Mannheim (1991)
- Schöning, U.: Logik für Informatiker. Spektrum Akad. Verlag, Heidelberg, Aufl. (2000)
- Schönthaler, F., Nemeth, T.: Software-Entwicklungsgerüste: Methodische Grundlagen. Teubner, Stuttgart (1990)
- Schrödinger, E.: What is life? The physical aspect of the living cell. Cambridge University Press, Cambridge. (Deutsche Ausgabe: Was ist Leben? Lehnen, München 1951)
- Schwinn, W.: Grundlagen der Roboterkinematik. Verlag Liborius Schwinn, Schmalbach (1992)
- Smolensky P (1988): On the proper treatment of connectionism. Behavioral and brain sciences 11(88): 1-74.
- Stoffer, T.: Perspektiven konnektionistischer Modelle: Das neuronale Netzwerk als Metapher im Rahmen der kognitionspsychologischen Modellbildung. In: Meinecke/Kehrer a. a. O
- Stroustrup, B.: The C++ programming language, 3. Aufl. Addison-Wesley, Reading (Mass.) (1997) (deutsch: Die C++ Programmiersprache, Addison-Wesley, München 2002)
- Strube, G. (Hrsg.): Wörterbuch der Kognitionswissenschaft. Klett-Cotta. Stuttgart (1996)
- Thompson, R.F.: Das Gehirn. Spektrum der Wissenschaft, Heidelberg, (1990)
- Turing, A.M.: Computing machinery and intelligence. Mind. 59, 433-460. (Deutsche Übersetzung Kann eine Maschine denken? In Zimmerli und Wolf 1994)
- Varela, F.: Kognitionswissenschaft – Kognitionstechnik. Eine Skizze aktueller Probleme. Frankfurt (1990)
- Wirth, N.: Grundlagen und Techniken des Compilerbaus. Addison-Wesley, Bonn (1996)

Durch die Validierung soll in den nächsten Abschnitten festgestellt werden, ob die bisher entwickelten Modellannahmen ausreichen, um ein kognitives Robotersysteme durch entsprechende Implementierung so zu realisieren, dass dieses System die geforderten „intellektuellen“ Leistungen erbringen kann. Dabei wird sich an einem Anwendungsfall aus der mobilen Robotik orientiert, da dort komplexe kognitive Fähigkeiten nicht nur erforderlich sind, sondern die Erbringung solcher Leistungen auch entsprechend empirisch und anschaulich untersucht werden können. So müssen solche mobilen Systeme sich nicht nur zuverlässig und sicher in natürlichen, sich verändernden Umgebungen zurechtfinden und bewegen, sondern auch mit den Objekten dieser Umgebungen interagieren, kooperieren und damit im Sinne dieses Buches intoperieren können. Im Verlauf dieses Kapitels wird daher ein *Autonomer Raum Explorer* (ARE) auf Basis eines kognitiven Robotersystems entwickelt, der sich autonom in einem unstrukturierten räumlichen Umfeld orientieren, fortbewegen und bestimmte Objekte durch Interoperation auffinden kann. Das Ziel des Robotersystems besteht also in der Erforschung eines unbekannten Raumes und das Finden definierter Gegenstände. Hierzu werden die bisher erarbeiteten Erkenntnisse mit denen aus den Bereichen der autonomen Navigation, Modellierung und Cognitive Computing verknüpft.

---

## 6.1 Problem

Das Problem soll mit Hilfe eines autonomen und mobilen Roboters gelöst werden. Dabei wird unter einem solchen autonomen, mobilen System eine Maschine verstanden, die sich in einer natürlichen Umgebung aus eigener Kraft und ohne Hilfestellung von außen bewegen und dabei ein ihr gestelltes Ziel erreichen und das ihr aufgetragene Problem lösen kann. Die Wahrnehmung der Umwelt erfolgt über Sensoren, die Einwirkung auf die Umwelt über Aktoren und die intelligente Aussteuerung der Sensoren und Aktoren erfolgt über die Brainware.

Der Problemraum lässt sich zur ersten Orientierung untergliedern in die Bewegungsplanung, die Kollisionsverhinderung, die Konstruktion von Raumkarten aus den gewonnenen Sensordaten, die Entscheidungsfindung in Bezug auf Interoperationsalternativen sowie maschinelles Lernen auf Basis vollzogener Interoperationen. Anhand von Prototypen auf Basis unterschiedlicher Bausätze werden diese Probleme in unterschiedlichen Programmierweisen und mit unterschiedlichen Programmiersprachen gelöst, um am Ende durch einen integrierten und hybriden Lösungsansatz ein Maximum an systemischer Intelligenz zu erreichen.

Diese Aufgabenstellung wurde gewählt, da sie eine Vielzahl von Aufgabenstellungen aus der aktuellen Robotik in sich vereint: Konstruktion von Roboterträgersystemen, Testen verschiedener Sensoren, Motoren und Kommunikationsschnittstellen, sowie Konstruktion und Programmierung entsprechender Brainware. Sie wurde auch deshalb gewählt, da damit ein möglichst umfangreicher Einsatz von Sensoren und Aktoren notwendig wird.

Zur Realisierung der Modelle in Form von lauffähigen Robotersystemen wurde auf die Roboterbausätze von LEGO Mindstorms zurückgegriffen. Diese Bausätze ermöglichen nicht nur den eher spielerischen Umgang mit Problemstellungen der Robotik, sondern haben sich im Einsatz von Forschung und Lehre mehr als bewährt. Der erste Versuch, LEGO mit Computertechnologie zu kombinieren, zeigte sich 1998 durch die erfolgreiche Markteinführung von LEGO Mindstorms in den USA. Seit Ende 1999 wird LEGO Mindstorms auch in Europa angeboten. Mit diesem Produkt kann ein Roboter gebaut und mittels Computer programmiert werden. Das Herzstück dieser Roboter ist das so genannte RCX, ein batteriebetriebener programmierbarer LEGO-Baustein“, der zur Programmierung über eine Infrarotschnittstelle an einen PC angeschlossen werden kann. Er verfügt über ein LCD und Anschlüsse für verschiedene Sensoren (Temperatur-, Licht-, Rotations- und Berührungsensoren) und 3 Elektromotoren. Von der LEGO-Block-Programmiersprache bis Java existieren diverse Programmiersprachen für den RCX. Im Herbst 2006 brachte LEGO die neue Serie Mindstorms NXT (Next Generation) auf den Markt, die eine konsequente Weiterentwicklung von Mindstorms darstellt. Damit verbunden sind verbesserte Sensoren und der Verzicht auf die Noppenstruktur der Bauelemente zugunsten eines stabileren und leichteren LEGO TECHNIC Konstruktionssystems.

Bezüglich der Ausgestaltung der Brainware wird sich in diesem Buch auf die sogenannten exekutiven Funktionen beschränkt, also jenen mentalen Prozessen, die zur Steuerung und flexiblen Anpassung des Roboter systems benötigt werden und die die Impulskontrolle, Interoperationsplanung und Emotionsregulierung umfassen.

---

## 6.2 Hardware

### 6.2.1 Roboterbausätze

#### 6.2.2 Robotic Controller Explorer (RCX)

Das Robotics Invention System ist das basale Set, das für die Entwicklung von LEGO-Robotern benötigt wird. Zu den wichtigsten Bestandteilen des Sortimentes gehören der RCX

**Abb. 6.1** RCX

und eine Infrarot-Schnittstelle für den PC. Das Sortiment enthält aber auch Aktoren und Sensoren. Zur Grundausstattung gehören zwei Motoren sowie zwei Tastsensoren und ein optischer Sensor. Zum Anschluss der Aktoren und Sensoren werden vier kurze und zwei lange Anschlusskabel mitgeliefert. Darüber hinaus besteht das Robotics Invention System aus verschiedenen Basisbausteinen, Achsen und Zahnrädern, sowie einem Differentialgetriebe. Es lassen sich Roboter bauen, die mit Rädern oder einem Raupenantrieb bewegt werden können. Aus den Zahnrädern lassen sich sehr verschiedene Arten von Getrieben bauen, die bereits in den theoretischen Kapiteln dieses Buches behandelt wurden. Für Experimente mit dem Lichtsensor steht eine Testunterlage zur Verfügung, auf der u. a. ein schwarzes Oval aufgedruckt ist. Anhand dieser Linie lassen sich Experimente durchführen, bei denen ein Roboter sich entweder innerhalb des Kreises bewegen muss oder bei dem er – etwas anspruchsvoller in der Programmierung – dem Grenzbereich zwischen schwarz und weiß folgt (Abb. 6.1).

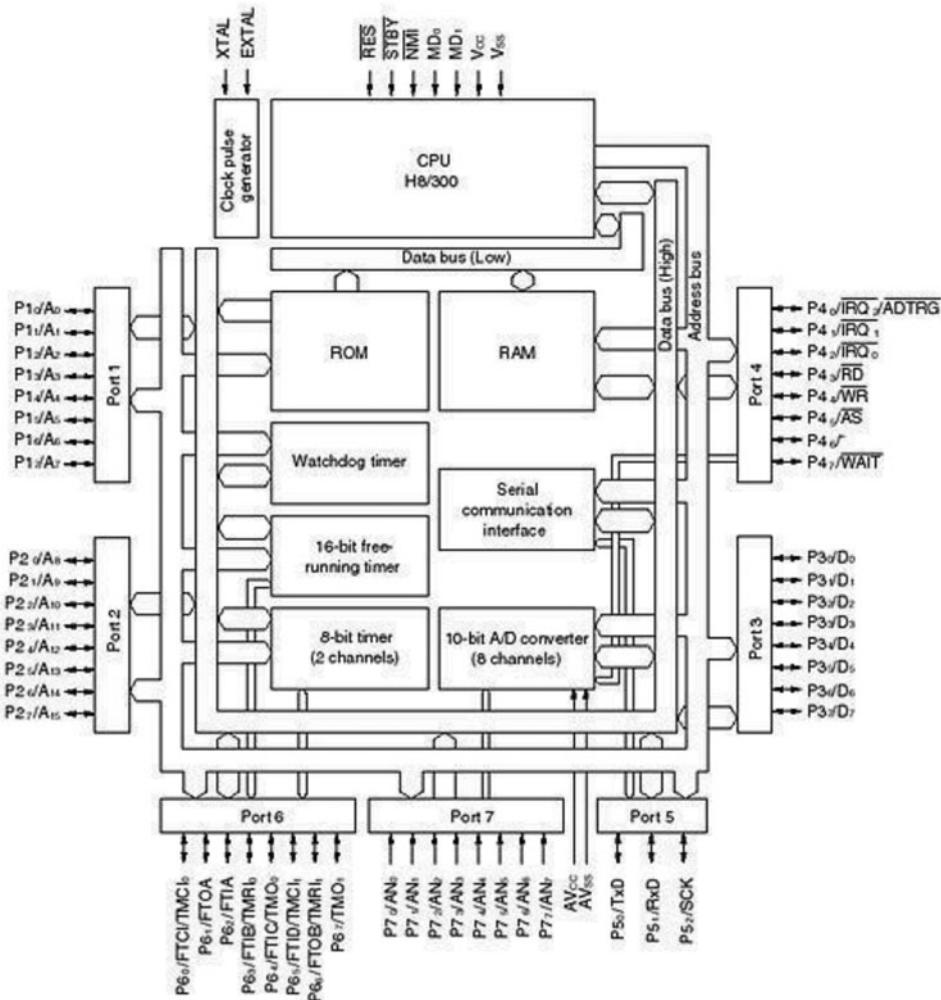
Die zentrale Intelligenz aller mit dem Robotic Invention System entwickelten Roboter-Modellen stellt der *Robotic Controller Explorer* (RCX) dar. Beim RCX handelt es sich um einen kleinen 16 MHz Microcomputer auf der Basis eines Hitachi HS-Prozessors. Der H8/3292 ist ein kleiner 8-Bit-Chip, dessen Rechenleistung sich natürlich nicht mit denen der modernen Personal Computer messen kann. Jedoch liegen seine Stärken in der relativen Einfachheit seiner Programmierung und in seiner engen Abstimmung mit den Werkzeugen des RCX. Zu einem Microcomputer gehören natürlich auch Schnittstellen und Peripheriegeräte. Der RCX besitzt anstelle eines hochauflösenden Monitors nur ein sehr kleines LC Display und die Tastatur verfügt über lediglich vier Funktionstasten. Auch

Programmbereich 1	Programmbereich 2	Programmbereich 3	Programmbereich 4	Programmbereich 5	
Firmware					RAM
Hardware ROM					ROM

**Abb. 6.2** Mindstorms RCX

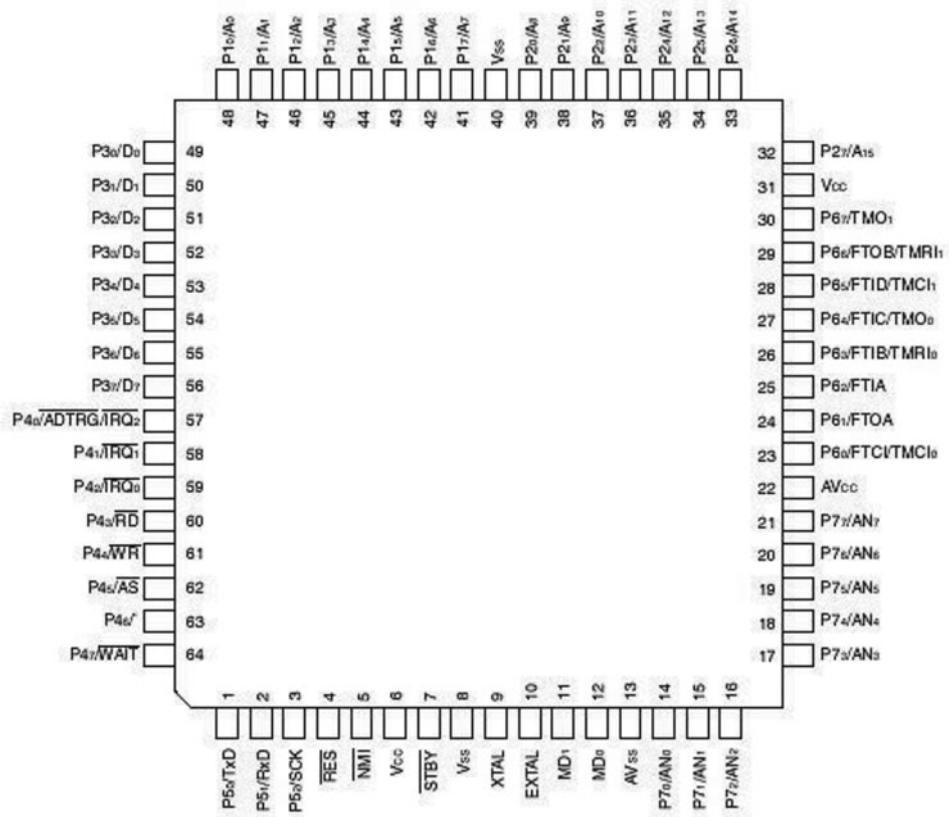
bei den Schnittstellen ist der RCX sehr bescheiden. Es existieren drei Aktoren-Ausgänge, drei Sensoreingänge, die bidirektional arbeitende Infrarot-Schnittstelle und ein kleiner akustischer Signalgeber.

Sein Speichervermögen ist auf die Kapazität des Chips begrenzt und das ist gemessen an den heute bei einem modernen PC üblichen Speicherdimensionen recht wenig. Der gesamte RAM-Bereich auf dem Controllerchip umfasst gerade einmal 32 kByte, was ungefähr 32768 Zeichen entspricht oder – je nach Schriftgröße – acht bis 16 DIN A4-Seiten Text. Der frei programmierbare Bereich liegt demnach bei ungefähr 6 kByte, dann reicht der Speicherbereich gerade mal für einen etwas ausführlicheren ASCII-Text-Brief. Dennoch reicht diese Speicherkapazität aus, um dem 8-Bit-Prozessors fünf leistungsfähige und voneinander unabhängige Programme einzuladen. Der RCX ist *multitaskingfähig*. Ein Mikroprozessor ist in seinem Arbeitsschema ein sehr „primitives“ Gebilde. Er liest einen kleinen Datenblock aus dem Speicher und interpretiert diesen als einen Befehl für eine Rechenoperation. Bei Bedarf liest der Prozessor nun weitere Daten ein und verknüpft diese nach den Regeln seiner Konstrukteure. Ist der Mikroprozessor mit der Abarbeitung eines Befehls fertig, dann liest er den nächsten Datenblock ein, den er wieder als Befehl ansieht. Alles geschieht sequentiell. Es werden nicht mehrere Operationen gleichzeitig durchgeführt. Dennoch erheben Multitasking-Systeme den Anspruch, gerade diese Gleichzeitigkeit zu leisten. Hierzu werden die verschiedenen Tasks bei der Abarbeitung einzelner Befehle abwechselnd bedient. Damit sind nicht die NQC-Kommandos gemeint, sondern der Wechsel in der Abarbeitung der Tasks geschieht auf der Ebene der Maschinensprache. Die Wechsel sind selbst bei einem einfachen Prozessor, wie dem H8 von Hitachi so schnell, dass diese Vorgänge der Anwender überhaupt nicht erkennen kann. Insofern sieht es aus Sicht des Anwenders so aus, als liefen alle Programme zeitgleich ab. Die Programme können dabei mit dem grafischen RCX-Code, einer anderen RCX-kompatiblen Programmiersprache (z. B.: Mindscript oder NQC) oder aber mit der objektorientierten Programmiersprache Java geschrieben werden. Allerdings stellen diese Arbeitsprogramme nur einen Teil der Software dar, die benötigt wird, damit der RCX auch tatsächlich funktionsfähig ist. So muss zum einen eine Verbindung zwischen der Entwicklungsumgebung und dem RCX aufgebaut werden. Darüber hinaus muss der RCX auch verstehen, was die einzelnen Programmsequenzen überhaupt tun sollen und wie die Programme auf die fünf Programmplätze zu verteilen sind. Insofern benötigt auch der RCX eine Basisprogrammierung in Form einer *Firmware*, die beispielsweise die Übertragung über das Infrarot-Interface koordiniert. Diese Firmware setzt sich aus zwei Komponenten zusammen (Abb. 6.2 und 6.3).



**Abb. 6.3** Blockschaltbild des Hitachie H8/3000

Die erste Komponente ist fest im RCX programmiert und enthält die absolut wichtigsten Grundfunktionen. Ohne sie könnte nicht einmal die Startsoftware in den Speicher des RCX geladen werden. Die Startsoftware – der Ladevorgang wird bei der ersten Benutzung oder nach einem Batteriewechsel erforderlich – ist die zweite Firmware-Komponente. Um eigene Firmware-Module zu entwickeln, bedarf es der Auseinandersetzung mit dem H8-Prozessor selbst und dessen Entwicklungs-Toolkits. Insofern setzt sich das Softwaresystem eines Mindstorms-Roboters aus drei Komponenten zusammen: einer Basis-Firmware, die fest im Baustein gespeichert ist, eine zu Beginn zu ladende Firmware und der frei programmierbaren Arbeitsprogrammen des Roboters (Abb. 6.4).



**Abb. 6.4** PIN-Belegung des Hitachie H8/3000

Die *Infrarot-Schnittstelle* des RCX hat drei Aufgaben:

- Empfang der Programme vom PC,
- Empfang von Steuerinformationen einer Infrarot-Fernbedienung (Ultimate Accessory Set, 3801) oder vom PC,
- Kommunikation zweier RCX untereinander (ein zweites Robotics Invention System, bzw. ein zweiter RCX ist erforderlich).

Da für die zweite und dritte Anwendung noch weitere Sets erforderlich sind, reduziert sich der Sinn der Infrarot-Schnittstelle beim ausschließlichen Erwerb eines Robotics Invention System zunächst einmal auf die Programmierung des Roboters. Dazu wird mit dem Robotics Invention System ein Infrarot-Transmitter geliefert, der in der aktuellen Version über den USB an den PC angeschlossen wird. Da verschiedene Computer – insbesondere Notebooks – bereits mit einer IrDA-Infrarot-Schnittstelle ausgerüstet sind, sollte an dieser Stelle betont werden, dass das LEGO-System mit einem proprietären Kommunikationsprotokoll

arbeitet. Der Infrarot-Transmitter ist daher auch dann zu verwenden, wenn der PC bereits eine IrDA-Schnittstelle besitzt. Bei der Benutzung der Infrarot-Schnittstelle ist darauf zu achten, dass die serielle Schnittstelle des Computers funktioniert und frei belegt werden kann. Auch empfiehlt es sich, sich zu merken, an welcher der beiden seriellen Schnittstellen der IR-Transmitter angeschlossen wird. Darüber hinaus ist zu beachten, dass bei älteren Modellen an der Vorderseite des IR-Transmitters ein kleiner Umschalter zu finden ist (bei den neueren Modellen wird diese Umschaltfunktion über die Software realisiert). Wer den IR-Transmitter ausschließlich zur Programmierung einsetzt, kann darüber eine Einstellung für eine geringe Sendereichweite wählen und damit Batterie-Energie einsparen. Oft befindet sich der RCX aber nicht in unmittelbarer Nähe zum IR-Transmitter. In diesen Fällen sollte die Einstellung für eine größere Reichweite gewählt werden. Allerdings hat die Vorgabe einer geringen Reichweite nicht immer nur den Grund, Batterie-Energie einzusparen. Insbesondere dann, wenn im gleichen Raum mehrere RCX programmiert werden sollen, was beispielsweise bei Wettkämpfen der Fall sein kann, empfiehlt es sich, mit geringer Leistung zu arbeiten. Ansonsten besteht das Risiko einer gegenseitigen Beeinflussung fremder RCX-Bausteine. Allerdings muss stets darauf geachtet werden, dass der Infrarot-Transmitter und der RCX eine direkte optische Verbindung haben. Im Klartext bedeutet dies, dass beide Infrarot-Schnittstellen direkt aufeinander ausgerichtet sein müssen. Optische Hindernisse können zu einem Problem werden.

Robotersysteme müssen auf äußere Einflüsse adäquat reagieren können, wozu *Sensoren* benötigt werden. Sensoren kann man durchaus mit den Sinnesorganen vergleichen. So lässt sich mit Fingern tasten, die Haut erkennt Berührungen, die Augen sehen und mit den Ohren vermag man zu hören. Insofern verfügt der menschliche Organismus für jede Art der Wahrnehmung über ein besonderes Sinnesorgan. Etwas Ähnliches muss man dem Robotersystem angedeihen lassen, soll auch dies auf äußere Einflüsse reagieren, diese erkennen und auswerten können. Der RCX besitzt drei Sensoreingänge, an die verschiedene Sensoren angeschlossen werden können. Wenn diese Sensoren später einmal programmiert werden sollen, dann muss dem Robotersystem zum einen mitgeteilt werden, welcher Eingang für die einzelnen Sensoren vorgesehen ist und zum anderen, welche Art von Sensor sich an diesem Eingang befindet. Allerdings versagt auch die beste Programmierung, wenn kein eindeutiger Input kommt. Ein Sensor ist damit als Informationsquelle beinahe so wertvoll, wie ein gutes Programm. Die Vielfalt der möglichen Sensoren ist nahezu unbegrenzt. Auch das LEGO-Mindstorms-Programm bietet bereits eine gut überlegte Auswahl von Sensoren an. Dazu gehören unter anderem die folgenden externen Sensoren: Berührungssensoren, optische Sensoren, Rotationssensoren und Temperatursensoren. Neben diesen existieren auch noch weitere „interne“ Sensoren. Diese können durchaus sehr nützliche Dienste leisten und die Funktion des Roboters unterstützen: Timer, Zähler, und Infrarot-Empfänger. Technisch betrachtet, sind folgende Sensoren möglich: Spannungs- und Stromsensoren, Magnetsensoren, Neigungssensoren, Drucksensoren, Gassensoren, Mikrofone, Radioaktivitätssensoren, Infrarot-Sensoren, Windsensoren, u. v. m.

Das LEGO-Sortiment beinhaltet im Grundbaukasten des Mindstorms-Programmes drei externe Sensoren: zwei Berührungssensoren und einen Lichtsensor. So lassen sich

über geeignete Hebel Sensoren zur Erkennung des Endes einer Tischplatte oder zur Erkennung eines Hindernisses bauen. Mit dem Lichtsensor, der mit einer eigenen Lichtquelle arbeitet, können Linien verfolgt oder auf Veränderungen der Raumhelligkeit reagiert werden. Neben diesen externen Sensoren verfügt auch der RCX über interne Sensoren. So besitzt der kleine Microcomputer des RCX drei Timer, die von den Programmen ausgewertet werden können. Damit lassen sich bestimmte Aktivitäten des Roboters zeitlich begrenzen oder Wartezeiten programmieren. Neben den Timern gibt es Zähler. Damit kann das Robotersystem veranlasst werden, nicht auf den ersten Sensorimpuls, sondern auf eine bestimmte Anzahl von Impulsen zu reagieren. Als interner Sensor fungiert der Registerspeicher des RCX, die Timer- und Zählerstände ständig mit den in ihnen gespeicherten Werten vergleichen. Zusätzlich dient der Infrarot-Empfänger des RCX zur Steuerung über den PC oder über eine Fernbedienung aber auch zur Programmierung des Robotersystems. Steht ein zweiter RCX zur Verfügung, so können auch zwei Roboter über die Infrarot-Schnittstelle miteinander kommunizieren. Der *Berührungssensor* ist ein sehr einfacher kleiner Taster, mit dem die Zustände „Kontakt geschlossen“ und „Kontakt offen“ simuliert werden können. Der Berührungssensor liefert in diesem Fall quasi eine digitale Information (0 oder 1). Der Roboter kann nun anhand des geschlossenen oder geöffneten Kontaktes entscheiden, welche Aktivität er ausführen soll. Dieser Umstand lässt sich programmtechnisch ausnutzen, indem Produktionsregeln definiert und implementiert werden können:

WENN

Kontakt an Sensoreingang 1 geschlossen,

DANN

Starte Motor am Ausgang A des PCX.

Allerdings ist die 0-1-Auswertung des Berührungssensors nur der am häufigsten verwendete Idealfall, denn der Sensor ist keineswegs ein digitales Element. Es handelt sich nämlich in Wirklichkeit um einen variablen Widerstand, der durchaus verschiedene Messwerte zulässt. Damit ein sinnvoller Einsatz auch möglich ist, darf der Schalter auch nur bei den gewünschten Ereignissen betätigt werden. Im Robotics Invention System ist neben den beiden mechanischen Berührungssensoren auch ein *optischer Sensor* beigelegt. Mit diesem lassen sich anhand von Markierungen auf einer Unterlage Entfernen ermitteln oder den Roboter entlang einer Linie navigieren. Dieser optische Sensor besteht aus einer kleinen Leuchtdiode (LED) und einem lichtempfindlichen Bauteil. In der Regel werden als lichtempfindliche Bauteile Photodioden oder Phototransistoren verwendet, es können aber auch lichtempfindliche Widerstände (LDR = Light Depending Resistor) zum Einsatz kommen. Mit der Lichtquelle und dem eigentlichen Lichtsensor, die zusammen den optischen Sensor darstellen, lassen sich sehr interessante Projekte realisieren. So kann die Photodiode des Sensors allein als Helligkeitsmesser eingesetzt werden. Dabei kann mit dem Sensor die Umgebungshelligkeit gemessen und das Ergebnis in einem Programm ausgewertet werden. Setzt man ein kleines Röhrchen vor die Photodiode, dann blockt

man den Einfall seitlichen Lichts ab und erreicht damit eine punktuelle Auswertung eines Helligkeitswertes. Im Zusammenspiel mit der eigenen Lichtquelle ist der Sensor unabhängig von der Beleuchtung seiner Umgebung. Soll ein Roboter beispielsweise einer Linie folgen, dann kann der nach unten gerichtete Sensor die Unterlage beleuchten und auch in dunkler Umgebung auswerten, ob der Sensor auf einem hellen oder dunklen Untergrund gerichtet ist. Das liegt daran, dass eine schwarze Unterlage das Licht absorbiert, also gewissermaßen jede Lichtenergie in sich verschlingt. Eine weiße Unterlage reflektiert das Licht dagegen, was vom Sensor erkannt werden kann. Eine Sonderform eines optischen Sensors stellt eine Kamera dar, die zur Erfassung von Bildern eingesetzt wird, die am PC als Fotos oder als Video gespeichert werden können. Eine intelligente Software macht die Kamera auch zu einem echten Sensor, denn es werden Bewegungen in einzelnen Bildausschnitten erkannt. Ein *Rotationssensor* ist ein recht komplexes Element, das Drehbewegungen einer Achse zählt, die durch den Sensor geführt wird. Eine volle Umdrehung entspricht 16 Zählimpulsen. Darüber hinaus kann die Rotationsrichtung bestimmt werden. Dabei bedient man sich des folgenden mathematischen Zusammenhangs: der Umfang eines Rades kann mit Hilfe der Formel

$$U = d * \pi,$$

berechnet werden, wobei U der Umfang, d der Durchmesser und pi die Kreiskonstante mit dem Wert 3,14... darstellt. Je nach Drehrichtung zählt der Sensor aufwärts oder abwärts. Für die Auswertung in einem Programm für den RCX gibt es verschiedene Möglichkeiten:

- der Rotation Sensor Watcher-Block bzw. der Sensor Monitor (in RIS 2.0),
- der „Check and Choose“ Stack-Controller-Block bzw. in RIS 2.0 der „Ja-oder-Nein“-Block,
- der „Repeat While“ Stack-Controller-Block (Schleifen)
- und der „Wait until“ Stack-Controller-Block (entspricht in RIS 2.0 dem „Warte bis“-Block).

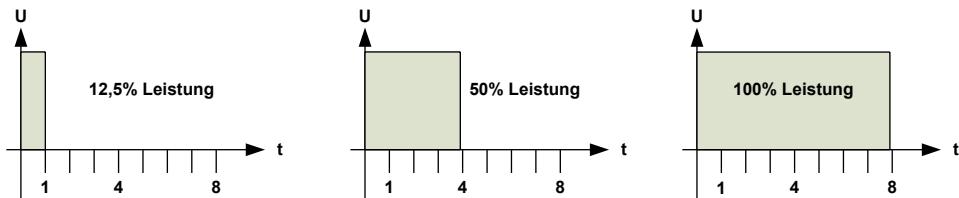
Beim Rotations-Sensor Watcher Block-handelt es sich um ein Programm-Modul, das den Sensor permanent überwacht und dessen Zählerstand auswertet. Die drei anderen Blöcke stellen Bestandteile von Abfragen und Schleifen während eines Programmablaufes dar. Sie sind nur dann aktiv, wenn sie gerade vom aktuellen Programm aufgerufen werden. Der Rotationssensor findet in verschiedenen Modellen Anwendung. Er kann für die Steuerung eines Motors eingesetzt werden. Soll der Motor eine Drehung in einem bestimmten Winkel ausführen, muss er nach dem Anlauf rechtzeitig an der gewünschten Stelle wieder gestoppt werden. In der Regel setzt man sogenannte Schrittmotoren für solche Zwecke ein. Einfacher kann man die Bewegung eines Motors steuern, wenn man seinen Drehwinkel misst. Der Rotationssensor ist dabei ein gutes Messwerkzeug, denn er liefert bei jeder Umdrehung Impulse und darüber hinaus eine Aussage zur Drehrichtung. Mit einem geeigneten Programm kann der Motor beispielsweise angewiesen werden, nach rechts

zu drehen und beim Stand des Drehzählers von 32 zu stoppen. Damit wird der Motor seine Achse zweimal rechts herum drehen. Für eine halbe Drehung wäre der entscheidende Zählerstand acht und für eine Viertel Drehung entsprechend vier. Die doch recht grobe Auflösung eines 1/16-Rasters kann übrigens durch den geschickten Einsatz eines Getriebes optimiert werden. Eine zweite mögliche Anwendung eines Rotationszählers ist die Längenmessung. Die Überlegung dabei ist, dass ein Rad bei einer einzigen Umdrehung exakt den Weg zurücklegt, der seinem Umfang entspricht. Ist also der Umfang des Rades bekannt, kann anhand der gezählten Umdrehungen die zurückgelegte Strecke bestimmt werden. Wird darüber hinaus auch die dafür gebrauchte Zeit ausgewertet, kann die durchschnittliche Geschwindigkeit errechnet werden. Sind die Messintervalle sehr kurz, dann stellt ein solches System quasi einen Tachometer dar, wie er in jedem Auto zu finden ist. Der Hitachi-H8- Microcomputer enthält verschiedene Register, in die Zählerwerte eines Sensors eingetragen werden können bzw. die automatisch fortlaufend mit einem aktuellen Zählerstand belegt werden, der aus dem Systemtakt des kleinen Rechners abgeleitet wird. Da die Frequenz des Systemtaktes bekannt ist, können daraus Zeitwerte in Sekunden errechnet werden. Mit Hilfe eines so entstandenen Timers hat man die Möglichkeit, beispielsweise einen Motor eine bestimmte Zeit lang laufen zu lassen. Neben dem internen Timer können Zähler aber auch dazu eingesetzt werden, um die Anzahl der Aktivitäten eines Berührungssensors zu ermitteln oder die Wechsel der Lichtverhältnisse an einem optischen Sensor zu zählen. Damit können beispielsweise Positionsbestimmungen durchgeführt werden.

Damit das Robotersystem in der Lage ist, mit den Eindrücken, die ihm die Sensoren liefern, auch etwas anzufangen zu können, benötigt es noch einen anderen Typ von Peripherie: die *Aktoren*. LEGO bietet einige externe aber auch RCX-interne Aktoren an. Das Mindstorms Sortiment bietet folgende Aktoren an: Motoren, Lampen, die Infrarotschnittstelle des RCX (Sender) und den Lautsprecher des RCX. Denkbar wären aber auch Robotersysteme, die mit weitergehenden Aktoren ausgestattet sind. Hier könnten beispielsweise folgende Aktoren zum Einsatz kommen: Elektromagnete, elektrische Hydraulik- oder Pneumatikpumpen, Sirenen, Hochfrequenzsender und Laser.

Ein Roboter zeichnet sich in erster Linie durch seine Beweglichkeit aus. Der originale Mindstorms-Motor hat relativ geringe Ausmaße, er ist intern auch mit einem Getriebe ausgestattet. Durch dieses interne Getriebe werden dem Motor nicht nur sehr gleichmäßige Laufeigenschaften verliehen, er wirkt auch im direkten Antrieb bereits sehr kräftig. Andere Motoren, bei denen die Welle direkt angetrieben wird, zeichnen sich zwar durch eine hohe Drehzahl aus, jedoch erreichen sie diese nur im Leerlauf. Unter Belastung bricht die Drehzahl merklich ein und – dies ist aus Sicht eines möglichen häufigen Batteriewechsels im RCX durchaus interessant – führt zu einer sehr hohen Stromaufnahme.

Neben dem internen Getriebe und der geringen Größe hat das Gehäuse des Motors eine bemerkenswerte Form. An der Seite des Motorgehäuses befinden sich vier kleine Schlitze. Im Sortiment des Robotics Invention System sind entsprechende kleine Adapter zu finden, die in diese Schlitze eingeschoben werden und dem Motor damit rechts und links eine Auflagefläche anbieten. Diese hat die Größe eines flachen Zweier-Blocks. Die Ge-



**Abb. 6.5** Motoren und ihre Leistung

häuseform ermöglicht einen ausgesprochen platzsparenden Einbau des Motors, der für die Robotermodelle unbedingt erforderlich ist. Auf andere Motoren sollte nur bei Bedarf zurückgegriffen werden. Beispielsweise dann, wenn sich ein Roboter auf einer Schiene bewegen soll. Dies kann bei einem Modell eines Hochregallagers sehr sinnvoll sein. Die Motoren unseres Roboters werden über beliebige Aktorenausgänge am RCX (A, B und C) angesteuert. Dazu wird eine Verbindung mit einem der mitgelieferten Verbindungskabel hergestellt. Weiterhin muss die Motoraktivität programmiert werden. Generell gibt es die Möglichkeit, einen bestimmten Aktorenausgang gezielt ein- oder auszuschalten. Darüber hinaus kann dessen Drehrichtung verändert werden. Dies lässt sich durch einen einfachen Polaritätswechsel erreichen. Zusätzlich kann auch die Geschwindigkeit des Motors vorgegeben werden. Alle diese Optionen zur Steuerung eines Motors können im grafischen RCX-Programmcode definiert werden. Dabei gilt es zu bedenken, dass ein RCX ein digitaler Baustein ist, dessen Ausgänge nur die Zustände *Ein* oder *Aus* kennen. In Bezug auf einen Motor übersetzt, könnte man sagen, dass sich der Motor entweder dreht oder nicht dreht. Dennoch ist es sehr oft erforderlich, die Geschwindigkeit des Motors nicht digital, sondern eher analog zu beeinflussen, d. h., es wird die Höhe der Spannung verändert, um die Drehzahl des Motors zu beeinflussen. Eine hohe Spannung bedeutet, dass sich der Motor schnell dreht. Eine kleine Spannung steht für eine langsame Fahrt. Der RCX arbeitet nicht mit einem Transformator. Auch wäre ein leistungsfähiger Digital-Analog-Wandler recht aufwendig. Man beschritt daher bei der Entwicklung des RCX einen anderen Weg und verzichtete auf die Veränderung der Betriebsspannung zur Leistungsregelung am Aktorenausgang und veränderte in schnellen Intervallen die Einschaltzeiten. Man nennt dies eine Pulsweitenmodulation (PWM). Das Prinzip einer solchen Pulsweitenmodulation ist sehr einfach. Bei voller Leistung ist der Ausgang permanent eingeschaltet. Wenn dem Motor jedoch eine geringere Leistung angeboten werden soll, dann wird der Ausgang in sehr schnellen Intervallen ein- und ausgeschaltet. Die Höhe der Leistung, die dem Motor zur Verfügung steht, hängt vom zeitlichen Verhältnis der Ein- und Ausschaltzeiten ab. In der minimalen Leistungsstufe des RCX dauert die Einschaltperiode nur ungefähr 1 ms. Dagegen steht die Ausschaltperiode von 7 ms. Dem Motor stehen damit nur 12,5 % der maximalen Leistung des RCX zur Verfügung (Abb. 6.5).

Die Darstellung der Pulsweitenmodulation durch saubere Ein- und Ausschaltphasen ist natürlich der optimale Zustand an den Ausgängen. In der Realität wird der Verlauf des Spannungspegels ganz erheblich von sehr verschiedenen Einflüssen beeinträchtigt. Die Tatsache nämlich, dass ein Motor sowohl als Antrieb als auch als Stromerzeuger arbeiten

kann, erklärt die merkwürdigen Verlaufsverformungen. In den Zeiten, in denen der RCX keine Spannung auf den Aktorenausgang schaltet, dreht sich unser Motor nämlich immer noch weiter und wird damit selbst zur Stromquelle, dessen Spannung Abweichungen erzeugen. Je schneller der Motor bei kurzen Aus-Perioden am RCX dreht, umso größer ist die vom Motor in dieser Zeit erzeugte Spannung. Aber auch wenn man den Aktorenausgang ohne einen Motor und damit quasi im „Leerlauf“ seine Arbeit verrichten lässt, wird der den idealen Verlauf eines pulsweiten modulierten Ausgangssignals nicht erzielen. Die Schaltkreise des RCX müssen vor unangenehmen elektrischen Wirkungen geschützt werden, wie sie beispielsweise Motoren in der bereits beschriebenen Form erzeugen. Aus diesem Grunde werden die Aktorenausgänge mit Kondensatoren gepuffert. Ein solcher Kondensator wirkt wie ein Akku. Er kann geladen und wieder entladen werden. Im Gegensatz zum Akku, der in einem Batterieersatz zum Einsatz kommt, sind die Lade- und Entladezeiten bedeutend kürzer. In der Impulsphase, also in der Zeit, in der der RCX seinen Aktorenausgang einschaltet, wird der Pufferkondensator aufgeladen. In der Pausenphase entlädt sich dieser über die angeschlossenen Schaltkreise allmählich. Diese Entladung verläuft umso schneller, je mehr Leistung dem RCX abverlangt wird. Am wenigsten Leistung wird am offenen Ausgang abverlangt, wodurch die Entladezeit ein Minimum beträgt.

Die hauptsächliche Aufgabe des Aktors *Lampe* ist die Funktion als Signalgeber. So kann die Lampe als Kontrollleuchte den Status eines Programmes anzeigen. Kontrollleuchten spielen in der Technik eine wichtige Rolle, denn es lassen sich ohne größeren Aufwand die Betriebszustände elektrischer Anlagen überwachen. Unter anderem eignen sich Lampen recht gut zur Simulation oder Überprüfung von Roboter-Programmen, bei denen es darum geht, das Ansprechen der Aktoren zu überprüfen, ohne dabei einen kompletten Roboter bauen zu müssen. Allerdings haben Lampen als Aktoren-Prüfer einen großen Nachteil, denn sie leuchten nicht „vorwärts oder rückwärts“. Sie leuchten einfach nur. Zur Feststellung einer Bewegungsrichtung sind Lampen also ungeeignet. Natürlich kann eine Lampe auch zur Steuerung eines Roboters eingesetzt werden. Wenn diese auf einen Lichtsensor gerichtet wird, kann ein Roboter daraus eine gewisse Aktivität ableiten. Eine solche Lampe wird im Fachjargon als *Ohm'sche Last* bezeichnet, weil sie keinerlei speichernde Funktion hat wie der Kondensator (Kapazität) und auch keine Spannungen induziert, wie es beispielsweise die Motorwicklungen (Induktivitäten) tun. Es gilt einzig und allein das Ohm'sche Gesetz:  $U = R \cdot I$ .

Ein *Lautsprecher* dient der Ausgabe von Tonsignalen. So kann die Ausgabe von Tönen durchaus als Warnsignal von Bedeutung sein. Nicht ausgeschlossen ist auch, dass mit einem Roboter gesprochen und ihm dadurch mit menschlicher Sprache Befehle gegeben werden können. In diesem Fall muss der Roboter natürlich in der Lage sein, in entsprechender Weise ein Feedback zu geben. Der Hitachi-Prozessor des RCX ist natürlich nicht in der Lage, Sprache zu verarbeiten, aber man kann die Tonausgabe dennoch einsetzen, um auch von einem LEGO-Roboter ein gewisses akustisches Feedback zu bekommen. Dies ist beispielsweise unmittelbar nach der Programmierung bei den ersten Testläufen recht sinnvoll. Das akustische Signal, das in verschiedener Weise durch verschiedene Tonfrequenzen und -längen programmiert werden kann, ist sehr gut geeignet, um die erreicht-

ten Programmstellen zu kennzeichnen. So lässt sich sehr schnell anhand eines akustischen Signals erkennen, ob das Robotersystem beispielsweise auf einen Sensor richtig reagiert und den korrekten Programmablauf startet. Die Experimente mit dem Mindstorms-Programm lassen sich noch interessanter gestalten, wenn man zwei oder mehr RCX-Module einsetzen kann. Jeder RCX ist dabei ein eigenständiger Roboter, jedoch können alle Module miteinander kooperieren, wenn sie miteinander kommunizieren können. Eine sichere Übermittlung von Informationen ist per Funk oder – wie mit dem RCX – per Infrarot möglich. Die *Infrarot-Schnittstelle* des RCX dient als Empfänger für die Signale der Infrarot-Fernbedienung aus dem Ultimate Accessory Set (380 l). Im RCX-Code gibt es beispielsweise das Programmierelement *Send Message*, mit dem über die Infrarot-Schnittstelle Informationen verschickt werden können. Auf diese Weise lassen sich Meldungen über betätigte Sensoren und Zählerstände an einen zweiten RCX übermitteln, der dann seinerseits diese Informationen für den eigenen Programmablauf auswerten kann.

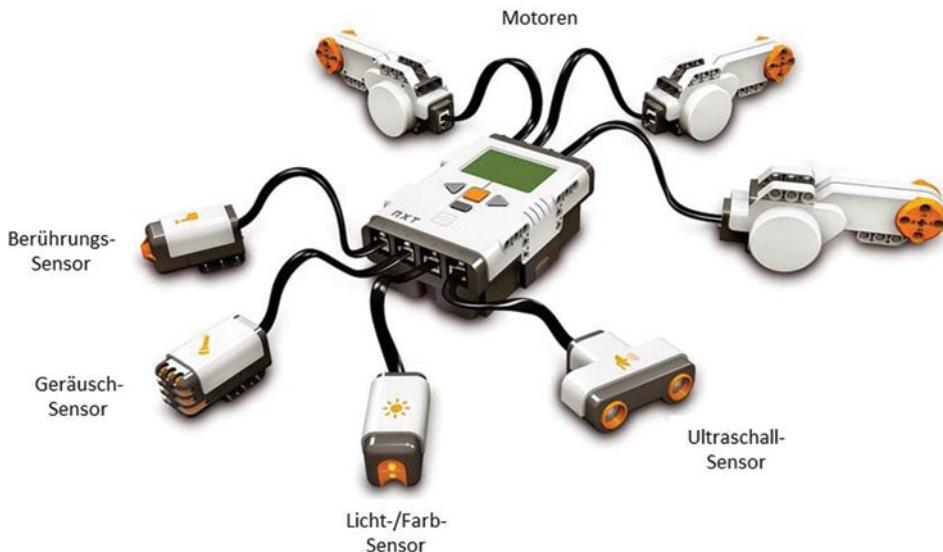
## NXT

Bei der neuen Mindstorms NXT Serie wurden der Tastsensor sowie der Lichtsensor verbessert. Neu entwickelt wurden der Ultraschallsensor, der Tonsensor und der NXT-Baustein, der das neue Herzstück des Roboters darstellt. Dieser NXT-Baustein kann nun erstmals auch von einem Apple-Computer aus programmiert werden. Die vielseitige Software baut auf dem LabView-Programm von National Instruments auf. Dank der neuen Bluetooth-Schnittstelle können Programme kabellos auf den NXT-Baustein übertragen werden und der Roboter kann durch Fernbedienen gesteuert werden (z. B. von einem Handy oder einem PDA aus). Außerdem ist auch eine Verbindung via USB-Kabel möglich. Die 3 Servomotoren besitzen neu eingebaute Rotationssensoren zur präzisen Geschwindigkeitskontrolle. Für grösere Stabilität des Roboters sorgen 519 LEGO-TECHNIC Elemente, welche wie oben erwähnt grösstenteils keine Noppenstruktur besitzen.

Der Lego Roboter NXT besitzt einen 32-Bit ARM Prozessor mit 48 MHz Taktfrequenz, 256 KB Flash-RAM, 64 KB RAM, einen USB Anschluss sowie einen Bluetooth v2.0 Chip mit EDR (Enhanced Data Rate). Ebenfalls, wie beim Vorgänger, gehören zum Lieferumfang des Bausatzes mehrere Sensoren und Aktuatoren sowie zahlreiche Lego Technik Bausteine. Speziell dieses Lego Technik System bietet die Basis für die Roboterkonstruktionen. Mit dem NXT Baustein hat Lego aber auch die nächste Generation der programmierbaren Steuerung eingeleitet. Um sensortechnisch abwärts kompatibel zu bleiben, bietet Lego auch Adapterkabel an, um das alte System nutzen zu können. Seit Veröffentlichung der Firmware als Open Source, kann man in Zukunft Programme von Drittanbietern erwarten (Abb. 6.6).

Diverse neue und modifizierte Sensoren gibt es bereits. Insgesamt umfasst der Bausatz die folgenden Inhalte:

- NXT Baustein
- Sensoren (Ultraschall, Berührung, Licht und Geräusch)



**Abb. 6.6** NXT

- 3 Servomotoren mit einer Art Resolverfunktion (ähnlich Schrittmotor – Rückmeldung der Position des Motors)
- Bauteile (rund 600 Stück) um diverse Fahrzeuge und roboterähnliche Figuren bauen zu können
- 6-adriges Kabel unterschiedlicher Länge
- Software auf CD für Windows und Mac
- Testpad im ca. DIN A2 Format um die entwickelten Programme zu testen

Der NXT-Baustein besitzt:

- Drei *Ausgabe Ports* zum Anschliessen der Motoren: Port A, B und C
- Vier *Eingabe Ports* zum Anschliessen der Sensoren: Port 1, 2, 3 und 4 (Port 4 beinhaltet einen IEC 61158 Type 4/EN 50 170 kompatiblen Erweiterungssport für zukünftigen Gebrauch)
- *USB 2.0 Schnittstelle* (12 Mbit/s) für die Verbindung zum Computer (PC oder Mac).
- *Bluetooth* (Class II V2.0, BlueCore 4 Chip, 460.8 Kbit/s) Schnittstelle für die drahtlose Verbindung zum Computer sowie zur Fernsteuerung. Unterstützt das Serial Port Profile (SPP), besitzt 1 MB externen FLASH Speicher für den Bluetooth stack.
- *Lautsprecher* für das Abspielen von Soundeffekten: 8 kHz Soundqualität, Soundkanal mit 8 Bit Auflösung und 2–16 kHz Sample-Rate
- *Schaltflächen*: Orange: On/Enter/Run, Hellgrau: Navigation im NXT-Menü, Dunkelgrau: Clear/Go back

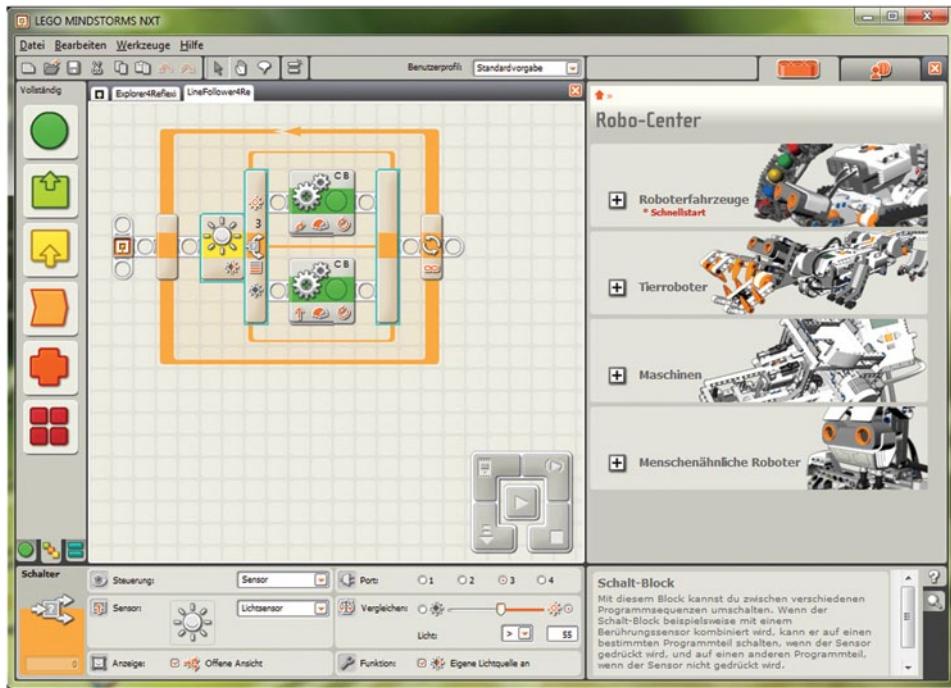
- *Bildschirm:* 100×64 Pixel LCD Display
- *Prozessoren:*
- CPU: 32-bit ARM7 Mikrocontroller – AT91SAM7S 256 (256 kBytes FLASH, 64 kBytes RAM, 48 MHz – 0.9 MIPS/MHz);
- Co-Prozessor: 8-bit AVR Mikrocontroller – ATMEGA48 (4 kBytes FLASH, 512 Byte RAM, 4 MHz – 1 MIPS/MHz)

Die *Sensorik* umfasst unterschiedliche Sensoren mit folgenden Leistungsumfang:

- Der *Tastsensor* gibt dem Roboter einen Tastsinn, er detektiert 1, wenn er betätigt wird und 0, wenn er wieder freigegeben ist.
- Der *Tonsensor* kann sowohl unangepasste Dezibelwerte [dB] wie auch angepasste Dezibelwerte [dBA] ermitteln. Die Einheit Dezibel ist ein Mass für den Schalldruckpegel, wobei Werte zwischen 55 dB und 90 dB gemessen werden können. Diese Messwerte werden durch den NXT in Prozenten ausgedrückt [%]. Je niedriger der Prozentwert, desto leiser ist die Tonquelle.
- Der *Lichtsensor* ermöglicht dem Roboter, zwischen Hell und Dunkel zu unterscheiden. Er kann die Lichtintensität in einem Raum (Helligkeit) oder einer farbigen Oberfläche messen.
- Der *Ultraschallsensor* ermöglicht dem Roboter Abstände zu Objekten zu berechnen, indem er die Zeit misst, die benötigt wird, bis eine von ihm ausgesendete Schallwelle an ein Objekt stösst und deren Echo wieder empfangen wird. Er misst Abstände in der Einheit Zentimeter oder Zoll und ist in der Lage Distanzen bis zu 255 cm mit einer Präzision von ± 3 cm zu messen.

Die *Aktorik* bietet 3 Motoren, die den Roboter mit der Fähigkeit ausstatten, sich zu bewegen. Jeder Motor hat einen eingebauten Rotationssensor, welcher die Ansteuerungspräzision erhöht. Der Rotationssensor misst Umdrehungen in Winkelgrad (Genauigkeit von einem Grad). Dank dieses Sensors kann durch Veränderung der Powereinstellung jeder Motor mit unterschiedlicher Geschwindigkeit betrieben werden.

Die LEGO Mindstorms NXT-Entwicklungsumgebung ermöglicht, den selbst konstruierten NXT-Robotern zu programmieren und die Programme über die USB- oder Bluetooth-Schnittstelle auf den NXT-Baustein zu übertragen. Diese Mac und PC kompatible visuelle Programmierung gemäss dem „drag and drop“- Prinzip, die auf dem LabView Programm aufbaut, besitzt zum jetzigen Zeitpunkt vier Roboterkonstruktionen sowie dazugehörige Programmieranleitungen. Das Software User Interface stellt ein Arbeitsfenster (work area) bereit, in dem einzelne Blöcke (z. B. Move, Display, Sound, Record/Play, Wait, Loop, Switch) aus der Programmierpalette grafisch einfach verknüpft werden können (Abb. 6.7).



**Abb. 6.7** Mindstorm Entwicklungsumgebung

### 6.2.3 Roboterkonstruktion

Der Roboter als Trägersystem der Hardware, Software und Brainware wurde so einfach wie möglich und so komplex wie nötig konstruiert, so dass er die folgenden Anforderungen erfüllt:

- Fahrbare Konstruktion, die an Ort und Stelle Drehungen von  $\pm 90$  Grad vollführen kann, ohne dass sich der Mittelpunkt des Roboters um mehrere Zentimeter verschiebt, damit das Wenden in engen Räumen möglich ist.
- Sensorarm, an dem der Licht- und Tonsensor vertikal nach oben gerichtet montiert werden kann, damit einerseits die Lichtqualität des Raumes wiedergegeben wird und andererseits Motorengeräusche durch die Ausrichtung des Mikrofons abgeschwächt werden.
- Drehbare Plattform, auf welcher der Ultraschall Sensor montiert werden kann, damit sich dieser unabhängig von der fahrenden Basis bewegen kann.
- Der Schwerpunkt des Roboters sollte möglichst nahe bei der Motorenachse liegen, um eine möglichst hohe Stabilität zu erreichen.
- Die Dimensionierung des Roboters sollte möglichst kompakt sein, damit sich dieser auch in engen Räumen fortbewegen kann.

## Sensorik

Zunächst unterscheidet man die Sensoren und Aktoren hinsichtlich ihres Wirkbereiches in externe und interne Sensoren bzw. Aktoren. Interne Sensoren überwachen den inneren Zustand eines Systems, und externe Sensoren sammeln Informationen über die Umgebung. Interne Aktoren verändern die Stellung des Roboters und externe manipulieren Gegenstände in der Umgebung des Roboters. In der Aktorik sind dabei die Übergänge zwischen internen und externen Aktoren eher fließend, da man nicht immer eine klare Grenze zwischen den Wirkbereichen ziehen kann.

Vergleichbar mit den Sinnesorganen von Organismen ist ein Sensor ein mechanisch-elektronisches Bauteil, das eine messbare physikalische oder chemische Größe (Temperatur, Druck, Geruchstoff, Farbe, Entfernung etc.) in ein geeignetes elektrisches Signal umwandelt. So werden zur Lösung des Problems zunächst *taktile Sensoren* verwendet, die Informationen durch den mechanischen Kontakt bestimmter Elemente bereitstellen. Solche taktilen Sensoren dienen zur Positionsbestimmung, Objekterkennung oder zur Ermittlung der Oberflächenbeschaffenheit. Mit Hilfe der taktilen Sensoren lassen sich Kräfte als auch deren Verteilung bestimmen, um beispielweise die gefundenen Objekte sicher zu halten. Man unterscheidet dabei zwischen tastenden, gleitenden und den Kraft-Moment-Sensoren.

Taktil tastende Sensoren können zur Stabilisierung von Greifvorgängen oder Haltevorgängen eingesetzt werden. Taktil gleitende Sensoren liefern Informationen über Oberflächenbeschaffenheit und geometrischen Strukturen eines Objektes. Mit Hilfe von Kraft-Moment-Sensoren lassen sich vorgegebene Bewegungsabläufe kontrollieren und bei Abweichungen von Ist- zu Solldaten gezielt abbrechen.

Zusätzlich werden *Näherungssensoren* in Form nichttaktiler Sensoren eingesetzt, um die Ortung von Objekten zu realisieren. Diese Sensoren bringen den Vorteil mit sich, dass sie aufgrund fehlender Verschleißteile, eine längere Haltbarkeit wie andere Sensoren aufweisen. Ein weiterer Vorteil zeigt sich in dem Umstand, dass die zu ortenden Objekte nicht berührt werden müssen und so auch kein Schaden entstehen kann. Hierbei kann man zwischen verschiedene Typen des Näherungssensors wählen: induktiv, kapazitiv, optisch und akustisch.

Ein induktiver Näherungssensor reagiert auf nahe liegende metallische Objekte auch ohne mit ihnen in Kontakt kommen zu müssen. Vorteil dieser Sensoren ist, dass sie sehr zuverlässig arbeiten und durch ihre Konstruktion als sehr robust auch gegenüber störenden Umwelteinflüssen gelten. Ein kapazitiver Näherungssensor unterscheidet berührungslos zwischen in der Nähe sich befindliche leitenden und nicht leitenden Objekten. Der Vorteil dieser Sensoren liegt darin, dass sie trotz großer Abstände ziemlich exakt reagieren und auch in unzugänglichen bzw. gefährlichen Umgebungen (z. B. radioaktive Umgebungen) zuverlässig arbeiten können. Optische Näherungssensoren arbeiten nach dem Reflexionsprinzip auch bei großen Entfernen zuverlässig. Dabei sendet eine Sendereinheit einen gerichteten Lichtstrahl an den Empfänger und wenn dieser Strahl unterbrochen wird, löst dies ein entsprechendes Signal

aus. Auch die akustischen Näherungssensoren eignen sich für große Entferungen, arbeiten allerdings nach dem Prinzip des Ultraschalls. Gemäss diesem werden Schallsignale gesendet, die anschließend vom Objekt reflektiert werden. Die Entfernung dieses Objektes wird durch die Messung der Zeit zwischen dem Absenden des Signals bis zum Empfang berechnet.

Ebenso werden *Abstandssensoren* eingesetzt, um die Entfernung zwischen dem Sensor und den Gegenständen in der Umgebung zu messen. Sie dienen dem Robotersystem nicht nur zur Orientierung in seiner Umgebung, sondern auch der effektiven Kollisionsvermeidung. Dieser Sensortyp hat gegenüber den Näherungssensoren den Vorteil, dass er eine auch größere Reichweiten erfassen und die Geometrie von Objekten erkennen kann. Dabei unterscheidet man zwischen optischen und akustischen Abstands- sowie Radarsensoren.

So kann der optische Abstandssensor als eine Erweiterung des Näherungssensors aufgefasst werden. Er erzeugt elektromagnetische Wellen im sichtbaren Bereich. Hierdurch zeigen diese Sensoren ein von Fremdeinwirkungen unbeeinflusstes Verhalten und arbeiten entweder nach dem Laufzeitverfahren, Triangulation oder Stereoskopie. Akustische Abstandssensoren arbeiten im Ultraschallbereich und dienen der Hinderniserkennung sowie der Entfernungsmeßung. Dabei greift man auch das Echoprinzip zurück, bei dem der Sensor einen kurzen Ton aussendet, der von einem Körper reflektiert wird. Dabei gilt es zu beachten, dass man beim Einsatz akustischer Absstandssensoren mit Ungenauigkeiten rechnen muss und es zu Überlagerung von Reflexionen (Crosstalk) kommen kann. Radar ist das Akronym für „Radio Detection and Ranging“ und ein solcher Sensor liefert Information über Entfernung und genaue Position eines Objektes.

*Visuelle Sensoren* liefern Informationen über die Strukturierung der Umgebung bzw. über Struktureigenschaften erfasster Objekte. Sie arbeiten ebenfalls berührungslos und zeichnen sich durch eine recht hohe Messgenauigkeit aus.

Zur Problemlösung werden in diesem Fall positionsempfindliche Photodioden eingesetzt, die eruieren, mit welcher Intensität und an welchem Ort das Licht auftrifft.

Zu den externen Sensortypen zählen vor allem solche Sensoren, die den inneren Zustand eines Roboters ermitteln, wie beispielsweise die Position, die Geschwindigkeit und die Orientierung.

Ein Positionssensor gibt Information über die Position eines Greifers oder über die entsprechende Position des Robotersystems. Solche Positionssensoren sind beispielsweise Potentiometer, optische Codierer, Differentialtransformatoren oder magnetisch-induktive Gebersensoren. Ein optischer Encoder, wie er in diesem Fall in den Motoren und dort für die Bestimmung der Geschwindigkeit zum Einsatz kommt, richtet dazu einen Lichtstrahl auf einen Fotodetektor aus. Auf einem rotierenden Rad, das sich vor dem Detektor befindet, ist ein verschlüsseltes Muster abgebildet. Der ausgerichtete Lichtstrahl wird durch diesen Code in regelmäßigen Abständen unterbrochen, der Encoder beobachtet diese Markierungen und wertet die durch die Unterbrechungen entstandenen Impulse aus. Um die Geschwindigkeit des Motors zu bestimmen, werden die Umdrehungszahlen der Scheibe ausgewertet.

## Aktorik

*Aktoren* (engl. Actuators) sind sozusagen das Pendant zu den Sensoren. Unter einem Aktor werden die beweglichen Bauteile eines Roboters subsummiert, die es ermöglichen, dass der Roboter seine Form, Position verändern und durch Interoperationen auf seine Umwelt einwirken kann. In der Fachliteratur werden oftmals die Aktoren, die im direkten Kontakt mit der Umwelt stehen, auch als Effektoren bezeichnet. Im Rahmen der Validierung kommen bei dem autonomen Robotersystem hauptsächlich Räder, Ketten und Greifer zum Einsatz. Dabei wird die Fortbewegungsart dem späteren Einsatzgebiet entsprechend angepasst.

So wird beispielsweise auf glattem Untergrund ein fahrendes Robotersystem von Vorteil sein, da kein Lastwechsel erfolgen muss. In unwegsamen Gelände hingegen kann ein laufender Roboter sich als vorteilhaft erweisen, indem er durch eine insektenartige Fortbewegungsweise auch größere Hindernisse bewältigen kann.

Damit das Robotersystem sich seiner Umwelt fortbewegen kann, werden in diesem Lösungsansatz Räder und Ketten anstelle von Beinen verwendet. So werden beispielsweise Räder gewählt, wenn eine einfache Mechanik, eine einfache Bauweise und ein geringes Eigengewicht gefordert sind. Der Nachteil an Rädern zeigt sich in einer geringen Leistung, wenn das System auf unebenem Grund interperieren soll. Für solche Fälle gilt es Ketten als Alternative zu wählen. So werden Roboter, die in der Natur eingesetzt werden, mit Ketten ausgestattet. Solche Kettenroboter können mit Hilfe ihrer großen Lauffläche auch größere Hindernisse überwinden und sind auch gegenüber der Bodenbeschaffenheit (Sand oder Geröll) eher leidenschaftslos. Damit einher geht allerdings der Nachteil, dass durch die extreme Reibung innerhalb der Ketten, die Leistung eher abnimmt und durch den Druck der Ketten gegen den Boden mit Energieverlusten zu rechnen ist. Auch an die Positionsbestimmung des Roboters werden hohe Ansprüche gestellt, da der Roboter unter Umständen seine Position je nach der Umdrehungszahl der einzelnen Räder der Kette neu bestimmen muss.

Bezüglich der Antriebsarten bei radgetriebenen Robotersystemen lassen sich gleich mehrere Ansätze unterscheiden. Dabei stellt der *Differentialantrieb* sicherlich die einfachste Form der Roboterantriebe in Bezug auf deren Konstruktion und Programmierung dar. Hier sind zwei separat angetriebene Räder gemeinsam auf einer Achse angeordnet, das dem Roboter ermöglicht, geradeaus zu fahren, sich auf der Stelle zu drehen und sich auf einer Kreisbahn fortzubewegen. Ein Nachteil dieser Antriebsform könnte sich in Bezug auf das Gleichgewicht des Roboters ergeben, indem oftmals ein oder mehrere Stützräder angebracht werden, die als frei schwenkbare Räder für die Balance sorgen. Die Anzahl der benötigten Stützräder hängt unter anderem auch von der Gewichtsverteilung bzw. der Schwerpunktverlagerung des Roboters ab.

Hingegen sind beim *Synchronantrieb* die Räder so miteinander verbunden, dass sie sich alle immer gleichzeitig bewegen, was eine weitaus komplizierte Mechanik erforderlich



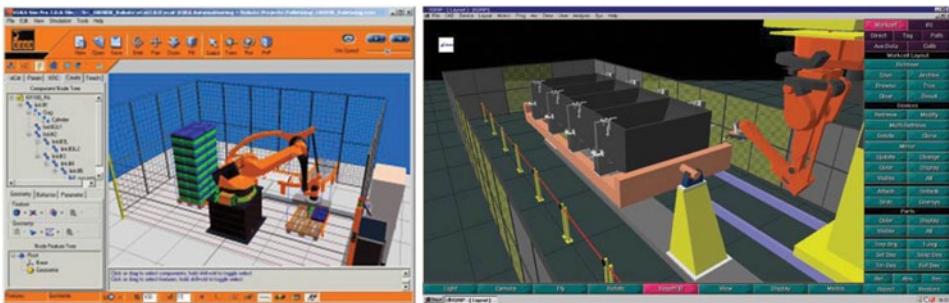
**Abb. 6.8** Impressionen der Radaantriebe

macht. Beim sogenannten *Dreiradantrieb* wird das lenkbare Vorderrad von einem Motor angetrieben und die beiden feststehenden Hinterräder von einem zweiten, separaten, Motor gesteuert. Die Motorgeschwindigkeiten müssen bei dieser Antriebsform nicht wie beim Differentialantrieb kontrolliert werden, um ein Fahren geradeaus zu ermöglichen. Vielmehr reicht es auch, sicherzustellen, dass das Vorderrad in die jeweilige Fahrtrichtung positioniert ist. Ein Nachteil des Dreiradantriebs ist sicherlich die Tatsache, dass sich ein mit dieser Antriebsform ausgestatteter Roboter nicht um die eigene Achse zu drehen vermag. Aus dem Bereich der Kraftfahrzeuge ist die sogenannte *Ackermannlenkung* bekannt, die aus zwei feststehenden Antriebshinterrädern und zwei synchron lenkbaren Vorderrädern besteht. Beim Einsatz von von drei oder vier angetriebenen sogenannten Allseitenrädern (Omni-Wheels, Mecanum-Räder) ist der Roboter in der Lage, sich in alle Richtungen zu bewegen ohne sich vorher um die eigene Achse drehen zu müssen (Abb. 6.8).

### 6.3 Software

Bezüglich der Programmierung von Robotern existieren viele verschiedene Verfahren, die sich allerdings alle in die beiden Kategorien der Online- und Offlineprogrammierung eingliedern lassen. Bei der Onlineprogrammierung ist der Roboter während der Programmierung aktiv ist, nicht abgeschaltet und demnach „online“. Zur *Onlineprogrammierung* zählt man insbesondere das „Teach-In“ – Verfahren bzw. das „Play-Back“ – Verfahren zugehörig ist, wobei die Programmierung an sich meistens über sogenannte „Control Panels“ erfolgt.

Unter dem „Teach-In“ – Verfahren versteht man alle Verfahren, bei denen einem Roboter sein späteres Verhalten durch „Learning by doing“ so lange angelernnt wird, bis das gewünschte Ergebnis erzielt ist. Dazu zählt das Play-Back-Verfahren, bei dem das Robotersystem in seinem Bewegungsablauf bzw. Verhalten direkt geführt wird. Dazu zählt aber auch die direkte Parametereingabe der für die Bewegungsabläufe notwendigen Parameter über dedizierte Schnittstellen, Computer oder Control Panels.



**Abb. 6.9** Simulationsumgebung

Der Nachteil der Onlineprogrammierung besteht sicherlich darin, dass durch die notwendige Anlernphase kostbare Zeit in der Produktion verloren geht, zumal in der Regel der gesamte Produktionsprozess für die Umprogrammierung eines einzelnen Roboters angehalten werden muss.

Insbesondere deshalb stellt die *Offlineprogrammierung*, sofern der eingesetzte Roboter diese Art der Programmierung zulässt, eine willkommene Alternative dar. Hierzu zählen die Programmierung per 3D-Simulation, die Erstellung und Kompilierung von Quellcode, sowie das Editieren von Quelldateien die von der Robotersteuerungssoftware direkt via „download“ geladen werden müssen.

Derzeit koexistieren noch eine Vielzahl verschiedener speziell angepasster Roboterprogrammiersprachen zur Quellcodekodierung. Dabei sind viele dieser Sprachen speziell und damit proprietär auf bestimmte Robotertypen/-familien ausgerichtet. Allerdings setzen sich parallel hierzu immer mehr freie Programmiersprachen oder Erweiterungen für gängige Programmiersprachen wie C++ und JAVA zur Programmierung von Robotersystemen durch. Diese Sprachen bieten neben deren allgemeinem Komfort alle speziell für Roboter benötigten Funktionen und Leistungsmerkmale (Abb. 6.9).

Die folgende Aufzählung von Sprachen, die für die Programmierung von Robotersystemen herangezogen werden können, ist sicherlich unvollständig: AL, AML, AR-BASIC, ARLA, BAPS, COSIMIR, IRL, JARS, KAREL, PA-LIB, RAPID, ROBEX, SRL, TPE, QC, VAL/VAL2/VAL3, KRL, RCCL. In der ersten Auflage dieses Buch wurde auch NQC behandelt, eine Sprache („Not Quite C“), die sich an C bzw. C++ orientiert. Sie umfasst zwar weniger Funktionen als C und C++, ist dafür aber sehr viel kompakter und der für einen Roboter benötigte Betriebeskern, sowie die darauf aufbauenden Programme, beanspruchen die knappen Hardwareressourcen relativ gering. Diese Eigenschaften macht die Sprache beispielsweise für die Lego-Mindstorm Roboter Bausatz sehr nützlich. VAL, VAL2, VAL3 und KRL hingegen sind Sprachen, die speziell für Industrieroboter entwickelt wurden. Die VAL-Sprachfamilie wurde von dem Industrieroboterhersteller Stäubli und KRL von KUKA entwickelt.

In Bezug auf die Programmierung wird derzeit intensiv geforscht. So versucht man beispielsweise den Robotern die natürliche Sprache des Menschen, das Reden, Deuten, Zeigen, das Fühlen „einzutragen“. Dieses Kapitel stellt allerdings die Fähigkeit, Reflexe zu

besitzen, Entscheidungen zu fällen, der explorative Umgang mit Dingen der Umwelt, das Lernen, das Kommunizieren bzw. Kooperieren und somit das Interoperieren in den Vordergrund.

---

## 6.4 Brainware

Dieser Abschnitt beschreibt die Brainware des mobilen Robotersystems, die für die „intelligente“ Steuerung zuständig ist. Dabei wird nicht nur auf den aktuellen Stand der Technik aufgesetzt, sondern dieser konsequent zu sogenannten Musterlösungen (solution patterns) ausgebaut, die für weitere Entwicklungen angepasst und wiederverwendet werden können.

### 6.4.1 Anforderungen

Bereits in den 60er Jahren unterteilte man die Steuerung eines mobilen Robotersystems in drei in sich abgeschlossene Funktionsbereiche:

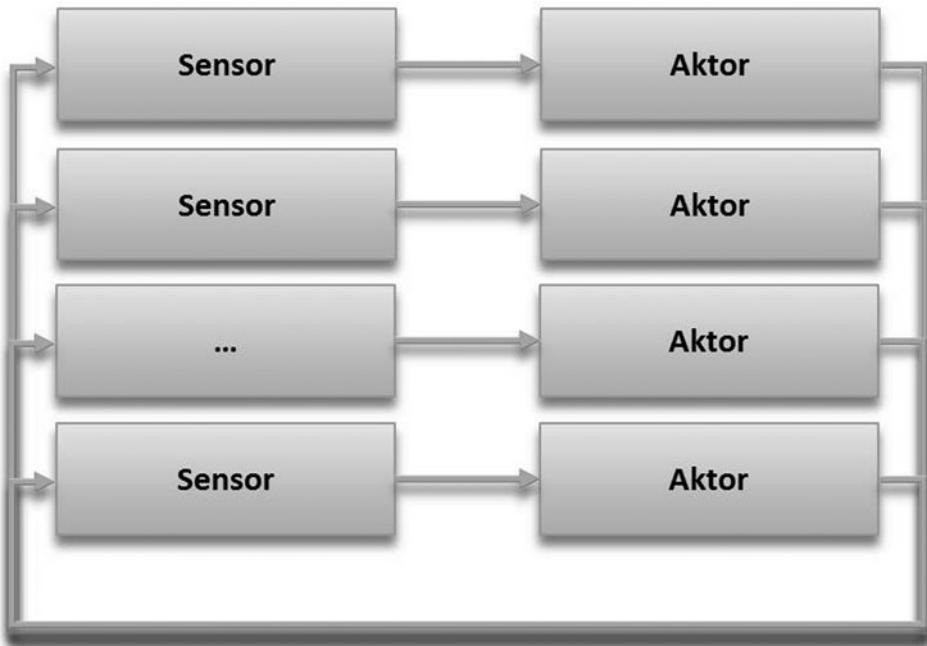
- *Sensorik*: dient zur Datenerfassung mit Hilfe von Sensoren, wobei letztere oftmals durch Funktionen der Datenvorverarbeitung angereichert werden.
- *Planung*: Basierend auf den durch die Sensorik bereitgestellten Daten werden Entscheidungen getroffen und entsprechende Aktionen ausgewählt.
- *Aktorik*: Entsprechend der ausgewählten Aktionen werden die einzelnen Akteure angesteuert, wobei letztere durch Funktionen der Datennachverarbeitung ergänzt werden können.

Dieser Ansatz lässt sich auch mit der zentralen Hypothese der klassischen AI in Verbindung bringen, nach der natürliche kognitive Systeme deshalb intelligent sind, weil es sich hierbei um die Verarbeitung physikalischer Symbole handelt. Wenngleich man sich damit dem Problem aussetzt, den durch die Sensorik und Aktorik übermittelten Daten eine Bedeutung zukommen lassen zu müssen. Insgesamt sieht dieser „klassische“ Ansatz also vor, dass die Funktionsbereiche zyklisch durchlaufen werden, was den Vorteil mit sich bringt, dass die dadurch notwendigen *sequentiellen Strukturen* relativ einfach zu implementieren sind bzw. die hierfür aufzubringenden Rechenzeiten recht bescheiden ausfallen (Abb. 6.10).

Gleichwohl des einfach erscheinenden Ansatzes lassen sich durch unterschiedliche Verschaltung der Funktionsbereiche auch unterschiedliche Entscheidungs- bzw. Verhaltensweisen realisieren. Bei einer *reakтив-reflexiven Architektur* handelt es sich in der Regel um ein zustandsloses Steuerungssystem. Aus den Sensordaten werden direkt die Vorgaben für die Akteure berechnet, ohne auf vorherige Daten zurückzugreifen oder zukünftige Situationen vorauszuplanen. Dadurch entfällt die Notwendigkeit einer Planung, wobei sich mehrere Sensor-Aktor-Sequenzen parallel ausgeführt werden können (Abb. 6.11).



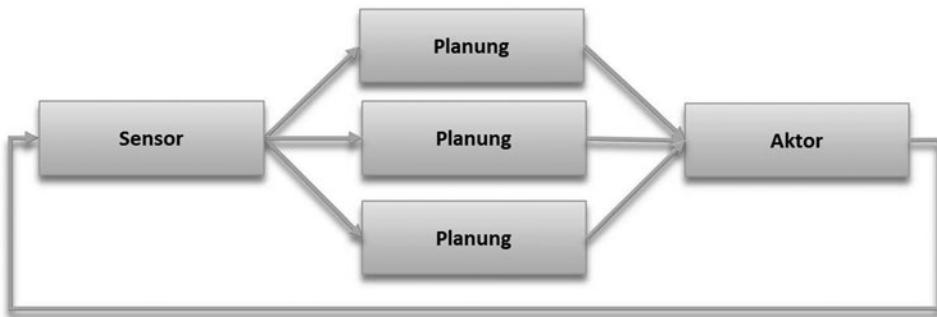
**Abb. 6.10** Sequentielle Robotersteuerung



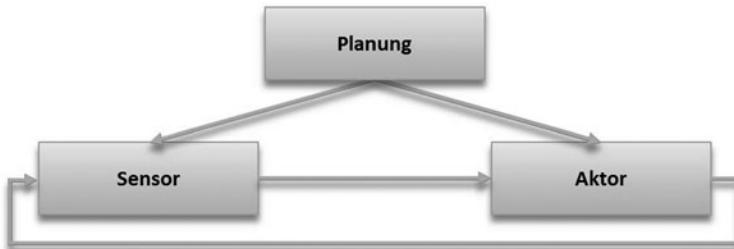
**Abb. 6.11** Reaktiv-reflexive Robotersteuerung

Eine *verhaltensbasierte Architektur* wurde bereits in den späten 80er Jahren des letzten Jahrhunderts entwickelt, bei der verschiedene Verhalten statt konkrete Aufgabenlösungen implementiert wurden. Dabei mussten interne Zustände berücksichtigt und für die Planung vorgehalten werden, was jedoch den Umstand dennoch zuließ, dass die Planungsergebnisse widersprüchlich sein konnten und daher ein Konfliktmanagement den Aktoren vorgeschaltet werden musste (Abb. 6.12).

Insofern lag es nahe, die geringe Komplexität des reaktiven Ansatzes zu nutzen, um aber dennoch die Planung auch komplexer Aufgaben zu ermöglichen. Dies führte zu einer Architektur, bei der die einzelnen Schichten der Steuerung entkoppelt wurden. Dabei arbeiten die Sensoren und Aktoren unabhängig von der Planung, jedoch kann die parallel arbeitende Planungsinstanz bei ausreichender Parametrisierung die Sensoren und Aktoren beeinflussen. Weitere Variationen solcher klassischer Ansätze sind möglich und in der



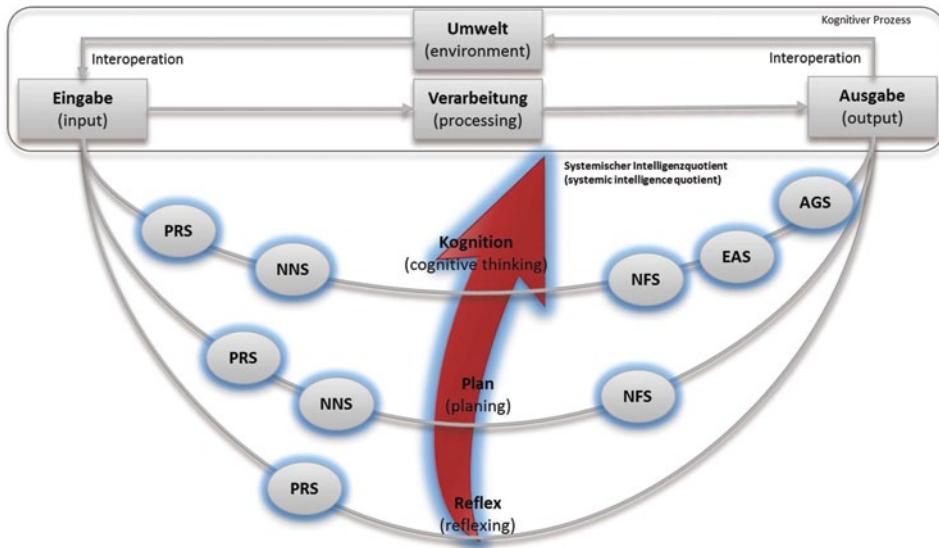
**Abb. 6.12** Verhaltensbasierte Robotersteuerung



**Abb. 6.13** Hybrid-deliberative Robotersteuerung

Vergangenheit auch entsprechend umgesetzt und in der Praxis entsprechend eingesetzt worden (Abb. 6.13).

Das Verhalten des Robotersystems in diesem Buch wird neben deren wahrgenommenen Rolle als Problemlöser und ausgezeichneter Persönlichkeitsmerkmale vor allem durch dessen intrinsische artifizielle Kognition gesteuert. Diese artifizielle Kognition ist in der Brainware realisiert und als solche die für die Verhaltensauswahl zuständige Instanz, d. h. sie entscheidet über die Ausgestaltung dieses Verhaltens in Form von Interoperationen. Diese Brainware bedient sich dabei eines situativen Umweltmodells, das sowohl durch die von der Simulationsumgebung an den Roboter als Agenten gelieferte Wahrnehmung als auch durch das eigene Verhalten ausgeprägt wird. Dabei ermöglicht dies auch *Reflexe*, die unter Umgehung der Brainware als nicht unterdrückbare Interoperationen direkt das Verhalten beeinflussen und dadurch auf die Umgebung einwirken können. Die Brainware des Robotersystems überprüft dabei nach der Verarbeitung der Wahrnehmung, ob eventuell die aktuelle Situation durch einen *Reflex* gemeistert werden kann. Ist ein solcher Reflex möglich und sinnvoll, dann wird er ausgelöst und es werden keine weiteren Überlegungen bezüglich des Verhaltens in dieser Situation angestellt. Dadurch wird der Erfahrung aus dem Alltag Rechnung getragen, dass sich Reflexe und Automatismen im Wettkampf bzw. zur Problembewältigung unter Umständen gegenüber höheren kognitiven Verhaltenswei-



**Abb. 6.14** Kognitionsbögen der systemischen Intelligenz

sen auszahlen. Nachdem das Robotersystem seine Positionierung vorgenommen hat, kann es sich davon ausgehend ein „Bild“ seiner Situation machen und die darin vorkommenden Objekte in einem *situativen Umweltmodell* speichern. Ebenfalls werden in diesem Modell auch die nicht sichtbaren, sondern eher antizipierten Objekte berücksichtigt. Diese antizipativen Fähigkeiten betreffen die Tatsache, dass das Robotersystem sowohl andere Robotersysteme als auch das Zielobjekt dann wahrnehmen kann, wenn sie sich hinter dem Robotersystem, sozusagen in seinem Rücken befinden. Weiterhin umfasst die Antizipation das eigene Verhalten, indem das Robotersystem die Effekte dieses Verhaltens vorausberechnen kann. Die Qualität des Umweltmodells wird dahingehend verbessert, indem die einzelnen Robotersysteme als Agenten miteinander kommunizieren und sich so über die jeweiligen individuellen Umweltmodelle untereinander austauschen können. Dabei gilt es zu beachten, dass die Simulationsumgebung als auch die Roboterbausätze nur eine beschränkte Kommunikation ermöglichen.

Reichen die Reflexe jedoch nicht aus, um das Problem in der gegebenen Situation zu lösen, dann prüft das Robotersystem, ob die vorhandenen Pläne eine Problemlösung herbeiführen können oder ob ein Plan hierzu situativ noch entwickelt werden muss. Wird auch dem planerischen Umgang mit der Problemsituation keine adäquate Lösungswahrscheinlichkeit zugemessen, wird auf weitere Komponenten des kognitiven Prozesses bzw. die hierzu notwendigen Technologien des Cognitive Computings zurückgegriffen und somit der Kognitionsbogen weiter gespannt (Abb. 6.14).

Die Probleme, die es hierbei zu lösen gilt, beziehen sich vor allem auf die *Navigation* und die *Identifikation* von Hindernissen beziehungsweise der entsprechenden Zielobjekte. Dabei kommt der Navigation die zentrale Aufgabe zu, das Erreichen eines Ziels zu

ermöglichen, wozu eine entsprechende Bewegungssteuerung notwendig ist. Man unterscheidet hierbei zwischen einer sogenannten lokalen Navigation, die nur die unmittelbare Umgebung berücksichtigt und auch eine Hindernisumgehung beinhaltet von der sogenannten globalen Navigation, die einen Routenplan zum Zielpunkt ermittelt und diesen an die lokale Navigation zur weiteren Verarbeitung übergibt.

Für eine Navigation ist die Kenntnis der eigenen Position und für das Ermitteln des Routenplans einer Karte der Umgebung erforderlich. Dabei reicht in diesem Fall eine zweidimensionale Karte aus, die dann auch für die Hindernisumgehung verwendet werden kann. Für die Realisierung solcher Karten kann man zwischen drei Modellen wählen:

- *Gitterbasiertes Modell:* Die Umgebung wird in einzelne Zellen zerlegt, wobei für jede Zelle die Information gespeichert wird, ob sie frei oder durch ein Hindernis besetzt ist. Um auch mit Ungenauigkeiten umgehen zu können, erfolgt die Realisierung oftmals mit Hilfe von Wahrscheinlichkeitswerten in Form reeller Zahlen im Bereich zwischen 0 und 1. Der Vorteil dieser Modellvariante liegt in der recht einfachen Implementierung durch beispielsweise zweidimensionale Arrays. Ein Nachteil liegt sicherlich darin, dass bei sehr großen Karten durch den entsprechenden Speicherbedarf unter Umständen mit hohen Laufzeiten für die Routenplanung gerechnet werden muss.
- *Merkmalbasiertes Modell:* In diesem Modell werden Art, Position und Ausrichtung der vorkommenden Entitäten der Umgebung gespeichert. Die Speicherung der Merkmale orientiert sich dabei auf die Erkennbar- und Detektierbarkeit der eingesetzten Sensorik. So lassen sich beispielsweise durch Einsatz von Kameras ganze Objekte speichern und verarbeiten.
- *Topologisches Modell:* Die Umgebung wird als Graph gespeichert, wobei die Kanten die Routen und die Knoten die Abzweigungen repräsentieren.

Bei der gitterbasierten und merkmalbasierten Variante kann die Lokalisation durch *Map-Matching* erfolgen, wohingegen bei der topologischen Variante oftmals die *Graphensuche* zum Einsatz kommt.

Die *Hindernisumgehung* hat die zentrale Aufgabe, eine Kollision mit Hindernissen bzw. mit Objekten, die als solche erachtet werden, zu vermeiden. Dabei gilt es aber gleichzeitig sicherzustellen, dass die vorgegebenen Ziele erreicht bzw. „nicht aus den Augen“ verloren werden.

Entsprechend dieser Anforderung gestaltet sich der Technologieeinsatz, indem Produktionsregelsysteme, Fuzzy Logic und Neuronale Netze vorgesehen werden.<sup>1</sup>

So bieten sich Produktionsregeln dazu an, die Steuerung der Reflexe zu realisieren, indem das Verhalten auf einzutreffende Situationen in Wenn-Dann-Strukturen festgelegt wird. Die Fuzzy-Logic bietet die Möglichkeit, einen Aktor basierend auf linguistischen Variablen und transparenten Regeln auch in solchen Fällen zu konzipieren, wo eher weiche Bedingungen verarbeitet werden müssen. So könnte ein Regelsatz für die Hindernisum-

---

<sup>1</sup> Vgl. Brunak und Lautrup 1993.

gehung durch entsprechende Klassen und Methoden in einem Java-Programm wie folgt sich gestalten:

```
If (klareSicht) then Geschwindigkeit = MaxGeschwindigkeit;  
If (keineSicht) then Rotation = 90 and Geschwindigkeit = 10;  
If (erreichtZiel) then Rotation = 0 and Geschwindigkeit = 0;
```

Hier gilt es sicherzustellen, dass widersprüchliche Befehle durch ein entsprechendes Konfliktmanagement aufgelöst werden müssen, so dass beispielweise vermieden wird, dass sich die Regeln gegenseitig aufheben und das Robotersystem vor einem Hindernis einfach stehen bleibt. Eine weitere Schwierigkeit besteht darin, aus den Sensordaten geeignete Werte für die linguistischen Variablen so aufzubereiten, dass diese entsprechend von der Fuzzy Logik auch ausgewertet werden können. Der Einsatz eines neuronalen Netzes ist dafür vorgesehen, um die Sensordaten direkt in das Netz einzuspeisen und die Ausgänge so zu konzipieren, dass die Akteure je nach Musterbeispiel für den Antrieb angesteuert werden. Insbesondere für den Fall, dass eine hinreichend realistische Experimentier- bzw. Simulationsumgebung zur Verfügung steht, lässt sich solch ein Netz gut trainieren, um den Anforderungen zu genügen.

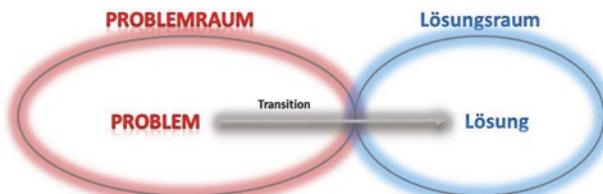
#### 6.4.2 Musterlösung

In den folgenden Abschnitten wird das artificielle kognitive System in Form eines Lösungsmusters (solution pattern) beschrieben, indem auf Basis der Architektur die Teilfunktionen des Systems aus Sicht der Implementierung aufbereitet werden. Insofern wird unter der Architektur eines Systems in diesem Abschnitt zunächst seine Außensicht, d. h. die verschiedenen Bestandteile (Klassen, Schnittstellen) und ihre Beziehungen zueinander, verstanden.

Die Wahl des didaktischen Mittels bzw. des Entwurfprinzipes, das kognitive System als Lösungsmuster zu beschreiben, hat gleich mehrere Gründe:

- Das Lösungsmuster dokumentiert die Ergebnisse zahlreicher Entwurfsversuche und der damit verbundenen Entwurfsproblemen.
- Dabei lassen sich Abstraktionen finden, die sich auf einer Ebene oberhalb der von einzelnen Klassen und Objekten bewegen, die allerdings die einzelnen Klassen und Objekte identifizieren, charakterisieren und letztlich spezifizieren.
- Das Lösungsmuster dient zur Dokumentation von Lösungsarchitekturen und dient der ziel- und problemorientierten Lösungsentwicklung. Damit verbunden ist der Einbezug

**Abb. 6.15** Problem-Transition-Lösung



von nichtfunktionalen Eigenschaften wie Änderbarkeit, Zuverlässigkeit, Robustheit, Testbarkeit und Wiederverwendbarkeit.

- Es ist es möglich, auch komplexe Lösungsarchitekturen zu entwickeln, indem das Lösungsmuster als basaler Ausgangspunkt für die weitere Entwicklung verwendet wird. Insofern gewährleisten sie die Beherrschung der Komplexität, die von solchen Problemen zu erwarten ist.
- Lösungsmuster stellen einen Ausgangspunkt, aber keine vollständigen Lösungen dar.

Insofern beschreibt das in diesem Abschnitt vorgestellte Lösungsmuster sowohl das Entwurfsproblem einer artifiziellen Kognition als auch das generische Schema seiner Lösung (Abb. 6.15).

Ein Lösungsmuster kann dabei als ein Tripel verstanden werden, dass sich aus drei Bestandteilen zusammensetzt:

- *Problem*: In diesem Teil der Musterlösung wird ein bestimmtes Problem beschrieben, dass innerhalb des Problemraumes auftritt. Dabei gilt es das Wesen des Problems zu erfassen. Zur Beschreibung können beispielsweise Anforderungen (requirements), Randbedingungen (constraints) oder angestrebte Eigenschaften (properties) herangezogen werden.<sup>2</sup>
- *Lösung*: Dieser Teil der Musterlösung beschreibt die Lösung, wie das Problem annähernd gelöst werden kann. Dabei gilt es zu beachten, dass die Musterlösung an sich nicht schon die Lösung bzw. einen vollständigen Lösungsentwurf darstellt, sondern eher als ein generisches Lösungsprinzip aufzufassen ist.<sup>3</sup>
- *Transition*: Dieser Teil beschreibt die Konfiguration bzw. Orchestrierung von Elementen oder Diensten (services), die sicherstellen, dass das Problem in eine Lösung überführt werden kann. Insofern wird auch beschrieben, wie Komponenten oder Beziehungen von Komponenten mit den Mitteln einer Programmiersprache oder Notation implementiert werden können. Zur Beschreibung lassen sich beispielsweise die Diagramme der UML oder aber direkt Programmiersprachen Java, C++, etc. verwenden.

<sup>2</sup> Siehe auch Leffingwell und Widrig 2000.

<sup>3</sup> Siehe auch Buschmann et al. 1986.

Lösungsmuster kommen dabei in drei Ausprägungen vor, die sich vor allem in Bezug auf ihren Detailierungsgrad und ihrem Einsatzzeitpunkt während des Entwicklungsprozesses unterscheiden:

- *Architekturmuster* beschreiben das Strukturierungsprinzip von Lösungssystemen, indem die Menge der Subsysteme, Komponenten oder Klassen, deren jeweiliger Zuständigkeitsbereich als auch deren Beziehungen untereinander definiert werden.
- *Entwurfsmuster* beschreiben die Strukturen der einzelnen Subsysteme, Komponenten oder Klassen sowie deren Dienste, die vorausgesetzt werden müssen, damit diese Subsysteme, Komponenten oder Instanzen von Klassen die gestellte Aufgabe oder das Problem lösen. Man kann diese Entwurfsmuster als *Makrocode* zur Entwicklung einer Problemlösung ansehen.
- *Programmmuster* beschreiben den Lösungsansatz eines Subsystems, Komponente oder Klasse in einer bestimmten Programmiersprache. Programmuster sind Muster auf der niedrigsten Abstraktionsebene und orientieren sich in Bezug auf die Lösungsmöglichkeiten an den Grenzen der jeweiligen Programmiersprache. Man kann diese Programmmuster als *Mikrocode* zur Entwicklung einer Problemlösung ansehen.

Architekturmuster können zu Beginn des Lösungsentwicklungsprozesses eingesetzt werden, um damit die technologischen Rahmenbedingungen zu setzen. Entwurfsmuster kommen während des gesamten Entwicklungsprozesses und Programmiersprache in der Implementierungsphase zur Anwendung.

Die Lösungsmuster müssen in einer angemessenen Form und in einem angemessenen Detailierungsgrad dargestellt werden. Ausgehend von der Problem-Lösungsstruktur werden im Handbuch Cognitive Computing ein Schema zur Beschreibung von Lösungsmuster erarbeitet. Dieses gründet sich auf den folgenden Bestandteilen:

- *Name*: Sprechender Name des Lösungsmuster
- *Zusammenfassung*: Kurze Beschreibung des Lösungsmusters
- *Synonyme*: Andere Bezeichnungen oder Namen, unter denen das Lösungsmuster in der Literatur Verwendung findet bzw. in der Praxis zur Anwendung kommt.
- *Beispiel*: Beispiel aus der Praxis, das die Problematik als auch die Daseinsberechtigung des Lösungsmusters aufzeigt.
- *Problemraum*: Die Situationen bzw. der Kontext, in denen das Muster anwendbar ist.
- *Problem*: Eine Beschreibung des Problems, welches das Lösungsmuster lösen hilft.
- *Lösung*: Das grundsätzliche Lösungsprinzip des Lösungsmusters.
- *Struktur*: Bildliche Darstellung Struktur des Lösungsmusters einschließlich Diagramme der UML.
- *Dynamische Aspekte*: Typische Szenarien, die das Verhalten des Musters zur Laufzeit beschreiben. Diese Szenarien werden mit Hilfe von Diagrammen der UML veranschaulicht.

- *Implementierung*: Anweisungen und Richtlinien für eine Implementierung des Lösungsmusters. Diese Richtlinien stellen lediglich einen Vorschlag dar, sie sind keine zementierten Axiome. Die Implementierung sollte den jeweiligen Erfordernissen und den Grenzen der jeweiligen Implementierungssprache angepasst werden.
- *Beispieldlösung*: Diskussion aller wichtigen Aspekte für die Lösung des Beispiels, die noch nicht in den vorangegangenen Abschnitten zur Sprache gekommen sind.
- *Varianten*: Eine kurze Beschreibung der Varianten und Spezialisierungen des Lösungsmusters.
- *Anwendungen*: Beispiele für die Anwendung des Lösungsmusters aus praktischen Problemstellungen.
- *Implikationen*: Mögliche Vorteile, Nachteile, Risiken und Kosten einer Anwendung des Lösungsmusters.
- *Verweise*: Verweise auf Lösungsmuster, die ähnliche Probleme lösen, und auf Muster, die bei der Wiederverwendung und der dortigen Verfeinerung des gerade beschriebenen Musters dienlich sein können.

### 6.4.3 Lösungsmuster

Als übergeordnetes Architekturmuster wurde das sogenannte *Chunkboard-Muster* gewählt, das vor allem in solchen Problemfällen zur Anwendung kommt, wo keine exakten bzw. deterministischen Lösungsstrategien vorliegen. Vielmehr arbeiten mehrere Komponenten gemeinsam an der Lösung, indem diese unvollständige Annäherungslösungen erarbeiten und der Gemeinschaft zur weiteren Bearbeitung bzw. Komplettierung zur Verfügung stellen. Insofern verfügen die einzelnen Komponenten nicht über das Wissen des vollständigen Problemsbereiches, sondern lediglich über Ausschnittswissen über den Problembereich. Erst die Gesamtheit der Komponenten ist in der Lage, den Problem- und Lösungsbereich abzudecken. Dieses Architekturmuster entspricht also von seinem Ansatz her dem des Kognitionsmodells, indem sich der kognitive Prozess als Orchestrierung unterschiedlicher Teilkomponenten ergibt, die unterschiedliche Kompetenzen besitzen und demnach unterschiedliche Teilprobleme lösen. Die Lösungen dieser Teilprobleme erfordern unter Umständen die Bearbeitung unterschiedlicher Repräsentationen und das Verfolgen unterschiedlicher Lösungsparadigmen. Erschwerend kommt hinzu, dass sich Situationen ergeben können, wo sich keine übergeordnete Strategie angeben lässt, wie die Teillösungen zu einer Gesamtlösung zusammengeführt werden können. Insofern kann in diesen Fällen auch keine genaue Folge der Lösungsschritte festgelegt werden. Zusätzlich kann die Verarbeitung von unsicherem, unscharfen und vagen Wissen notwendig sein, was dazu führt, dass unter Umständen die einzelnen Komponenten nicht nun mehrere Alternativlösungen erarbeiten, sondern diese Alternativen sogar in Konkurrenz zueinander stehen. Eher erleichternd hingegen wirkt sich dagegen der Umstand aus, dass in vielen Fällen keine optimale Lösung gefunden werden muss und man mit einer supoptimalen Lösung „gut fahren“ kann. Wichtig ist in diesem Zusammenhang ist, dass das System seine Grenzen kennt. Zusammengefasst ergibt sich die Eignung des Chunkboard-Muster als übergeordnetes Architekturmuster aus folgenden Gründen:

- Eine vollständige Sichtung des Problem- und Lösungsraumes ist nicht möglich.
- Eine vollständige Transparenz bezüglich des Problemraumes ist nicht gegeben, wozu das Problem in Teilprobleme zerlegt werden muss.
- Eine exakte Reihenfolge der Abarbeitung des Problems kann nicht angebenen werden, vielmehr müssen unterschiedliche Komponenten und deren Algorithmen in locker gekoppelter Form zusammenarbeiten.
- Die Komponenten sind voneinander unabhängig und arbeiten hierarchiefrei an der Gesamtlösung.
- Die Erarbeitung der Gesamtlösung erfordert die Berarbeitung unterschiedlicher Repräsentationen und den Einsatz unterschiedlicher Algorithmenparadigma.
- Dennoch arbeiten die Komponenten auf Basis eingereicherter Lösungen anderer Komponenten.
- Eine Parallelisierung der Lösungsschritte kann sinnvoll, nützlich und damit eine Sequentialisierung ausgeschlossen sein.
- Es gibt unsicheres, unscharfes, vages und sich widersprechendes Wissen.
- Der Einsatz von Technologien des Cognitive Computings kann notwendig bzw. unabdingbar sein.

Konkret wird dieses Muster als Sammlung unabhängiger Komponenten realisiert, die miteinander auf einer gemeinsamen Datenstruktur (Perzept) arbeiten. Jede dieser Komponente kommt eine bestimmte mentale Rolle zu und ist damit auf eine bestimmte Aufgabe spezialisiert. Diese spezialisierten Komponenten sind dabei völlig unabhängig voneinander, dahingehend entkoppelt, dass sie weder gegenseitig aufrufen noch in einer bestimmten Reihenfolge aktiviert werden. Vielmehr entscheidet der Erkenntnisfortschritt, die das kognitive System nimmt, die Aktivierung der einzelnen Komponenten. Insofern fungiert das kognitive System als übergeordnete Steuerungskomponente, die den aktuellen Erkenntniszustand bewertet, die beteiligten Komponenten von diesem Zustand informiert und den Lösungsprozess sozusagen moderiert. Während des Lösungsprozesses arbeitet das kognitive System mit unvollständigen Annäherungslösungen, die im Laufe des kognitiven Prozesses erweitert, verändert, oder gar wiederufen werden können. Die Menge der Lösungen spannen den Lösungsraum, der durch sogenannte Kognitionsbögen unterteilt wird. Den untersten Bereich dieses durch Spannungsbögen unterteilten Raum stellt die unbehandelte Eingabe dar. Je nach „kognitiver Veredelung“ dieser Eingabe überwindet das Lösungsangebot die einzelnen Kognitionsbögen, bis hin zum höchsten Bereich der optimalen Lösung.

Das ChunkBoard ist das zentrale Perzept, dass alle Daten und Informationen des Lösungsraumes speichert und diese den einzelnen Komponenten zur Verfügung stellt. Dieses Chunkboard stellt hierfür eine Schnittstelle bereit, die allen am kognitiven Prozess beteiligten Komponenten einen lesenden und schreibenden Zugriff auf das Chunkboard ermöglicht. Insofern stellt dieses Chunkboard mit seinen Elementen sozusagen einen Hypothesenraum dar, in dessen Grenzen einzelne Hypothesen erzeugt, ausgebaut oder gelöscht werden. Das Chunkboard wird bei der Initialisierung mit den rohen Eingabedaten versorgt und während des kognitiven Prozesses angereichert bis hin zur optimalen Ausgabe. Die unabhängigen Komponenten des Kognitionssystems kommunizieren nicht direkt mit-

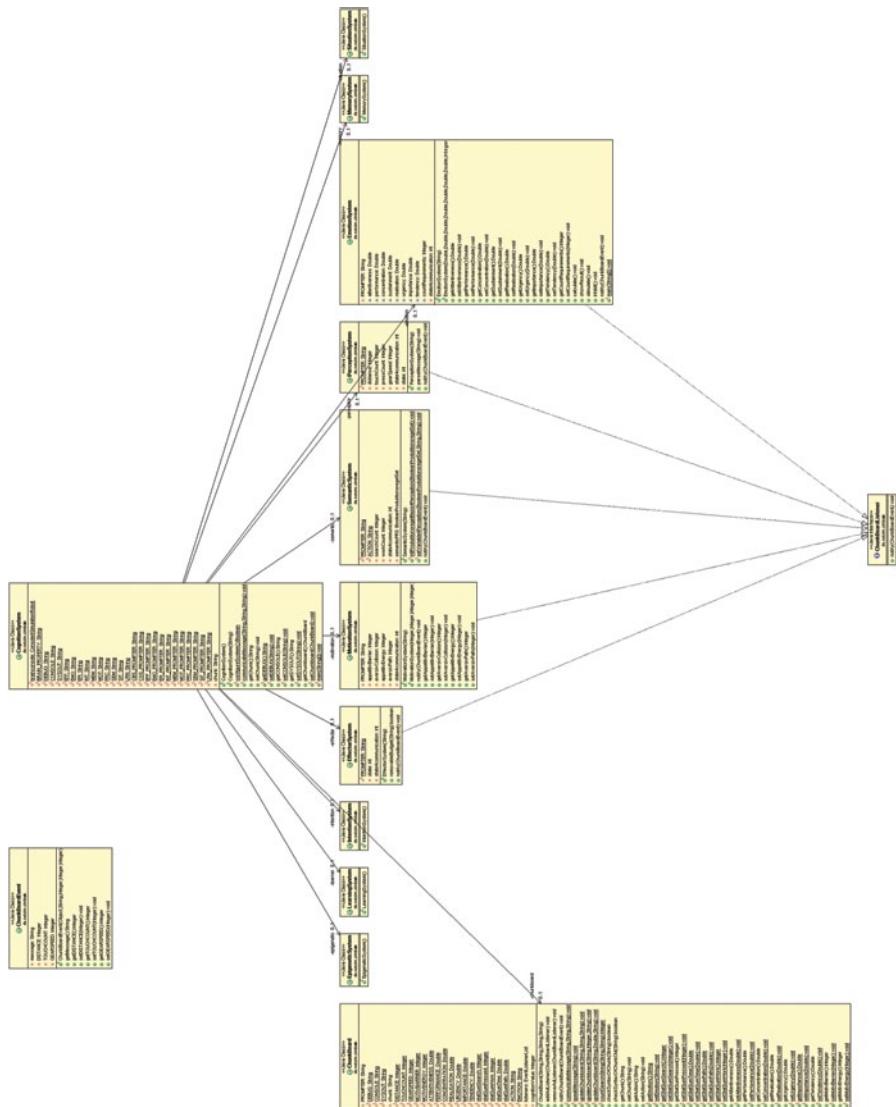
einander, sondern lesen und schreiben ausschließlich auf das ChunkBoard. Dabei verfügt jede Komponente über das Wissen, welchen Beitrag sie zur Gesamtlösung beizutragen hat. Insofern ist jede Komponente mit einem Produktionsregelsystem ausgestattet, dass im Bedingungsteil den aktuellen Zustand des ChunkBoards evaluiert und je nach Ausgang die im Aktionsteil vorgesehenen Aktionen ausführt.

Im Rahmen der architektonischen Festlegungen müssen auch Überlegungen bezüglich der Kommunikation zwischen den beteiligten Komponenten, sowie des Gesamtsystems zu seiner Umwelt angestellt werden. Die Ergebnisse dieser Überlegungen führen dann zu einem sogenannten *Strukturmuster*, dass die Struktur der Beziehungen der Komponenten und deren Kommunikationsverhalten festlegt. Insofern basieren diese Entwurfsmuster auf dem übergeordneten Architekturmuster, können jedoch letzteres in seiner Ausprägung mehr oder weniger stark beeinflussen. Für die Lösung des vorliegenden Problems wurde ein *Emergenz-Muster* gewählt, das gemäß dem Motto „Das Ganze ist mehr als die Summe seiner“ Komponenten zu einer semantischen Komponenten zusammenfasst. Dabei gilt es zu beachten, dass eine Vorgabe eines Strukturmusters nicht unbedingt die Zerlegung einer Funktionalität in einzelne Komponenten strikt vorgibt. Eine solche Zerlegung orientiert sich vielmehr an den jeweiligen Anforderungen des Problem- bzw. Anwendungsfalls und damit des Problem- und Lösungsraumes. Auf jeden Fall gibt ein vorgeschlagenes Strukturmuster Hinweise, wie die Beziehungen zwischen den beteiligten Komponenten zu implementieren und welche Aufgaben den beteiligten Komponenten zuzuordnen sind. Das Emergenz-Muster nun kapselt die einzelnen Komponenten und steuert die Kooperation dieser Komponenten so, dass sich aus deren Zusammenspiel eine semantische Komponente mit einer zugesicherten Gesamtfunktionalität ergibt. Insofern zeigt die zusammengesetzte Komponente ein Verhalten, dass sich unter Umständen nicht einfach aus der Existenz der Einzelkomponenten ergeben würde. Die Kombination der einzelnen Komponenten erzeugt vielmehr ein neues Verhalten oder eine neue Eigenschaft, zeigt ein neues Verhalten und damit Emergenz. Wichtig dabei ist, dass die Gesamtkomponente von außen über definierte Schnittstellen angesprochen und der direkte Zugriff auf ihre Einzelkomponente verboten wird.

#### 6.4.4 Architektur

Die einzelnen Bestandteile des kognitiven Systems, sowie deren Beziehungen zueinander, zeigt das folgende Klassendiagramm (Abb. 6.16).

Die zentrale Moderations- und Steuerungskomponente CognitionSystem beobachtet den Erkenntniszustand bzw. die Veränderungen auf dem ChunkBoard und entscheidet, welche der registrierten Komponenten zur Bearbeitung des ChunkBoards aufgefordert werden. Insofern koordiniert CognitionSystem zwar die Aufrufe der einzelnen Komponenten gemäß einer verfolgten Strategie, jedoch bilden die Daten und Informationen des ChunkBoard als auch die Kompetenzen der jeweiligen Komponenten die Grundlage für den Erkenntniszuwachs. Auch bleibt es der Komponente CognitionSystem überlassen, die



**Abb. 6.16** Klassendiagramm des kognitiven Systems (Brainware)

Bearbeitung abzubrechen und den erarbeiteten Erkenntniszustand als endgültiges Ergebnis an das übergeordnete Ziel- bzw. Trägersystem weiterzureichen. Dabei wird derzeit das Erreichen dieses Zeitpunktes davon abhängig gemacht, inwieweit eine angemessene Hypothese gefunden, der Speicherplatz erschöpft oder ob eine kritische Berechnungszeit überschritten ist.

Insgesamt ergibt sich damit folgende Dynamik des kognitiven Prozesses:

1. Die zentrale Komponente CognitionSystem wird vom Trägersystem initiiert, indem über die Sensoren das perzeptive System mit den entsprechenden Signalen oder Daten versorgt wird.
2. CognitionSystem aktiviert über die Methode nextComponent() die gemäss der Lösungsstrategie vorgesehene mentale Komponente, die möglicherweise etwas zum Erkenntniszuwachs beisteuern kann.
3. Die aktivierte Komponente liest den Inhalt des ChunkBoards aus und bestimmt über deren inhärentes Produktionsregelsystem, ob ein Beitrag geleistet werden kann. Sollte dies möglich sein, wird dieser Beitrag geleistet und das Ergebnis auf das ChunkBoard über die update()-Methode gestellt. Sollte kein Beitrag möglich sein, wird der mentale Prozess der Komponente abgebrochen und CognitionSystem darüber informiert.
4. CognitionSystem überprüft die Abbruchbedingung bezüglich des kognitiven Prozesses und wählt ggf. daraufhin eine weitere mentale Komponente entsprechend dem Erkenntnisstatus von ChunkBoard aus.

Bezüglich der Implementierung dieses Architekturmusters müssen folgende Schritte durchlaufen werden:

1. Das zu lösende *Problem* bzw. der komplette Problemraum gilt es inhaltlich zu erfassen bzw. zu beschreiben.
2. Es sind alle *Sensoren* und deren möglichen Ergebnistypen zu erfassen. Insbesondere betrifft dies die Datentypen, deren Strukturen, Bereiche und Qualität. Moderne Sensoren führen beispielsweise eine Vorverarbeitung dahingehend durch, dass nur entscheidungsrelevante Daten angeliefert werden. Gegebenenfalls müssen Mechanismen wie Complex Data Processing bzw. Complex Event Processing vorgeschaltet werden, um den Datenumfang bzw. deren Komplexität beherrschbar zu gestalten.
3. Ebenso sind alle *Aktoren* und deren möglichen Ergebnistypen zu erfassen und entsprechend aufzubereiten. Hier können zusätzliche Aspekte wie Korrektheit bzw. Robustheit eine Rolle spielen, indem die Ausgabe über das effektorische System diesen Aspekten Rechnung tragen muss.
4. Eventuelle *Eingriffsmöglichkeiten* von außen oder manuelle Interaktionen müssen überdacht werden, wobei der Architekturansatz des ChunkBoards zunächst davon ausgeht, dass das CognitionSystem als autonomes System auch ohne solche externen Eingriffsmöglichkeiten auskommt.

5. Nunmehr gilt es den *Lösungsraum* weitestgehend zu erfassen und zu beschreiben. Dieser Raum kann dabei auch Näherungs-, Zwischen- bzw. Teillösungen vorsehen. Gegebenenfalls lassen sich Beziehungen zwischen diesen Teillösungen und der idealtypischen Gesamtlösung finden, was sich unter Umständen auf die verfolgte Lösungsstrategie auswirken kann.
6. Diese *Lösungsstrategie* ist zu formulieren und das dazu notwendige Wissen hinsichtlich Art und Qualität zu spezifizieren. In diesem Schritt ist auch ein Bewertungsmaßstab bezüglich des Erkenntniszustandes zu definieren. So kann dies beispielsweise durch eine Zahl auf einer Skala zwischen 0 und 100 bzw. zwischen 0.1 und 1.0 ausgedrückt werden. Ein hoher Erkenntniswert wäre dann erreicht, wenn ein gewisser Schwellenwert 85 bzw. 0.85 überschritten wird.
7. Sowohl das Wissen als auch die Schritte der Lösungsstrategie sind den einzelnen *mentalen Prozesse* zuzuordnen und somit die Rolle bzw. die Funktion der hierfür verantwortlichen Komponenten verbindlich festzulegen.
8. Auf Basis dieser Vorarbeiten und Überlegungen kann das *ChunkBoard* spezifiziert werden, indem die Repräsentation der Sensorik und Aktorik als auch der notwendigen Ergänzungen sich in dieser Struktur niederschlagen muss. Eventuell müssen ontologische Transfermechanismen zwischen den einzelnen Repräsentationen vorgesehen werden, was als Funktion des ChunkBoard anzusehen ist. Je mehr dieser Unterstützungsfunktionen innerhalb des ChunkBoard gekapselt werden können, desto flexibler lässt sich die Partizipation der einzelnen Komponenten an dem kognitiven Prozess realisieren.
9. Es ist das *CognitionSystem* zu spezifizieren, indem die vorgesehene Lösungsstrategie bzw. Alternativen zu dieser implementiert werden. Dabei ist darauf zu achten, dass das CognitionSystem die Inhalte des ChunkBoard lesen und entsprechend verarbeiten kann. Beispielsweise ist auch das ChunkBoard mit einem Produktionsregelsystem auszustatten, dass je nach Erkenntnisszustand des ChunkBoards die Bedingungen aller Komponenten auswertet und dementsprechend die Aktivierungsfolge der Komponenten steuert. Aber auch die Aktivierung per Zufall oder die „Fitness“-getriggerte Aktivierung kann ein adäquates Mittel darstellen. Insofern kommt der Entwicklung einer geeigneten Lösungsstrategie eine zentrale Bedeutung zu und wird sich als der Garant für die Problemlösung auszeichnen.
10. Zuguter Letzt ist sich nochmals den einzelnen Komponenten des mentalen Prozesses zuzuwenden, um dort die notwendigen *Daten-Informations- oder Wissensverarbeitungsprozesse* durch die entsprechenden Cognitive Computing-Technologien auszuimplementieren. Unter Umständen kann es erforderlich sein, innerhalb einer Komponente je nach Wissensart oder Wissensqualität gleich mehrere Technologien (Produktionsregelsystem, Neuronale Netze, Fuzzy Systeme, Evolutionäre Algorithmen etc.) implementieren zu müssen.

Das folgende RobotermodeLL teilt den Roboter in mehrere, interagierende Teilsysteme ein, wobei grob betrachtet zwischen dem perzeptorischen, motorisch-effektorischen und dem

dem kognitiven System unterschieden wird. Diese Teilsysteme ergeben in ihrer Zusammenarbeit als Ganzes das kognitive System, welches die Interoperationen mit der Umwelt im Allgemeinen und mentale Fähigkeiten wie Wahrnehmung, Motorik, Entscheidungskompetenz und Lernmechanismen im Speziellen realisiert. Insgesamt soll damit das kognitive System die Fähigkeiten des Schlussfolgerns, des Problemlösens, des Lernens bzw. den Erwerb und Nutzung von Fertigkeiten sicherstellen. Dabei fungiert die Komponente CognitionSystem als Gesamtkomponente, die die Sammlung der mentalen Komponenten repräsentiert. Es ist über die Schnittstelle von CognitionSystem sichergestellt, dass nur dessen Dienste nach außen hin sichtbar sind. Die nunmehr folgenden Komponenten resultieren aus der folgenden Vorgehensweise, die sich aus der Orientierung am Emergenzmuster ergeben.

1. Zunächst gilt es die öffentliche Schnittstelle der Gesamtkomponente zu entwerfen. Dabei orientiert man sich den Funktionalität, die diese Komponente dem Trägersystem des kognitiven Systems anbieten muss.
2. Die Gesamtfunktionalität wird in kapselbare Teilfunktionalitäten aufgeteilt. Diese Aufteilung basiert in der Regel auf einer Theorie oder eines aus einer Theorie resultierenden Modells. In diesem Fall liegt das in diesem Buch vorgestellte Kognitionsmodell der Zerlegung zugrunde. Ebenso muss dem Prinzip der Wiederverwendung entsprochen werden.
3. Nach der Zerlegung analysiert man den bestehenden Fundus (Bibliotheken, APIs, Repositories, etc.), ob im Rahmen der Wiederverwendung auf bereits entwickelte Komponenten zurückgegriffen werden kann.
4. Nunmehr gilt es zu überprüfen, ob die Summe der zerlegten Teilfunktionalitäten auch wirklich die erforderliche Gesamtfunktionalität ergibt. Ebenso gilt es festzulegen, ob eine von außen an die Gesamtkomponente gerichtete Funktionalität als Dienst an die Teilkomponenten weitergeleitet werden kann. In diesem Fall benötigt die Teilkomponente nicht den gesamten Problem- und Lösungskontext. In dem Falle, dass die Gesamtkomponente die Funktion als Dienst an die Teilobjekte delegieren muss, sind die Teilkomponenten mit dem Problem- und Lösungskontext zu versorgen. Es kann auch erforderlich sein, dass die Teilkomponenten miteinander kooperieren und damit interagieren, um die an sie delegierten Funktionen zu erfüllen.
5. Danach sind die Teilkomponenten zu implementieren, d. h. in diesem Fall werden die mentalen Systeme des Kognitionsmodells ausimplementiert.
6. Nunmehr ist die Gesamtkomponente und deren Dienste zu implementieren und durch entsprechende Integrationstest sicherzustellen, dass die erwartete Gesamtfunktionalität zur Verfügung steht.

Der Registrierungsmechanismus von CognitionSystem stellt sicher, dass die Lebenszyklen der Teilobjekte verwaltet werden können. Dies ist insbesondere deshalb wichtig, da die Teilkomponenten nicht nur von der Hauptkomponente „ins Leben gerufen“ werden, sondern auch mit der Zerstörung von CognitionSystem ihre Daseinsberechtigung einbüßen

und ebenfalls „sterben“. Das bedeutet, dass CognitionSystem zuständig ist für das Anlegen und Zerstören der mentalen Komponenten und damit die Verwaltung dieser Komponenten übernimmt.

In Anlehnung an das aus der Softwareentwicklung bekannte Publisher-Subscriber-Muster ist mit der Registrierung auch die Festlegung verbunden, in welcher Art und Weise die mentalen Komponenten sich mit dem Zustand des ChunkBoard synchronisieren. Insofern ist mit der Registrierung eine Art von Abonnement verbunden, das festlegt, ob die Synchronisation in Form eines Push- oder Pull-Verfahrens erfolgt. Beim Push-Verfahren sendet CognitionSystem den geänderten Zustand des ChunkBoards an die registrierten Komponenten. Dies bedeutet, dass die mentalen Komponenten keinen Einfluss darauf haben, ob und wann sie von dieser Zustandsänderung erfahren. Hingegen fragen beim Pull-Verfahren die mentalen Komponenten beim ChunkBoard nach, ob eine Zustandsänderung vorliegt und entscheiden dann selbst, ob sie das ChunkBoards bearbeiten möchten.

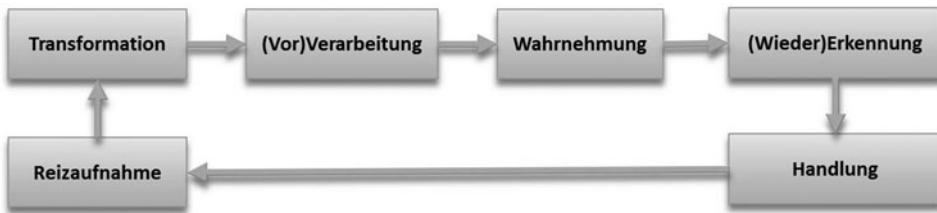
Ebenfalls obliegt es CognitionSystem, eine eventuell erforderliche multimodulare Redundanz in die Wege zu leiten, wenn mehrere mentale Komponenten von einander unabhängig Ergebnisse erarbeiten müssen, um diese dann in ein Gesamtergebnis überführen zu können. So kann es beispielsweise vorgesehen sein, dass nur dann ein verwertbares Ergebnis vorliegt, wenn eine Mindestanzahl unterschiedlicher mentaler Komponenten das gleiche Ergebnis liefert.

## Kognitives System

Aufgrund der Annahme, dass Kognition als ein Berechnungsprozess beschreibbar ist, gehört die Architektur des Verarbeitungssystems zu den zentralen Ausgestaltungsmöglichkeiten des kognitiven Systems. Dem Paradigma der Objektorientierung folgend, besteht die Architektur im Rahmen der Implementierung aus elementaren Klassen und Methoden, aus denen sich das kognitive System konstituiert bzw. die den darin ablaufenden kognitiven Prozessen zugrunde liegen.<sup>4</sup> Sie ermöglichen es einem solchen System, Daten, Informationen und Wissensinhalte in Form von Strukturen zu speichern und anzuwenden, um Ziele zu erreichen. Dazu gehören Methoden für die Interoperation mit der Umgebung unter Echtzeitbedingungen, für den Erwerb und die Repräsentation von Wissen und für die Auswahl von Alternativen an Entscheidungspunkten. Diese Klassen und Methoden spezifizieren die vergleichsweise stabile Architektur eines Systems, die zwischen Problemdomänen und Aufgaben- bzw. Problemklassen invariant bleibt. Je nach Anwendungsfall bzw. Problemstellung kann dieses Muster im Detail ausimplementiert werden.

Das Kognitionssystem (CognitionSystem) beruht auf der Gegenüberstellung eines perzeptiven Systems (Wahrnehmungssystem) und eines effektorischen Systems (Handlungs- bzw. Verhaltenssystem) einer Außenwelt (Umgebung). Der kognitive Prozess besteht aus Statusübergängen, die jeweils auf ihren Folgestatus Einfluss ausüben und an jeder Art von Wahrnehmung und Verhalten in genau dieser Reihenfolge beteiligt sind (Abb. 6.17).

<sup>4</sup> Siehe auch Jacobson et al. 2003



**Abb. 6.17** Geschlossene Prozesskette des kognitiven Systems

In der folgenden Variante ist der kognitive Prozess in sich geschlossen, d. h. der letzte Status beeinflusst wiederum den ersten Status des Prozesses. Allerdings sind auch andere Prozessvarianten durchaus denkbar.

Folgende Prozessschritte werden dabei durchlaufen:

- *Reizaufnahme*: Die Objekte in der Außenwelt emittieren Signale. Ein solches Signal in Form einer physikalisch messbaren Größe, das auf Eigenschaften des Objektes beruht, wird als „distaler Reiz“ bezeichnet.
- *Transformation*: Ein distaler Reiz trifft auf Sensoren bzw. Rezeptoren, wo er durch Interaktion mit diesen zum proximalen Reiz wird. Sensoren und Rezeptoren sind spezialisierte Komponenten des Systems, die durch bestimmte Stimuli erregt werden. Sie wandeln im Rahmen einer Transformation verschiedene Arten von Energie (z. B. Licht, Schall, Druck) in Spannungsänderungen um.
- *Verarbeitung*: In den Sensoren oder Rezeptoren selbst findet oft eine massive Vorverarbeitung (Filterung, Hemmung, Konvergenz, Divergenz, Integration, Summation, etc.) der empfangenen Signale statt, besonders aber in allen folgenden Teilsystemen des kognitiven Systems.
- *Wahrnehmung*: Der nächste Schritt ist die bewusste Wahrnehmung zur Entwicklung des Perzepts.
- *Wiedererkennung*: Teilprozesse, wie Erinnern, Kombinieren, Erkennen, Assoziieren und Urteilen führen zum erfassenden Verständnis des Wahrgenommenen und bilden die Grundlage für Reaktionen auf den distalen Reiz.
- *Handlung*: Letztendliches Ergebnis des kognitiven Prozesses ist die adäquate Reaktion auf und die Interoperation mit der Umwelt. Diese Reaktion und Interoperation als Verhalten ist damit Bestandteil des kognitiven Prozesses, da diese darauf abzielen, den nächsten Durchlauf der Prozesskette zu beeinflussen, indem neue Eigenschaften der Umwelt dem Prozess wieder zugänglich gemacht werden.

Der kognitive Prozess arbeitet vertikal, d. h. zwischen einem Reiz und seiner Repräsentation im Kognitionssystem besteht ein kausaler, nachvollziehbarer Zusammenhang. Ist ein Glied der Prozesskette gestört, kann es zu Widersprüchen zwischen dem Reiz und der durch ihn ausgelösten Interoperation kommen, was einen gestörten kognitiven Prozess induziert. Diese Störung gilt es dann durch passende Korrekturmechanismen zu beheben.

Aufgrund des gewählten Strukturmusters in Form des Emergenz-Musters dient CognitionSystem als Gesamtkomponente, welches die nun folgenden Komponenten in sich kapselt. Dabei wird verhindert, dass von außen kein direkter Zugriff auf diese einzelnen Komponenten möglich ist. Durch eine entsprechende Schnittstelle von CognitionSystem wird gewährleistet, dass diese Schnittstelle als einziger Zugang auf die Funktionalität der einzelnen Komponenten fungiert. CognitionSystem mit seinen in sich gekapselten Komponenten erscheint damit nach außen hin als Gesamtkomponente. Dabei gilt es sicherzustellen, dass die Komponenten eher locker an die Gesamtkomponente gebunden sind, d. h. die Konstitution bzw. die Zusammensetzung dieser kann sich je nach Problemstellung ändern, insofern Komponenten aktiviert oder deaktiviert sein können. Wichtig dabei ist, dass die Komponentenbindung an das Gesamtobjekt durch diese Problemstellung (Situation des Robotersystems, Anwendungs- bzw Problemfall, etc.) bedingt ist und damit auch von dem über das ChunkBoard ausgedrückten Erkenntniszustand eingefordert wird.

## Perzeptives System

Das perzeptive System (PerceptionSystem) umfasst den Vorgang der Einreichung einer Gesamtheit von Reizen (Stimuli) der Umwelt und den inneren Zuständen des Systems. Es inkludiert auch das Filtern und Zusammenführen von Teil-Informationen zu sinnvollen Gesamteindrücken. Diese werden auch Perzepte genannt und laufend mit den als innere Vorstellungswelt repräsentierten Konstrukten oder Schemata abgeglichen. Der Umfang und die Qualität aller Einreichungen orientieren sich an den Möglichkeiten der Sensoren und Rezeptoren. Die Inhalte und Qualitäten des perzeptiven Systems können durch gezielte Steuerung der Aufmerksamkeit und durch Wahrnehmungsstrategien beeinflusst werden.

Grundsätzlich unterscheidet das kognitive System zwischen der Extero- und der Interozeption. *Exterozeption* bezeichnet dabei allgemein die Wahrnehmung der Außenwelt, wohingegen der Begriff *Interozeption* als Oberbegriff die Wahrnehmung des eigenen Systems über die Reflexion steht. Bei letzterem unterscheidet man *Propriozeption* als die Wahrnehmung von Systemlage und -bewegung im Raum und *Viszerozeption* als die Wahrnehmung von intrinsischen Teilsystemen.

Die Wahrnehmung der Außenwelt über das perzeptive System bezieht sich je nach Sensor- oder Rezeptortyp auf die „fünf Sinne“ (Riechen, Sehen, Hören, Schmecken und Fühlen). Das Fühlen (Tastsinn) wiederum kann einerseits nach der Wahrnehmung von Berührung, Systemanomalien und Temperatur (Oberflächensensibilität), andererseits aber auch in das aktive Erkennen (haptische Wahrnehmung) und das passive „kontakteert werden“ (Oberflächensensibilität) unterteilt werden.

Die Psychologie kennt die Begriffe der *Selbst-* und *Fremdwahrnehmung*, wobei erstere die Überzeugungen sind, die der Mensch von sich selbst beziehungsweise seinem Empfinden und Verhalten hat, während Fremdwahrnehmung die Eindrücke bezeichnet, die andere Menschen gewinnen.

Die Implementierung des perzeptiven Systems orientiert sich an den Konzepten der Psychologie und der Physiologie, wo Wahrnehmung die Summe der Schritte Aufnahme, Auswahl, Verarbeitung und Interpretation von sensorischen Informationen bezeichnet, und zwar nur jener Informationen, die der Adaptation durch Reaktionen oder Interoperationen des Systems an seine Umwelt dienen oder die dem System eine Rückmeldung über Auswirkungen des systemischen Verhaltens geben. Gemäß dieser Implementierung sind also nicht alle Reize wahrnehmungsrelevant, sondern nur diejenigen, die kognitiv verarbeitet werden und der Orientierung des Systems dienen. Wahrnehmung ermöglicht sinnvolles Handeln bzw. den Aufbau von mentalen Modellen der Welt und dadurch erfassendes und planerisches Denken. Die Wahrnehmung als Ergebnistyp des perzeptiven Systems bildet die Grundlage des kognitiven Prozesses im Allgemeinen und der Lernprozesse im Speziellen.

### **Semantisches und intentionales System**

Das artifizielle „Denken“ allgemein bezieht sich als Daten-, Informations- und Wissensverarbeitungsprozess auf systeminterne Vorgänge, die ausschließlich im kognitiven System präsent sind. Im Unterschied zur Wahrnehmung, die über die Situation informiert, dient diese Verarbeitung der inneren Vergegenwärtigung, die unabhängig von Raum und Zeit stattfindet. Die in der Wahrnehmung schon angesprochene Ordnungsleistung des Systems wird im semantischen und dem intentionalen System fortgesetzt, es werden logische und funktionale Zusammenhänge hergestellt, Begriffe und Sachverhalte zu Perzepten abstrahiert und Entscheidungsmöglichkeiten berechnet und verglichen.

Das artifizielle Denken kommt im Kognitionsmodell damit gleich zwei Stellen (SemanticSystem, IntentionSystem) und dort in unterschiedlicher Ausprägung vor: einmal im aufnehmenden Strang (erfassendes Denken), wozu Erfassen, Verstehen, Interpretieren, Klassifizieren und Folgern zählen, und zum anderen im einwirkenden Strang (planendes Denken), wozu Planen, Problemlösen und Entscheiden zählen.

Eine weitere Differenzierung unterteilt das artifizielle Denken in fünf Operationen:<sup>5</sup>

- Das Gedächtnis als reproduktive Leistung des Erinnerns von sensorischem Material des perzeptiven Systems.
- Erfassendes Denken bezieht sich hierbei auf das Erkennen und Wiedererkennen von Informationen.
- Wertendes Denken bezeichnet die Beurteilung von etwas Erfasstem oder Produziertem nach bestimmten Gütekriterien.
- Konvergierend-produktives Denken ist problemlösendes Denken, das auf das Finden der einen richtigen Lösung abzielt.
- Divergierend-produktives Denken ist auch problemlösendes Denken, wobei aber nicht von einer richtigen Lösung auszugehen ist, sondern verschiedene Richtungen mit verschiedenen Lösungen parallel gültig sind. Gemeinhin wird dieses Denken auch als kreatives Denken bezeichnet.

---

<sup>5</sup> vgl. Nolting und Paulus, S. 49 f.

Im Rahmen der Implementierung werden die Operationen des Erkennens, Wiedererkennens und der Beurteilung dem erfassenden Denken (*SemanticSystem*) und die Operationen des Findens und die kreative Komponente dem planenden Denken (*IntentionSystem*) zugeordnet. Die Gedächtnisoperationen werden in der beschriebenen Form, je nach Inhalt, in beiden Systemen implementiert.

Es ist damit auch möglich, das artifizielle Denken aufgrund des Komplexitätsgrades zu unterscheiden und so eine Hierarchie unter den Denkleistungen zu bestimmen: Wissen – Verstehen – Anwenden – Analyse – Synthese – Evaluation.

Wie schon in der Wahrnehmung, gibt es auch im menschlichen Denken den Prozess des über sich selbst Nachdenkens, was als *Metakognition* bezeichnet werden soll. Diese Metakognition wird im epigenetischen System realisiert.

Das artifizielle Denken wird auch durch die agentenindividuellen Dispositionen beeinflusst. So unterscheiden sich die Agenten in ihren kognitiven und kreativen Fähigkeiten, sie verwenden bei Problemen, je nach Erfahrungs- und Entwicklungsstand, unterschiedliche kognitive Leistungen. So kann der Lösungsweg, den ein Agent beschreitet, wenn kein spezifischer Wissensbestand vorliegt, sich aufgrund der Dispositionen erheblich von dem anderer Agenten unterscheiden.

## **Emotionales System**

Die artifizielle *Emotion* (*CognitionSystem*) ist ein Informationsverarbeitungsprozess, der durch die direkte und/oder indirekte Wahrnehmung eines Objekts über das perzeptive System oder einer Situation über das situative System ausgelöst wird und mit Zustandsveränderungen, spezifischen Kognitionen, Schwächung oder Verstärkung der Persönlichkeitsmerkmale und einer Veränderung der Verhaltensbereitschaft einhergeht. Die aus einer Emotion folgende Interpretation eines Objekts oder einer Tatsache bezeichnet man als artifizielles „Gefühl“. Damit ergibt sich die artifizielle *Emotionalität* als der Sammelbegriff für die individuell unterschiedliche Eigenart des Gefühlszustands eines Agenten sowie die Verhaltenssteuerung und die Art der Verarbeitung dieses Verhaltens. Als artifizielle *Emotivität* ist die ausgeprägte Erregbarkeit des Agenten zu verstehen.

Im Vergleich zu menschlichen Stimmungen sind die artifiziellen Emotionen relativ kurz und intensiv. Während menschliche Stimmungen und deren Auslöser oft unbemerkt bleiben, sind bei den artifiziellen Emotionen das auslösende Moment als Situation und die epigenetische Komponenten im Fokus der Verarbeitung.

Während menschliche Emotionen stets auf etwas gerichtet sind (z. B. Wut auf jemanden, Trauer um jemanden), richten sich Stimmungen auf kein spezifisches Objekt, d. h. man kann z. B. gereizt sein ohne einen bestimmten Grund oder Ereignis. Ähnlich, wie meist nur kurzzeitige Gefühlseindrücke, vermögen Stimmungen die Wahrnehmung „einzufärben“ und man erlebt die Wirklichkeit durch eine „Gefühlsbrille“.

Betreffen Emotionen im Rahmen der Simulation Verhaltensintentionen oder lösen sie Verhalten aus, die nicht mehr oder in geringerem Maße kontrollierbar sind, dann spricht man von artifiziellen Affekten. Die Emotionen als Ergebnis dieses Systems spielen bei der Motivation dank deren Wertcharakters eine wichtige Rolle als verhaltensaushörendes Moment.

Während beim menschlichen Gefühl der kognitive Aspekt durchaus fehlen kann – so muss man beispielsweise um Schmerz zu fühlen, nicht unbedingt verstehen, was passiert –, beinhaltet Emotionen immer auch irgendeine Art von Verständnis. Dies trifft auch auf artifizielle Affekte zu, die mit einem Werturteil wie „richtig“ oder „falsch“ verbunden sind.

Nicht aus jeder Emotion erfolgt auch ein Verhalten oder eine Verhaltensbereitschaft, da diese zur Reduktion einer Ist-Soll-Diskrepanz führen soll und bei eingetretenem gewünschtem Ereignis der Ist-Soll-Zustand schon erreicht wurde. Umgekehrt aber stellt das Verhalten einen direkten oder indirekten Grund einer Emotion dar. Damit ist für ein artifizielles Verhalten sowohl das *inhaltliche Ziel*, d. h. auf das, was das Verhalten bezogen ist, als auch die *emotionalen Gründe* des Verhaltens, welche auf die Fragen Warum oder Wozu abzielen, relevant. Das Verhalten resultiert also aus der Hoffnung, ein bestimmtes positives Erleben einer Emotion zu erhalten/verbessern oder im Umkehrschluss ein negatives Erleben zu vermeiden/reduzieren.

Mit der Funktion der Emotionsregulation werden im emotionalen System Prozesse realisiert, die der mentalen Verarbeitung emotionaler Zustände dienen.

## **Motivationales System**

Als artifizielle *Motivation* (MotivationSystem) wird das auf emotionaler Aktivität beruhende Streben des Agenten nach Zielen bezeichnet. Motivation steigert die Verhaltensbereitschaft und ist somit eine „Triebkraft“ für Verhalten. Die artifizielle Motivation veranlasst einen Agenten zu zielgerichtetem Verhalten.

So bezeichnet die Motivationspsychologie mit dem Begriff *intrinsische Motivation* das Bestreben, etwas um seiner selbst willen zu tun (weil es einfach Spaß macht, Interessen befriedigt oder eine Herausforderung darstellt). Bei der *extrinsischen Motivation* steht dagegen der Wunsch im Vordergrund, bestimmte Leistungen zu erbringen, weil man sich davon einen Vorteil (Belohnung) verspricht oder Nachteile (Bestrafung) vermeiden möchte.

Die Agenten können mit unterschiedlichen Motivationsmodellen ausgestattet werden. Dabei lassen sich Inhaltsmodelle von Prozessmodellen unterscheiden. Während Inhaltsmodelle Verhalten allein aufgrund bestimmter epigenetischer Inhalte erklären, führen Prozessmodelle das Verhalten auch auf bestimmte Vorgänge zurück. Diese *Inhaltsmodelle* beschäftigen sich mit Inhalt, Art und Wirkung von Motiven. Eine Taxonomie von Motiven wird implementiert und bestimmt, nach welchen Gesetzmäßigkeiten welche Motive verhaltensbestimmend werden. Die *Prozessmodelle* versuchen hingegen zu erklären, wie

Motivation formal und losgelöst von Bedürfnisinhalten entsteht und auf das Verhalten wirkt. Das Ziel des Verhaltens ist unbestimmt, aber der Agent will den erwarteten Nutzen maximieren. Im Rahmen der Implementierung wird ein Prozessmodell auf Basis eines Erwartungs-mal-Wert-Modells verfolgt.

Dies ist dadurch motiviert, dass seit der sogenannten kognitiven Wende Motivation oft als eine multiplikative Verknüpfung von Erwartung und Wert konzipiert wird. Diese Modelle werden auch als Erwartungs-Valenz-Modelle bezeichnet, wo der *Erwartung* die Wahrscheinlichkeit des Eintritts und der *Valenz* die Attraktivität eines Ziels entspricht.

Gemäß diesem Erwartung-mal-Wert-Modell geht Motivation auf die Erwartung bestimmter Verhaltensergebnisse sowie auf deren (positive oder negative) Bewertung zurück. Es ist damit ein Modell, das die Motivation, ein bestimmtes Verhalten auszuführen, durch die Erwartung, mit dem Verhalten eine bestimmte Konsequenz herbeizuführen und den Wert der Verhaltenskonsequenz, erklärt. Es basiert auf dem mathematischen Konstrukt des Erwartungswertes. Die Motivation M, ein bestimmtes Verhalten zu zeigen, ergibt sich aus dem Produkt vom Wert der Verhaltenskonsequenz (emotional oder rational) W und der Erwartung E, mit dem Verhalten, die erwünschte Konsequenz zu erzielen.

$$M = W \times E.$$

Motivational betrachtet kann somit ein hoher Wert geringe Erfolgsaussichten kompensieren, oder ein geringer Wert wird durch hohe Erfolgsaussichten kompensiert. Diese Basisannahme wird nun dahingehend erweitert, dass die Frage, ob ein Agent eine Verhaltensmotivation zeigt, nicht nur das Resultat seines epigenetischen Zustands, sondern vor allem von Faktoren der Situation abhängig ist. Insofern gilt es, durch die Erweiterung neben den eher inhaltlichen Aspekten vor allem der Dynamik der Motivation Rechnung zu tragen. Diese Erweiterung sieht vor, die Intention des Verhaltens aus der Wertigkeit der Ziele, der Orchestrierung der Verhaltensmöglichkeiten für das Erreichen dieser Ziele und der Wahrscheinlichkeit, dieses Verhalten auch zeigen zu können, zu errechnen. Dabei konzentriert sich das motivationale System speziell auf jene Prozesse, die zur Entscheidung für eine bestimmte Verhaltensalternative führen.

Dabei ist von zentraler Bedeutung, dass zwischen Erwartung und Orchestrierung unterschieden wird. Beide sind Einschätzungen des Agenten. So gilt für die *Erwartung* ein Verhaltens-Ergebnis-Zusammenhang: Die angenommene Wahrscheinlichkeit, dass durch den Einsatz die Voraussetzungen für das Verhaltensergebnis erfüllt werden wird. Diese Erwartung hängt überwiegend von den epigenetischen Merkmalen und damit den Fähigkeiten des Agenten ab. Für die *Orchestrierung* gilt ein Ergebnis-Folge-Zusammenhang: Die Wahrscheinlichkeit, mit der das Ergebnis der eigenen Leistung zu den erwünschten Verhaltensfolgen führt. Diese wird überwiegend durch das Verhalten anderer Agenten und übergreifenden Regeln beeinflusst.

Um eine motivationale Entscheidung zu treffen, sind drei Aspekte, Orchestrierung, Valenz und Erwartung zu berücksichtigen.

- *Orchestrierung* steht hier für die Beziehung zwischen dem Verhaltensergebnis und den daraus resultierenden Folgen, indem die Verhaltensergebnisse günstige oder ungünstige Auswirkungen haben können. Hinter dem Begriff der Orchestrierung verbirgt sich also zum einen die Frage, inwieweit das Ergebnis des Verhaltens wünschenswerte Konsequenzen nach sich ziehen kann, aber auch die Tatsache, dass optimale Einzelhandlungen in der Summe nicht automatisch optimales Verhalten ergeben muss. Ein Verhaltensergebnis kann also, bezogen auf verschiedene Verhaltensfolgen, gleichzeitig positive und negative Auswirkung haben.
- *Valenz* bezeichnet den Wert, den bestimmte Zustände für einen Agenten haben. Valenz gibt also den Grad an, in dem ein bestimmter Zustand für einen Agenten wünschenswert oder wichtig ist. Dabei bezieht sich die Valenz V auf das Verhaltensergebnis, die Valenz V' auf die Verhaltensfolgen. Denn die Valenz V des Verhaltensergebnisses E ergibt aus den Valenzen V' und der Orchestrierung O der Verhaltensfolgen F. Es wird postuliert, dass ein Agent für jede Verhaltensfolge die spezifische Orchestrierung des Verhaltensergebnisses kalkuliert. Wie bedeutsam diese Auswirkungen im Einzelnen für die Bewertung des Verhaltensergebnisses sind, hängt von der jeweiligen Valenz der Verhaltensfolge ab. Durch die Valenz einer bestimmten Verhaltensfolge erhält ihre Veränderung durch das Verhaltensergebnis mehr oder weniger Gewicht. Wie wünschenswert das direkte Verhaltensergebnis ist, ergibt sich so aus der Summe der gewichteten Orchestrierung aller Verhaltensfolgen.

$$\text{Valenz (E)} = \sum_{i=1}^n (V(F_i) * O(F_i))$$

- *Erwartung* beschreibt den Grad der wahrgenommenen Eintretens-Wahrscheinlichkeit eines Ergebnisses. Die Erwartung misst man auf einer Skala von 0 bis 1: Bei einer Erwartung von 0 hält der Agent das Auftreten eines Ereignisses infolge eines bestimmten Verhaltens für unwahrscheinlich, bei einer Erwartung von 1 wird das Auftreten eines Ereignisses für sicher gehalten.

Die Entscheidung eines Agenten darüber, ob er eine Verhaltensmöglichkeit in ein konkretes Verhalten überführt oder nicht, ergibt sich demnach aus dem Zusammenspiel von

$$\text{Entscheidung} = \text{Valenz (E)} \text{ Erwartung (E)}$$

Erwartungen und der Wertigkeit des Verhaltensergebnisses:

Der Agent wird demnach zu jener Verhaltensalternative greifen, die den höchsten F-Wert besitzt. Die multiplikative Verknüpfung von Valenz und Erwartung zeigt, dass beide As-

pekte ein Mindestmaß erreichen müssen, damit ein Agent bereit ist, das entsprechende Verhalten zu zeigen. Ein extrem wünschenswertes Ziel wird sich trotzdem nicht leistungssteigernd auswirken, wenn der Agent annimmt, dass dieses Ergebnis nicht zu erreichen ist. Umgekehrt werden Verhaltensziele, die sehr leicht zu erreichen wären, trotzdem nicht motivierend wirken, wenn sie keine positive Valenz besitzen. Mit anderen Worten kann man dies auch als einen Weg-Ziel-Ansatz (Path-Goal Approach) beschreiben. Demnach wird Leistung („Weg“) von Agenten nur dann als erstrebenswert angesehen, wenn damit ein erwünschtes Ziel erreicht werden kann. Entsprechend orientiert sich das Ausmaß an gezeigter Leistung an dem Aufwand, der notwendig ist, um dieses Ziel zu erreichen. Dieser Ansatz beruht auf dem Paradigma des Nutzenmaximierers, d. h. die Wahrnehmung eines „relativen Nutzens“ ist wesentlich ausschlaggebend für die Bereitschaft zur Leistungserbringung.

### **Epigenetisches System**

Das artifizielle *epigenetische System* (EpigeneticSystem) betrifft die Eigenschaften und die Merkmale des Agenten, die eine relativ überdauernde (zeitstabile) Bereitschaft (Disposition), die bestimmte Aspekte des Verhaltens des Agenten in einer bestimmten Klasse von Situationen beschreiben und vorhersagen sollen.

Vom Begriff der artifiziellen „Persönlichkeitseigenschaft“ abzugrenzen ist der Begriff des aktuellen *Zustandes* (state) eines Agenten, der über Situationen hinweg variiert. Ebenfalls nicht zu den Eigenschaften gerechnet werden Verhaltengewohnheiten (habit), also die erlernten Reaktionen auf spezifische Reize. Diese werden im Lernsystem entwickelt und entsprechend verwaltet. Die epigenetischen Eigenschaften in einem weiten Sinn umfassen somit alle softwaretechnologisch fassbaren individuellen Differenzen des Verhaltens sowie ihre systembedingten Grundlagen in der Individualität (Konstitution) des Agenten.

Im Rahmen der Implementierung ist mit Eigenschaft heute nicht ein direkt beobachtbares Verhalten oder ein feststehender Wesenszug gemeint, sondern eher eine *Disposition* im Sinne einer Verhaltensbereitschaft. So kann sich ein extrovertierter Agent in unterschiedlichen Situationen „gesellig“, „impulsiv“ und „lebhaft“ verhalten, in anderen Situationen zeigt sich diese Disposition nicht. Ob sich die Disposition auswirkt, hängt von den jeweiligen äußeren und inneren Bedingungen ab. Disposition als theoretisches Konstrukt beschreibt also die mehr oder minder große Wahrscheinlichkeit, dass sich der Agent in ähnlichen Situationen erneut so verhalten wird.

Wie ausgeprägt die individuelle Disposition ist, kann in gültiger und zuverlässiger Weise nur erschlossen werden, wenn mehrere miteinander zusammenhängende (konsistente) Indikatoren, wie Testfälle sowie verschiedene Situationen im Rahmen der Simulation berücksichtigt werden.

Daneben werden im epigenetischen System auch die metakognitiven Fähigkeiten berücksichtigt. Die *Metakognition* wird dahingehend realisiert, dass das System im epigenetischen

System über die jeweiligen Teilsysteme reflektiert und das Ergebnis dieser Reflexion im Verhältnis zu anderen Agenten oder verschiedenen Aufgabentypen bewertet. Die Metakognition kann durch das Erkennen und Steuern eigener geistiger Aktivitäten beide Stränge beeinflussen. Innerhalb des epigenetischen Systems bezeichnet Reflexion demnach auch eine bestimmte Form der Selbstreferenz des kognitiven Systems. Diese Selbstreferenz dient der autopoietischen Reproduktion, d. h. der Reproduktion des Systems aus sich selbst heraus. Konkret bedeutet Reflexion oder *Introspektion*, dass ein Agent als Programm bzw. eine Komponente als Bestandteil des Agenten seine eigene Struktur kennt und diese, wenn nötig, modifizieren kann.<sup>6</sup>

Reflexion ermöglicht es, bei objektorientierter Programmierung, beispielsweise zur Laufzeit Informationen über Klassen oder deren Instanzen abzufragen. Bei einer Methode sind das unter anderem deren Sichtbarkeit, der Datentyp des Rückgabewertes oder der Typ der Übergabe-Parameter. Die Umsetzung der Abfragemöglichkeiten ist sprachspezifisch.

### **Situatives System**

Im Rahmen des *situativen Systems* (SituationSystem) wird unter *Situation* die *Lage* oder *Position*, die Gebundenheit an Gegebenheiten oder Umstände, aber auch die systemtechnische Beschaffenheit bzw. Wirksamkeit einer definierten oder eingegrenzten Region oder eines Gebietes verstanden. Damit umfasst der Begriff der Situation die Rahmenbedingungen, vor die der Agent gestellt ist und die als konkrete Bedingungen die Möglichkeiten des Entscheidens und Verhaltens stellen und begrenzen. Demgegenüber beschreibt der Begriff *Lage* einen mehr objektiv vorhandenen Zusammenhang.

In Bezug auf die Simulation verwendet man den Begriff *Situation*, um die Perspektivität des Wahrnehmens durch den Agenten zu beschreiben. Die artifizielle Wahrnehmung bildet nur einen Ausschnitt der simulierten Welt ab, wobei dieser Ausschnitt zusätzlich vom Zustand und der Motivation des Agenten bestimmt wird.

Im Rahmen der Implementierung erfolgt auch die Aufarbeitung der geografischen Situation. Sie bedeutet die Einbettung des Agenten in die Simulationsumgebung, also das räumliche Umfeld des Agenten und gibt damit den räumlichen Zusammenhang der topografischen Sachverhalte wieder, in denen der Agent interoperiert. Diese geografische Situation wird ergänzt um die epigenetische Situation.

Letztere wird damit begründet, dass auch dem Menschen nicht alle Bedingungen, die für das Verhalten relevant sind, auch bewusst sein müssen. Insofern könnte man zwischen bewussten und nicht bewussten Bedingungen unterscheiden. Es wird hier also ein Modell verfolgt, dass die Situation stärker sein kann als ein noch so stabiler Charakter. Mit anderen Worten: Wird auf jemanden starker Druck ausgeübt, kann er auch gegen seine persönlichen Grundsätze (Persönlichkeitsstruktur) handeln. Damit wäre der Dominanz der Situation gegenüber den Persönlichkeitsstrukturen Rechnung getragen.

---

<sup>6</sup> Siehe auch Gruhn und Thiel 2000.

## Intentionales System

Im Rahmen des *intentionalen Systems* (IntentionSystem) realisiert die artifizielle *Intuition* die Fähigkeit, Einsichten in Sachverhalte, Sichtweisen, Gesetzmäßigkeiten oder die Stimmigkeit von Entscheidungen unter Umgebung von Schlussfolgerungsmechanismen zu gewinnen. Intuition ist damit ein Teil kreativer Entscheidungen und Entwicklungen. In solch einem Fall führt der Agent nur noch aus. Als grundlegende Kompetenz verstanden, ist die artifizielle Intuition eine Fähigkeit zur Informationsverarbeitung und zur angemessenen Reaktion bei großer Komplexität der zu verarbeitenden Daten. Sie führt sehr oft zu richtigen bzw. optimalen Ergebnissen.

Neue Forschungsergebnisse deuten darauf hin, dass man mit der Intuition gerade in komplexen Situationen zu besseren Entscheidungen kommt als mit dem bewussten Verstand. Das Unbewusste ist in der Lage, weitaus mehr Informationen zu berücksichtigen als das Bewusstsein, das zwar sehr präzise ist, jedoch mit nur wenigen Informationen zurechtkommt.<sup>7</sup>

## Effektorisches System

Vom Verhalten als *Systemverhalten* eines Agenten (EffecterSystem) spricht man dann, wenn eine Veränderung des Zustandes bzw. der Zustandsgrößen des Systems auf der Makroebene beobachtet werden kann. Als *Ereignis* wird der Übergang von einem Zustand in einen anderen bezeichnet. Über ein solches Verhalten lassen sich bereits ohne Kenntnis der Mikroebene Gesetzmäßigkeiten erkennen. Erklärt werden können diese Gesetzmäßigkeiten aber nur durch die Systemstruktur.

Dies entspricht auch der psychologischen Auffassung von Verhalten, das sich vor allem auf die von anderen unmittelbar beobachtbaren Handlungen bezieht, und damit die Gesamtheit aller von außen beobachtbaren Äußerungen eines Menschen umfasst. Der Begriff der Handlung umfasst dabei neben den Verhaltenskomponenten meist noch diejenigen der Motive, also innere bzw. subjektive Elemente.

*Verhalten* aus systemtheoretischer Sicht bezieht sich auf alle äußerlich wahrnehmbaren und daher auch mit soft- und hardwaretechnischen Hilfsmitteln erfassbaren, aktiven Veränderungen, Bewegungen, Stellungen, Körperhaltungen, Gesten und kommunikativen Äußerungen eines Agenten, die in irgendeiner Form der Verständigung dienen. Als Verhalten kann einerseits die Gesamtheit solcher Interaktions- und/oder Interoperationsvorgänge bezeichnet werden, andererseits können als Verhalten aber auch einzelne Merkmale in einer bestimmten Zeitspanne bezeichnet werden. Verhalten wird in dieser Arbeit auch als eine durch das epigenetische System und durch Lernvorgänge beeinflusste Anpassungsleistung eines intakten Agenten an seine Umwelt verstanden.

Das effektorische System differenziert den Begriff der *Verhaltensweise* als eine beobachtbare Interaktions- bzw. Interoperationsfolge eines Agenten, die von einer anderen

---

<sup>7</sup> Siehe auch Dörner 1989.

Folge unterscheidbar ist. Als Verhaltensmuster wird eine Abfolge von Verhaltensweisen bezeichnet, die in bestimmten Situationen regelmäßig zu beobachten ist.

Ein bestimmtes Verhalten kann sowohl durch einfache innere Reize als auch durch komplexere, aber gleichfalls epigenetisch veranlagte Komponenten ausgelöst werden. Verhalten kann ferner als Reaktion auf Veränderungen in der Umwelt ausgelöst werden; in diesem Fall wird es durch exogene Reize ausgelöst. Verhalten ist stets an existierende Agenten gebunden.

Auch Steine können von einer Klippe abbrechen und sich so abwärts bewegen; diese Bewegung wurde aber vollständig von äußeren Einflüssen verursacht. Sie ist keine „Eigenleistung“ eines aktiv agierenden oder reagierenden Subjekts, für das die als Verhalten bezeichnete Veränderung, Bewegung, Haltung oder Äußerung eine bestimmte Funktion (einen Zweck, eine Bedeutung) hat. Für eine Zecke, die sich von einem Strauch auf ein warmblütiges Tier fallen lässt, hat das Fallen hingegen zweifelsfrei eine Funktion.

Der Begriff Verhalten wird daher in diesem Buch nur auf Agenten mit der Möglichkeit zur Informationsverarbeitung durch das kognitive System angewandt, die zum aktiven Verhalten fähig sind. Das Verhalten ist nicht nur an sichtbares Verhalten oder Veränderungen eines Agenten gebunden. Das Verhalten eines Agenten äußert sich auch in Erscheinungen wie Ruhe, Schlaf, Starre oder Lauerstellung, die über eine bestimmte Zeitspanne hinweg stationäre Zustände sind.

### **Gedächtnissystem**

Unter dem *Gedächtnis* (MemorySystem) versteht man die Fähigkeit des Agenten, wahrgenommene und verarbeitete Informationen zu behalten, zu ordnen und wieder abzurufen. Die gespeicherten Informationen sind das Ergebnis von direkten (bewussten) oder indirekten (unbewussten) Lernprozessen.

In diesem Sinne wird der Begriff des Gedächtnisses auch allgemein für die Speicherung von Informationen in anderen biologischen und technischen Gebieten benutzt.

Je nach Dauer der Speicherung der Information wird zwischen dem sensorischen Gedächtnis, dem Kurzzeitgedächtnis und dem Langzeitgedächtnis unterschieden. Je nach Art der Gedächtnisinhalte unterscheidet man beim Langzeitgedächtnis ferner zwischen *deklarativem* und *prozeduralem* Gedächtnis. Das deklarative Gedächtnis speichert Fakten bzw. Ereignisse, die entweder zur Biographie des Systems gehören (*episodisches* Gedächtnis) oder das so genannte *Weltwissen* eines Systems ausmachen, wie zum Beispiel Kenntnisse, Fakten etc. (*semantisches* Gedächtnis). Das prozedurale Gedächtnis beinhaltet Fertigkeiten, die automatisch, d. h. ohne Vorverarbeitung eingesetzt werden, wozu vor allem motorische Abläufe gehören. Prozedurale Gedächtnisinhalte werden durch implizites Lernen, semantische durch explizites Lernen erworben.

Das *deklarative Gedächtnis*, auch *Wissensgedächtnis* oder *explizites Gedächtnis*, speichert Tatsachen und Ereignisse, die direkt wiedergegeben werden können. Man unterteilt das deklarative Gedächtnis in zwei Bereiche:

- Das *semantische Gedächtnis* enthält das Weltwissen, als von dem Agenten unabhängige, allgemeine Fakten.
- Im *episodischen Gedächtnis* finden sich Episoden, Ereignisse und Tatsachen aus dem bisherigen Lebenszyklus des Agenten.

Das *prozedurale Gedächtnis*, auch *Verhaltensgedächtnis*, speichert automatisierte Handlungsabläufe bzw. Fertigkeiten.

Dabei müssen oftmals komplexe Bewegungen ausgeführt werden, deren Ablauf man gelernt und oft geübt hat, die nun aber *ohne nachzudenken* abgerufen werden können, ohne dass sich das Bewusstsein um Bewegungsimpulse an verschiedenste Muskeln und ihre Koordination kümmern müsste. Verschiedene subkortikale Regionen (nicht im Neocortex gelegen und damit nicht dem Bewusstsein zugänglich) erbringen die Leistung des prozeduralen Gedächtnisses.

Neben den vorgestellten drei Speicherarten (sensorisches Gedächtnis, Kurzzeitgedächtnis und Langzeitgedächtnis) gibt es stets auch einen weiteren Speicher für „überflüssige“ Informationen für die Funktion des Vergessens („Papierkorb“).

Der Prozess, in dem das kognitive System durch Lernprozesse die Art und Weise beeinflusst, in der bestimmte Reize eine Emotion hervorrufen, wird als „emotionales Gedächtnis“ bezeichnet.

Die im Rahmen des *Gedächtnissystems* berücksichtigten Gedächtnis-Modelle unterscheiden sich sowohl hinsichtlich des Formats gespeicherter Daten, Information und Wissenseinheiten als auch in der Implementierung der einzelnen Gedächtnisfunktionen: Assoziation oder Gedächtnisspuren. So basieren assoziative Modelle auf Netzwerken von Knotenpunkten und Verknüpfungen. Die Knoten repräsentieren kognitive Einheiten, z. B. Begriffe, die Verknüpfungen semantische oder episodische Relationen.

Die deklarative Gedächtnisfunktion umfasst alles Faktenwissen eines informationsverarbeitenden Systems und wird in Form eines propositionalen Netzwerks beschrieben. Die Nutzung dieses Wissens erfolgt durch Ausbreitung von Aktivierungen. Das prozedurale Gedächtnis umfasst hingegen als kleinste Wissenseinheit Produktionen (Produktionensysteme). Das sind aus Bedingungs- und Aktions-Komponenten zusammengesetzte Einheiten.

Weiterhin sind *konnektionistische Modelle* vorgesehen.<sup>8</sup> Dabei werden einfache Verarbeitungselemente zugrunde gelegt, über denen Muster definiert werden können. Die Elemente repräsentieren kleine, merkmalsähnliche Einheiten (*microfeatures*). Sie bilden

<sup>8</sup> Siehe auch Helm 1991.

durch vielfältige Verknüpfungen komplexe Systeme und werden ähnlich wie die Neuronen aufgefasst. Die Verknüpfungen sind unterschiedlich stark. Sie sind in Modulen organisiert. Jedes Element eines Moduls erhält Information von anderen Modulen und leitet Informationen an andere Module weiter. Die Verarbeitung von Information hinterlässt Spuren im Modul in Form von Veränderungen der Gewichte von Verknüpfungen zwischen den Elementen. Das Aktivationsmuster über den Elementen des Systems gibt an, was das System zu einem Zeitpunkt repräsentiert. Wissenserwerb und Lernen erfolgt durch Veränderung der Verknüpfungsgewichte. Erinnern wäre demnach zum einen ein rekonstruktiver Prozess, der Muster-Vervollständigung vergleichbar und entspricht einem Bewusstwerden von Erfahrungsinhalten. Zum anderen ist das intentionale Erinnern ein komplexer artifiziell-kognitiver Prozess, der sich aus mehreren Teilprozessen zusammensetzt. Ausgelöst wird dieser Prozess stets durch eine Frage- oder Problemstellung. Aus ihr werden die lösungshinweisenden Suchelemente extrahiert, die wiederum bestimmte Gedächtnisinhalte aktivieren bzw. aktualisieren. Was aktiviert werden kann, hängt sowohl von den Suchelementen, als auch von den im Gedächtnis repräsentierten Daten, Informationen und Wissen ab. Die aktivierte Gedächtnisinhalte bilden dann die Suchmenge, die daraufhin zu überprüfen ist, ob sie wenigstens Teile der gesuchten Daten und Informationen bzw. von Wissen enthält, die für die Konstruktion einer Antwort, Entscheidung oder Lösung benötigt wird. Reichen die gefundenen Gedächtnisinhalte für die Antwort- oder Lösungskonstruktion noch nicht aus, dann können andere Suchelemente ausgewählt werden, und der Erinnerungszyklus startet erneut. Abgeschlossen wird der Erinnerungsprozess durch die Konstruktion einer Antwort oder Lösung, die als Ganzes noch einem Bewertungsprozess unterliegt. So können Erinnerungsleistungen beispielsweise mit der Methode der Wiedererkennung oder der freien Reproduktion in Form eines Abrufes bewertet werden. Gedächtnisinhalte, die erfolgreich wiedererkannt werden können, müssen vorher bereits gespeichert gewesen sein. Können solche Inhalte wiedererkannt, aber nicht erinnert werden, weist dies auf Störungen des Abrufs aus dem Gedächtnis hin. Bei einem Abruf von im Gedächtnis gespeicherten Daten, Inhalten und Wissen wird durch die Abfrage selbst bestimmt, welche Inhalte tatsächlich abgerufen werden. Hierbei gilt es dann, zwischen dem Anteil der abgerufenen Inhalte und dem darin enthaltenen Anteil der relevanten Inhalte zu unterscheiden. Im Idealfall werden alle relevanten Inhalte und keine irrelevanten Inhalte abgerufen.

Konkrekt enthält das Kurzzeitgedächtnis (Arbeitsgedächtnis)  $7 + 2$  Chunks als effektive Kapazität, wobei jedes einzelne Chunk aus Einzelinformationen besteht, die nach bestimmten Ähnlichkeitskriterien geordnet oder kombiniert werden. Die Speicherzeit dieser Chunks als auch die Zugriffszeit auf die Informationen kann parametrisiert und damit je nach Anwendungsfall variiert werden. Die semantische bzw. assoziative Struktur des Langzeitgedächtnis wird in Form semantischer Netze bzw. episodischer Netze realisiert. Alternativ oder zusätzlich kommen Scripts bzw. Frames zum Einsatz.

## Lernsystem

Unter *Lernen* (LearnerSystem) versteht diese Arbeit das absichtliche (intentionales Lernen), das eher beiläufige (inzidentelles und implizites Lernen) und den agentenindividuellen oder kollektiven Erwerb von Kenntnissen, Fähigkeiten und Fertigkeiten. Dabei wird das lernpsychologische Konzept des Lernens als ein Prozess der relativ stabilen Veränderung des Verhaltens und Schlussfolgerns aufgrund von Erfahrung oder neu gewonnenen Einsichten und des Verständnisses (verarbeiteter Wahrnehmung der Umwelt) realisiert.

Lernen soll im Gedächtnis ebenso Spuren hinterlassen (agentenbezogener Anteil), wie in der Umwelt (objektivierender Anteil) durch Interoperationen. Lernen geschieht damit aktiv und passiv. Zum Grundinstrumentarium des Lernens gehören neben dem Lernprozess auch die Fähigkeit zur *Erinnerung* (Gedächtnis) und des *Abrufens* (Anwendung von Erlerntem oder Lerntransfer). Jedoch ist Lernen mehr als das reine Abspeichern von Informationen. Lernen beinhaltet die Wahrnehmung und Bewertung der Umwelt, die Verknüpfung mit Bekanntem (Erfahrung) und das Erkennen von Regelmäßigkeiten (Mustererkennung).

Der lernende Agent beginnt mit dem Lernen nicht als unbeschriebenes Blatt (tabula rasa). Jeder Lernprozess setzt auf einem Lerntypen auf, einer implementierten Eigenschaft und damit Ausprägung bei der Nutzung von sensorischen Kanälen oder der Fähigkeit, durch verschiedene Lernprozesse sich Wissen anzueignen.

Lernprozesse werden in der Literatur nach verschiedenen Kriterien klassifiziert. Ist das Kriterium die Art des gelernten Verhaltens, kann zwischen dem Erlernen von Bewegungsabläufen (motorisches Lernen), dem Erlernen sprachlicher Inhalte (verbales Lernen), dem Erlernen von sozialen Normen (Sozialisation) usw. unterschieden werden. Ein anderes Kriterium zur Klassifizierung von Lernprozessen ist die Komplexität des gelernten Verhaltens. Einfache Anpassungen werden durch Sensitivierung und Habituation erworben. Eine komplexere Form ist das assoziative Lernen. Dabei werden zwei Ereignisse miteinander verknüpft (assoziiert). Beim sog. S-S-Lernen sind dies zwei Reize, beim S-R-Lernen ein Reiz mit einer Reaktion. Zwei bekannte Arten des assoziativen Lernens sind die Klassische Konditionierung und die operante Konditionierung. Weitere Formen assoziativen Lernens sind die Prägung, das Lernen am Erfolg sowie Generalisierungs- und Diskriminationslernen. Komplexere Verhaltensweisen werden durch Lernen durch Einsicht, durch Lernen lernen und durch strukturelles Lernen erworben. Ein weiteres Kriterium zur Klassifizierung von Lernprozessen ist die Rolle des Lernenden. Dabei wird unterschieden zwischen inzidentiellem Lernen, intentionalem Lernen, entdeckendem Lernen, selbstbestimmtem Lernen, expansivem Lernen, widerständigem Lernen usw.

Im Rahmen der Implementierung können je nach Komplexität der Problemstellung dabei unterschiedliche *Wissensrepräsentationen* Anwendung finden. Die möglichen Wissensrepräsentationsformate unterscheiden sich bezüglich ihrer Ausdrucksvermögen bzw. ihrer Expressivität, je nachdem also, welche Wissensstrukturen sie darzustellen vermögen. In der derzeitigen Version des Cognitive Computing lassen sich *Konzepte* bzw. Instantiierungen von Konzepten als Elemente einer Wissensrepräsentation einsetzen (Beispiele: Apfel, Baum, Mensch, etc.). Weiterhin lassen sich diese Konzepte bzw. deren Instanzen

durch Merkmale anreichern, die diesen Konzepten bzw. Instanzen zukommen oder fehlen (Beispiele: roter Apfel, grüner Baum, lebender Mensch). Durch *Attribut-Wert-Strukturen* lassen sich Klassen spezifizieren (Beispiele: Farbe:rot, Oberbegriff:Obst.) Durch die Angabe von *Relationen* (Propositionen, Prädikate) lassen sich Beziehungen zwischen Entitäten ausdrücken (Beispiel: hat-Farbe, ist-ein-Apfel.). Prozedurales Wissens wird in Form von *Produktionsregeln* der Form „WENN p DANN q“ und komplexe *relationale Strukturen* durch Schemata repräsentiert (Beispiele: frames, scripts).

Wie aus dem kognitiven Modell ersichtlich, wird diese Wissensrepräsentation ausschließlich auf der Ebene von Daten-, Informations- und Wissensstrukturen, Objekten und den entsprechenden kognitiven Funktionen in Form von Methoden (Algorithmen) realisiert, wobei die Implementierung auch hier unterschiedlich sein kann.

In den Objekten ist deklaratives Wissen prozedural gekapselt, d. h. nur durch eine Nachricht und die darauf folgenden Ausführung einer Prozedur zugänglich.

So werden derzeit neben gängigen Programmiersprachen, Merkmalslisten und Netzen mit Aktivitätsausbreitung, Attribut-Wert-Strukturen, semantische Netze und Frames, Logik, Produktionsregeln, Neuronale Netze, Fuzzy Logic, Evolutionäre Algorithmen eingesetzt.

In konnektionistischen Systemen im Allgemeinen und in neuralen Netzen oder Fuzzy Systemen im Speziellen steckt „Wissen“ in der Verbindungsstruktur, den Gewichten der einzelnen Verbindungen sowie auch in den Eigenschaften der Verarbeitungselemente.

Lernen kann unter Zuhilfenahme von Lehrmethoden, Lernstrategien und dem Einsatz von Technologien funktional und prozessual zum maschinellen Lernen ausgestaltet werden.

Unterschiedliche Formen des Lernens sind bekannt und werden von verschiedenen Lerntheorien beschrieben. Die genaue Funktionsweise des Lernens ist allerdings wissenschaftlich noch nicht geklärt und durchaus umstritten, weshalb sich verschiedene Lerntheorien in Ansätzen und Herangehensweisen durchaus widersprechen können.

*Maschinelles Lernen* ist dabei ein Oberbegriff für die „künstliche“ Generierung von Wissen aus Erfahrung: Ein künstliches System lernt aus Beispielen und kann nach Beendigung der Lernphase verallgemeinern. Das heißt, es lernt nicht einfach die Beispiele auswendig, sondern es „erkennt“ Gesetzmäßigkeiten in den Lerndaten. So kann das System auch unbekannte Daten beurteilen.

Das Thema ist eng verwandt mit Data-Mining, bei dem es jedoch vorwiegend um das Finden von neuen Mustern und Gesetzmäßigkeiten geht.

Beim maschinellen Lernen spielen Art und Mächtigkeit der Wissensrepräsentation eine wichtige Rolle. Man unterscheidet zwischen symbolischen Systemen, in denen das Wissen – sowohl die Beispiele als auch die induzierten Regeln – explizit repräsentiert ist, und subsymbolischen Systemen, wie neuronale Netze, denen zwar ein berechenbares Verhalten „antrainiert“ wird, die jedoch keinen Einblick in die erlernten Lösungswege erlauben, d. h. dass hier Wissen implizit repräsentiert ist. Bei den symbolischen Ansätzen werden aussagenlogische und prädikatenlogische Systeme unterschieden.

Die praktische Realisierung erfolgt mittels Algorithmen. Die verschiedenen Algorithmen aus dem Bereich des maschinellen Lernens lassen sich einteilen:

- *Überwachtes Lernen (supervised learning)*: Der Algorithmus lernt eine Funktion aus gegebenen Paaren von Ein- und Ausgaben. Dabei stellt während des Lernens ein „Lehrer“ den korrekten Funktionswert zu einer Eingabe bereit. Ein Teilgebiet des *überwachten Lernens* ist die automatische Klassifizierung.
- *Unüberwachtes Lernen (unsupervised learning)*: Der Algorithmus erzeugt für eine gegebene Menge von Eingaben ein Modell, das die Eingaben beschreibt und Vorhersagen ermöglicht. Dabei gibt es Clustering-Verfahren, die die Daten in mehrere Kategorien einteilen, die sich durch charakteristische Muster voneinander unterscheiden. Ein wichtiger Algorithmus in diesem Zusammenhang ist der EM-Algorithmus, der iterativ die Parameter eines Modells so festlegt, dass es die gesehenen Daten optimal erklärt. Er legt dabei das Vorhandensein nicht beobachtbarer Kategorien zugrunde und schätzt abwechselnd die Zugehörigkeit der Daten zu einer der Kategorien und die Parameter, die die Kategorien ausmachen. Eine Anwendung des EM-Algorithmus findet sich beispielsweise in den Hidden Markov Models (HMMs). Andere Methoden des unüberwachten Lernens verzichten auf die Kategorisierung. Sie zielen darauf ab, die beobachteten Daten in eine einfachere Repräsentation zu übersetzen, die sie trotz drastisch reduzierter Information möglichst genau wiedergibt.
- *Bestärkendes Lernen (reinforcement learning)*: Der Algorithmus lernt durch Belohnung und Bestrafung eine Taktik, wie in potenziell auftretenden Situationen zu handeln ist, um den Nutzen des Agenten (d. h. des Systems, zu dem die Lernkomponente gehört) zu maximieren.

Des Weiteren unterscheidet man zwischen Batch-Lernen, bei denen alle Eingabe/Ausgabe Paare gleichzeitig vorhanden sind und kontinuierlichem (sequentiellen) Lernen, bei dem sich die Struktur des Netzes zeitlich versetzt entwickelt.

Als *Erfahrung* wird in diesem Buch zweierlei bezeichnet: im Einzelfall ein bestimmtes Erlebnis eines Agenten in Form eines von ihm selbst wahrgenommenen Ereignisses, oder allgemein – und dann im Sinne von „Lebenszykluserfahrung“ - die Gesamtheit aller Erlebnisse, die ein Agent in deren Lebenszyklus jemals gehabt hat.

Das entspricht auch der Auffassung seitens der Pädagogik, die zwischen *Primärerfahrung* und *Sekundärerfahrung* unterscheidet. Primärerfahrungen sind unmittelbare Erfahrungen, die in direktem Kontakt mit Mitmenschen oder einem Objekt gemacht werden. Erfahrungen, die man aus der Wahrnehmung anderer übernimmt, sind Sekundärerfahrungen. Hierzu zählen beispielsweise Erfahrungen, die durch Medien vermittelt werden.

Mit anderen Worten unterscheidet das Lernsystem zwischen einer intrinsischen (inneren) Erfahrung von einer extrinsischen (äußereren) Erfahrung. Äußere Erfahrung bezieht sich auf das Erleben von „äußeren“, d. h. in der Umwelt stattfindenden Ereignissen, während innere Erfahrungen sich vollständig im Bereich des Systems durch Reflexion abspielen. So kann Erfahrung an Phänomenen wie Wissen, Fähigkeiten, Überzeugungen und Meinungen oder auch an der Herausbildung individueller wie kultureller Weltbilder maßgeblich beteiligt sein. Der Erfahrungsbegriff betont dabei im Unterschied zu anderen möglichen Formen des Wissens, dass dieses durch unmittelbares Verhalten zustande gekommen ist. Dieser verhaltensorientierte Ansatz von Erfahrung ist implementierungsbedingt immer nur auf einen bestimmten Agenten bezogen, kann allerdings auch die gesamte Population der Agenten erreichen.

Die im artifiziellen Lernen verwendeten Repräsentationsformalismen zur Darstellung solcher Erfahrungen lassen sich in drei Kategorien einteilen: durch Eigenschaftsvektoren, durch Attributvektoren und durch relationale Strukturen. Bei der einfachsten Repräsentation in Form von Eigenschaftsvektoren (Eigenschaftslisten) werden Erfahrungen durch Listen binärer Werte dargestellt. Jede Liste beschreibt ein Objekt und die binären Werte beschreiben das Vorhandensein oder Fehlen einer Eigenschaft des Objekts. Ein mächtigerer Formalismus ist gegeben, wenn auch relationale (z. B. strukturelle) Beschreibungen möglich sind. Diese Möglichkeiten bieten Repräsentationsformalismen, die auf Prädikatenlogik basieren.

Der Zusammenhang zwischen Lernen und Problemlösen gestaltet sich in drei Phasen.<sup>9</sup>

- Am Anfang des Wissenserwerbs steht die Phase der interpretativen Verwendung deklarativer Daten-, Informations- und Wissensbestände. Die Grundidee besteht darin, dass neue Daten und Informationen zunächst in einer deklarativen Form enkodiert werden. Diese deklarativ gespeicherte Form kann nicht unmittelbar in Verhalten umgesetzt werden, weil die entsprechenden Verhaltensanweisungen bzw. -empfehlungen fehlen.
- Das Ziel der zweiten Phase besteht in der Bildung bereichsspezifischer Regeln, in denen zuvor deklarativ kodierte Daten und Informationen in unmittelbar verhaltenswirksame prozedurale Wissensbestände überführt werden.
- Die dritte Phase der Wissensoptimierung bzw. Wissensrevision führt zu weiteren Verbesserungen. Die Ausführung der Fertigkeiten gelingt immer besser und schneller. Die damit einhergehende Feinabstimmung der prozeduralen Wissensbestände erfolgt auf der Grundlage von den drei Mechanismen der Generalisierung, Diskrimination und Verstärkung.

---

<sup>9</sup> Siehe auch Mayer 1979.

Das Ergebnis dieses Lernprozesses kann sowohl eine Veränderung der Daten, Informations- und Wissensstrukturen des Systems als auch eine Veränderung der Repräsentation dieser Strukturen sein. Bereits repräsentiertes Wissen an neue Gegebenheiten im Prozess der Revision anpassen zu können bzw. zu vervollkommen, muss als unabdingbare Fähigkeit kognitiver Systeme betrachtet werden. Der Prozess der Wissensrevision umfasst das Erkennen einer Unvollkommenheit, die Identifizierung des Wissens, das diese Unvollkommenheit auslöst, die Berechnung möglicher Revisionsalternativen, die anschließende Bewertung der Alternativen, die Auswahl der optimalen Revision und die Durchführung der Revision.

Erschwert wird die Wissensrevision aufgrund der in formalen Systemen erforderlichen Widerspruchsfreiheit. Menschen hingegen scheinen relativ problemlos mit Widersprüchen leben zu können. Erschwerend kommt hinzu, dass neben dem Umgang mit inkonsistentem Wissen auch unvollständiges und unsicheres Wissen eine Revision bedingen kann. Man wird hier mit dem allgemeinen Problem der Nichtmonotonie von Erweiterungen einer Wissensbasis konfrontiert, das immer dann gegeben ist, wenn neu hinzugefügtes Wissen in Widerspruch gerät zu bereits vorhandenem Wissen. Aus diesem Grund ist man an Repräsentationsformalismen interessiert, die es erlauben, *Standardannahmen* auszudrücken, also Fakten und Regeln, die für den typischen Fall gelten, ohne dass die Ausnahmen benannt werden müssen. Das maschinelle Schließen mit Standardannahmen erfordert unter Umständen die Rücknahme bereits abgeleiteter Formeln. Konkrete Beispiele spielen deshalb eine wichtige Rolle bei der Bewertung nicht monotoner Logiken. Lässt sich ein Beispiel angeben, in dem eine bestimmte Form der nichtmonotonen Ableitung zu „nicht intuitiven“ Ergebnissen führt, so wird das häufig zum Anlass genommen, um einen neuen Kalkül vorzuschlagen, der für dieses Beispiel ein „intuitiveres“ Ergebnis liefert.

Zu den im artifiziellen Lernen untersuchten Ansätzen zur Repräsentationsmodifikation gehören das erklärbasierte Lernen, die Bildung von Makro-Operatoren und die Wissenskompilierung. Zu den im artifiziellen Lernen untersuchten nicht wahrheitserhaltenen Transformationen gehören das induktive Lernen aus Beispielen, das abduktive Schließen, das Bilden neuer Konzepte, das Erkennen und Nutzen von Analogien, die induktive Wissensrevision und das Verstärkungslernen (reinforcement learning).

Das *induktive Lernen aus Beispielen* gehört zum überwachten (supervised) Lernen, d. h. zusammen mit den Eingaben und den Ausgaben wird genau spezifiziert, was und aus welchen Daten gelernt werden soll. Die meisten *induktiven Lernverfahren* induzieren (disjunktiv verknüpfte) Mengen konjunktiver Regeln. Die Bevorzugung möglichst weniger Regeln mit einer möglichst geringen Anzahl von Prämissen bildet eine typische Lernpräferenz dieser Lernverfahren. Mit der Suchstrategie „Allgemein-nach-Speziell“ (*top down*) wird, ausgehend von der allgemeinsten darstellbaren Hypothese, durch die Anwendung von Spezialisierungsoperatoren nach Hypothesen gesucht, die weniger negative Beispiele falsch klassifizieren. Mit der Strategie „Speziell-nach-Allgemein“ (*bottom up*) wird, ausgehend von einer speziellen Hypothese, die typischerweise aus einem positiven Beispiel generiert wird, nach allgemeineren Hypothesen gesucht, die mehr positive Beispiele richtig

klassifizieren. Die Generalisierung einer Hypothese geschieht durch das Ersetzen einzelner Bedingungen durch weniger restriktive Bedingungen oder das Löschen von Bedingungen. Zur Verfolgung verschiedener Generalisierungs- bzw. Spezialisierungsmöglichkeiten einer Hypothese werden bei vielen Lernverfahren Standardsuchstrategien in Form von Breiten-suche oder Tiefensuche verwendet. Auch das *Lernen in neuronalen Netzen* ist dem induktiven Lernen zuzuordnen, da neuronale Netze nicht nur auf Informationen reagieren, die in einer Lernphase verarbeitet wurden. Dabei kann, wie bei symbolischen Lernverfahren, zwischen überwachtem und unüberwachtem Lernen unterschieden werden. Hinsichtlich der Verständlichkeit der Lernergebnisse sind symbolische Verfahren deutlich im Vorteil.

Beim Lernen oder Trainieren von künstlichen neuronalen Netzen unterscheidet man zwischen überwachtem und unüberwachtem Lernen. Beim überwachten Lernen gibt es sogenanntes Lehrersignal, das den richtigen Ausgabewert (oder Ausgabevektor) des Netzes angibt oder zumindest sagt, ob die Ausgabe richtig oder falsch ist. Beim unüberwachten Lernen gibt es keine vorgegebene richtige Antwort des künstlichen neuronalen Netzes.

Die Bildung neuer Konzepte und deren Integration in existierende Repräsentationen von Wissen wird als *Begriffsbildung* bezeichnet. Das Lernziel der Begriffsbildung ist die Konstruktion einer Hierarchie nützlicher Konzepte aus Erfahrungen und Hintergrundwissen, wobei jedes Konzept intensional beschrieben ist.

*Einsicht* bezeichnet das Verstehen der strukturellen, funktionalen oder kausalen Zusammenhänge, die einer Problemstellung und ihrer grundsätzlichen Lösung zugrunde liegen. Die damit einhergehenden Umstrukturierungsvorgänge erfordern in der Regel einen Wechsel der Analyseebenen (z. B. abstrakt versus konkret), der Problemlösestrategie (z. B. ziel- versus datenorientiert) oder mentalen Repräsentation des Problems (z. B. symbolisch versus analog).

Das *Lernen durch Analogiebildung* weist starke Ähnlichkeit mit dem fallbasierten Schließen auf. Wie beim fallbasierten Schließen wird die (Teil-) Lösung eines Falles auf einen anderen Fall übertragen, indem nach einem ähnlichen Fall gesucht wird, dessen Lösung an die gegebene Situation adaptiert wird. Da das Vokabular zur Beschreibung unterschiedlicher Sachbereiche typischerweise verschieden ist, erfordert Analogiebildung das Erkennen struktureller Ähnlichkeiten.

Beim *Verstärkungslernen* (reinforcement learning) wird jede Verhaltensweise des Systems bewertet. Ziel des Lernens ist die Veränderung des Verhaltens des Systems in eine Richtung, die zur höchsten Bewertung des Gesamtverhaltens führt. Typisches Anwendungsbeispiel für Verstärkungslernverfahren ist das Optimieren der Effektorsteuerung von Robotersystemen.

Das Lernen durch wahrheitserhaltende Transformationen konzentriert sich auf erklärbasisiertes Lernen sowie der Wissenskompilierung. *Erklärbasisiertes Lernen* ist solch ein wahrheitserhaltendes Lernverfahren für regelbasierte Systeme, das, geleitet durch ein vorliegendes Beispiel einer Problemlösung, eine Regel deduziert, mit der das gleiche oder sehr ähnliche Probleme effizient gelöst werden können. Die *Wissenskompilierung* ist

ebenfalls eine wahrheitserhaltende Transformation von Wissen, die in einer oder verschiedenen Formen deklarativ repräsentiert ist, in eine effizientere prozedurale Repräsentation, die dann auf eine bestimmte Nutzung ausgerichtet ist. Im Gegensatz zum erklärbungsisierten Lernen beinhaltet Wissenskompilierung den Austausch des Repräsentationsformalismus.

Da ein kognitives System in der Regel auf mehrere Wissensarten zugreift, muss ein solches System über verschiedene Lernverfahren verfügen. Durch die unterschiedlichen Lernausrichtungen und damit Lernleistungen kann es außerdem notwendig sein, mehrere Lernverfahren zu kombinieren und kooperierend oder nacheinander zum Lernen einer Art von Wissen zu verwenden.

In Bezug auf die Funktionen des epigenetischen Systems ist insbesondere zwischen terminologischem Wissen, Problemlösungswissen (Kontrollwissen) und Wissen über Objekte der jeweiligen Problemdomäne zu unterscheiden. Die im Rahmen der Wissenstheorie erarbeiteten Wissensarten können deklarativ und prozedural repräsentiert sein. Eine deklarative Repräsentation hat Vorteile, wenn die Verständlichkeit der Darstellung wichtig ist, während prozedurale Repräsentationen häufig gewählt werden, wenn die Effizienz im Vordergrund steht. Eine hybride Repräsentation liegt vor, wenn die Repräsentation deklarative und prozedurale Anteile hat.

In seiner allgemeinen Bedeutung meint der handlungsorientierte Ansatz des Cognitive Computing mit *Lernerfahrung* praktisches Wissen, erworben im konkreten Umgang mit einem Gegenstandsbereich innerhalb einer Problemdomäne bzw. Umgebung. In seiner engeren Bedeutung umfasst Erfahrung nur jene kognitiven Inhalte, die sich scheinbar ohne systemische Bewertung präsentieren. In diesem engeren Sinne kann man die Lernerfahrung als eine „äußere“ Erfahrung, d. h. eine sich rein aus der Wahrnehmung und dem Verhalten konstituierende Erfahrung bezeichnen. Eine solche Lernerfahrung wird in diesem Ansatz daher von der „inneren“ Erfahrung, wie Motivation, Introspektion und Intuition abgegrenzt.

#### 6.4.5 Systemische Intelligenz

Dieses Buch und im Rahmen der Validierung wird unterstellt, dass ein Robotersystem dann über eine systemische Intelligenz verfügt, wenn es ein definiertes Ziel anstrebt und dabei versucht, dieses Ziel mit möglichst geringem Energieaufwand zu erreichen.<sup>10</sup> Zu der Erreichung dieses Ziels kann das Robotersystem selbständig durch Interoperationen mit der Umwelt (Umwege wählen, Einbahnngassen vermeiden, Kollisionen verhindern, bzw. Pläne entwickeln, Emotionen eingrenzen), um Konflikte auf dem Weg zur Zielerreichung zu vermeiden. Die Intelligenz des Robotersystems wird dabei mittels einer Kennzahl angegeben, die als systemischer Intelligenzquotient ( $IQ_s$ ) bezeichnet wird.

<sup>10</sup> Siehe auch Cobalus 1988.

Im Rahmen der Validierung wird der Tatsache Rechnung getragen, dass die systemische Intelligenz eines Robotersystems zunächst kein direkt beobachtbares Merkmal wie beispielsweise sein Gewicht ist. Insofern lässt sich der Grad der Intelligenz nicht aus den systemindividuellen Merkmalen ermitteln, sondern ergibt sich als Resultat aus den Interoperationen mit der Umwelt. Umgangssprachlich wird damit der Sachverhalt ausgedrückt, dass der systemische Intelligenzquotient die Fähigkeit des Robotersystems ausdrückt, zweckvoll bzw. zielorientiert zu interopieren, vernünftig Schlussfolgerungen zu ziehen, um damit auf die Umwelt einzuwirken. Wenngleich damit die systemische Intelligenz nicht unabhängig von den Leistungsmerkmalen der Teilsysteme, insbesondere des Lernsystems, des Gedächtnissystems und des Wahrnehmungssystems zu sein scheint, ergibt sich der Grad der Intelligenz vor allem aus dem kognitiven Prozess und seiner Leistungsfähigkeit. Demzufolge errechnet sich die systemische Intelligenz aus der multiplikativen Verknüpfung der perzeptiven, effektorisch-motorischen und kognitiven Komponente, was schließlich auch der Eingangs vorgenommenen Grobgliederung des Robotersystems entspricht. Ein Robotersystem ist demnach dann intelligent, wenn es über eine hohe Kompetenz zur Lösung der im Rahmen der Interoperationen auftretenden Probleme verfügt. Ein Problem wiederum wird als eine Diskrepanz zwischen einem Ist- und einem Zielzustand aufgefasst, wobei der Lösungsweg nicht unbedingt vorgegeben, sondern beispielsweise im Rahmen einer intuitiven und kreativen Zuwendung zum Problem erst gefunden werden muss. Dieses Finden der Problemlösung kann als Suche in einem Problem- und Lösungsraum beschrieben werden, wobei der Problemraum den Ausgangszustand, eventuelle Zwischenzustände, Operationen als Mittel zum Zweck, Anwendungsbedingungen als Restriktionen sowie das Weltbild des Robotersystems umfasst. Der Lösungsraum hingegen enthält den Zielzustand oder hierzu alternative Zustände, die es anzustreben gilt. In diesem Sinne gestaltet sich das Problemlösung als Schrittfolge, wobei der erste Schritt darin besteht, den Problemraum entsprechend aufzufalten. Im zweiten Schritt wird durch die Anwendung von Operatoren und der Durchführung rekursiver Ziel-Mittel-Analysen der Problemraum durchschritten, um in den Lösungsraum zu gelangen und dort den Zielzustand einzunehmen. Dieses Durchschreiten kann unter Umständen dazu führen, dass sowohl der Problemraum als auch der Lösungsraum entweder expandiert oder komprimiert wird, je nachdem ob Probleme oder Lösungen hinzukommen oder sich auflösen. Dieser Problemlösungsansatz zeigt sich konkret darin, dass die einzelnen Systeme durchaus als Produktionsregelsysteme aufgefasst werden können, deren Bedingungs- und Aktionskomponente das Durchschreiten des Problem- und Lösungsraum regelt.

---

## Literatur

- Brunak S, Lautrup B (1993) Neuronale Netze: Die nächste Computer-Revolution. Hanser, München  
Buschmann F, Meunier R, Rohnert H, Sommerlad P, Stal M (1986) Pattern oriented software architecture: a system of patterns. Wiley, New York  
Cliff DT, Husbands P, Harvey I (1993) Evolving recurrent dynamical networks for robot control.  
School of Cognitive and Computing Sciences University of Sussex

- Cobalus N (1988) Grenzen der Intelligenz. Eine Einführung in die Kritik der Computertechnologie.  
LIT Verlag, Frankfurt a. M.
- Dörner D (1989) Die Logik des Mißlingens. Strategisches Denken in komplexen Situationen.  
Rowohlt Verlag, Reinbek
- Gruhn V, Thiel A (2000) Komponentenmodelle, Addison-Wesley, München
- Helm G (1991) Symbolische und konnektionistische Modelle der menschlichen Informationsverarbeitung. Springer, Berlin u. a.
- Leffingwell D, Widrig D (2000) Managing software requirements, a unified approach, addison-wesley. Addison-Wesley, Reading, Massachusetts
- Mayer R (1979) Denken und Problemlösen. Springer, Berlin

---

## 7.1 Implikationen einer artifiziellen Kognition

In diesem Buch standen auch das Cognitive Computing und dort die symbolischen und subsymbolischen Ansätze sowie deren Ausimplementierung zu softwaretechnologischen Lösungen im Fokus des Interesses. Dabei haben sich Gemeinsamkeiten herauskristallisiert, die als Basis für die zünftige Entwicklung grundlegend sind. So gehen die Ansätze von einer Nichtableitbarkeit des Kognitiven aus, d. h. in den Ansätzen wird eine bestimmte Qualität kognitiver Zustände angenommen, die sich auf keine andere Ebene wissenschaftlicher Beschreibung reduzieren und demnach modellieren lässt. Dies ist auch der Grund dafür, dass zur theoretischen Aufarbeitung dieser Ansätze reduktionistisch operiert und damit die Annahme vertreten wird, dass kognitive Phänomene auf rein physikalischer Ebene wissenschaftlich derzeit nicht hinreichend erklärt werden können. Auf Basis dieser Annahme geht der symbolische Ansatz von einer irreduziblen Symbolebene aus, um kognitive Phänomene zu erzeugen, hingegen erachtet der subsymbolische Ansatz ein neuronales Netz für im Prinzip fähig, Kognition als emergentes Phänomen hervorzubringen. Die Entwicklung dieser beiden Ansätze und die im Rahmen der Implementierung erfolgte Kombination von symbolisch und subsymbolischen Techniken hat aber auch gezeigt, dass die Reduktion von „Informationsverarbeitung“ einmal als regelgeleitete Technik (implementiert als sequenzielle Abarbeitung von symbolisch dargestellten Problemstellungen) und dann als „adaptives Verhalten“ (implementiert in Form struktureller Plastizität auf Basis einer neuronalen Anpassung an erlernbare Problemsituationen) als methodischer Garant für die Entwicklung artifizieller kognitiver Robotersysteme gesehen werden muss. So wurde auf Basis einer kognitiven Modellierung die natürliche Kognition in Form eines Simulationsprogrammes rekonstruiert, d. h. die Bestandteile des natürlichen kognitiven Prozesses wurden nach einem dekompositionalen Prinzip in einzelne Bestandteile zerlegt, die entsprechende Komponenten mit den jeweiligen Funktionen ausimplementiert und im Rahmen der Simulation einerseits dahingehend validiert, dass jede Komponente die von ihr und damit von anderen Komponenten abgegrenzte und klar definierte Aufgabe erfüllt.

Andererseits wurden im Rahmen der Simulation nicht nur die einzelnen Bestandteile des Systems und ihre Funktionsweise zu- und miteinander validiert, sondern das gezeigte Eingabe-Verarbeitung-Ausgabe-Verhalten des Gesamtsystems einer kritischen Überprüfung unterzogen. Dabei wird in Bezug auf die Validierung das technologische Simulationskriterium für ausreichend erachtet, um sowohl theoretischen Überlegungen, die Modellierung dieser Überlegungen als auch die Implementierung dieser Modelle mittels formaler Programmiersprachen empirisch zu überprüfen. In diesem Sinne gelten die Theorien, Modelle und Lösungen als validiert, wenn sich für das gestellte Problem im Rahmen der Simulation die Daten-, Informations- und Wissensverarbeitung als „Problemlösen“ sowie Lernen als „Adaption an die Umwelt“ bewährt haben.<sup>1</sup> Insgesamt verfolgt der symbolische Ansatz dieses Buches die Exemplifikation der natürlichen Kognition durch Simulationsprogramme zur Problemlösung. Auch wenn diese Sichtweise eventuell dazu herausfordert, ist damit eine strukturelle Identität zwischen Gehirn und Geist beim Menschen und Computer und Brainware nicht gegeben. Der entwickelte subsymbolische Ansatz verhält sich in dieser Frage eher neutral, indem er lediglich die Position zulässt, dass die kognitiven Strukturen des Denkens als emergente Phänomene durch die Konnektivität neuronaler Netze und ihrer Aktivitätsmuster und Schwellenwerte entstehen. Mit anderen Worten, wird demnach ein subsymbolisch arbeitendes System „trainiert“, während dessen ein symbolisch arbeitendes System eher „programmiert“ wird. Damit lässt der symbolische und subsymbolische Anteil aber die Annahme zu, dass „Mensch“ und „Maschine“ in strukturell analoger Form, d. h. in ähnlich gelagerter Art und Weise, emergente Phänomene hervorzubringen in der Lage sind. Gleichwohl sich das Cognitive Computing mit seinen symbolischen und subsymbolischen Ansätzen bzw. Modellen im wissenschaftlichen Stadium der Grundlagenforschung befindet, sowie deren (Aus)Implementierungen zu artifiziellen kognitiven Systemen derzeit eher noch den technologischen Charakter von Prototypen zeigen, lassen sich die Erfolgsaussichten der technologischen Leistungsversprechen nach dem bisherigen Stand der Forschung durchaus positiv bewerten.

Die traditionellen Techniken der künstlichen Intelligenz, die zur bisherigen Programmierung von Robotern genutzt wurden, sind in den vergangenen Jahren in eine Sackgasse geraten. Da sie in der Regel Wahrnehmungs-, Planungs- und Aktionskomponenten getrennt betrachten, denen jeweils ein bestimmtes Modell zugrunde gelegt wird, funktionieren diese Robotersteuerungen meist nur unter ganz speziellen Umweltbedingungen und bereits geringe Abweichungen führen zu einem unvorhersagbaren Verhalten. Ein vernünftiges und stabiles Verhalten kann nur dann entstehen, wenn unterschiedlichste Verarbeitungsprozesse zwischen Wahrnehmung, Planung und Aktion auf verschiedenen Ebenen möglich sind. Ein solches System kann allerdings nicht mehr durch logische Ableitungen entworfen werden. Stattdessen bietet es sich an, die simulierte Evolution zu nutzen, um die Interoperationen sich selbst herausbilden zu lassen. Meist werden für diese Regelungsaufgaben künstliche neuronale Netze benutzt, die die Sensordaten auf Aktionen beispielsweise zur Bewegung des Roboters abbilden. Alternativ zu künstlichen neuronalen Netzen

---

<sup>1</sup> Siehe auch Ursul 1970.

kommt genetisches Programmieren zum Einsatz. Da die direkte Durchführung der simulierten Evolution auf einem echten Roboter meist sehr zeitaufwendig und schwierig aufzusetzen ist, wird die Optimierung in der Regel mit Simulationsprogrammen durchgeführt. Anschließend wird entweder das Resultat direkt auf den Roboter übertragen oder es findet dort noch eine kurze Anpassungsphase statt.<sup>2</sup>

Roboter sind gemäß klassischer Auffassung zunächst Maschinen, die mit Aktoren und/oder Sensoren ausgestattet sind und die von Computern gesteuert werden, um Aufgaben in einer Problemdomäne durchführen zu können. In dieser Hinsicht ist die klassische Robotik eine Disziplin, die sich mit dem systematischen Aufbau, der funktionellen Ausgestaltung (Programmierung) und der Anwendung aufgabenorientierter Roboter beschäftigt, die nach biologischen Vorbildern sehen, gehen (fahren, kriechen etc.), fühlen, greifen und logische Schlüsse (Inferenzen) ziehen können, so dass sie Aufgaben so durchführen, wie sie von Menschen gemeistert werden. Kognitive Robotik hingegen basiert auf den Ansätzen und Technologien des Cognitive Computing, um nicht wie üblich auf die o. a. formal-logische Fähigkeiten eingeschränkt zu bleiben, sondern auch kognitive Vorgänge und Handlungen mit einzubeziehen.

So konzentriert sich ein Teil der Forschung auf die Entwicklung „emotionaler“ Systeme, die in der Lage sind, „intuitive“ Entscheidungen schnell zu treffen, ohne dafür zeitaufwändige und umfangreiche Berechnungen vornehmen zu müssen.

Das Ziel der kognitiven Robotik besteht darin, Robotersysteme mit intrinsischen, kognitiven Fähigkeiten auszustatten, so dass sie nicht nur durch Nachahmung menschlicher Arbeitsweisen, die ihnen zugeteilte Teilaufgaben autonom erledigen, sondern auch autonom Probleme lösen. Die Forderung nach Selbstständigkeit, Lernfähigkeit und Problemlösungsfähigkeit beschränkt sich nicht auf individuelle Roboter, sondern schließt auch Gruppen bzw. Teams solcher Robotersysteme ein, die sich durch Kooperationen in Form von Multiagentensystemen selbst organisieren können.

Ein solches Multiagentensystem im Sinne des Cognitive Computing ist ein aus zwei oder mehr Agenten bestehendes daten-, informations- und wissensverarbeitendes System, bei dem die Agenten interoperieren, so miteinander in Wechselwirkung treten und gemeinsam Probleme lösen, die typischerweise nicht von einem der Agenten allein, sondern nur durch die Kooperation der Agenten gelöst werden können. Kennzeichnend für die Agenten ist, dass sie voneinander unabhängig sind, d. h. über eigene Ziele, eigenes Wissen, eigene Problemlösemethoden und eigene Schnittstellen zu anderen Agenten und zur Umwelt verfügen. Charakteristisch für Multiagentensysteme ist es ebenfalls, dass die Agenten ihre Zusammenarbeit selbst organisieren und dabei ein entsprechendes Konfliktmanagement betreiben, wenn es zu Konflikten zwischen den Zielen, Aktionen und Teillösungen der partizipierenden Agenten kommen sollte.

<sup>2</sup> Vgl. Stanley und Bak 1991.

Solche autonomen, intelligenten Robotersysteme werden durch eine Reihe von Leistungsmerkmalen festgelegt.

- Selbständige Entscheidungsfähigkeit durch Sehen, Planen, Schlussfolgern und Abschätzen der Entscheidungskonsequenzen.
- Gewährleistung der Unabhängigkeit durch Vorgaben und Regelungen.
- Selbständige Durchführung aufgabenorientierter Zielvorgaben durch die Kombination von Planungs- und Überwachungsschritten.
- Selbständiges Lernen und Beseitigen von Fehlern (Störungsbehandlung) durch maschinelles Lernen.
- Fähigkeit zur Kooperation und Interaktion mit anderen Maschinen, um nicht nur individuelle Aufgabenstellungen zu erfüllen, sondern auch Teamarbeiten durch Absprachen bzw. Verhandlungen erfolgreich bewältigen zu können.
- Bewältigung von ungeahnten Situationen (Situations- und Konfliktmanagement) und Lösen von nicht vorhersehbaren Problemen (Problemmanagement),
- Vernünftiges Handeln des Einzelnen, trotz des Vorhandenseins einer nur relativen und nicht totalen Freiheit.
- Ein freies Verhältnis zu einer Sache und zu sich selbst (Maschinelles Bewusstsein).

In diesem Sinne erweitert sich der Begriff Robotik um die multidisziplinäre Zuwendung an bisher fremde Wissensgebiete und dies im Gegensatz zu einer rein ingenieurmäßigen Roboterentwicklung. Aus dieser erweiterten Sichtweise schlagen Robotersysteme die Brücke zwischen Sensorik und Aktorik/Motorik mittels deren kognitiver Fähigkeiten. Kognitive Robotik avanciert so zu der Wissenschaftsdisziplin, die Sensorik mittels Brainware mit Aktorik/Motorik im Dienste der Ermöglichung systemischer Intelligenzprofile verknüpft. Aus Sicht der kognitiven Robotik ist ein Roboter damit ein wissensbasiertes System mit der Fähigkeit des maschinellen Lernens.

Dieses maschinelle Lernen umfasst sowohl symbolische als auch subsymbolische Ansätze. Im Unterschied zu den meisten symbolischen Ansätzen beschreiben konnektionistische Modelle adaptive, lernfähige Systeme, die kontinuierlich einer Selbstkonfiguration unterliegen. Die Adaptionsfähigkeit ist immanenter Bestandteil des konnektionistischen Ansatzes aufgrund des Konstrukts der modifizierbaren (synaptischen) Verknüpfungen zwischen gleichzeitig aktiven funktionellen Einheiten (Neuronen).

Bezogen auf die klassische Strukturierung fließen Sensordaten in einem kontinuierlichen Strom in die Datenbasis, wobei letztere auch den aktuellen Zustand des Roboters speichert. Die Wissensbasis enthält alle Regeln, Selbstmodelle, Umweltmodelle, Sensor- und Steuerungsalgorithmen. Die Brainware als solches wählt aufgrund des Zustandes der Datenbasis aus der Wissensbasis die geeigneten Sensor- und Steuerungsalgorithmen aus und aktiviert die ausgewählten Akteure. Aufgrund dieses generischen Ansatzes mittels einer Brainware

werden je nach Anwendungsbereich die unterschiedlichen Roboterarten (Industrie-, Service- und Personal Roboter etc.) unterstützt.

Gerade diese in Aussicht gestellten kognitiven Fähigkeiten zukünftiger Systeme, als auch die Möglichkeiten der durch deren Echteinsatz vorgelagerte Simulationen nehmen Einfluss auf den wissenschaftlichen Prozess als solchen. Aus dieser wissenschaftstheoretischen Perspektive betrachtet, sind vor allem Simulationen aufgrund der immer weiter fortschreitenden Möglichkeiten der Computertechnologie in allen Bereichen der Wissenschaft, Forschung und Technik zum unentbehrlichen Werkzeug geworden.<sup>3</sup> So ist in vielen Disziplinen der wissenschaftlichen Forschung eine direkte Beobachtung aus den verschiedensten Gründen nicht möglich (zu hohe Kosten, ethische Bedenken, prinzipielle naturgesetzliche Grenzen, etc.). In diesen Fällen können Beobachtungen am interessierenden Realitätsausschnitt durch Beobachtungen an Simulationen ersetzt werden. Aber auch im Bereich der industriellen Forschung und Entwicklung tragen Simulationen dazu bei, dass Kosten gespart werden können, weil beispielsweise die Einführung neuer Produktionsmethoden durch Einsatz von Robotersystemen zunächst am Bildschirm in ihrer Wirkung getestet werden kann. Simulationen sind damit Werkzeuge, die das Vorgehen nach dem Prinzip von Versuch und Irrtum wesentlich ungefährlicher und kostengünstiger machen. Der Ersatz von Beobachtung durch Simulationsergebnisse kann daher durchaus als weiterer Schritt hin zu einer pragmatischen oder gar instrumentalistischen Auffassung von Wissenschaft aufgefasst werden. Dieser Schritt hätte unter Umständen zugleich den befreienden Aspekt, dass Wissenschaft nicht länger nur durch eine wie auch immer geartete Wahrheitssuche legitimiert wäre, sondern dass das von ihr produzierte Wissen in Form von Modellen durch Technologien im Rahmen von Simulationen validiert werden könnte. Etwas pathetischer formuliert, zeigt dieser Schritt, dass nach den *Mythen*, die die mündliche Kultur beherrscht haben, und der *Theorien* und *Modelle*, welche durch die Schrift ermöglicht wurde, nun ein neuer Typ von Erkenntnis durch den Einsatz der Informationstechnologie als logische und praktische Konsequenz möglich wird, der auf *Simulation* beruht. So standen in den einzelnen Abschnitten dieses Buches unter anderem kognitive und mentale Phänomene im Fokus. Es war hierbei oft von Bewusstsein, Wünschen, Zielen, Intentionen oder auch mentalen Zuständen, Prozessen, Entitäten oder Ereignissen die Rede. Der Versuch, mit wissenschaftlichen Methoden den Gegenständen, auf die diese Begriffe bezogen werden, näher zu kommen, kann zur Folge haben, dass diese Gegenstände aus dem „Blickwinkel“ verschwinden. Der Einsatz der Simulation als Werkzeug kann diesen „blinden Fleck“ zumindest kompensieren. Auch die eingangs behandelte Differenzierung zwischen realistischer und funktionalistischer Simulation hat deutlich gemacht, dass immer auch eine Erörterung der Ziele von Wissenschaft dem Zugriff auf Theorien oder dem Einsatz von Werkzeugen vorher zu gehen hat. Ist das Ziel Erkenntnis einer wie auch immer gearteten Realität, können Simulationen unter bestimmten Umständen dann sinnvolle Werkzeuge sein.

<sup>3</sup> Siehe auch Becker 1986.

Im Verlauf der Erkenntnisgewinnung hat sich die Simulation kognitiver und mentaler Phänomene dabei als Erklärung derselben entwickelt, weil sie nicht nur die Simulation, sondern die Replikation dieser kognitiven und mentalen Zustände bedeuten. Da es aber möglich war, im Rahmen der Konzeptionalisierung und Implementierung das Zustandekommen der replizierten mentalen Zustände im Rahmen des Symbolverarbeitungs- oder des subsymbolischen Ansatzes (Konnektionismus) zu erklären, ist das Zustandekommen mentaler Zustände zumindest in artifiziell kognitiven Systemen einer allgemeinen Theorie von Kognition näher gekommen. Im Rahmen der Validierung durch Simulation und dann durch Einsatz in Echtzeit wird letztlich der unabdingbaren Forderung Rechnung zu tragen sein, dass alle theoretischen Annahmen, die in einer Simulation enthalten sind, sich in der empirischen Überprüfung bewähren müssen. Nachdem die Simulationen bzw. die sie realisierenden Programme als Theorie aufgefasst wurden, stellt diese am Ende des Buches eine Theorie artifizieller kognitiver Systeme, aber nicht die Theorie aller kognitiven Systeme dar. Nicht mehr, aber auch nicht weniger!

Diese Bescheidenheit soll auch als didaktischer Warnhinweis verstanden werden. Gerade in den letzten Jahren haben „marktschreierische“ Äußerungen, dass in Zukunft beispielsweise nicht mehr von Schmerzen oder Wille, sondern von vom Feuern der C-Fasern gesprochen werden sollte, Gegensätze wie zwischen „Erklären“ und „Verstehen“ erst entstehen lassen, die nunmehr auch im Rahmen des Cognitive Computing mühevoll wiederaufgebaut werden müssen. Ein weiterer Gegensatz zeigt sich auch in der „klassischen“ Argumentation, dass nur die naturwissenschaftlichen Theorien erklärenden Charakter und prognostische Kraft besitzen, während die geisteswissenschaftlichen Methoden weder erklären noch vorhersagen können, sondern nur beschreibend verstehen (Hermeneutik).

Dieses Wenige wird Ausgangspunkt für eine fortschreitende Dynamisierung von Theorien sein. Die Theoriodynamik erhält ihren wesentlichen Impuls aus der weiteren Technologisierung, die zunehmend Bedeutung für die Forschung erlangt. In Form der Computersimulation zur Generierung und Überprüfung von Theorien fundiert sie darüber hinaus die zukünftige Transdisziplin „Cognitive Computing“ als eigenständigen Forschungsbereich. Insofern lautet das wissenschaftstheoretische Fazit, dass sich die Technologie des Cognitive Computing, eingebettet in kognitive Robotersysteme immer mehr zu einer zur Technik gewordenen Kognitionstheorie entwickeln wird. Damit hat dieses Buch aber auch gezeigt, dass sich das Cognitive Computing und die Kognitive Robotik zu einer „Transdisziplin“ entwickelt, die zunächst ganz ohne ein zentrales und damit hart „verdrahtetes“ Paradigma oder einem Paradigmenwechsel auskommt, da andere Momente, unter anderem die Technologisierung, die Theoriodynamik bestimmen.

Dies zeigt sich auch durch die gegenwärtige Situation der Forschung, die dadurch gekennzeichnet ist, dass sie sich zur Erkenntnisgewinnung der Computersimulation als theoriebildende und -validierende Methoden bedienen wird, um kognitive Leistungen zu modellieren, dann zu operationalisieren und in dieser simulativen Abbildung als lauffähige Programme zu validieren. Aufgrund der „harmonisierenden“ Bedeutung der Simulation, indem sie die teils

---

komplementären Ansätze in der Lauffähigkeit des Systems vereint, wird sich kein Paradigmenwechsel ereignen, sondern die Komplementarität unterschiedlicher Forschungsansätze zum Tragen kommen.

Vielmehr erweist sich der Paradigmenbegriff als viel zu statisch und nur an revolutionären Veränderungen orientiert, um die eher evolutionären, permanenten theoriekonstitutiven technologischen Innovationen des Cognitive Computing und der Kognitiven Robotik erfassen und abbilden zu können. Gerade dadurch wird aber eine neue Forschungssituation entstehen, indem sich komplementäre Strukturen ausbilden, in denen es ein Mit- und Nebeneinander, eine Art Koexistenz der Ansätze geben wird und die die gegenwärtig noch teilweise verfolgten Abschottungstendenzen überwinden werden. Insgesamt werden neue Überlegungen und die Entwicklung weiterer technologischer Anwendungen auch zu keiner vollständigen Substitution bisheriger Modellvorstellungen, sondern eher zu einer Pluralisierung der Theorien und Hypothesen führen.

Dieser „weiche Gestaltwandel“ in Form eines schleichenden Übergangs von Vorstellungen schließt allerdings einen zukünftigen Paradigmenwechsel nicht *a priori* aus. So kann es sich sehr bald herausstellen, dass beispielsweise der Begriff der „Informationsverarbeitung“ als derzeitiges Paradigma zu wenig aussagekräftig erscheint und man damit dieses Paradigma ggf. verlassen muss.

Alles zusammen, Theorien, Modelle und deren Simulationen auf Basis lauffähiger Programmsysteme werden den Computer und den Roboter als Trägermedium in mehrfacher Hinsicht zu einer bisher qualitativ einzigartigen „Brainware“ ausgestalten. Letzteres begründet sich unter anderem auch darin, dass der Computer bzw. das Robotersystem nicht nur Algorithmen einfach abarbeitet, sondern dass er theorie- bzw. erkenntnisgenerierend eingesetzt werden wird, also auf mehreren Ebenen den Forscher in seiner mühevollen Aufgabe, über Kognition nachzudenken, mit seiner artifiziellen Kognition zu unterstützen vermag. Die wissenschaftstheoretische Implikation zeigt sich auch darin, dass in einer wissenschaftlichen Tätigkeit, die immer mehr auf dem Einsatz von Technologien basiert – und zwar nicht nur, um formalisierte Theorien einer Überprüfung zu unterziehen, sondern immer häufiger auch zur Theoriegenerierung selbst – dem Prozess der Implementierung innerhalb der Forschungsheuristik immer mehr Bedeutung zukommt.<sup>4</sup> Es sind damit nicht nur pragmatische Handlungsentscheidungen, welche innerhalb des Wissenserwerbs eine Rolle spielen, sondern durch die Verwendung von Technologie und mit dem Einsatz von Computersystemen im Rahmen von Simulationen kommt der „simulativen Vernunft“ eine entscheidende Bedeutung zu.<sup>5</sup> Damit soll an dieser Stelle lediglich zum Ausdruck gebracht werden, dass die theoretische und die

---

<sup>4</sup> Siehe auch Baudrillard et al. 1989.

<sup>5</sup> Siehe auch Haugeland 1987.

praktische Vernunft um ein Element des Spiels, der Exploration und der Simulation erweitert werden muss.

Die wissenschaftliche Ausarbeitung dieses Aspekts bleibt einer weiteren, dann aber philosophischen Arbeit vorbehalten.

Durch diese Erweiterung lässt sich Wissenschaft als ein regelbasiertes Entscheidungs- und Handlungssystem rekonstruieren, das sich über Simulationen aufbaut, indem die Formulierung und Überprüfung bestimmter wissenschaftlicher Hypothesen und Theorien anhand von Simulationsprogrammen erfolgen kann, denen dann im Ergebnis fundamentale Bedeutung zukommen können. Technologisch induzierte Wissenschaft vollzieht sich demnach immer mehr in einem Raum potentieller Möglichkeiten und Realisierungen, so dass zwischen dem Schritt von der theoretischen Vernunft zur praktischen Vernunft, die simulative Vernunft als technologische Realisierung von Hypothesen und Theorien mit ihrem Kriterium der Simulierbarkeit als erkenntnisgewinnender Zwischenschritt gegangen werden kann.<sup>6</sup> Dieser, der Erkenntnisgewinnung dienende Zwischenschritt, wirkt sich auch auf den Aspekt des Wissens aus, so dass in der Simulation die Wissensarten bzw. die Wissensqualitäten mit dem instrumentellen Wissen in einer neuen Art und Weise konvergieren, um dadurch eine neue Wissensform entstehen zu lassen. In diesem neuen Simulationswissen verschmelzen Intuition, Beschreibung, Modellierung und handlungsorientiertes Implementieren dergestalt, dass das, was simuliert wird, als beschreibbar und – da es funktioniert – als validierbar gilt. Die Simulation als ausführbares Modell liefert die Erklärung, als auch – bei positivem Ausgang der Validierung – den Beweis für das implementierte Verfahren. Somit ist Simulation Erklärung und Beweis zugleich und das daraus gewonnene Simulationswissen das Ergebnis. Wenn also in Wissenschaft bisher Experiment und Interpretation bzw. Wahrnehmung noch weitgehend auseinander fielen, so fusionieren sie nun in der Computersimulation auf eine neue Art und Weise, um dann als „Kernfusion“ einen neuen Wissenstypus entstehen zu lassen. Dieser Wissenstypus in Form von Simulationswissen ist vor allem dadurch gekennzeichnet, dass Theorie und Praxis sowie Modell und Technologie verschmelzen, wenngleich diese „Kernschmelze“ auch zur Besinnung auffordert. So besteht die Gefahr, dass rechnergestütztes Forschen sich ausschließlich auf das Kriterium der Implementierbarkeit stützt und dabei die dadurch bedingte technologische Zirkularität verdrängt wird.

Diese Zirkularität zeigt sich unter anderem darin, dass der Computer sowohl zur Entwicklung als auch zur Überprüfung von computertechnischen Experimenten und ihren theoretischen Annahmen herangezogen wird.

Weiterhin kann der technologiegetriebene Forscher der Versuchung unterliegen, durch die Verschmelzung von Theorie und Technik und der damit einhergehenden Vergegenständ-

---

<sup>6</sup> Siehe auch Weizenbaum 1978.

lichung der Logik, die Machbarkeits- und Realisierungsüberlegungen in den Vordergrund seines wissenschaftlichen Tuns zu stellen.<sup>7</sup> Damit würde die Technologie in kognitions-wissenschaftliche Fragestellungen eindringen, dort selbstreferentielle Züge annehmen, die Fragen nur innerhalb der eigenen Logik formulieren, um die dann demzufolge auch nur in diesem Rahmen bzw. in diesen engen Grenzen zu beantworten. Die Simulation avanciert so zwar zum Wissenschafts- und Fortschrittskriterium, welches allerdings wiederum nur technologisch eingelöst werden kann und deshalb zirkulär auf sich selbst beschränkt und bezogen bleibt. Diesem, durch die Technologisierung bedingten Zirkel an dieser Stelle zu entkommen bzw. die Beschränkung und Selbstbezogenheit aufzulösen, ist und bleibt weiterhin die Aufgabe des menschlichen Denkens.

---

## 7.2 Implikationen einer kognitiven Robotik

Derzeit finden in der Öffentlichkeit, der Industrie und Forschung Überlegungen statt, die einen *Paradigmenwechsel* in der Industrie einleiten soll. Treiber dieses Paradigmenwechsels ist die Tatsache, dass in der nächsten Dekade auf der Basis Cyber-Physischer Systeme (Internet of Things) neue Geschäftsmodelle möglich, heute bereits entwickelt und in der unmittelbaren Zukunft folglich realisiert werden. Cyber-physikalische Systeme (cyber-physical systems, CPS) gelten als eine neuartige Technologie für „EveryWare“ und fungieren als Basistechnologie für das Internet der Dinge und Dienste. Sie steuern von der Cloud aus Dinge der realen Welt, nehmen Sensordaten auf und regeln und optimieren damit Informations-, Material-, Güter-, und Ressourcenflüsse. Sie benötigen als Trägermedien entweder eingebettete Chips oder ganze Plattformen für sicherheits- und qualitätsbehaf-te Software.<sup>8</sup>

Laut namhaften Vertretern aus Politik, Wirtschaft und vor allem Wissenschaft soll Deutschland hierbei eine führende Rolle übernehmen, sprich „die erste Geige“ spielen. Um dieser Rolle gerecht zu werden, gilt es, in die Entwicklung und Integration neuer Technologien und Prozesse zu investieren. Produktionsstandort bleiben heißt demnach, sich heute schon fit zu machen für die vom Internet getriebene 4. oder x. industrielle Revolution.

Unabhängig davon, ob der Begriff eines Paradigmenwechsels angemessen ist, erscheint diese Entwicklung zumindest als logische Konsequenz der industriellen Entwicklungsge-schichte.

Die erste industrielle Revolution, die Einführung mechanischer Produktionsanlagen Ende des 18. Jahrhunderts und die zweite industrielle Revolution, die arbeitsteilige Massenproduktion von Gütern mit Hilfe elektrischer Energie (Fordismus, Taylorismus) seit der Wende zum 20.

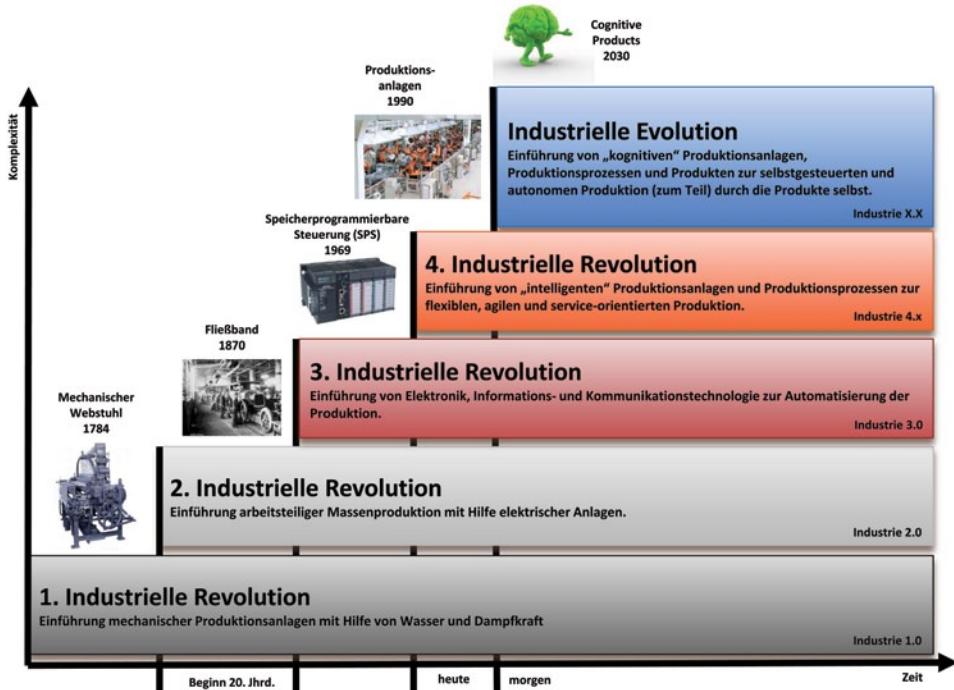
---

<sup>7</sup> Siehe auch Ropohl 1991.

<sup>8</sup> Vgl. Schäfer 2003.

Jahrhundert, mündeten ab Mitte der 70er Jahre in die bis heute andauernde dritte industrielle Revolution, mit der durch den Einsatz von Elektronik bzw. Informations- und Kommunikationstechnologie getriebenen weiteren Automatisierung von Produktionsprozessen.

Derzeit wird der Transformationsprozess in Richtung Automatisierung von Produktionsprozessen durch die Entwicklung intelligenter Überwachungs- und autonomer Entscheidungsprozesse dahingehend ausgestaltet, dass Unternehmen und ganze Wertschöpfungsnetzwerke sich in Echtzeit steuern und optimieren lassen. Aufbauend auf diese Entwicklung zeichnet sich als logische Konsequenz eine vierte industrielle Revolution ab, indem durch eine „Intelligenzierung“ von Produktionsanlagen und industriellen Erzeugnissen bis hin zu Alltagsprodukten durch eine Brainware mit integrierten Speicher- und Kommunikationsfähigkeiten, Funksensoren, eingebetteten Aktuatoren und intelligenten Softwaresystemen eine Brücke zwischen virtueller („cyber space“) und dinglicher Welt entsteht. Diese Brücke wird eine wechselseitige Synchronisation zwischen digitalem Modell und der physischen Realität gestatten. Im Rahmen dieser Synchronisation und durch die sukzessive Ausreifung der produktintrinsischen Brainware wird dies in der Industrie eine Evolution in Richtung eines Paradigmenwechsels einleiten, bei dem das zu produzierende Produkt erstmals eine aktive Rolle im gesamten Produktlebenszyklus übernimmt und diesen Zyklus damit aktiv mitbestimmt bzw. mitgestaltet. Konkret bedeutet dies, dass in der Zukunft weniger eine zentrale Steuerung, sondern quasi das Produkt „bestimmt“, wie es in den einzelnen Fertigungsschritten bearbeitet werden muss. Das entstehende Produkt steuert somit den Produktionsprozess selbst, überwacht über die eingebettete Sensorik die relevanten Umgebungsparameter und löst bei Störungen entsprechende Interventionen aus. Das entstehende Produkt ist gleichzeitig Beobachter und Akteur, das Produkt „interoperiert“ sozusagen mit seiner Umwelt. Die Ausstattung der Produkte mit einer solchen Brainware führt zum einen zu einer vertikalen Vernetzung eingebetteter Systeme mit betriebswirtschaftlicher Steuerungssoftware, was zunächst das Potenzial der Entwicklung neuartiger Geschäftsmodelle eröffnet. Zum anderen ergeben sich dadurch erhebliche Optimierungspotentiale im Bereich der Logistik und der Produktion. Indem die Produkte als autonome Produkte direkt am Ort des Geschehens (Point of Production) interoperieren, ergeben sich kürzeste Reaktionszeiten bei Störungen und eine optimale Ressourcenutzung in allen Prozessphasen. Die Produkte werden mit allen relevanten Prozessdaten versorgt und können so just-in-time „entscheiden“, wie sie sich unter Abwägung der aktuellen Produktionssituation verhalten. Damit wird es beispielsweise auch möglich sein, nicht nur den ökonomischen, sondern auch den besonderen ökologischen Anforderungen einer „grünen Produktion“ besser gerecht zu werden. Neben der betrieblichen Prozessoptimierung ergeben sich auch Potenziale für neuartige und „verrechenbarer“ Dienstleistungen, da die Produkte selbst über intelligente Dienste verfügen werden. Letzteres wird nicht nur die derzeitigen Produktlebenszyklen nachhaltig beeinflussen, sondern auch neue Produktplanungs-, -steuerung und Fertigungsprozesse erfordern. Sowohl das Ziel ressourcensparender, weil effizienter Produktionsprozesse kann damit angestrebt, als auch der Tatsache, dass damit das Internet der Dinge bzw. das Internet der Entitäten zum Garanten und Treiber für neue „smarte“ Geschäftsmodelle, der Entwicklung intelligenter und damit



**Abb. 7.1** Entwicklungspfad der industriellen Revolution und Evolution

„smarter“ Produkte und dem Bereitstellen mehrwertiger Dienstleistungen avanciert, entsprochen werden (Abb. 7.1).

Bei all dem kommt der Informatik eine zentrale Bedeutung zu. Die Informatik hat dabei die kritische Phase einer Wissenschaft erreicht, indem mit Theorie (Modelle) und Praxis (Technologien) die Konstruktion solcher artifiziellen Systeme möglich wird, die kognitive Fähigkeiten aufweisen. Das Zeitalter des Cognitive Computings hat begonnen und wird sich sowohl als Jahrhundertwissenschaft im Kanon der Wissenschaftsdisziplinen etablieren als auch im praktischen Einsatz sich bewähren. Gerade als wissenschaftliche Disziplin wird sie über einen längeren Zeitraum wissenschaftliche und öffentliche Aufmerksamkeit auf sich ziehen, indem sie

- erfolgreich gegen „klassische“ und gegen ihre eigenen Grundlagen verstößt,
- mit diesen und auf ihnen aufbauend neue, „evolutionäre“ Erkenntnisse gewinnt,
- diese in praktische Anwendungen (Simulationen) umsetzt,
- dabei ganz neue Systeme, Begriffe, Teilchen, Stoffe, Zustände, Prozesse, Komponenten und Lösungen „synthetisiert“ entwickelt,
- also Natur und Realität konstruiert und umkonstruiert, aber
- eben dadurch auch an die Grenze des Erlaubten stößt und
- diese Grenze vielleicht sogar überschreitet.

Diese Wissenschaftsdisziplin wird nicht nur neue Einsichten bringen, sondern auch viele praktische Anwendungen zulassen. Sie wird kognitive, intelligente Systeme nicht nur durchschauen, sondern auch verändern. Denkbar sind auch Systeme, die es in der Natur bisher nicht gab. Eine erfolgreiche Bereicherung der Informatik durch Cognitive Computing wird das gesamte Wissenschaftsgefüge im Allgemeinen und die Informatik im Speziellen verändern. Die Grenzen zwischen Natur- und Geisteswissenschaften würden, falls es sie je gab, ein weiteres Mal durchbrochen. Typische mentale Merkmale und Leistungen – Sprechen, Erkennen, Denken, Bewerten, Urteilen, Entscheiden – werden zu Anforderungen des Cognitive Computing.<sup>9</sup> Auch philosophische Probleme, wie das der Willensfreiheit oder das Leib-Seele-Problems, bisher ungelöst und augenblicklich unlösbar, können der empirischen Forschung (Modellierung) zugänglich, können womöglich (Implementierung) gelöst werden.<sup>10</sup> Insofern wird die durch Cognitive Computing erweiterte Informatik den Menschen eine weitere Kränkung zufügen. Aber auch das ist typisch für eine Jahrhundertwissenschaft. Insofern wird sich die Informatik den kognitiven Fähigkeiten des Menschen nicht nur vermehrt zuwenden, sondern diese Fähigkeiten auch in artifiziellen Systemen abbilden können. Die Gewichtung der Informatik verlagert sich damit in Richtung kognitiver Lösungsansätze. Dabei arbeiten Wissenschaftler der unterschiedlichsten Fachrichtungen zusammen, um soft- und hardwaretechnische Systeme mit neuen, definierten Eigenschaften zu konzeptionalisieren und zu implementieren:

- So wird neben der *Philosophie* des Geistes<sup>11</sup> auch die Naturphilosophie<sup>12</sup> fruchtbare Beiträge liefern können.<sup>13</sup> Grundlegend sind die Sprachphilosophie,<sup>14</sup> die Erkenntnistheorie<sup>15</sup> und die Wissenschaftsphilosophie, die hier unbedingt mit einbezogen werden müssten.<sup>16</sup>
- Die *Psychologie* bietet eine Vielzahl von Experimenten und Ergebnissen, um zu exakten, empirisch messbaren Begriffen wie Emotion, Motivation usw. zu gelangen. Die klinische *Neuropsychologie* hat eine Vielzahl von experimentellen und klinisch-empirischen Befunden über den Zusammenhang von Gehirn und Erleben und Verhalten deutlich gemacht. Für die Psychologie im Allgemeinen, die Psychiatrie oder die Psychosomatik erscheint aber neben ihrer neurobiologischen Ausrichtung auch ein eigenständiger Weg der Theoriebildung sinnvoll zu sein, der die neuere theoretische Psychologie und die Systemwissenschaft berücksichtigt.

---

<sup>9</sup> Siehe auch Allman 1990.

<sup>10</sup> Siehe auch Bungo 1984.

<sup>11</sup> Vgl. Pauen 2001.

<sup>12</sup> Vgl. Bartels 1996.

<sup>13</sup> Vgl. Bieri 1993.

<sup>14</sup> Vgl. Runggaldier 1990.

<sup>15</sup> Vgl. Klaus 1966.

<sup>16</sup> Vgl. Kanitscheider 1984.

- Die *Biologie* hat durch die Biochemie viele ihrer Phänomene auf molekularbiologische Mechanismen zurückführen können und gilt als die zentrale Disziplin in Form der *Neurobiologie*. Besonders relevant sind die neueren genetischen und molekularbiologischen Befunde zu psychiatrischen Erkrankungen. Hier gilt es, in Zukunft die Defizite an Methodenkritik und Mängel an sprachlicher und argumentativer Präzision anzugehen.
- Die *Mathematik* verdeutlicht die Möglichkeiten der Berechenbarkeit sowie die Formalisierbarkeit von nicht-linearen Phänomenen, von Selbst-Referenz oder Komplexität (Graphentheorie). Sie liefert den mathematischen Untergrund mit der Algebra, Matrizenrechnung und der Graphentheorie.<sup>17</sup>
- Die *Informatik* hat heute noch die Form des systemwissenschaftlich-kybernetischen Ansatzes der 1960er und 1970er Jahre. Sie verfügt über das Know-how beim Umgang mit komplexen Datenbanken und Software, mit denen solche Probleme anwendergünstig gelöst werden können (Number Crunching). Durch die Systembiologie bekommt allerdings der systemtheoretische Ansatz gegenwärtig wieder einige Schubkraft. Gerade die naturwissenschaftliche Systemforschung ermöglicht einen neuen Brückenschlag zwischen den an der Gehirn-Geist-Debatte beteiligten Disziplinen.<sup>18</sup>
- Die *Physik* verfügt über die längsten Erfahrungen mit Experimenten und Theorien und gilt heute als Königsdisziplin der Naturwissenschaften und auch als Basisdisziplin der Biologie und Chemie. Die Physik hat in dieser Hinsicht bereits seit Hunderten von Jahren einen breiten Bereich der impliziten und expliziten *Wissenschaftsphilosophie* ihres Faches aufgebaut.
- Die *Sprachwissenschaften*, die *Kulturwissenschaften*, die *Sozialwissenschaften* und auch die *Rechtswissenschaften* sind ebenfalls einzubinden.

Der Kanon der interessanten und interessierten Fächer lässt sich sicherlich noch weiter spannen. Damit diese Kommunikationen aber nicht ausufern und damit konturlos werden, weil die disziplinspezifische Methodenreflexion zu kurz kommen dürfte, muss hier die Philosophie mit ihren Spezialausrichtungen als Integrationsinstanz wirken. Die Institutionalisierung einer derartig multidisziplinär fundierten Wissenschaftsdisziplin wäre von großem Wert für ein integrales Verständnis von menschlicher und artifizieller Erkenntnis.

So regt Neurobiologie zur Modellbildung neuer Typen von Computern und Algorithmen an. Generell werden in Zukunft die biologischen Konzepte, insbesondere jene, die die Evolution betreffen, die Entwicklung der Computerwissenschaft im Allgemeinen und des Cognitive Computing im Speziellen, beeinflussen. Am Beispiel der Neurobiologie zeigt sich, dass gerade diese wechselseitige Beeinflussung, die Synchronisation Konvergenz und einst auseinanderliegender Wissenschaftsdisziplinen zu neuen Synthesen im Cognitive Computing und damit in der Kognitionswissenschaft führen werden.

<sup>17</sup> Siehe auch Brill 2001.

<sup>18</sup> Siehe auch Drieschner 1981.

Vor allem aufgrund der Entwicklung der Wissenschaften, wie insbesondere der Psychologie und Systemforschung und selbstverständlich der Neurobiologie, scheint es daher sinnvoll zu sein, ein eigenes inter- und transdisziplinär fundiertes Gebiet einer Wissenschaftsphilosophie, oder besser: *Technologiephilosophie*, zu etablieren.<sup>19</sup> Die Basis dieser neuen Disziplin gilt es in der Philosophie zu etablieren, die mit ihrer bisherigen wissenschaftsphilosophischen Ausrichtung eine fruchtbare Plattform darstellen könnte, um die einzelwissenschaftlichen Arbeitsansätze mit den klassischen philosophischen Positionen und Methoden in Beziehung zu setzen. Auf diese Weise könnten die empirischen und die theoretischen Wissenschaften in die philosophische Reflexion eingebunden werden. Der Nutzen eines derartig weit gespannten und ausgewogenen Diskurses wird äußerst vielfältig sein. Insofern stellt dieses Buch nicht nur ein Plädoyer für die Institutionalisierung eines philosophisch multidisziplinär fundierten Diskurses zur Kognitionsthematik in Form einer um Kognitions- und Simulationsaspekte erweiterten Technologiephilosophie dar, sondern liefert hierfür auch eine mögliche Grundlage. Ziel einer solchen erweiterten *Technologiephilosophie* ist dabei u. a. auch die Entwicklung einer neuartigen Methodik, die es ermöglicht, empirische Fakten und theoretische Begriffe nicht nur nebeneinander zu stellen, sondern direkt systematisch aufeinander zu beziehen. Zur Bearbeitung der Problemkomplexe des Cognitive Computing und der kognitiven Robotik ist die Entwicklung einer neuen transdisziplinären Methodik notwendig. Im Rahmen einer solchen Methodik können philosophische Theorien zur Kognition und neurowissenschaftliche Hypothesen zur Funktionsweise des Gehirns direkt und systematisch ohne Verlust der jeweiligen Gelungsansprüche und Bedingungskomplexe aufeinander bezogen werden. Und darüber hinaus muss eine solche Philosophie versuchen, philosophische Theorien und neurowissenschaftliche Hypothesen nicht nur intuitiv aufeinander zu beziehen, sondern systematisch zu vollziehen, d. h. nach bestimmten methodischen Regeln, die speziell in Hinsicht auf die Technologisierung durch das Cognitive Computing entwickelt sind.

Um diesen Fortschrittsprozess zu intensivieren und konstruktiver zu gestalten, sind allerdings auch einige institutionelle Veränderungen vor allem im Wissenschafts- und Lehrbetrieb notwendig und sinnvoll, wenngleich in einem ersten Schritt unter Umständen nur eine Aktualisierung klassischer Arbeitsansätze erforderlich erscheint.

Eine solche *Technologiephilosophie* wird auch dadurch begründet, dass die artifiziellen Systeme sich durch neue Komponenten sowie neuartige Technologien derart ausgestalten, dass solche intelligenten Leistungen möglich sind, die bisher nicht möglich erschienen. Dabei nimmt die Bedeutung evidenzbasierter Vorgehensweisen in Modellierung und Implementierung zu. Vor allem wird diese Vorgehensweise weg vom bisherigen Analysieren und Modifizieren hin zum Synthetisieren und Konstruieren führen. Mit dieser Vorgehensweise und mit dem Zugriff auf entsprechende Technologien wird angestrebt, die „Brainware“ in ihren Merkmalen so gezielt zu verändern, dass sie, mit grundlegend neuen Eigen-

---

<sup>19</sup> Siehe auch Hoyningen-Huene und Hirsch 1988.

schaften versehen, kognitive Leistungen vollbringen. Insofern ist das spezifische Merkmal des Cognitive Computing und der kognitiven Robotik, dass sie bestehende software- und hardwaretechnische Systeme mit kognitiven Fähigkeiten anreichert und damit wesentlich verändert oder aber durch Orchestrierung solcher Cognitive Computing-Komponenten zu neuen Systemen kombiniert. Dabei können unter Umständen Eigenschaften entstehen, wie sie in den bisher vorkommenden Systemen bislang nicht bekannt sind. Damit verbunden ist das Ziel, Soft- und Hardwarekomponenten (*SoftBricks*, *HardBricks* und *BrainBricks*) zu konstruieren, die sozusagen als „Chassis“ im Rahmen von Frameworks ein Mindestmaß an unentbehrlichen Funktionalitäten zur „fabrikmäßigen“ Entwicklung intelligenter Systeme nach dem Baukastenprinzip zur Verfügung stellen. Insofern verfolgt das Cognitive Computing und die kognitive Robotik als ein weiteres Ziel die Entwicklung von sogenannten Minimalkomponenten, die essenzielle kognitive Funktionen als Dienste anbieten.

Das Cognitive Computing stellt dabei eine konsequente Weiterentwicklung bestehender Methoden und Technologien der Künstlichen Intelligenz und des Künstlichen Lebens dar und besitzt ein großes Innovationspotenzial, von dem sowohl die Grundlagenforschung als auch die industrielle Anwendung und damit die allgemeine Robotik profitieren werden. Dabei wird der Erfolg des Cognitive Computing maßgeblich davon abhängen, inwieweit es gelingen wird, die verschiedenen Forschungsdisziplinen zusammenzuführen und in einer adäquaten Infrastruktur zu integrieren bzw. zu bündeln, um Synergien zu erzeugen.

Eine unsystematisierte Behandlung der angesprochenen Probleme und Themen führt zu einer unproduktiven beziehungsweise kontraproduktiven Verwirrung. Dies wird durch die Faktenschwämme und die Explosion der Nachrichten aus den einzelnen Forschungsdisziplinen sogar eher verstärkt. Der Aufbau und die Etablierung einer interdisziplinären und transdisziplinären *Diskurs-Plattform* zum Cognitive Computing scheint daher sehr dringlich zu sein. Letzteres ist allerdings nur dann zielführend, wenn eine relativ autonome Institution dieses Projekt trägt, die vielseitig disziplinär mit den Wissenschaftsdisziplinen vernetzt ist. Ein im Wissenschaftsbetrieb etabliertes Institut oder eine eigens eingerichtete Fakultät könnte eine solche Plattform darstellen und den Austausch zwischen Industrie, Wissenschaft, Gesellschaft, Politik und Ingenieuren gewährleisten. Darüber hinaus könnte eine solche Institutionalisierung Erfahrung und Erkenntnis verbinden und somit zur Entwicklung intelligenter Lösungen beitragen. Die ökonomisch interessanten Möglichkeiten einer solchen Etablierung des Cognitive Computing beziehungsweise deren Institutionalisierung liegen in der erhöhten Produktivität durch die Verbesserung der Entwicklungsprozesse, der Gewinnung neuartiger Lösungen und Produkte, der Beschleunigung von Entwicklungszeiten durch Standardisierung der soft- und hardwaretechnischen Bauteile und der Etablierung neuer Entwicklungs- und Lösungskonzepte. Zu guter Letzt gilt es, durch Information und Kommunikation eine Transparenz zu schaffen, die zur Akzeptanz dieser neuen Forschungsrichtung und deren Ergebnisse in der Öffentlichkeit, der Forschung und der Wirtschaft beiträgt.

Kognitive Robotik wird dadurch eine weitere Generation neuer, weil intelligenter, Robotersysteme ermöglichen. Derzeit lassen sich drei Robotergenerationen ausmachen, wobei jede dieser Generationen sich durch ein ihr eigenes Programmierparadigma auszeichnet: Erste Generation (Teach-in-Verfahren), zweite Generation (Roboter-orientierte Programmierung) und dritte Generation (Aufgaben-orientierte Programmierung). Aufgabenorientierte Robotersprachen sind beispielsweise Hochsprachen, bei denen sich alle Angaben auf die zu bearbeitenden Objekte und nicht auf den Roboter beziehen. Der „Compiler“ dieser Sprache ist ein Aufgabenplaner. Er übersetzt die symbolischen, objektbezogenen Anweisungen in explizite roboterorientierte Anweisungen. Die neue Generation der kognitiven Roboter wird neben der Simulationstechnik auf die Technologien des Cognitive Computing zugreifen. Dieser Einbezug der Technologien des Cognitive Computing wird nachhaltig zu neuen Architekturen von Robotersystemen führen, was dann wiederum von zentraler Bedeutung für die Ausgestaltung dieser Systeme mit intrinsischer Intelligenz sein wird. Die Kernvoraussetzung einer solchen artifiziellen Intelligenz im Bereich der Robotik ist zum einen die Fähigkeit, selbstständig denken und damit aus sich heraus Schlüsse ziehen zu können. Zum anderen wird maschinelles Lernen und sensorgestütztes Handeln, das Interoperieren, das Bewegen in einer dynamischen Umwelt, deren Zustände sich permanent ändern und daher unsicher sind, erst ermöglichen.

Durch die multiplikative Verknüpfung von systemischem Denken, Lernen und Handeln wird es notwendig sein, die bisherige Architektur von Robotersystemen zu überdenken und in neue Hierarchie-Ebenen (Abstraktionsebenen) aufzuteilen, um im Sinne eines kognitiven und wissensbasierten Systems in jeder Ebene die bisherige Dreiteilung Aufgabenplaner (Inferenzmaschine, Kontrollkomponente), Umweltmodell (Wissensbasis) und Sensordatenverarbeitung (Datenbasis) neu zu arrangieren. Auch wird die bisherige Unterscheidung in drei Entscheidungsarchitekturen:

- zentrale Architekturen: eine Entscheidungsinstanz,
- dezentrale Architekturen: mehrere Entscheidungsinstanzen, die miteinander kooperieren und
- zelluläre Architekturen: mehrere Entscheidungsinstanzen, die auch konkurrieren können,

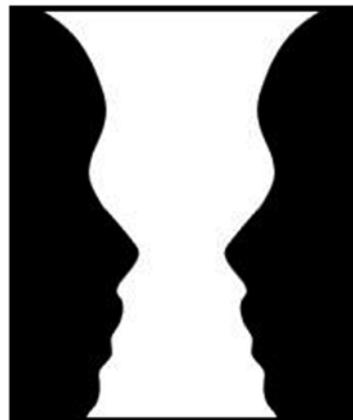
eher aufweichen, indem die Entscheidungsarchitekturen fließend ineinander übergehen. Ein spezieller Robotertyp der neuen Generation, der sogenannte konnektionistische Roboter, folgt beispielsweise eng dem biologischen Vorbild des Verhaltens von natürlichen Zellen. Eine solche konnektionistische Zelle ist dabei eine elementare Einheit, die mit Rechen- und Speicherleistung ausgestattet ist und sozusagen als zellulärer Automat den Knoten eines neuronalen Netzes darstellt. Diese Zellen werden nach ihrer funktionalen Leistungsfähigkeit klassifiziert in Sensor-, Aktor- und Brainware-Zellen. Zellen aggregieren zu Zellverbänden (zelluläre Module), und diese zellulären Module werden wiederum zu solchen funktionalen Einheiten zusammengefasst, die autonom sind (Task-Module).

Zelluläre Roboter werden vor allem dafür eingesetzt, Erkenntnisse über notwendige Hierarchieschichten, über effiziente und robuste Autonomiebausteine zu gewinnen und um Fragen der Selbstorganisations-Mechanismen und der dabei auftretenden Emergenz von Intelligenz zu beantworten.

Als artifizielle Emergenz werden Eigenschaften komplexer, insbesondere dynamischer Systeme bezeichnet, die nicht Eigenschaften einzelner Bestandteile des Systems sind, sondern aus deren Wechselwirkungen resultieren. Insbesondere können die kognitiven Prozesse und deren Resultate als emergente Eigenschaften des kognitiven Systems aufgefasst werden. Eine solche Anschauung bezeichnet man als emergentistischen Materialismus.

Ein weiterer Zweig, die verteilte kognitive Robotik, geht von einem Top-down-Ansatz aus. Die einzelnen Roboter sind dabei intelligent, koordinieren und kooperieren. Hier charakterisiert der Begriff dar Koordination die zeitlich synchronisierte Zusammenarbeit von intelligenten Robotersystemen im Dienste einer Aufgabendurchführung bzw. Problemlösung. Die Kooperation zwischen autonomen Robotern als Multiagentensysteme durch Verhandlungen (Communication) und Vertrag (Contracting) wird die höchste Form der Selbstorganisation darstellen. Sie hat zum Ziel, schwierige Aufgaben und Problemstellungen durch Arbeitsteilung zu lösen (task-sharing), an Resultaten zu partizipieren (result-sharing), gewonnenes Wissen zu verteilen (knowledge sharing) oder Betriebsmittel gemeinsam zu nutzen (resource-sharing). Kooperierende Robotersysteme müssen Wissen besitzen über andere Roboter (Absichten, Fähigkeiten, etc.), über verteilte Entscheidungsfindungen (Konfliktbehandlungsmuster, Rollen, etc.), über die Organisation (Zuständigkeiten, Aufgabenzuweisungen, verfügbare Ressourcen, Gruppenwissen, etc.), über die Umwelt (Barrieren, Hindernisse etc.) und vor allem über die Kommunikation (Sprache, Verhandlungsmuster, Medium, etc.). Kognitive Robotik wird auch die Frage beantworten, wie aus einer Menge von Robotersystemen heraus eine höhere Gruppenintelligenz auftauchen kann.

Das Plädoyer für eine kognitive Robotik zeigt das uneigentliche Bild eines schleichen den Übergangs bezüglich der Grundorientierung der Forschung bzw. lässt schemenhaft einen entscheidenden „evolutionären Paradigmenübergang“ am Horizont des Wissenschaftsbetriebes und seiner Strukturen erkennen. Diese neuen Strukturen spiegeln in gewisser Hinsicht den permanenten Wandel, der durch die Technologie induziert wird. So unterliegen die technologischen Innovationen einem dauernden Neuerungsprozess, so dass sich die einzelnen Wissenschaften ebenfalls einem dauernden Wandlungsprozess in Bezug auf den Einsatz der jeweils neuesten technischen Möglichkeiten unterziehen. Die dadurch bedingte, sich verstärkende Technologisierung der Wissenschaft, sorgt für Innovationen, die auf Problemstellungen und Lösungsversuche Einfluss nehmen, ohne dass zunächst paradigmatische Veränderungen in der Disziplin unbedingt eintreten müssen. Aber auch dieser schleichende Übergang birgt einen fundamentalen Aspekt in sich, da er die erkenntnistheoretischen Ausgangspositionen zukünftiger Forschungen betrifft.

**Abb. 7.2** Kippbild

Wenn auch die Welt mit dem Wechsel eines Paradigmas nicht wechselt, so arbeitet doch der Wissenschaftler danach in einer anderen Welt.<sup>20</sup>

Ein solcher Wechsel in eine „andere Welt“ findet statt, wenn die wissenschaftliche Gemeinschaft das Kriterium der Implementier- und Simulierbarkeit als Wissenschaftlichkeitskriterium akzeptiert und damit den Übergang von der Kognitionswissenschaft zum technologisierten Cognitive Computing vollzieht. Diese Ausgangssituation wird durch das Cognitive Computing mögliche Wissen-Können als Realisieren-Können via Simulation im und am Computer verändert, indem der technologische Entwicklungsstand dieser Disziplin deren Ausgangsposition wesentlich bestimmt.

Selbstverständlich stellen auch weiterhin Paradigmen einen konstitutiven Beitrag zur Entwicklung wissenschaftlicher Weltbilder dar, aber erst die Simulationsprogramme werden die Garanten dafür sein, dass im Rahmen des Cognitive Computing als neue Wissenschaftsdisziplin die notwendige Theoriedynamik entsteht (Abb. 7.2).

Das Wechselbild soll verdeutlichen, dass ein Wandel einem sogenannten „Gestaltswitch“ aus der Gestaltpsychologie gleicht, d. h. die neue Sichtweise wandelt sich vom bloß rhetorischen zum konstitutiven Beitrag im Forschungsprozess.

Ähnlich, wie ein Bild zwei oder mehrere Perspektiven beinhalten kann, so kann auch die Wirklichkeit bei gleichbleibendem Bild allein durch eine Veränderung der Annahmen, beispielsweise durch neue Blickwinkel, anderes Erkenntnisinteresse und hier durch den Zugang mittels neuartiger Technologien, eine völlig andere Gestalt annehmen.<sup>21</sup> Dieser Gestaltwandel ereignet sich nicht auf dem Bild, sondern in den Augen des Betrachters, denn er konstruiert aktiv den Bildausschnitt seiner Wahrnehmung, je nachdem, welche Perspektive er einnimmt. Ob dieser Wandel sich als graduelle bzw. evolutionäre Verän-

<sup>20</sup> Siehe auch Kuhn. 1976, Seite 133.

<sup>21</sup> Siehe auch Heisenberg 1989.

derung, oder analog den Wechsel- oder Kippbildern als eine totale Wandlung der einen in eine andere Sichtweise vollzieht, wird sich sicherlich bald zeigen. Zumindest die Methodologie des Cognitive Computing geht jedoch derzeit von einem wissenschaftlichen Fortschritt als evolutionäre Entwicklung und nicht von einem revolutionären Paradigmenwechsel als Bruch oder Umbruch aus. Im Verlaufe dieses Fortschrittes wird sich dann die perspektivische Gewichtung weiter zu kognitiven Modellen sowie zu kognitiven Lösungen als deren implementierte Realisierungen zu kognitiven Robotersystemen verlagern und damit neue Erkenntnisse ermöglichen.

---

### 7.3 Empfehlungen als Plädoyer

In diesem Buch ist man an der zentralen Stelle der systemischen Intelligenz zu dem vorgestoßen, was seine Peripherie genannt werden kann, d. h. zu Überlegungen bezüglich der Auswirkungen des Lebenszyklus von solchen Systemen auf Kognition und Agieren. Nun sind natürlich die Wissenschaftler, die an den eher klassischen Konzepten festhalten, der Auffassung, dass die vorgetragenen Probleme nur temporär außerhalb des präzise begrenzten Bereichs der gegenwärtig lösbar scheinenden Probleme liegen; andere wiederum gehen soweit zu behaupten, dass derartig „unscharfe“ und „philosophische“ Aspekte in einem praxisorientierten Handbuch nichts verloren haben. Zur Veranschaulichung dieser Spannungen mögen die folgenden Begriffsoppositionen dienen:

Aufgabenbezogen ↔ kreativ; Problemlösung ↔ Problemdefinition; abstrakt, symbolisch ↔ historisch, körperbezogen; universal ↔ kontextsensitiv; zentralisiert ↔ verteilt; sequentiell, hierarchisch ↔ parallel; vorgegebene Welt ↔ hervorgebrachte Welt; Repräsentation ↔ wirk-sames Agieren; Implementation durch ↔ Implementation durch; Konstruktion bzw. Technik ↔ Interaktionstheorie und intelligente Technologien; Abstrakt ↔ substantiell.

Die Erkenntnisse aus den Forschungen der letzten Jahre zeigen, dass echte Adaptation und systemische Intelligenz sich nur dann in einem Robotersystem ausprägen können, wenn diese artifiziellen Systeme nicht nur über einen an ihre Aufgaben angepassten, idealerweise einen anpassungsfähigen Korpus, sondern auch über situierte Reaktions- und Kommunikationsfähigkeiten und ein hohes Maß an Handlungsautonomie verfügen. Ausgehend von dieser Tatsache wird ein substantieller Fortschritt nur bei einer engen Zusammenarbeit der verschiedenen und getrennt agierenden Disziplinen möglich sein. Dies betrifft auch bzw. gerade solche Disziplinen, die bislang nicht unbedingt mit der Robotik in einen direkten Zusammenhang gebracht worden sind: Kognitionswissenschaften und Neuroforschung, theoretische Biologie, Materialwissenschaften, Philosophie<sup>22</sup> und weitere.

Eine kognitionsorientierte Neuroinformatik in Verbindung mit der Möglichkeit, dass sich in Abhängigkeit von der Aufgabe vieldimensionale Körperstrukturen über geeignete

---

<sup>22</sup> Vgl. Beckermann 1999.

Materialien selbsttätig ausbilden können, stellt sicherlich einen weiteren Meilenstein für neue, wesentlich leistungsfähigere Robotersysteme dar. Weiterhin müssen hierzu unter Umständen die bestehenden Lernparadigma erweitert oder gänzlich neue Paradigmen erfunden werden. Die Zusammenarbeit zwischen Robotikern, Kognitions- und Neurowissenschaftlern wird also immer dringender erforderlich. Mit steigender Komplexität technischer Systeme, die sich zunehmend einer analytischen Betrachtung entziehen, könnte es hilfreich sein, von den Neurowissenschaften methodisch zu lernen, auf welche Weise komplexe hochdynamische Systeme arbeiten bzw. wie sie sich ontogenetisch entwickeln und dabei über das gesamte Leben vom Einzell-Stadium bis zur vollen Ausprägung immer stabil und zielgerichtet arbeiten.

Unabdingbar seitens der Robotik ist die Entwicklung eines Verständnisses für kognitive Prozesse, die auf Robotersystemen bislang nur partiell implementiert wurden, aber für intelligentes Verhalten unverzichtbar wichtig sind. Diese Überlegungen schließen die über ein artifizielles Bewusstsein mit ein. Für die Steuerung von Robotersystemen in mehreren Modalitäten ist es von größter Wichtigkeit, über genaue Vorstellungen von menschlicher Wahrnehmung, Kommunikations- und Interoperationsverhalten zu verfügen, weshalb die systemische Interoperationsforschung in den nächsten Jahren konsequent ausgebaut werden muss.

Ebenfalls wird in den nächsten Jahren die intelligente Prothetik eine direkte Nervenkopplung von Robotersystemen an den Menschen für unterstützende Funktionen umsetzen. Auch hierzu wird ein vertieftes Verständnis derjenigen Bereiche im Gehirn zu entwickeln sein, die die Steuersignale für das angekoppelte Robotersystem erzeugen können. Eine systematische Untersuchung des Gehirns und des zentralen Nervensystems auf Strukturen, die die Umsetzung in artifizielle Systeme ermöglichen und letzteren wesentliche neue Eigenschaften verleihen, die Rolle von Genen in artifiziellen Entwicklungsprozessen und die Ausprägung von sehr hohen kognitiven Fähigkeiten, wie Sprachverständigen sowie situatives Problemlöseverhalten, sind weiter zu erforschen. Aus dieser gegenwärtigen Perspektive erscheint folgende Zukunftsbilanz durchaus realistisch:

- In den nächsten 5 Jahren: Robotersysteme, die ihre sensorisch-kognitiven Fähigkeiten und ihr damit auf verschiedenen Ebenen gekoppeltes Interoperationsrepertoire verfeinern, basierend auf multimodaler/multisensorieller Rückkopplung. Typische Vertreter dieser Spezies werden intelligente mehrbeinige Laufmaschinen und humanoide Robotersysteme sein.
- In den nächsten 10 Jahren: Robotersysteme, die strukturell neue sensomotorische Fertigkeiten in permanenter enger Kopplung mit der Umgebung entwickeln und dabei in gewissen Maßen über an die Aufgabenstellung anpassbare Korpusteile verfügen.
- In den nächsten 20 Jahren: Robotersysteme, die ihre Steuerungssysteme zusammen mit einem auf die typischen im Laufe ihres Lebens angetroffenen Aufgaben zu optimierenden Korpus evolvieren.

Gerade die letzte Klasse von Robotersystemen stellt schon heute eine wirklich langfristige Herausforderung zur transdisziplinären Zusammenarbeit dar.

Strebt man die Entwicklung von Robotersystemen an, die solche grundlegend neuen Fähigkeiten zur Anpassung an Umweltgegebenheiten und ebenso eine neue Qualität von kognitiven Leistungen zu Umweltwahrnehmung und Lernen aufweisen, zur Kooperation und Interoperation mit dem Menschen und mit ihresgleichen, so ergeben sich also drei miteinander verwobene Problemkreise:

- Entwicklung von Erkenntnistrukturen,
- Entwicklung der Körperlichkeit und
- soziale interaktive Kommunikation.

Bei der Frage nach der Ausprägung von eigenständigen *Erkenntnissstrukturen*, also der autonomen Entwicklung von Fähigkeiten, die bei den Robotersystemen zur Sammlung von Wissen und Verhalten führen können, zeichnet sich derzeit ein Wandel von einem eher repräsentations- zu einem handlungsorientierten Ansatz ab. Kognition avanciert zum aktiven Prozess der Weltgestaltung. Korpus und Kognition werden als Resultat des Lebenszyklus (Historie) von (physischen, sensorischen, sozialen) Interoperationen zwischen Systemen und ihrer Umwelt aufgefasst, die sich in struktureller Kopplung befinden und permanent aufeinander einwirken. Während nun aber sowohl kognitivistisch als auch konnektionistisch inspirierte Ansätze für technische Lösungen zu teilweise beeindruckenden Systemlösungen geführt haben, muss die systemische Interoperationstheorie den Nachweis erbringen, dass die konsequente technische Umsetzung des handlungsorientierten Ansatzes zu intelligenteren Robotersystemen führt als die beiden anderen Ansätze für sich allein. Ein Robotersystem, dass gemäß der Interoperationstheorie modelliert und mit Hilfe von intelligenten Technologien (aus)implementiert wird, müsste von einem minimalen Satz an basalen, d. h. unter Umständen vorgegebenen, Verhaltensweisen ausgehend, lernen, zu beobachten, Beobachtungen zu kategorisieren und damit eigene artifizielle Kognitionsstrukturen aufzubauen, in welchem sich der Zusammenhang zwischen der individuellen sensorischen Wahrnehmung des Robotersystems mit seinen spezifischen Handlungsoptionen spiegelt. Bei einem solchen Ansatz werden keine expliziten, symbolischen und vom Entwickler interpretierbaren Repräsentationen vorgegeben, die der Abbildung der Außenwelt gemäß einem vom Entwickler vorgegebenen Kategoriensystem dienen. Vielmehr wird das Robotersystem mit einer basalen Grundstruktur für Informationsverarbeitung und Verhaltensweisen ausgestattet, die dann die Plastizität der restlichen Systemstrukturen steuert.

Eine weitere Problemstellung ergibt sich bezüglich der Austattung bzw. dem Ausbau einer Wissensbasis innerhalb des Robotersystems. Dies betrifft Wissen über den eigenen Korpus (Geometrie, Anpassungsmöglichkeiten), über mögliche Verhaltensweisen, Aufmerksamkeitssteuerung, Bewusstheit über Handlungsoptionen oder gar eigenes Bewusstsein in Bezug auf Raum, Zeit und inneren Zustand. Solche Robotersysteme wären dann möglicherweise in der Lage, zu wissen und erklären zu können, was sie tun und warum

sie etwas tun. Sie könnten auch entscheiden, welches Wissen aus der internen und bzw. vorher erworbenen Wissensmenge ableitbar ist und wofür zusätzlicher sensorischer Input benötigt wird. Ferner könnten diese Systeme möglicherweise auch zur Introspektion fähig sein, also Aussagen über ihre inneren Zustände formulieren und aus diesen dann weitere Schlussfolgerungen ziehen.

Mit dem Problemkreis der Entwicklung der *Körperlichkeit*, der Anpassung der aktori-schen Fähigkeiten an die Umwelt und der Veränderung der Umwelt durch das Robotersys-tem selbst sind weitere Fragen verbunden. Dazu zählt zunächst das Problem der Beschleu-nigung und Optimierung der Erkenntnisprozesse durch eine geeignete physische Gestalt, die auch die Frage nach der sensorischen Ausstattung einschließt. Es betrifft vor allem aber die Frage, wie interne Repräsentationen über den Körper genutzt werden können, um die eigene Verhaltensplanung durch Prob behandeln unter Berücksichtigung des dynamischen Eigenverhaltens durch antizipatorischen Einbezug von Sensormustern (Verhalten anderer Systeme), durch Nutzung von Erfahrungen aus anderen Kontexten zu ermöglichen.

Der Aufbau *sozialer und interaktiver Kommunikation* zwischen dem Robotersystem und Mensch erfordert zum einen die bidirektionale Nutzung möglichst vieler, der dem Menschen zur Verfügung stehenden Modalitäten (optisch-visuell, sprachlich-auditiv, ges-tisch-mimisch, haptisch-taktil, etc.), und zum Anderen die Fähigkeit des Robotersystems zur Vorhersage von Bewegungs-, Handlungs- und Kommunikationsabläufen, einschließ-lich eines Verständnisses von menschlichen Emotionsäußerungen sowie der Darstellung innerer „emotionaler Zustände“ des Systems. Der erste Punkt bedingt neben der Erken-nung und Produktion von Äußerungen in allen Modalitäten auch die Erkennung und Be-folgun g menschlicher Dialogmuster. Der zweite Punkt ist die Voraussetzung für einsichti-ges Verhalten, d. h. dass bei Vorlage einer Aufgabe die verschiedenen Lösungswege mental durchgespielt und der beste in Handlung umgesetzt wird.

Um die Akzeptanz solcher zukünftiger, intelligenter Robotersysteme gewährleisten zu können, bedarf es der Transformation der inneren Konzepte des Robotersystems in die der Lebenswelt des Menschen. Dazu gehört die Klärung der Fragen, wie Konzepte und Begriffe der menschlichen Vorstellungswelt gelernt werden können, wie Bedeutungen ent-stehen und diese in den Sensor- und Handlungsmustern des Robotersystems gegründet sind. Die Schwierigkeit des Robotersystems verdoppelt sich dabei: nicht nur muss es die eigene Konzeptwelt samt Handlungsmustern lernen, sondern es muss auch die des Men-schen verstehen und dann die möglichen Korrespondenzen zwischen beiden konstruieren lernen. Dies wird es ermöglichen, dem Menschen einen Zugang zu den Erfahrungen des Robotersystems zu schaffen und gleichzeitig die Grundlage dafür zu legen, dass Mensch und Robotersystem gemeinsame Erfahrungen im Umgang miteinander sammeln können.

Allgemeiner ausgedrückt stellt sich also die Frage, wie ein Bauplan eines autonomen Systems aussehen kann, das seine kognitiven Fähigkeiten so weit wie möglich selbst ent-wickelt, durch Interoperation mit seiner Umwelt (dem Menschen) sowie durch Modifikation/ Konstruktion seiner Effektoren eine (nur ihm verständliche) Wissensbasis aufbaut, damit es zu maschineller Erkenntnis (im Gegensatz zum reinen Lernen) und zur Entwicklung neuer, nicht vorprogrammierter Aktionsmuster fähig ist und diese auch dem Menschen in

einer ihm verständlichen Form mitteilen kann. Es muss also darum gehen, systematische Untersuchungen vorzunehmen, um die Wirkungsprinzipien biologischer Systeme auf der Signalverarbeitungs- und Konzeptebene im Hinblick auf ihre Übertragbarkeit auf die kognitiven und aktorischen Leistungen von Robotersystemen zu untersuchen. Dies betrifft insbesondere

- die Evolution von Sensorik, kognitiven Leistungen und aktorischen Fähigkeiten während der Lebenszeit der artifiziellen Systeme und/oder über eine Vererbungskomponente;
- die Steuerung bzw. Nutzung von Wachstumsdynamiken des sensorischen Apparats und der äußeren Gestalt (Morphologie). Statt voller Kodierung (Spezifikation) des Entwicklungs- bzw. Wachstumsprozesses sollte nur eine gewisse Disposition vorgegeben und damit Prinzipien der Selbstorganisation ausgenutzt werden (in einem die Funktion optimierenden Sinne),<sup>23</sup>
- die Herstellung einer kognitiven Basis zum Lernen von Strukturen in verschiedenen Kontexten: Kategorienbildung, Konzeptlernen, Objektbenennung (beispielsweise durch Imitation) und Transformation der Erkenntnisse in eine für den Menschen nachvollziehbare Form;
- die Entwicklung geeigneter Substrate/Materialien und Realisierungstechniken (Hardware: analog, programmierbare Logikbausteine, hybrid, biologisch);
- die strukturelle Kopplung der Robotersysteme an die Umwelt, sowie die Kopplung an den Menschen bzw. an andere Maschinen durch soziale und kommunikative Interaktion über geeignete dynamische Ontologien.

Gegen die in diesem Buch vorgestellten Theorien, Modelle und Lösungsansätze mag es sicherlich wissenschaftspolitisch kluge Einwände geben. Diesen möchte der Autor am Schluss des Buches folgendes zu bedenken geben: die im vorangegangenen geschilderten Grundlagen der Robotik als Wissenschaft und die sich daraus ergebenden Konsequenzen führen dazu, dass Probleme, Verfahren, Ziele, Werte und Ergebnisse dieser wissenschaftlichen Arbeit im Vergleich zu bisherigen „klassischen“ *insgesamt* (nicht nur partiell) neu und anders gesehen werden müssen. Das bedeutet: die Robotik als Wissenschaft betrachtet als ihre Aufgabe

- die Beschreibung und Erklärung von Prozessen in und um Robotiksystemen,
- die Anwendung empirischer Verfahren aus den diversen Wissenschaften und die empirische Prüfbarkeit der erzielten Ergebnisse,
- das Bereitstellen anwendbaren Wissens für Forscher innerhalb und außerhalb der Forschungsdisziplinen,
- die Erarbeitung expliziter empirischer Theorien als Problemlösungsstrategien

---

<sup>23</sup> Vgl. Eigen und Schuster 1979.

- interdisziplinäre Arbeit, die Intersubjektivität der verwendeten Fachsprache und rationale Argumentation voraussetzt,
- Teamarbeit im Rahmen expliziter Theorien und Methoden,
- die Orientierung an gesellschaftspolitischen Wertvorstellungen, die sich auf Prinzipien wie Selbstorganisation, Ordnung durch Fluktuation und Freiheit, Selbstbestimmung, Primat von Prozessen über Strukturen beziehen.

Wenn aber sowohl im intellektuellen als auch im normativen Bereich, im Hinblick auf die Konzeptionen von Wissenschaft, Robotik und ihre gesellschaftspolitischen Implikate als auch im Hinblick auf Konzepte von Sprache, Text, Bedeutung und Ästhetik zugleich grundlegende Veränderungen in einem Forschungsprogramm tatsächlich auftreten, dann kann wohl im Kuhnschen Sinne von Paradigmawechsel gesprochen werden, auch wenn die bis jetzt entwickelten *Theorien* noch nicht den Stand erreicht haben, der wissenschaftstheoretisch als ausgereift bezeichnet werden kann.<sup>24</sup> Wenn sich daher mit der Entwicklung der Robotik als Wissenschaft tatsächlich ein evolutionärer Paradigmawechsel andeuten sollte, dann *können* nicht einfach alte Probleme, Theorien, Modelle, Konzepte und Werte ganz oder partiell übernommen werden. Denn im neuen Paradigma haben sich, aufgrund der erkenntnistheoretischen, wissenschaftstheoretischen und gesellschaftspolitischen Entscheidungen, die Konstruktionsregeln und -normen für Wirklichkeit, Sinn, Wert und Identität gewandelt. Kurz gesagt: es hat sich nicht nur eine neue Theorie über artifizielle Systeme entwickelt, sondern das „natürliche“ Leben ist anders geworden.

Durch die Technologisierung der Wissenschaft und des Alltags wird sich die Sichtweise der Welt und damit die Welt selbst verändern. Diese Chance, oder je nach Sichtweise, Gefahr, motiviert den Autor, in diesem Plädoyer ein ganzes Jahrhundert zu fordern, um sich dem Wissenschaftsbereich der Robotik zu verschreiben. Mehr noch, es wird vorgeschlagen, sich im Rahmen eines weltumspannenden „Robotik Science Program“ der Förderung der Robotik und deren tangierten Wissenschaften einen breiten Raum einzuräumen. Die Beweggründe für diese gezielte Intensivierung der Robotikforschung sind ebenso vielfältig wie die möglichen Folgen der erhofften Ergebnisse. Zunächst drängen gesellschaftliche Gründe, indem in vielen Gebieten ein intelligentes Robotersystem kompetent und wirtschaftlich sinnvoll Aufgaben wahnimmt und damit übernehmen kann, bei dem der Mensch eben weniger kompetent und ökonomisch ineffizienter zum Einsatz kommt oder aber sich einfach „nur schwer tut“.

- So erwarten alte Menschen von einem Krankenpfleger tatkräftige Hilfe und vor allem individuellen Zuspruch. Doch diese Zuwendungen werden bereits im nächsten Jahrzehnt Robotersysteme übernehmen müssen. Nur durch den Einsatz solcher intelligenten, systemischen Nachwuchspfleger lassen sich die Probleme einer alternden Gesellschaft bewältigen.

---

<sup>24</sup> Vgl. Hoyningen-Huene 1989.

- Ebenfalls älteren Menschen werden systemische Putzhilfen dienlich sein, wenn es gilt, hohe Fenster, raumgreifende Kachelwände oder aber nicht erreichbare Ecken zu säubern.
- Sicherheitsroboter werden in Zukunft noch mehr in Haus, Hof und Garten patrouillieren. In übersichtlichen und leicht zugänglichen Umgebungen ist diese Dienstleistung bereits gelebte Realität.
- Entschärfungsroboter werden in Zukunft noch intelligenter und sicherer Bomben entschärfen, und die Menschen von dieser lebensgefährlichen Arbeit befreien.
- Robotersysteme werden in dem immer enger werdenden Verkehrschaos Fahrfunktionen selbstständig und autonom übernehmen, und so wesentlich zur Erhöhung der Verkehrssicherheit und Zuverlässigkeit beitragen. Erste Ansätze zum vollautomatischen Auto sind bereits umgesetzt.
- Aber nicht nur am Boden, sondern auch in der Luft werden Robotersysteme Aufgaben übernehmen müssen, wenn es beispielsweise gilt, in kritischen Luftgebieten zielsicher zu manövrieren. Dies gilt übrigens auch für die zivile Luftfahrt.
- Die sukzessive Erkundung uns ferner Planeten wird ebenfalls nicht ohne Robotersysteme auskommen. Wann immer ein Raumschiff von der Erde einen fernen und fremden Himmelskörper ansteuert, wird es einen Roboter an Bord haben, der Gesteinsproben entnimmt, seine Umgebung erforscht und die Resultate zur Erde funk.
- Nicht von außen zugängliche Umgebungen, wie beispielsweise Rohre, Leitungen in Atomkraftwerken etc. können durch winzige Robotersysteme gefahrlos untersucht werden. Dies gilt auch für das Verlegen von Kabeln in solchen Tiefen, in denen der Mensch nicht gelangen kann.
- Was im Groben funktioniert, greift auch im medizinisch Kleinen. Roboter im Nanobereich werden sich durch Adern und Organe des menschlichen Körpers kämpfen, um am Ort der Störung schadhafte Teile auszubessern, oder aber Verengungen auszuweiten bzw. Verstopfungen aufzulösen. Und selbst die schwierige Lage der Hirnverletzten und Schlaganfallpatienten könnte sich als weniger ausweglos erweisen, als es bisher scheinen mag.

Die eben aufgezählten Perspektiven der Robotikforschung sind aber nicht die einzigen Gründe für ein wachsendes gesellschaftliches Interesse an der Robotik. Auch tangierte Wissenschaftsgebiete dürften von solch einem Vorhaben unmittelbar profitieren. So zeigte sich bisher, dass artifizielle intelligente Systeme, bei all jenen Problemen versagen, die von natürlichen Systemen mit besonderer Eleganz und Leichtigkeit gelöst werden. Der Grund ist, dass die bisher entwickelten Systeme nach gänzlich anderen Prinzipien organisiert sind als ihre natürlichen Vorbilder. Zwar lassen sich beispielsweise in den Neurowissenschaften gewisse Analogien zwischen den logischen Funktionen einzelner Nervenzellen und den Schaltelementen in Rechnern herstellen; die Architekturen, in welche diese logischen Elemente jeweils eingebettet sind, unterscheiden sich jedoch radikal.<sup>25</sup> Da sich die Organisati-

<sup>25</sup> Vgl. Pauen und Roth 2001.

onsprinzipien des Gehirns offenbar weder durch Selbsterkenntnis noch durch angestrengetes Nachdenken erschließen lassen, richtet sich die Hoffnung auf die Neurowissenschaften und auch auf die Erfolge der Robotik.<sup>26</sup>

Eine weitere Attraktion der Robotikforschung liegt darin, dass natürliche Systeme als ideale Modelle für das Studium von Wechselwirkungen in komplexen, sich selbst organisierenden Systemen erkannt werden. In keiner anderen uns bekannten Struktur sind so viele Einzelemente zu einem funktionstüchtigen Ganzen verkoppelt. Das Nervensystem ist lebender Beweis dafür, dass komplexe, stark vernetzte Systeme stabile Zustände einnehmen können und zu zielgerichtetem Handeln fähig sind, obgleich sie einer übergeordneten Steuerzentrale entbehren. Die Hoffnung ist nun, dass ein vertieftes Verständnis solcher Zusammenhänge helfen wird, jene Regeln zu erkennen, die zur Stabilisierung und Selbstorganisation hochkomplexer, dynamischer Systeme beitragen. Diese Regeln sind deshalb von erheblicher Bedeutung, da ähnliche Organisationsprobleme in Öko- und Wirtschaftssystemen, aber auch in sozialen Systemen auftreten.<sup>27</sup>

Während die bisherige Erforschung von Robotersystemen ausschließlich Domäne der Naturwissenschaften ist, stellt das weite Gebiet der Robotik auch für Psychologen, Linguisten, Psychiater, Neurologen, Verhaltensforscher und Informatiker eine faszinierende Herausforderung dar. Bis vor wenigen Jahren entwickelten sich diese Wissensgebiete jedoch recht autonom. Besonders tief war natürlich die Trennung zwischen den Verhaltenswissenschaften und den biologischen Disziplinen. Insbesondere in Deutschland gab es bis vor einigen Jahren kaum Kontakte zwischen Informatikern, Philosophen, Psychologen und Neurobiologen. Jetzt aber erfolgt Annäherung der verschiedenen Wissensgebiete, wodurch die Robotik in eine besonders erkenntnisträchtige Phase eintreten kann.

In Einzelfällen kann man jetzt für bestimmte Verhaltensleistungen von natürlichen Systemen die zugrunde liegenden Prozesse über die verschiedenen Ebenen hinweg bis hinunter zu den molekularen Vorgängen fast lückenlos angeben. So konnten auch Teilleistungen komplexer Gehirne auf neuronaler Ebene analysiert werden. Hierzu zählen die Vorverarbeitung von Sinnessignalen, das Erkennen von Mustern, Fern- und Gedächtnisvorgängen und das Entwerfen von Handlungsfolgen. Mithilfe leistungsfähiger Rechenanlagen ließen sich zum Beispiel die Neuronengruppen orten, die eine Erinnerung an nur kurz sichtbare Objekte ermöglichen und die sicherstellen, dass eine spätere Greifbewegung zu dem nun unsichtbaren Objekt dennoch zum Ziel führt. Die entsprechenden Zellen werden aktiv, sobald das Objekt erscheint, halten ihre Aktivität aufrecht, auch nachdem es wieder verschwunden ist, und verstummen erst, wenn Minuten später die Bewegung abläuft.

Den Roboterforschern wird es zunehmend möglich sein, auch für menschliches Verhalten enge Beziehungen zwischen Struktur und Funktion darzustellen. Dadurch lassen sich mentale Prozesse wie das Aufrufen von Gedächtnisinhalten, das Vorstellen von Szenen, das stumme Sprechen und das Planen von Handlungen simulieren. Psychologische Modelle über die Struktur und Repräsentation kognitiver und mentaler Vorgänge können auf

---

<sup>26</sup> Vgl. Kandel et al. 1995.

<sup>27</sup> Vgl. Krohn und Küppers 1989.

diese Weise mit Abläufen im Gehirn in Verbindung gebracht werden. Ebenfalls umgesetzt werden kann das Ergebnis aus der Analyse von Gedächtnisleistungen, dass es verschiedene Arten von Gedächtnis geben muss: ein prozedurales Gedächtnis, welches das Erlernen und Wiederaufrufen motorischer Fertigkeiten, etwa Fahrradfahren, ermöglicht, ein räumliches Gedächtnis, ohne das wir uns in einer bekannten Stadt nicht zurechtfinden könnten, ein episodisches Gedächtnis, das uns erlaubt, eigene Erlebnisse zu erinnern, und schließlich ein deklaratives Gedächtnis, das wir brauchen, um bekannte Objekte benennen zu können.

Einen weiteren Input vermag die Sprachforschung zu leisten, indem Linguisten aufgrund sprachenanalytischer Untersuchungen zu dem Schluss kamen, dass es für die Repräsentation von Lexikon und Grammatik, für das Vokabular und das Regelwerk der Sprache verschiedene funktionell unterscheidbare Module geben müsse. Diese sollten zudem für Erst- und Zweisprachen unterschiedlich organisiert sein. Neuropsychologen haben dies bestätigt und weitere unerwartete Ergebnisse erzielt. So zeigte sich zum Beispiel, dass Eigennamen an anderen Stellen gespeichert werden als funktionsbezogene Bezeichnungen und hier wiederum Begriffe für Lebewesen anders abgelegt werden als Worte für unbelebte Objekte. Die Entwicklung und Gestaltung natürlicher Dialogschnittstellen wird davon profitieren und die Kommunikation zwischen natürlichen und artifiziellen Systemen revolutionieren.

Die Konvergenz vormals getrennter Wissensbereiche muss in zunehmendem Maße auch ihre institutionelle Verankerung in eigenen, multidisziplinär strukturierten Forschungseinrichtungen finden. In den Vereinigten Staaten gibt es bereits zahlreiche Institute im Bereich der „robotic science“. In Deutschland, sind solche Einrichtungen jedoch noch selten, wenn überhaupt vorhanden. Die Robotik wird vorwiegend von Lehrstühlen in den klassischen Disziplinen der Informatik oder Mathematik und Physik vertreten, gelegentlich auch von naturwissenschaftlich angehauchten Fachbereichen der Sozialwissenschaften. An dediziert ausgerichteten Instituten fehlt es ganz. Dagegen haben sich an vielen physikalischen Instituten neuerdings Arbeitsgruppen konstituiert, die auf dem Niveau eines Teilgebiets „Robotik“ oder „Mechatronik“ tätig werden. Hier mangelt es jedoch meist an der Verbindung oder Vernetzung zu andern Disziplinen. Entsprechend verschlungen sind oft die Wege, über die Studenten den Zugang zur Robotik finden. Es scheint an der Zeit, nicht mehr nur darüber nachzudenken, ob wir das fächerübergreifende Unternehmen Robotik nicht durch einen eigenen Studiengang und durch multidisziplinäre Robotikforschungsinstitute besser koordinieren sollten. Vielmehr gilt es zu handeln!

Die Tatsache, dass im Bereich der Robotik in Zukunft immer häufiger Brücken geschlagen werden zwischen Beschreibungssystemen, die auf ganz verschiedene Phänomene angewandt werden, wirft auch erkenntnistheoretische Fragen auf. Die Wissenschaftler setzen voraus, dass sich die Funktionen eines komplexen Systems aus dem Zusammenspiel seiner Einzelkomponenten ergeben, die, jede für sich genommen, weniger komplexe Eigenschaften aufweisen als das Gesamtsystem. Dieser reduktionistische Ansatz ist seit jeher Gegenstand philosophischer Auseinandersetzungen, führt aber im Zusammenhang mit der Robotik und den tangierten Wissenschaften zu besonders spannenden Fragen. Da er impliziert, dass sich auch psychische und seelische Phänomene Mechanismen zuordnen

lassen, die in und zwischen Nervenzellen ablaufen, also an ein materielles Substrat gebunden sind, röhrt er an die Grundfesten unseres Selbstverständnisses. Das uralte Leib-Seele-Problem<sup>28</sup>, die Frage nach dem Verhältnis von Geist und Materie<sup>29</sup>, ist mit einem Male nicht mehr nur Gegenstand philosophischer Diskurse, sondern auch ein zentrales Thema der Robotikforschung.<sup>30</sup> Aber auch die neuropsychologischen Befunde und vor allem die entwicklungsbiologischen Erkenntnisse belegen eindrucksvoll, dass mentale Funktionen aufs engste mit der Funktion der Nervennetze verbunden sind. Lässt sich doch bei der Erforschung der Hirnentwicklung Schritt für Schritt nachvollziehen, wie aus der Aggregation einfacher Grundbausteine der Materie zunehmend komplexere Strukturen entstehen und wie der jeweils erreichte Komplexitätsgrad des Systems mit der Komplexität der erbrachten Leistung zusammenhängt. Die Entwicklung von intelligenten Systemen, ausgestattet mit einer Hard- und (Soft-)Brainware stellt sich somit als stetig und im Rahmen der bekannten Naturgesetze erklärbar dar.<sup>31</sup>

Insgesamt muss die Robotikforschung auch eine Annäherung zwischen diesen scheinbar unversöhnlichen Positionen erreichen. Zunächst muss man sich hierzu klarmachen, dass sowohl die Aussagen der Robotikforschung wie die skizzierten philosophischen Positionen nur innerhalb der jeweiligen Beschreibungssysteme Gültigkeit beanspruchen können. In den Geisteswissenschaften wie in den Naturwissenschaften erfolgt alles Erklären, alles Verstehen ausschließlich innerhalb abgegrenzter Bezugssysteme. Als wahr oder zutreffend wird akzeptiert, was innerhalb dieser Wissensgebiete widerspruchsfrei und mit den Phänomenen des jeweiligen Objektbereiches vereinbar ist. Bei allen Konstrukten und Theorien handelt es sich jedoch immer nur um Beschreibungen von Erfahrungen, die wir als forschende Subjekte machen, also um Produkte der geistesgeschichtlichen Entwicklung. Diese Entwicklung setzte ein, als menschliche Gehirne damit begannen, ihre Umwelt abzubilden, Begriffe und Symbole für Erfahrungen zu erfinden, sich darüber zu verständigen, sich gegenseitig zu beschreiben. Kurzum, sie fügten der vorgefundenen materiellen Welt eine weitere Ebene hinzu, die aus immateriellen Konstrukten, Beschreibungen und Zitaten besteht, eine Ebene, die zu analysieren sich bisher die geisteswissenschaftlichen Disziplinen vorgenommen haben.

Nun lehrt uns aber die Sinnesphysiologie, dass dem Gehirn durch die Sinnesorgane nur ein sehr eingeschränkter Bereich der umgebenden Welt zugänglich ist. Aus diesem Ausschnitt werden überdies nur jene Merkmale verarbeitet, die zu kennen dem Überleben dienlich sind. Aus diesem bereits stark eingeschränkten Spektrum von Signalen aus der Umwelt synthetisiert das Gehirn dann in einem aktiven Vorgang seine „Erfahrungen“. Die Wahrnehmungspsychologie liefert eindrucksvolle Beispiele dafür, dass Wahrnehmen ein deutender Vorgang ist, in dem wir unsere Bilder von der Welt konstruieren. Die Regeln, nach denen wir diese Bilder entwerfen, sind naturgemäß im Gehirn verankert und

---

<sup>28</sup> Vgl. Bühler 1990.

<sup>29</sup> Vgl. Carrier und Mittelstraß 1989.

<sup>30</sup> Vgl. Maar et al. 1996.

<sup>31</sup> Siehe auch Eigen und Winkler 1975.

durch dessen funktionelle Architektur vorgegeben. Somit reflektieren sie das in den Genen gespeicherte „Wissen“ ebenso wie die Erfahrungen, die während der Individualentwicklung gewonnen wurden. Akzeptiert man aber, dass unsere Weltbilder Hirnkonstrukte sind, dann erscheint der Konflikt zwischen dem reduktionistischen Ansatz der modernen Robotikforschung und den geisteswissenschaftlichen Positionen lösbar. Die Untergliederung der Welt in die Ebenen der unbelebten Materie, der lebenden Organismen sowie der psychischen und geistigen Prozesse spiegelt dann nur die Koexistenz von Beschreibungssystemen für unterscheidbare Erfahrungen. Aus der Existenz von verschiedenen Beschreibungssystemen folgt jedoch noch nicht, dass sich die in ihnen angesprochenen Phänomene nicht aufeinander beziehen lassen. Es könnte sich zum Beispiel verhalten wie mit den verschiedenen Ansichten eines Gegenstandes. Reduktion oder Erklären würde dann nichts anderes bedeuten, als das Herstellen von Bezügen zwischen Phänomenen, die von unterschiedlichen Positionen aus eine unterschiedliche Beschreibung erfahren haben. Dies jedoch ist ein Vorgang, der in den Naturwissenschaften unangefochten und mit großem Erfolg vollzogen wird. Es wird auch darum gehen, Phänomene, die in unterschiedlichen Beschreibungssystemen erfasst und definiert wurden, miteinander zu verbinden. Konkret bedeutet dies, dass Beschreibungen für Phänomene, die von den Verhaltens- und Geisteswissenschaften mit Begriffen wie Lernen, Vorstellen, Erinnern oder Empfinden belegt wurden, über Brückentheorien mit Beschreibungen verbunden werden, die die Naturwissenschaften für vergleichbare Phänomene bereithalten. Gelänge dies, käme es im Rahmen dieser intertheoretischen Reduktionen zu einem direkten Brückenschlag zwischen Geistes- und Naturwissenschaften.<sup>32</sup>

In Gesprächen wird immer wieder die Befürchtung geäußert, die Robotik banalisiere unser Menschenbild, zerstöre metaphysische Dimensionen und degradiere den Menschen zu Maschinen unterschiedlicher Komplexität. Sie erzeuge langfristig eine Weltsicht, in der für Freiheit, Intentionalität, Moral und Religion kein Platz mehr sei.<sup>33</sup> Innertheoretische Reduktionen führen aber lediglich zu neuen Beschreibungen, die als Brücken zwischen bereits bestehenden Beschreibungen aufgebaut werden. Sie heben jedoch nicht die in den jeweiligen Systemen dargestellten Inhalte auf. Somit bleibt es uns Menschen belassen, an den erfahrbaren Wirklichkeiten festzuhalten. Jeder Brückenschlag wird dann Begriffe von Materie und Dinglichkeit eher aufwerten, nicht aber unser Menschenbild entwerten. Schließlich sind wir es auch in Zukunft, die Weltbilder entwerfen und uns unseren Platz darin zuweisen. Eine weitere, ernst zu nehmende Befürchtung besteht darin, vermehrtes Wissen über artifizielle Systeme könnte Manipulationsmöglichkeiten erschließen, die unsere Fähigkeiten zum verantwortungsvollen Umgang mit Wissen übersteigen. Hier sind, und das gilt für alle Wissensbereiche gleichermaßen, unsere Erziehungs- und Bildungssysteme gefordert. Sind sie in der Lage, uns die moralischen Kategorien und Handlungsmaximen an die Hand zu geben, die wir brauchen, um der Zunahme des Machbaren gewachsen zu sein?

<sup>32</sup> Vgl. Janich 1992.

<sup>33</sup> Siehe auch Searle 1987.

Trotz dieser offenen Fragen, und da wir aber nun einmal damit angefangen haben, planend und handelnd in jene Abläufe auf unserem Planeten einzugreifen, die uns hervorgebracht haben, können wir letztlich nicht umhin, uns weiter mit Wissen über die Bedingungen unserer Existenz zu versorgen. Dabei macht es die Sache nicht leichter, dass die Verbindung zwischen Nanotechnologie, neuen Werkstoffen und neuen Softwaretechniken ganz neue Perspektiven versprechen, die heute aus einer gewissen Ferne eingeschätzt werden müssen. Daher ist vermutlich nichts gefährlicher, irgendwann als unwissend handeln zu müssen oder aber dann als wissend nicht handeln zu können!

---

## Literatur

- Allman, F.: Menschliches Denken – Künstliches Intelligenz. Von der Gehirnforschung zur nächsten Computer-Generation. München (1990)
- Bartels, A.: Grundprobleme der modernen Naturphilosophie. Schöningh, Paderborn (1996). (UTB 1951)
- Baudrillard, J., Flusser, W., von Foerster, H.: Philosophie der neuen Technologie. Berlin (1989)
- Becker, B.: Wissen und Problemlösung im Spiegel neuer Entwicklungen der Computertechnologie. Dortmund (1986)
- Becker, B.: Künstliche Intelligenz. Frankfurt a. M. (1992)
- Beckermann, A.: Analytische Einführung in die Philosophie des Geistes. Gruyter, Berlin (1999)
- Bieri, P. (Hrsg.): Analytische Philosophie des Geistes. Athenäum-Hain-Hanstein, Bodenheim (1993)
- Brill, M.: Mathematik für Informatiker. Hanser, München (2001)
- Böhler, K.-E. (Hrsg.): Aspekte des Leib-Seele-Problems. Philosophie, Medizin, Künstliche Intelligenz. Würzburg (1990)
- Bungo, M.: Das Leib-Seele-Problem. Ein psychobiologischer Versuch. Tübingen (1984)
- Carrier, M., Mittelstraß, J.: Geist. Gehirn, Verhalten. Berlin (1989)
- Drieschner, M.: Einführung in die Naturphilosophie. Wissenschaftliche Buchgesellschaft, Darmstadt (1981)
- Eigen, M., Schuster, P.: The Hypercycle – A Principle of Natural Self-Organization. Springer, Heidelberg (1979)
- Eigen, M., Winkler, R.: Das Spiel: Naturgesetze steuern den Zufall. Piper, München (1975)
- Haugeland, J.: Künstliche Intelligenz – programmierte Vernunft. Hamburg (1987)
- Heisenberg, W.: Ordnung der Wirklichkeit (1942). Piper (1989)
- Hoyningen-Huene, P.: Die Wissenschaftsphilosophie Thomas S. Kuhn. Braunschweig (1989)
- Hoyningen-Huene, P., Hirsch, G.: Wozu Wissenschaftsphilosophie? Positionen und Fragen zur gegenwärtigen Wissenschaftsphilosophie- Berlin (1988)
- Janich, P.: Grenzen der Naturwissenschaft. Beck, München (1992)
- Kandel, E., Schwartz, J.H., Jessell, T.M.: Neurowissenschaften: eine Einführung. Spektrum Akademischer Verlag, Heidelberg (1995)
- Kanitscheider, B. (Hrsg.): Moderne Naturphilosophie. Königshausen + Neumann, Würzburg (1984)
- Klaus, G.: Kybernetik und Erkenntnistheorie. Deutscher Verlag der Wissenschaften, Berlin (1966)
- Kuhn, T.S.: Die Struktur wissenschaftlicher Revolutionen. Frankfurt a. M. (1976)
- Krohn, W., Küppers, G.: Die Selbstorganisation der Wissenschaft. Frankfurt a. M. (1989)
- Maar, C., Pöppel, E., Christaller, T. (Hrsg.): Die Technik auf dem Weg zur Seele. Rowohlt, Reinbek (1996)

- Pauen, M.: Grundprobleme der Philosophie des Geistes. Fischer, Frankfurt a. M. (2001)
- Pauen, M., Roth, G. (Hrsg.): Neurowissenschaften und Philosophie. Fink, München (2001)
- Ropohl, G.: Technologische Aufklärung. Beiträge zur Technikphilosophie. Frankfurt a. M. (1991)
- Runggaldier, E.: Grundprobleme der analytischen Sprachphilosophie. Kohlhammer, Stuttgart (1990)
- Schäfer, G.: Netzsicherheit. dpunkt, Heidelberg (2003)
- Searle, J.R.: Intentionality. Cambridge University Press, Cambridge. (Deutsche Ausgabe: Intentionalität, Suhrkamp, Frankfurt a. M.) (1987)
- Stanley, J., Bak, E.: Neuronale Netze. Computersimulation biologischer Intelligenz. München (1991)
- Ursul, A.D.: Information. Eine philosophische Studie. Dietz, Berlin (1970)

---

## 8.1 Musterlösung: Vorgehensmodell

Das 21. Jahrhundert wird ein Zeitalter umfassender, wenn nicht sogar revolutionärer Veränderungen. Explodierende Städte, eine alles umfassende Globalisierung und ein sich manifestierender Klimawandel stellt die Menschheit vor große Herausforderungen und Aufgaben. Herausforderungen, denen man sich mit Technologien stellen, Aufgaben, die man mit Hilfe intelligenter Lösungen lösen kann. Die Entwicklung solcher Lösungen wird dadurch erschwert, dass die heutigen Unternehmen sich neben der Globalisierung und kürzeren Innovationszyklen vor allem einem hohen Kostendruck und Preiskampf ausgesetzt sehen. Das bewirkt, dass auf die damit verbundenen Herausforderungen immer schneller, häufiger, mit intelligenten Lösungen immer preisgünstiger und dennoch adäquat geantwortet werden muss. Insofern muss ein Vorgehensmodell zur Entwicklung solcher Lösungen den Aspekten der Zeit, Innovation, Technologie, Kosten und der Angemessenheit entsprechen.

In dieser Musterlösung wird ein Vorgehensmodell zur Entwicklung von prozessorientierten und dynamischen Lösungen im Allgemeinen und von Roboterlösungen im Speziellen beschrieben. Dieses Vorgehen gestaltet sich als ein problem- und wissensorientierter, iterativer Prozess, der die Prinzipien agiler Lösungsentwicklung in die Praxis umsetzt. Der Prozess besteht aus mehreren, ineinander geschachtelten Rückkopplungsschleifen mit den Hauptphasen der Konzeptionalisierung, Implementierung und der Validierung.

### 8.1.1 Philosophie

Die Philosophie dieses Vorgehens besteht darin, ein Robotersystem als ein Agentensystem aufzufassen, dieses als *wissensbasiertes Modellsystem* zu konzeptionalisieren und dieses System durch rechnerbasierte *Technologien* in funktionaler und prozessualer Hinsicht intelligent auszugestalten. Zu diesem Zweck entwickeln die folgenden Abschnitte eine exakte



**Abb. 8.1** Multiplikativer Ansatz

Begrifflichkeit, also eine wissenschaftliche Terminologie, und bringt sie im Rahmen klarer Verfahrensregeln, einer wissenschaftlichen Methodik, zur Anwendung.

So stellen Prozesse die Ablauforganisation einer Lösung unter Zuordnung von Ressourcen und Zuständigkeiten dar. Sie formulieren die Abfolge von Handlungen (Aktionen), die zum Erreichen eines Ziels notwendig sind. Es werden Abläufe und damit der Weg (das WIE) zur Zielerreichung beschrieben. Prozesse sind damit im Allgemeinen eine zusammengehörende Abfolge von Aktivitäten. Anders formuliert: Prozesse im Speziellen sind eine zusammengehörende Abfolge von Aktivitäten mit dem Zweck der Leistungserbringung im Dienste der Zweckerfüllung. Die den Prozessen inhärenten Regeln hingegen stellen Gegebenheiten einer Lösung dar. Die Darstellung bzw. Formulierung solcher Regeln erfolgt in einer „Wenn-Dann-Aussage“ oder in entsprechenden Entscheidungstabellen. Sie sind auf ein Ziel ausgerichtet, nicht auf den Weg. Regeln beschreiben das „WAS“ erreicht werden soll. Eine Regel ist somit eine Direktive (guideline), die ein intelligentes Verhalten beeinflusst oder leitet.

Cognitive Computing hat sich zum Ziel gesetzt, das Verständnis und die Realisierung intelligenten Verhaltens in rechnerbasierten Systemen zu ermöglichen. Im Rahmen des Cognitive Computings werden Technologien wie Produktionsregelsysteme, Neuronale Netze, Fuzzy Systeme, Evolutionäre Algorithmen, Zelluläre Automaten, Memetische Systeme, Boolesche Netzwerke und Agentensysteme eingesetzt und miteinander entweder additiv oder multiplikativ kombiniert.

Aus Sicht dieser Philosophie werden die Begriffe *Wissen* und *Technologie* in den Mittelpunkt gestellt, indem im Rahmen dieses wissensorientierten Vorgehensmodells, auf diesen Begriffen aufbauend, intelligente Systemlösungen entwickelt werden. Durch die multiplikative Verknüpfung von Technologie (Soft-, Hard- und Brainware) und Wissen (Prozesse und Methodik) lassen sich Lösungen entwickeln, die sich durch einen noch nie erreichten Wirk- und Effizienzgrad in der Praxis sowie durch einen hohen inhärenten, systemischen Intelligenzquotienten auszeichnen (Abb. 8.1).

Dieser multiplikative Ansatz der Lösungsentwicklung besagt, dass eine Lösung nur dann eine Lösung ist, wenn sowohl die Software und Hardware, als das zur Entwicklung herangezogene Fach- und Methodenwissen, eingebettet in der Brainware, den konkreten Anforderungen entspricht. Wenn auch nur einer der Faktoren nicht diesen Anforderungen gerecht wird, d. h. dieser Indikator gegen Null strebt, wird auch die Lösung gegen Null streben.

Insofern basiert diese Entwicklungsmethodik im Form eines Vorgehensmodells auf einem Prozess, der sowohl den Anforderungen von prozessorientierten Problemstellungen der Robotik als auch Fragestellungen des diese Robotik umgebenden Umfeldes (Unternehmen, Produktionsumgebung, etc.) Rechnung trägt.

Die Entwicklung als Ganzes und die darin vorgesehenen Aktivitäten im Einzelnen gewährleisten einen agilen Entwicklungsprozess, der um die Aspekte der Wissensbasierung und Wiederverwendbarkeit von Ergebnistypen erweitert wurde. Unabhängig dieser Erweiterungen basiert die Methode auf dem allen agilen Methoden gemeinsamen Mindset. Dieser Bezug trägt auch der Tatsache Rechnung, dass unter Umständen mehrere Parteien an dem Vorhaben beteiligt sind und sich der Entwicklungsprozess trotz dieser „Verteiltheit“ als ein problem- und wissensorientierter, iterativer Prozess gestaltet und dadurch die Prinzipien agiler Lösungsentwicklung bedingt.

### 8.1.2 Prozessmodell

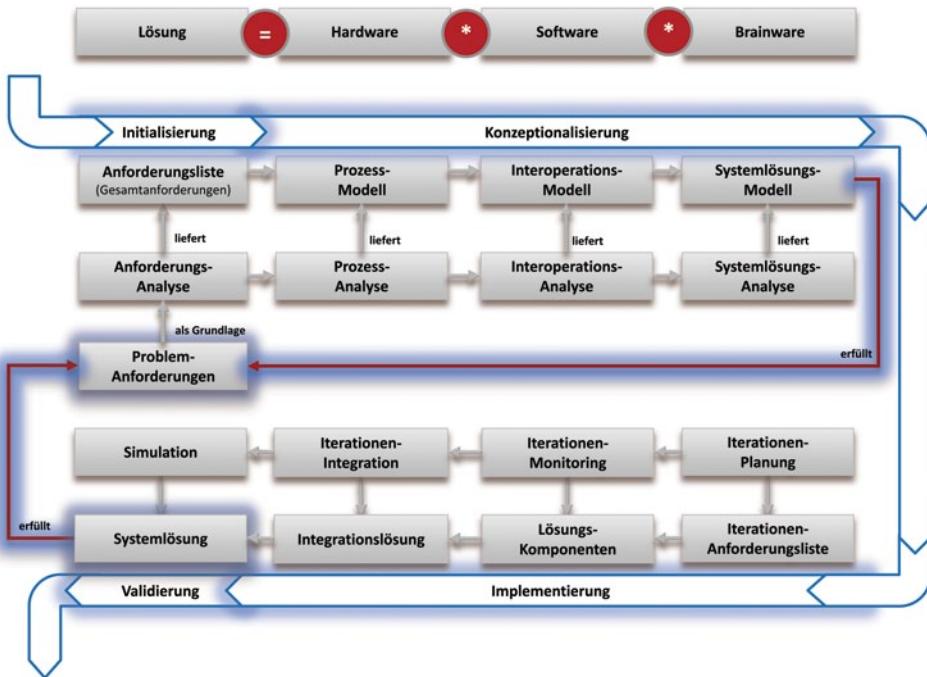
Der Entwicklungsprozess basiert immer auf einem Prozessmodell. Gemäss diesem Modell durchläuft der Entwicklungsprozess im Kern den Phasenstrang der *Konzeptionalisierung*, *Implementierung* und *Validierung*. Je nach Problemstellung können diese Phasen sequentiell durchlaufen werden (Strang) oder aber durch notwendige Rückkopplungsschleifen (Feedback-Loops) sich zu einer iterativen Vorgehensweise (Spirale, Kreis) ausgestalten. Unabhängig der jeweiligen Ausprägungsform bleiben die Aktivitäten (Anforderungsanalyse, Prozess- und Aktivitätsanalyse, etc.) und die Ergebnistypen (Anforderungsliste, Prozess- und Aktivitätsmodell, Lösungsmodell, etc.) im Modell verbindlich (Abb. 8.2).<sup>1</sup>

Im Rahmen der *Initialisierung* werden sämtliche Vorbereitungen getroffen, um das Entwicklungsprojekt starten zu können. Dies betrifft sowohl Fragen der Projektorganisation als auch die Erteilung des Entwicklungs- und Projektauftrages. Um den Auftrag definieren zu können, ist es unabdinglich, sich über die Anforderungen bereits in diesem frühen Zeitpunkt einen groben Überblick zu verschaffen.

In der Phase der *Konzeptionalisierung* wird die reale Welt bzw. der Ausschnitt aus dem Problemraum auf die Existenz von Entitäten und Entitätsbeziehungen hin untersucht und ein entitätsorientiertes Ontologiemodell dieser realen Welt bzw. dieses Ausschnitts erstellt. Es wird gefragt, *was mit welchen Entitäten warum* geschieht oder geschehen soll. Denken in Entitäten (= Erkenntnisobjekte, Begriffe etc.) heißt verallgemeinern, das Gemeinsame herausheben. Auf Basis dieses Ontologiemodells wird das Lösungsmodell entwickelt, das sich zum einen an den Anforderungen orientiert und die beteiligten Komponenten, die Prozess- und Interoperationslogik beschreibt.

In der Phase der *Implementierung* wird das entitätsorientierte Lösungsmodell zunächst in die Welt der Zielarchitektur (Hardware, Software, Brainware) und damit in den Lö-

<sup>1</sup> Siehe auch Hitz und Kappel 2003.



**Abb. 8.2** Vorgehensmodell als Musterlösung

sungsraum übertragen, dort ggf. ergänzt oder modifiziert. Das Lösungsmodell wird so zu einer einsatzfähigen Lösung konkretisiert und überführt. Es wird genau festgelegt, *wie* alles im Detail funktioniert.

In der Phase der *Validierung* wird permanent überprüft, ob die geschaffene Lösung noch den Anforderungen entspricht bzw. ob diese Anforderungen noch Bestand haben oder aber sich geändert haben oder aber ggf. das Lösungsmodell und damit die Implementierung modifiziert werden müssen.

### 8.1.3 Beteiligte

Die in dieser Musterlösung vorgeschlagene agile Entwicklungsmethodik ist eine Vorgehensweise zur Entwicklung von Hard-, Soft- und Brainwarelösungen, die ganz bewusst auf einigen wenigen klaren Regeln und einer flachen Beteiligtenstruktur basiert. Zur Veranschaulichung dieser Struktur wird auf den Metapher des Orchesters zurückgegriffen. Die am Entwicklungsprozess Beteiligten lassen sich demgemäß durch die beiden Rollen des Entwicklungsdirigenten und des Mitglieds eines Entwicklungsteams beschreiben. Beide Rollen bilden das Entwicklungsteam als Ensemble, das auf Basis einer priorisierten Lösungsanforderungsliste die Entwicklung von Lösungssinkrementen innerhalb kurzer Entwicklungszyklen vorantreibt. Als agile und wissensbasierte Entwicklungsmethode ver-

körpern beide Rollen die Werte des agilen Manifests und stellen den Menschen und sein Wissen in den Mittelpunkt der Entwicklung.

Der *Entwicklungsdirigent* repräsentiert die Anforderungen an die spätere Lösung, dirigiert die Entwicklung und arbeitet mit dem Team über den gesamten Projektverlauf eng zusammen. Die Rolle vereint die klassischen Produkt- und Projektmanagementaufgaben in sich und ist zugleich fest in den Entwicklungsprozess integriert.<sup>2</sup> Der Entwicklungsdirigent ist Koordinator und verantwortlicher Leiter eines Teams in Form eines Ensembles. Er ist für das Erfassen der Lösungsanforderungen verantwortlich. Außerdem verantwortet er das Erreichen der Projektziele und damit den Erfolg des Entwicklungsprojekts. Er allein entscheidet über Auslieferungszeitpunkt, Funktionalität und Kosten der Lösungsversion. Folglich erstellt und aktualisiert er den Releaseplan und die einzelnen Releaseberichte. Der Entwicklungsdirigent schlägt also die Brücke zwischen Problem und Lösung beziehungsweise dem Kunden und dem Entwicklungsteam. Im Idealfall übernimmt der Lösungsdirigent die Aufgaben eines klassischen Produktmanagers, Projektleiters und Chefarchitekten.

Das *Team* fungiert als ein Ensemble und führt alle Arbeiten aus, die zur Umsetzung der Anforderungen in auslieferbare Lösungssinkemente notwendig sind. Als Ensemble entscheidet das Team, wie viele Anforderungen es innerhalb der nächsten Iteration in ein Lösungssinkrement umwandeln kann. Die Mitglieder des Teams entscheiden gemeinsam, welche Aufgaben notwendig sind, um das Iterationsziel zu erreichen, und wer welche Aufgaben übernimmt. Die Selbstorganisation des Teams erfolgt diszipliniert, indem die Liste der Iterationen-Anforderungen, der Iterationen-Burndown-Bericht und die tägliche Besprechung die Grundlagen für den gemeinsamen Projektfortschritt bilden. Durch die Anwendung kurzer Arbeitszyklen und das regelmäßige Feedback des Lösungsdirigenten lernt das Team, seine Verpflichtungen immer besser zu erbringen.

Das Team ist als Ensemble möglichst klein und autonom, d. h. es besteht in der Regel aus maximal fünf bis sieben Mitgliedern und muss in der Lage sein, das Ziel der Iteration ohne wesentliche externe Abhängigkeiten zu erreichen.

Aufgrund dieser Unabhängigkeit und Autonomie könnte man das Team auch als eine Art „Task Force“ auffassen.

Das Team wird aufgrund der Anforderungen interdisziplinär besetzt sein müssen, d. h. alle Rollen, die zur Umsetzung der Projekt- und Sessionsziele benötigt werden, müssen im Ensemble vertreten sein.

Es müssen alle räumlichen Rahmenbedingungen geschaffen sein, damit das Team als Ensemble optimal zusammenarbeiten kann. So muss einerseits für jedes Mitglied genügend Platz vorgesehen sein, damit es in Ruhe und Abgeschiedenheit seine Arbeiten ausführen kann. Zum anderen muss eine enge Kommunikation und Zusammenarbeit, inklusive Paararbeiten, gewährleistet werden. Neben diesen räumlichen Voraussetzungen ist es wichtig, dass sich das Team auf gemeinsame Arbeitsweisen und Normen verständigt.

---

<sup>2</sup> Siehe auch Bittner et al. 1995.



**Abb. 8.3** Inhalte der Anforderungsliste

### 8.1.4 Anforderungsanalyse

Am Anfang einer Entwicklung stehen das Problem und seine adäquate Beschreibung. Anhand dieser Problembeschreibung wird eine Liste von Anforderungen zusammengestellt. Diese Liste ist die Basis für die spätere Aufwandschätzungen, der Priorisierung dieser Anforderung und damit insgesamt die „Richtschnur“ für das Entwicklungsteam, das die Anforderungsliste sukzessive abarbeitet und in auslieferbare und ausführbare Lösungen umwandelt.<sup>3</sup>

Das zentrale Medium zum Erfassen und Managen von Anforderungen ist die *Anforderungsliste*. Sie beinhaltet alle bekannten Anforderungen und Arbeitsergebnisse, die zur Erreichung des Projektziels umgesetzt und erbracht werden müssen.<sup>4</sup> Hierzu zählen funktionale und nicht funktionale Anforderungen sowie Anforderungen an die Benutzerschnittstellen. Neben den zentralen Funktionen der Lösung sind es gerade die nicht funktionalen Anforderungen, wie Skalierbarkeit, Robustheit, Performanz und Anforderungen, die über eine erfolgreiche Entwicklung der Lösung mit entscheiden (Abb. 8.3).

Die Anforderungsliste enthält keine Aktivitäten oder sonstige Umsetzungsdirektiven. Insofern ist die Anforderungsliste von der Iterations-Anforderungsliste in dieser Hinsicht absolut getrennt zu halten. Das impliziert, dass im Normalfall erst am Ende eines Entwicklungsprojekts alle umgesetzten Anforderungen präzise, detailliert und damit terminal zementiert werden können.

Die Anforderungsliste wird nach Nutzen, Risiko und Kosten priorisiert. Unabhängig von der Form der Anforderungsliste sollte die Struktur so einfach wie möglich gehalten sein. Es gilt dabei der Grundsatz: So kurz und knapp wie möglich, so umfangreich und detailliert wie nötig. Allerdings gilt es auch zu beachten: Je mehr Risiko eine Anforderung enthält und je weniger Wissen das Team über die Anforderungen besitzt, desto genauer sollten die Anforderungen beschrieben werden. Ein Risiko ist dabei das mögliche Eintreten oder Ausbleiben eines Ereignisses, das den Projektverlauf negativ beeinflusst. Gerade die Projekte im Umfeld der Robotik weisen sich durch ihren hohen innovativen Charakter aus, was ein Mehr an Unsicherheit und Risiken mit sich bringt. Diesem Umstand Rechnung tragend, wird eine risikobehaftete Anforderung, die zum jetzigen Zeitpunkt nicht genau analysiert werden kann, als hochprior klassifiziert und in einem der nächsten Sessions implementiert. So kann das Risiko exploriert werden.

<sup>3</sup> Siehe auch Rupp 2001.

<sup>4</sup> Siehe auch Sommerville und Sayer 1997.

Gute und damit verwertbare Anforderungen sollten daher die folgenden Eigenschaften aufweisen:

- *Unabhängigkeit*: Anforderungen sollten möglichst unabhängig voneinander sein und daher in beliebiger Reihenfolge und einzeln umgesetzt werden können.
- *Verhandelbarkeit*: Die Anforderungen sollten so gekapselt und beschrieben werden, dass der Entwicklungsdirektør in den Sessions-Planungssitzung unter Aspekten der Priorisierung diskutieren kann.
- *Nützlichkeit*: Alle Anforderungen sollten sich durch einen klaren Nutzen wertschöpfend auszeichnen.
- *Schätzbarkeit*: Alle Anforderungen müssen klar und verständlich formuliert sein, damit diese verstanden und hinsichtlich des Aufwandes eingeschätzt werden können.
- *Handhabbarkeit*: Die Beschreibungen zu den Anforderungen sollten kurz, prägnant und lapidar gehalten werden. Sollten die Anforderungen einen bestimmten Komplexitätsgrad aufweisen, so können diese über die Benutzergeschichten ergänzend beschrieben werden.
- *Testbarkeit*: Alle Anforderungen müssen testbar sein. Daher ist es angezeigt, zu jeder Anforderung Test- und Akzeptanzkriterien zu formulieren.

Anders als in klassischen Schätzverfahren, die häufig auf Lines of Codes (LoC) oder Functions Points zur Aufwandsbestimmung setzen, werden Schätzpunkte verwendet. Schätzwerte, die auf solchen Punkten basieren, sind immer teamspezifisch und lassen sich sicherlich nicht über Teamgrenzen hinweg vergleichen. Dennoch lassen sie sich einfach einwenden und die unten aufgeführte Punktereihe entspricht der Fibonacci-Funktion: **0**: Kein Aufwand, **1**: Kleiner Aufwand: doppelt so groß wie ein sehr kleiner Aufwand, **3**: Mittlerer Aufwand: so groß wie ein sehr kleiner und ein kleiner Aufwand zusammen, **5**: Großer Aufwand: so groß wie ein kleiner und mittlerer Aufwand zusammen, **8**: Sehr großer Aufwand: so groß wie ein mittlerer und großer Aufwand zusammen und **13**: Extremer Aufwand: so groß wie ein großer und sehr großer Aufwand zusammen.

Um die Aufwände zu schätzen, werden Schätz-Besprechungen einberufen. Da sich die Anforderungsliste während des Projektverlaufes ändern kann, finden solche Besprechungen auch noch während des Projektes statt. Solche Besprechungen laufen folgendermaßen ab: Der Entwicklungsdirektør erklärt die Anforderungen und das Team schätzt. Hierzu muss das Team alle wesentlichen Arbeitsschritte berücksichtigen. Dies beinhaltet Konzeptionalisierung, Implementierung und Validierung (inklusive Dokumentation). Sind alle Einträge abgeschätzt oder ist die anberaumte Zeit abgelaufen, so endet die Besprechung. Die Ergebnisse der Schätzungen werden in der Anforderungsliste festgehalten.



**Abb. 8.4** Prozesse

### 8.1.5 Interoperationsanalyse

Die Interoperationsanalyse im industriellen Umfeld basiert auf der Philosophie, das Unternehmen als *kognitive Organisation* aufzufassen, dieses als prozess- und kognitives Modell zu konzeptionalisieren und dieses Modell durch rechnerbasierte, prozessorientierte und künstliche Kognition ermöglichte Technologien in funktionaler und prozessualer Hinsicht auszugestalten.

*Prozesse* stellen dabei die Ablauforganisation einer Organisation bzw. den Ablauf einer Problemlösung unter Zuordnung von Ressourcen und Zuständigkeiten dar. Sie formulieren die Abfolge von Handlungen (Aktionen, Interoperationen), die zum Erreichen eines Ziels notwendig sind. Es werden Abläufe und damit der Weg (das WIE) zur Zielerreichung beschrieben. *Prozesse* sind damit im Allgemeinen eine zusammengehörende Abfolge von klassischen Aktivitäten oder Interoperationen (Abb. 8.4).

Die Prozesse im Allgemeinen und die Makro- bzw. Mikroprozesse, Geschäftsprozesse<sup>5</sup> bzw. Produktionsprozesse im Speziellen und damit die Prozessmodellierung werden als Ausgangspunkt zur Entwicklung von Lösungen gesetzt. Die Modellierung der Prozesse hat den Zweck, die fachlichen Abläufe zu beschreiben. Es wird die Frage beantwortet: Was wird in welcher Folge fachlich und organisatorisch in einer konkreten Problemstellung oder in einem Bereich (Unternehmen, Abteilung, Gruppe, Team etc.) bearbeitet?

Die Modellierung der Prozesse bildet den zentralen Ausgangspunkt für die Entwicklung rechnerbasierter Lösungen im Allgemeinen und von Roboterlösungen im Speziellen. Solche prozessorientierten Lösungen orientieren sich nicht nur am jeweiligen Prozess, sondern bilden diesen später in der jeweiligen Produktionsumgebung ab. Dabei steht die Fragestellung im Vordergrund: Was wird in welcher Folge fachlich und organisatorisch in einer Problemdomäne mit welcher Ressource und warum bearbeitet? Ein weiterer wichtiger Punkt ist die Identifizierung und Beschreibung von Logik (Entscheidungslogik etc.), so dass einzelne Interoperationen eventuell automatisiert werden können. Insofern müssen im Rahmen der Prozessmodellierung alle diejenigen Prozesse, (Makro-Mikroprozesse im Embedded Bereich, Geschäftsprozesse, Produktionsprozesse, etc.), die die zukünftige Lösung unterstützen soll, modelliert werden. Modellieren heißt in diesem Fall: Die Prozesse vollumfänglich verstehen.<sup>6</sup>

<sup>5</sup> Vgl. Scheer 2001.

<sup>6</sup> Vgl. Seidmeier 2002.



**Abb. 8.5** Prozess-Analyse

Die angestrebte, durchgängige Modellierung der Prozesse bietet die Möglichkeit, Synergiepotenziale durch die Wiederverwendung oder Harmonisierung von Teilprozessen mit atomaren Tätigkeiten zu erkennen und zu nutzen. Die Modellierung von Prozessen hat die Intension, die fachlichen Abläufe korrekt zu beschreiben.

Die Durchführung der einzelnen Aktivitäten obliegt dem Prozessteam. Neben dem Prozessverantwortlichen beinhaltet das Prozessteam folgende Rollen: Prozessverantwortlicher, Fachexperte, Geschäftsprozessmodellierer, Geschäftsstrategie, Produktionsleiter, Business Developer, etc.<sup>7</sup>

An dieser Stelle gilt es darauf hinzuweisen, dass je nach Problemstellung und Problemdomäne die Prozessanalyse und die Interoperationsanalyse nicht immer sauber voneinander zu trennen sind bzw. eine solche Trennung auch keinen Sinn macht. Aber auf jeden Fall beeinflussen die Ergebnisse der Prozess- und der Interoperationsanalyse die Anforderungen an die zukünftige Lösung. Insofern kann man die Prozess- und die Interoperationsanalyse durchaus als eine erweiterte fachliche Anforderungsanalyse bezeichnen, die dazu beiträgt, dass alle Anforderungen vollständig und widerspruchsfrei erarbeitet werden können. Nach Möglichkeit sollten die Anforderungen so formuliert bzw. dokumentiert werden, dass diese prüfbar sind. Die Ergebnisse einer solchen Anforderungsanalyse werden in Form eines umfassenden Entwicklungshandbuches (EHB) dokumentiert.

Einer der zentralen Punkte für eine qualitativ hochwertige Anforderungsanalyse ist die Identifikation der Personen, die für die Ermittlung der Anforderungen berücksichtigt werden. Dabei ist insbesondere zu beachten, dass die späteren Anwender in die Analyse einbezogen werden, um alle Anforderungen praxisgerecht zu erfassen und auch die Akzeptanz des zu entwickelnden Systems zu verbessern. Nachdem die fachlichen Anforderungen (Sollkonzeption) abgeschlossen und im EHB dokumentiert wurden, kann dies je nach Komplexität einem speziellen Abnahmeteam vorgelegt werden. Dieses muss detailliert prüfen, ob die fachlichen Anforderungen (Sollkonzeption) fachlich korrekt sind.

Im Rahmen dieser Musterlösung umfasst die fachliche Analyse zwei separate Phasen der Prozessanalyse und Interoperationsanalyse, wobei das Ergebnis der Prozessanalyse automatisch den inhaltlichen Rahmen für die anschließende Interoperationsanalyse liefert (Abb. 8.5).

<sup>7</sup> Siehe auch Österle 1995.

Bei der Analyse des IST-Prozesses erfolgt im ersten Schritt ein Finden aller Interoperationen.

In der klassischen Prozess-Analyse werden Aktivitäten im Rahmen von Anwendungsfällen ausfindig gemacht. Diese Aktivitäten können damit als Ausgangspunkt für die Gestaltung von Interoperationen verwertet werden.

Im zweiten Schritt werden diese Interoperationen in eine korrekte, zeitliche/logische Reihenfolge gebracht. Insofern steht das Ziel der Findung aller relevanten Interoperationen eindeutig im Fokus dieses Schrittes. Fallweise kann entschieden werden, ob zu den einzelnen Interoperationen Kennziffern zu erheben sind. In diesem Sinne ist ein Prozess eine Zusammenfassung von organisatorisch evtl. verteilten, fachlich jedoch zusammenhängenden Interoperationen, die unbedingt notwendig sind, um einen Anwendungsfall ziel- bzw. ergebnisorientiert zu bearbeiten.

Die Interoperationen eines Prozesses stehen gewöhnlich in zeitlichen Folgen und logischen Abhängigkeiten zueinander.

So entsteht ein Anwendungsfall beispielsweise durch ein Ereignis (z. B. Antragseingang) und hat mindestens ein sichtbares fachliches Ergebnis (z. B. einen Vertrag). Im Rahmen von Produktionsprozessen kann beispielsweise die Anlieferung von Rohmaterial die Produktion anstoßen, um am Ende des Produktionsprozesses ein Produkt zu liefern.

Im Unterschied zum Prozess beschreibt nun eine Interoperation stets eine zeitlich ununterbrochene Interoperation eines oder mehrerer Ressourcen oder Akteure in Form von Rollen. Ein Prozess setzt sich somit aus einer wohldefinierten Ansammlung von Interoperationen zusammen. In diesem Schritt geht es auch darum, zu klären, welche Ressourcen oder Akteure (Rollen) an der jeweiligen Interoperation beteiligt sind. Diese Fragen werden in diesem Schritt grundsätzlich und auf einem möglichst abstrakten Niveau beantwortet. Die Interoperationen werden in diesem Schritt also bewusst reduziert auf die Absichten der prozessualen Akteure und möglichst abstrakt bzw. generisch beschrieben.

Bevor Soll-Prozesse modelliert werden, sollten die zugrunde liegenden Prozessziele geklärt sein. Die Beschreibung eines Ziels setzt sich dabei aus mindestens drei Teilen zusammen:

- **Zielinhalt:** Welches Ziel bzw. welcher Zustand soll überhaupt erreicht werden?
- **Zielausmaß:** In welchem Ausmaß soll dieses Ziel erreicht werden?
- **Zeithorizont:** Bis wann soll dieses Ziel erreicht werden?

Neben diesen Hauptdimensionen können Ziele hinsichtlich ihrer Reichweite (strategisch, taktisch oder operativ) und ihrer Abhängigkeiten analysiert und spezifiziert werden. Gibt es mehrere Ziele, müssen ihre Abhängigkeiten, d. h. deren Kompatibilität, Neutralität oder Kontrarität geklärt werden.



**Abb. 8.6** Interoperationen als Regeln

Nun kann man im nächsten Schritt das im Rahmen der Ist-Analyse entwickelte Modell mit den Zielen abgleichen und daraus wiederum ein Soll-Prozessmodell entwickeln.<sup>8</sup> Neben der Dokumentation der Anforderungen an einen Prozess (Ziele) ist auch die Abgrenzung des (Teil-) Prozesses und seiner Aktivitäten gegenüber anderen (Teil)-Prozessen, Anwendungen und Aktivitäten sinnvoll und notwendig. Diese Begrenzung ist Bestandteil des Soll-Prozessmodells. Ziel der Prozessabgrenzung ist insbesondere die Einschränkung bzw. Klarstellung der Zieldefinition.

Nachdem das Prozessmodell vorliegt, kann man sich den Interoperationen zuwenden. Interoperationen können in einer ersten Annäherung als Regeln aufgefasst werden. Solche Regeln stellen Gegebenheiten einer Organisation oder einer Problemlösung dar. Die Darstellung solcher Gegebenheiten kann oftmals in einer „Wenn-Dann-Aussage“ oder in entsprechenden Entscheidungstabellen erfolgen. Sie sind auf ein Ziel ausgerichtet, nicht auf den Weg. Regeln beschreiben demnach das „WAS“ erreicht werden soll. Eine Interoperation als Regel ist somit eine Direktive (Guideline), die ein Verhalten beeinflusst oder leitet (Abb. 8.6).

Bei der Interoperationsanalyse an sich werden manuelle Interoperationen wie auch systemische Interoperationen beschrieben (= ganzheitliche Sicht). Dabei gilt es zu beachten, dass nur die Interoperationen beschrieben werden, die im jeweiligen Problem- oder Modellierungskontext stehen.

### 8.1.6 Iterationsplanung

Die Iterationsplanung hat eher den Charakter einer Vorhersage, als den einer „in Stein gehauenen“ oder „zementierten“ Planung. Gleichwohl wird die Beplanung einer zu entwickelnden Lösung in Form eines Versionsplans oder einer „Roadmap“ für alle Beteiligten verbindlich festgehalten. Bei der Erarbeitung eines solchen Versionsplanes wird für jede Version die kleinste Menge an „vermarktbares“ Merkmalen identifiziert, also der minimale Funktionsumfang, der einen echten Mehrwert darstellt. Der erste Wert, der die Ausgestaltung des Versionsplanes wesentlich prägt, ist dabei der geschätzte Aufwand zur Realisierung der einzelnen Anforderungen. Neben dem Aufwand zur Umsetzung der Anforderungen in der Anforderungsliste ist die Entwicklungsgeschwindigkeit die zweite Größe, die den Versionsplan beeinflusst. Die Entwicklungsgeschwindigkeit ist die Summe aller Aufwände der am Ende einer Iteration vom Entwicklungsdirigent abgenommenen Entwicklungsergebnisse.

<sup>8</sup> Jacobson et al. 1999a.

**Tab. 8.1** Entwicklungsgeschwindigkeit

Verpflichtende Anforderungs-Nr.	Anforderungen angenommen?	Geplante Punkte	Erzielte Punkte
001	Ja	2	2
005	Ja	2	2
006	Ja	1	1
008	Ja	2	2
010	Nein	3	0
<i>Geschwindigkeit</i>		10	7

**Tab. 8.2** Schritte zum Erstellen des Versionsplans

Schritt	Maßnahme
❶	Es wird der benötigte Zeitraum bestimmt, um alle Einträge der Anforderungsliste umzusetzen. Hierzu werden der Aufwand in der Anforderungsliste, die Iterationslänge und die durchschnittliche Entwicklungsgeschwindigkeit der beteiligten Teams berücksichtigt
❷	Aus Basis der Priorität der Anforderungen aus der Anforderungsliste wird die Reihenfolge der zu realisierenden Anforderungen festgelegt. Dabei gilt, dass jede Iteration ein auslieferbares Lösungsskript erzeugen muss. Zugleich muss jede Iteration das Projekt seinem Ziel einen Schritt näher bringen
❸	Der Zeitpunkt wird festgelegt, zu dem die Lösung freigegeben wird. Dies kann am Ende einer Iteration, nach mehreren Iterationen oder erst am Ende des Projekts erfolgen

nisse. Dabei gilt das Gesetz, dass teilweise oder defekte Entwicklungsergebnisse nicht abgenommen werden können. Mit anderen Worten: Es gilt das „Alles oder nichts“-Prinzip.

Das folgende Beispiel illustriert die Ermittlung der Entwicklungsgeschwindigkeit: Das Team hat sich in der letzten Besprechung zur Umsetzung von fünf Anforderungen verpflichtet. Der Lösungsdirigent nimmt vier der fünf Anforderungen ab, da die Implementierung der letzten Anforderung einen die Abnahme verhindern den Bug aufweist. Die geplante Entwicklungsgeschwindigkeit des Teams betrug 10 Punkte. Tatsächlich erzielte das Team aber nur 7 Punkte. Ist das Team in der Lage, in der Regel 7 Punkte pro Iteration zu erzielen, so beträgt die durchschnittliche Entwicklungsgeschwindigkeit des Teams 7 Punkte (Tab. 8.1).

Die Ermittlung der durchschnittlichen Entwicklungsgeschwindigkeit ist wichtig, um den Versionsplan einigermaßen realistisch erstellen zu können. Ist dies zu Beginn des Projektes nicht möglich, weil beispielsweise noch keine verlässlichen Entwicklungsgeschwindigkeiten vorliegen, kann eine nachträgliche Anpassung des Versionsplans sinnvoll und notwendig sein. Generell empfiehlt es sich, durch das regelmäßige Aktualisieren des Versionsplans den ursprünglichen Plan mit der aktuellen Realität abzugleichen.

Um den Versionsplan zu erstellen, werden folgende Schritte ausgeführt (Tab. 8.2):

**Tab. 8.3** Auszug aus einem exemplatischen Versionsplan

Session	5	6	7
Starttermin	05.03.2012	01.04.2012	21.04.2012
Endetermin	25.03.2012	20.04.2012	30.04.2012
Sessionziel	UI fertigstellen	Roboter A	Integrationstest
Einträge aus der Anforderungsliste	17, 18, 19, 20, 21, 22, 25	30, 31, 32, 33, 34	50

Auch hier empfiehlt sich die Durchführung eines eigenen Workshops, um den Versionsplan zu erstellen.

Die folgende Abbildung zeigt ein Beispiel der wesentlichen Informationen, die der Versionsplan mindestens erhalten sollte (Tab. 8.3):

Auf der Basis eines solchen Versionsplanes lassen sich nun die einzelnen Iterationen angehen.

### 8.1.7 Iterationen

Iterationen sind demnach Arbeitszyklen, eine Art Mikroprojekt im Gesamtprojekt, die ein Lösungssinkrement erzeugen. Jede Iteration muss dabei einen Mehrwert schaffen. Solche Iterationen dauern maximal 20 Tage.

Die Iterationen-Anforderungsliste enthält alle Aktivitäten, die notwendig sind, um das Ziel einer Iteration zu erreichen und die zugehörigen Anforderungen in Form eines Lösungssinkrements umzusetzen. Um diese Liste zu erstellen, werden die Größen Zeit, Funktionalität und Kosten als Steuerungsgrößen herangezogen. Jede Iteration beginnt daher mit einer Iterationen-Planungsbesprechung. Das Team legt in dieser Besprechung fest, welche und wie viele der Anforderungen innerhalb der Iteration in ein Lösungssinkrement umgewandelt werden können. Alle Aktivitäten in der Anforderungsliste sind in Personenstunden geschätzt und sollten möglichst detailliert und präzise beschrieben sein. Die Aktivitäten sollten nicht größer als ein Nettoarbeitstag sein. Anschließend führt das Team die hierzu notwendigen Aktivitäten aus. Dabei gilt es zu beachten, dass die Aktivitäten der höchspriorenen Anforderungen als Erstes angegangen werden.

Das tägliche Iterations-Fortschritts-Meeting ist eine auf 20 min beschränkte Besprechung, die an jedem Arbeitstag am selben Ort zur selben Zeit stattfindet. In der täglichen Besprechung plant das Team den aktuellen Entwicklungstag. Dabei werden der aktuelle Fortschritt und mögliche Hindernisse berücksichtigt. Dabei ist zu beachten, dass die Aktivitäten im Regelfall nicht größer als ein Nettotag sein sollten.

Aus didaktischer Sicht ist es ratsam, die Besprechung am Morgen und im Stehen abzuhalten. So kann das Team zum einen die Tagesplanung am besten vornehmen und zum anderen wird verhindert, dass die Besprechung zeitlich ausufert. Durch das Zuwerfen eines „Sprachballs“ erhält das Mitglied das Rederecht, der den Ball auffängt.

Die Besprechung ist fragenorientiert, indem jedes Mitglied des Ensembles die folgenden Fragen kurz, prägnant und lapidar beantwortet:

- Welche Aktivitäten habe ich seit der letzten Besprechung abgeschlossen?
- Woran plane ich bis zur nächsten Besprechung zu arbeiten?
- Gibt es Probleme, die die Ausführung der Aktivitäten behindern oder gar verhindern?

Es ist darauf zu achten, dass die Besprechung nicht dazu gedacht ist, um Probleme zu lösen, sondern diese nur aufgezeigt werden sollen. Zur Lösung der an- und aufgezeigten Probleme wird bei Bedarf eine separate Besprechung anberaumt. Es empfiehlt sich, die angezeigten Probleme mit den betroffenen Teammitgliedern direkt im Anschluss an die Besprechung zu lösen.

Ziel ist es, so schnell als möglich, Geschäftsmehrwert zu generieren und Lösungen zu entwickeln, die die Probleme lösen. In vielen Fällen muss hierzu jedoch erst das notwendige Wissen erarbeitet werden, bevor auslieferbare Lösungssinkemente in der Iteration entwickelt werden können. Um das hierfür notwendige Wissen zu generieren, können ein oder mehrere *Forschungsiterationen* durchgeführt werden. Der wesentliche Unterschied zu den normalen Iterationen ist, dass im Rahmen dieser Veranstaltungen möglichst wieder verwendbare Prototypen anstelle auslieferbarer Lösungssinkemente entwickelt werden. Insofern sind solche Forschungsiterationen von Anfang an einzuplanen und dementsprechend in dem Releaseplan auszuweisen. Es ist dabei darauf zu achten, dass diese Forschungsiteration nicht länger als drei Entwicklertage in Anspruch nehmen.

Um ein Lösungssinkrement zu erzeugen, werden iterativ-inkrementelle Entwicklungs-techniken eingesetzt. Zusätzlich werden agile Entwicklungspraktiken, wie testgetriebene Entwicklung, Refaktorisieren und kontinuierliche Integration verwendet. Um nun ein Lösungssinkrement erstellen zu können, muss ein Durchstich durch alle tangierten Schichten der Architektur realisiert werden. Nur so ist ein Test der Lösung möglich und nur so ist der Lösungsdirigent in der Lage, die Lösung freizugeben. Existieren am Ende einer Iteration partiell fertig gestellte Arbeitsergebnisse, so werden diese in der nächsten Iteration als erste abgearbeitet, bevor neue Anforderungen umgesetzt werden.

Neben einer realistischen Versionsplanung ist ein permanenter und projektbegleitender Abgleich von Plan und Wirklichkeit notwendig, um den tatsächlichen Iterations- und damit Projektfortschritt zu verstehen und ggf. intervenieren zu können. In den Projekten werden Probleme gegebenenfalls bereits nach der ersten Iteration sichtbar. Eine Möglichkeit, den aktuellen Projektfortschritt zu beschreiben, besteht in dem Aufzeigen, wie sich die Aufwände über die Iterations-Grenzen hinweg verändern.

Sind Hindernisse aufgetreten? Hat sich das Team verschätzt? Waren die Anforderungen volatile und die Anforderungsliste häufigen Änderungen ausgesetzt? Sind die Probleme mit den zur Verfügung stehenden Technologien lösbar?

Das Aufzeigen dieser Aufwandsabnahme nach jeder Iteration ermöglicht es, gezielt Fragen bezüglich des Projektfortschrittes zu stellen. Dies erlaubt es, bereits im Laufe des Projekts Fehlerursachen zu identifizieren und abzustellen, aus möglichen Fehlern zu lernen und so die Produktivität des Teams zu erhöhen.



**Abb. 8.7** Teilprozesse der Wissensakquisition

## 8.2 Musterlösung: Wissensakquisition

Die Entwicklungsmethodik für wissensbasierte Systeme bedient sich zum einen der Erkenntnisse der bisher vorgestellten Entwicklungsmethodik, erweitert diesen aber an den Stellen, wo die Aspekte der *Wissensakquisition* (knowledge acquisition) zu berücksichtigen sind.<sup>9</sup> Dabei gilt zu beachten, dass die Wissensakquisition gemäß dem Ansatz dieses Buches ein Teilaспект des Wissensmanagements darstellt. Wissensakquisition stellt in vielerlei Hinsicht den „Flaschenhals“ eines jeden wissensbasierten Problemlösungsansatzes dar. Zum einen stellt sie den aufwendigsten und zugleich subjektivsten Teilprozess des Entwicklungsprozesses dar. Die strukturelle Differenz zwischen dem Repräsentationsmodell des Lösungssystems und dem mentalen Modell des Experten bestimmt letztendlich die Qualität der gesamten Wissensakquisition von der Erhebung über die Interpretation bis zur Repräsentation und Validierung des Wissens (Abb. 8.7).<sup>10</sup>

Die *Wissenserhebung* ist der initialisierende Teilprozess der Wissensakquisition. Aufgabe der Wissenserhebung ist die Verbalisierung und Dokumentation von Wissen einer oder verschiedener Wissensquellen oder Wissensträger. Als Quelle oder Träger solchen Wissens können dabei Textdokumente, Datenbanken, ganze Bücher oder menschliche Experten oder Expertensysteme selbst fungieren. Dazu kann man sich etablierter Techniken aus dem Bereich der Psychologie bedienen: verschiedene Formen des Interviews, Protokollanalyse oder Konstruktgitterverfahren. Das *strukturierte Interview* ist dabei ein sehr subjektives Akquisitionsmittel, wohingegen das Konstruktgitterverfahren eine weitgehend neutrale Erfassung des Wissens sicherstellt. Das erklärte Ziel der *Konstruktgitter-Technik* ist es, persönliche Einstellungen und Wertmaßstäbe eines Experten zu erfassen. Die besondere Schwierigkeit liegt hierbei darin, die Beobachtungen unbeeinflusst von Strukturierungsmethoden zu dokumentieren. Die Konstruktgitter-Theorie baut auf der Annahme auf, dass persönliche Beurteilungen und Wertungen von Ereignissen durch Aufstellen von Hypothesen generiert werden. Die aufgestellten Hypothesen bzw. Konstrukte (generische mentale Konzepte) werden an Hand von persönlichen Weltbildern verifiziert und, falls Deckungsgleichheit besteht, übernommen. Die Charakteristik der Konstruktgitter-Theorie lässt dieses Verfahren auch für die Wissensakquisition geeignet erscheinen. Denn auch hier ist man bestrebt, menschliche Erfahrung unabhängig von Befragungsmethoden zu erfassen.

<sup>9</sup> Siehe auch Bunke und Mey 1987.

<sup>10</sup> Siehe auch Harrington 1991.

Der praktische Ablauf dieser Methode in der Wissensakquisition beginnt damit, dass der Experte innerhalb eines bestimmten Problembereichs Konzepte benennt. Der Knowledge-Engineer wird anschließend aus diesen Konzepten Objekt-Tripel bilden und den Experten auffordern, Attribute zu generieren, die nur zwei Elemente dieser Tripel teilen; das dritte Element darf diese Eigenschaft nicht besitzen. Durch neue Tripelkombinationen kann das Verfahren nahezu beliebig fortgesetzt werden. So ist es möglich, Relationen und Konzepte der Expertendomäne weitgehend unbeeinflusst von einschränkenden Fragemethoden zu gewinnen. Diese Objekt-Attribut-Relationen können nun dazu verwendet werden, semantische Netze innerhalb einer Wissensbasis aufzubauen. Ein Nachteil dieser Technik ist, dass allein beim Entwickeln umfangreicher Wissensbasen große Mengen unbrauchbarer Konstrukte produziert werden, die für die eigentliche Problemlösung nicht relevant sind. Außerdem können bei Anwendung des Verfahrens auf unterschiedliche Konzept- und Objektmengen Inkonsistenzen auftreten. Ein ganz wesentliches Problem ist die Monotonie des Verfahrensablaufs. Diese wird vor allem vom Experten als ermüdend empfunden. Trotz dieser Nachteile werden Konstruktgitter-Verfahren bei Werkzeugen zur automatisierten Wissensakquisition eingesetzt.

Bei der Protokollanalyse werden Äußerungen des Experten protokolliert und analysiert. Die Protokollanalyse hat keinen eindeutig vorgeschriebenen Ablauf. Insbesondere bei der Protokollaufnahme sind unterschiedliche Techniken möglich. Je nach Lage des Problems kann es sinnvoll sein, den Experten bei der Protokollaufnahme in seiner gewohnten Arbeitsumgebung zu belassen oder durch eine Art Rollenspiel eine bestimmte Situation zu simulieren. Die Bedeutung der Arbeitsumgebung für die Inferenzen des Experten zeigen psychologische Untersuchungen zu diesem Thema. Demnach verfügen Experten nur in geringem Maße über metakognitives Wissen, d. h. Wissen über die von ihnen eingesetzten kognitiven Anwendungsstrategien. Dies bedeutet, dass Experten kaum in der Lage sind, ihre eingesetzten Strategien zu äußern, ohne subjektive Annahmen oder artfremde Einstellungen zugrunde zu legen. Ein Experte ist jedoch in der Lage, in einer konkreten Arbeitssituation sein Wissen für Problemlösungsprozesse einzusetzen. Durch den Prozeß des „lauten Denkens“, bei dem der Experte nur die Wissenselemente äußert, die sich zu einem konkreten Zeitpunkt im Arbeitsgedächtnis befinden, kann dieses Wissen protokolliert werden. Es kann davon ausgegangen werden, dass bei einem solchen Prozess keine subjektiven Problemlösungstheorien in den Expertenmonolog einfließen. Die Güte einer Protokollanalyse hängt demnach ganz entscheidend von der Protokollaufnahme ab. Nur wenn es sich tatsächlich um ein Protokoll „lauten Denkens“ und um eine fehlerfreie Dokumentation handelt, kann die anschließende Analyse erfolgreich ablaufen. Neben der Protokollaufnahme während einer konkreten Arbeitssituation besteht die Möglichkeit einer retrospektiven Aufnahme. Hierbei simuliert der Experte eine bestimmte Arbeitssituation aus der Erinnerung heraus. Bei diesem Vorgehen werden hohe kognitive Anforderungen an den Experten gestellt. Bieten sich der Problembereich und der Experte dazu an, ist diese Methode eine interessante Alternative zur on-line Protokollaufnahme. Auch Verfahren, die ein Gespräch zwischen Knowledge-Engineer und Experten vorsehen, werden oft zur Protokollanalyse gezählt. Der Protokollaufnahme folgt die Protokollanalyse.

*Skalierungstechniken* werden in der kognitiven Psychologie angewandt, um die Organisation von Konzepten im menschlichen Gedächtnis zu analysieren. Die praktische Vorgehensweise sieht so aus, dass Begriffe und Konzepte bezüglich ihrer gegenseitigen Distanz abgeschätzt und klassifiziert werden. Das Verfahren wird so lange weitergeführt, bis alle möglichen Kombinationen von Konzept- bzw. Begriffspaaren gebildet und bezüglich ihrer semantischen Korrelation numerisch bewertet wurden. Semantisch vergleichbare Konzepte werden nun zu sogenannten Clustern zusammengefasst und bezüglich der Distanz zu anderen Konzepten bzw. Clustern bewertet. Zur Generierung einer Wissensbasis ist eine zusätzliche Interpretation der klassifizierten Konzepte erforderlich. Das Verfahren eignet sich gut für Akquisitionen mit mehreren Experten.

*Strukturlege-Verfahren* haben das Ziel, Objekt- und Konzeptrelationen auf graphischem Wege zu generieren. Die Erzeugung solcher Relationsnetze kann sowohl manuell als auch rechnerunterstützt erfolgen. Der Wissensakquisitionsablauf sieht so aus, dass der Experte aufgefordert wird, Konzepte und Objekte, sogenannte Knoten, zu definieren. Diese Knoten kann er nun mit Hilfe von Relationsklassen in Beziehung setzen. Neben der Art der Relation („ist Teil von“, „ist identisch mit“, „kann dies und das“) können auch Qualitäten graphisch ausgedrückt werden. Strukturlege-Techniken sind sicherlich nicht geeignet, eine umfassende Wissensakquisition zu leisten. Durch ihre leichte Automatisierbarkeit und gute Visualisierung sind sie jedoch im Verbund mit anderen Wissenserwerbsmethoden eine interessante Alternative.

*Textanalytische Verfahren* sollen mittels Untersuchung verbaler Daten die Lokalisierung und linguistische Verarbeitung problembereichsrelevanter Textausschnitte leisten. Auf diese Weise kann die zeitintensive Lektüre der in Frage kommenden Fachliteratur wesentlich gestrafft und die Weiterverarbeitung der Texte im Hinblick auf die Generierung einer Wissensbasis rechnerunterstützt realisiert werden.

Bei der Akquisition komplexen Expertenwissens reicht die singuläre Anwendung der einzelnen Akquisitionsverfahren in der Regel nicht für den vollständigen Aufbau einer Wissensbasis aus. Vielmehr müssen die einzelnen Verfahren so miteinander kombiniert werden, um die Lücken zu schließen.

Um die akquirierten Daten speichern und später rekapitulieren, *interpretieren* und damit weiter verarbeiten zu können, müssen die Sitzungen protokolliert werden. Hierzu sind verschiedene Verfahren denkbar. Ein Beispiel ist die Aufzeichnung der Akquisitionssitzungen. Diese aufgezeichneten Gespräche werden zu einem späteren Zeitpunkt so weit aufgearbeitet und strukturiert, dass sie sich in die Repräsentationsformalismen der Wissensbasis einordnen lassen. Da dem Strukturierungsprozess eine Selektion der Datenmenge vorangehen muss, ist der Zeitaufwand für diese nachbereitenden Tätigkeiten entsprechend hoch. Ein anderer Weg ist die handschriftliche Dokumentation der Sitzungsinhalte durch den Knowledge-Engineer in Form vorbereiteter, auf das Problem angepasster Formulare. Hier besteht sicherlich die Gefahr, dass sich Übertragungsfehler oder Dokumentationslücken in das Protokoll einschleichen. Der Knowledge-Engineer hat aber mit Hilfe dieser Darstellungsweise die Möglichkeit, eine Vorselektion und -strukturierung simultan zum Sitzungsgespräch durchzuführen. Dies erspart ihm zum einen Teil die nachträgliche Auf-

bereitungsarbeit und stellt zum anderen eine Plausibilitätsprüfung der Gesprächsinhalte dar.

Zur Darstellung des erhobenen und interpretierten Wissens stehen inzwischen leistungsstarke Methoden zur Verfügung:

- Semantische Netze
- Objekt-Attribut-Wert-Tripel
- Regeln
- Frames
- Logische Ausdrücke

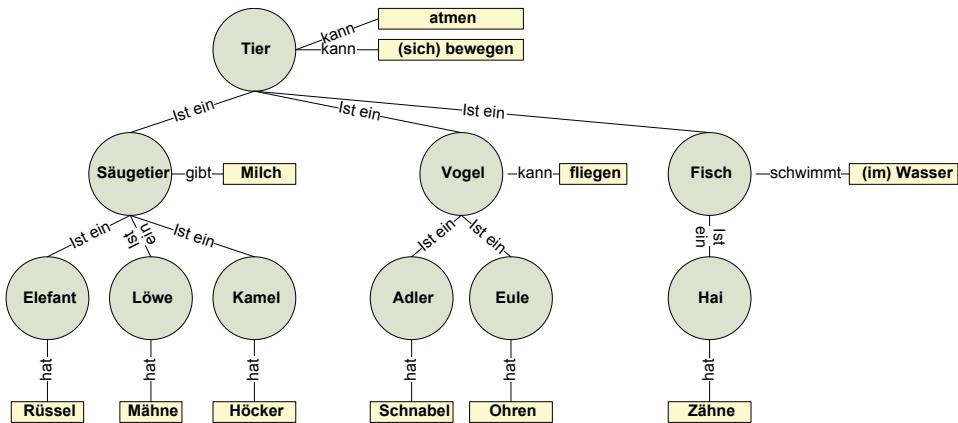
Das allgemeinste Repräsentationsschema ist das *semantische Netz* (oder Semantische Netzwerk). Ein semantisches Netz ist eine Sammlung von Objekten, die als Knoten (nodes) bezeichnet werden. Knoten sind miteinander durch Bögen (*arcs*) oder Glieder (*links*) verbunden. Normalerweise werden sowohl die Verbindungen als auch die Knoten mit Namen versehen. Es gibt keine strikte Übereinkunft darüber, wie die Knoten und Glieder zu benennen sind. In der Praxis haben sich allerdings die folgenden Konventionen bewährt:

- *Knoten* werden benutzt, um Objekte und Deskriptoren zu repräsentieren. *Objekte* können physische Gegenstände sein, die man sehen oder berühren kann. *Objekte* können auch gedankliche Elemente sein, wie z. B. Handlungen, Ereignisse oder abstrakte Kategorien. *Deskriptoren* liefern zusätzliche Informationen über Objekte.
- *Glieder* verbinden Objekte und Deskriptoren miteinander. Ein Glied kann jede Art von Relation repräsentieren. So wird das „*ist-ein*“-Glied oft dazu benutzt, um eine Relation zwischen Klasse und Einzelfall (*instance*) zu repräsentieren. Ein „*hat-ein*“-Glied kennzeichnet die Relationen zwischen Knoten, die ihrerseits Eigenschaften anderer Knoten sind. Sie repräsentieren Relationen zwischen Teilen und Teilelementen. Einige Glieder haben eine definierende Funktion. Andere Glieder geben heuristisches Wissen wieder.

Einer der Hauptvorteile dieses Repräsentationsschemas liegt in seiner Flexibilität hinsichtlich Quantität und Qualität. So können neue Knoten und Glieder nach Bedarf definiert werden. Oder wenn das Netzwerk eine Organisationshierarchie wiedergeben soll, könnte man zur Darstellung der Beziehungen zwischen den Hierarchieebenen die Glieder „gibt-Weisungen-an“ bzw. „empfängt-Weisungen-von“ verwenden.

*Vererbung (inheritance)* ist ein weiteres Merkmal semantischer Netze. Dieser Begriff bezeichnet den Sachverhalt, dass ein Knoten die Charakteristika anderer Knoten, mit denen er verbunden ist, „erben“ kann. Die Vererbung von Eigenschaften ist eine Folge der „*ist-ein*“-Relation und bedeutet, dass alle Einzelfälle einer Klasse sämtliche Eigenschaften der übergeordneten Klassen, denen sie angehören, übernehmen (Abb. 8.8).

Eine andere Methode, um den Informationsgehalt von Fakten zu repräsentieren, ist die Darstellung als *Objekt-Attribut-Wert-Tripel* oder *0-A-W-Tripel*. Bei diesem Schema können *Objekte* entweder physische Entitäten sein, wie beispielsweise eine Tür oder ein Transistor,

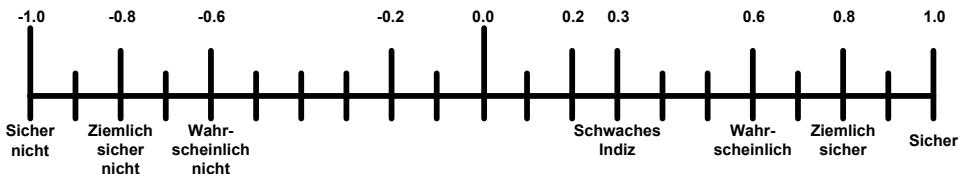


**Abb. 8.8** Ausschnitt aus der Tierwelt als semantisches Netz

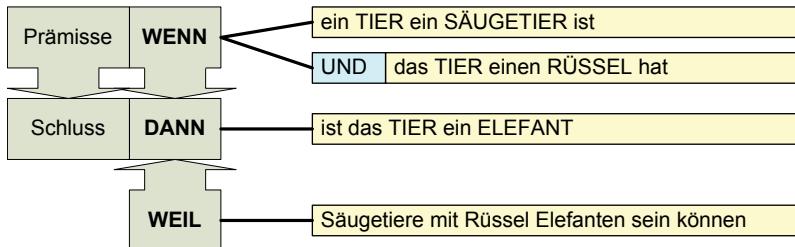
oder sie können begriffliche Einheiten sein, wie beispielsweise ein Sensor, ein Bankkredit oder ein Geschäftsvorfall. *Attribute* sind allgemeine Charakteristika oder Eigenschaften, die mit Objekten assoziiert werden. Größe, Form und Farbe sind typische Attribute von physischen Objekten. Der „Zinssatz“ ist ein Attribut eines Bankkredits, und „allgemeine Bedingungen“ können ein Attribut eines Geschäftsvorfalls sein. Das dritte Element des Tripels ist der *Wert* eines Attributs. Der Wert kennzeichnet die spezifische Beschaffenheit eines Attributs in einer bestimmten Situation.

So kann beispielsweise die Farbe eines Apfels rot sein oder der Zinssatz eines Bankkredits 12 % betragen.

Die Wissensrepräsentation durch Objekt-Attribut-Wert-Tripel ist ein Spezialfall der Methode der semantischen Netze. Zwei einfache Relationen treten an die Stelle mannigfaltiger Verbindungsglieder. Das Objekt-Attribut-Glied ist eine „hat-ein“-Relation, und das Attribut-Wert-Glied ist eine „ist-ein“-Relation. Ein weiteres wichtiges Merkmal der Repräsentation durch O-A-W-Triple ist die Art und Weise, wie die Objekte angeordnet sind und miteinander in Beziehung stehen. Diese Art und Weise des „In-Beziehung-stehens“ kann grafisch unterschiedlich dargestellt werden. So lassen sich beispielsweise baumartige Darstellungen verwenden, um Hierarchien abzubilden. Derartige grafische Darstellungen werden Bäume genannt. Das oberste Objekt wird als „Wurzel“ bezeichnet und dient als Ausgangspunkt für Schlussfolgerungen und zur Abfrage von Informationen. Solche hierarchischen Objektbäume können *verwickelt* sein, d. h. dass untergeordnete Objekte mit mehr als einem übergeordneten Objekt in Beziehung stehen können. Daraus folgt, dass das untergeordnete Objekt von mehr als einem übergeordneten Objekt Eigenschaften erbt. So wie die Ausnahmen einer Vererbungshierarchie, lassen sich auch verwickelte Bäume schwer eindeutig darstellen. Ein weiteres Merkmal der O-A-W-Schemata ist eine Verfahrensweise für den Umgang mit Ungewissheit. Es kann vorkommen, dass eine Tatsache nicht ganz eindeutig ist. Daher können die O-A-W-Triple durch eine Zahl,



**Abb. 8.9** Konfidenzfaktor



**Abb. 8.10** Produktionsregel

den sogenannten *Konfidenzfaktor* (certainty factor) modifiziert werden. Konfidenzfaktoren sind keine Wahrscheinlichkeitsfaktoren, sondern informelle Feststellungen, inwieweit man einer Tatsache oder einem Faktum vertrauen oder seiner sicher sein kann.

Man kann die Repräsentation des Wissens auch um ein einziges Objekt herum aufzubauen. In diesen Fällen werden Tatsachen als *Attribut-Wert-Paare* (A-W-Paare) und nicht als Tripel dargestellt. Dieses Repräsentationsschema verhält sich weitgehend wie ein O-A-W-Schema. Da es aber nicht mehrere Objekte repräsentieren kann, kann es den Vorteil der Vererbungshierarchie nicht nutzen (Abb. 8.9).

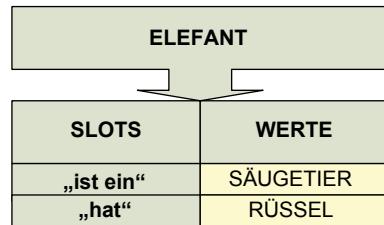
*Regeln* dienen zur Repräsentation von Beziehungen und können entweder mit A-W- oder mit O-A-W-Repräsentationen verwendet werden (Abb. 8.10).

Jede der zwei Teilprämissen wird als Ausdruck oder *Wenn-Teil* bezeichnet. Die darunterstehende Schlussfolgerung enthält einen einzigen Ausdruck, den *Dann-Teil*. Dieser könnte aber ebensogut gerade bei komplexeren Regeln auch mehr als einen Ausdruck enthalten. Die Sätze der Prämisse sind durch den logischen Operator „und“ konjunktiv miteinander verbunden. Sie könnten auch durch den Operator „oder“ disjunktiv verbunden sein, was andere Ergebnisse zur Folge hätte. In der folgenden Tabelle wurden die vier Ausdrücke, d. h. die vier O-A-W-Tripel, aus der sich die gesamte Regel zusammensetzt, nochmals anders dargestellt (Tab. 8.4):

Wenn einer der Wenn-Ausdrücke falsch ist, hört ein Expertensystem unter Umständen einfach auf, die Regel zu verarbeiten. Wenn alle Wenn-Ausdrücke richtig sind, wird gefolgert, dass auch das O-A-W-Tripel im Dann-Ausdruck richtig ist. Dieselbe Regel ließe sich für ein A-W-System umschreiben, indem man den Bezug zum Objekt ausklammert. Ebenso wie man Konfidenzfaktoren mit Tatsachen verbinden kann, kann man sie auch

**Tab. 8.4** Produktionsregel

	Attribut	Objekt	Wert	Konfidenz
WENN	Ist-ein	Tier	Säugetier	
	Hat-ein	Tier	Rüssel	0.7
DANN	Ist-ein	Tier	Elefant	

**Abb. 8.11** Frame

mit Regeln verbinden. Wenn man sich im obigen Beispiel nicht ganz sicher ist, ob das betreffende Tier einen Rüssel hat, dann lässt sich diese unsichere Relation durch einen (Un-) Gewissheitsfaktor repräsentieren, wie der Eintrag in der obigen Tabelle zeigt. Die durch ungewisse Regeln hergeleiteten Werte sind nicht völlig eindeutig. Auf diese Weise wird der methodische Umgang mit ungewissen Fakten und Relationen vereinheitlicht.

*Frames* (Rahmen) sind eine andere Methode der Darstellung von Fakten und Relationen. Ein Frame ist die Beschreibung eines Objekts und enthält sogenannte *Slots* (Attribute) für sämtliche mit dem Objekt assoziierten Informationen. In den Attributen können konkrete Werte gespeichert werden. Die Attribute können auch Vorgaben oder Vorbelegungen (*default values*) enthalten, oder auch Zeiger auf andere Frames, außerdem Gruppen von Regeln sowie Prozeduren, durch die man Werte erhält. Durch diese zusätzlichen Merkmale unterscheiden sich Frames von den O-A-W-Tripeln. Einerseits ermöglichen Frames eine umfangreichere Repräsentation von Wissen, andererseits sind sie komplexer und schwerer zu entwickeln als die einfacheren O-A-W-Regel-Systeme (Abb. 8.11).

Die obige Abbildung zeigt einen Frame aus der Tierwelt dieses Buches. Das Objekt ist ein Elefant, und es gibt Attribute (Slots) für deren Eigenschaften. Einige dieser Attribute könnten durchaus mit Belegungen vorbelastet sein, indem man davon ausgeht, dass es sich bei vielen Tieren um Säugetiere handeln wird, es sei denn, es liegen gegenteilige Informationen vor. Bei der Repräsentation von Wissen in Domänen, in denen Ausnahmen selten vorkommen, sind solche Default-Werte überaus nützlich.

Ein *prozeduraler Zusatz* (*procedural attachment*) ist eine andere Methode, wie ein Attribut in einem Frame belegt werden kann. In diesem Fall enthält das Attribut Anweisungen darüber, was eingegeben werden soll. Durch Einfügen von Prozeduren in Frames werden zwei einander ergänzende Methoden zur Feststellung und Speicherung von Fakten in einer einzigen Repräsentationsstrategie zusammengefasst: die *prozedurale* und die *deklarative* Repräsentation. Eine *deklarative Repräsentation* eines Faktums ist einfach eine Aussage, dass das Faktum wahr ist. „Das Tier ist ein Säugetier.“ ist eine solche deklarati-

ve Feststellung über die Klassifikation des Tieres. Eine *prozedurale Repräsentation* eines Faktums ist eine Gruppe von Anweisungen, die, wenn sie ausgeführt werden, zu einem Ergebnis führen, das mit dem Faktum übereinstimmt. So ließe sich beispielsweise zur Feststellung, ob eine Entität ein Rüssel ist, eine Liste von Eigenschaften angeben, die es dann im konkreten Fall abzuarbeiten gilt, um erst am Schluss dieser Verarbeitung definitiv festzustellen, ob es sich um einen Rüssel handelt oder nicht. Insofern liegt in den prozeduralen Anweisungen die Behauptung: „Das Tier hat einen Rüssel“ explizit nicht vor. Demzufolge sind die deklarative und die prozedurale Repräsentation alternative Strategien, die dieselben Ergebnisse erzielen. In dem Maße, in dem Fakten unabhängig und veränderlich sind, sind deklarative Ansätze für die Benutzer verständlicher bzw. transparenter und können aufgrund ihrer Modularität leichter gewartet werden. Experten und Benutzer gehen in der Regel lieber mit der deklarativen Form um. Demgegenüber sind prozedurale Repräsentationen im Einsatz effizienter, aber schwieriger in der Wartung. Das Resultat einer Prozedur lässt sich leichter zurückverfolgen, weil man den Instruktionsfluss leicht überprüfen kann. Knowledge Engineers bevorzugen in der Regel die prozedurale Form. Im Prinzip kann jede prozedurale Repräsentation in eine deklarative Repräsentation umgeschrieben werden und umgekehrt. Die beiden Formen, die als einander ergänzende Aspekte von Wissen angesehen werden, bezeichnet man oft als *duale Semantik*. Durch die Möglichkeit, deklarative und prozedurale Repräsentation zu integrieren, gewinnen Frames an Durchschlagskraft, Allgemeingültigkeit und Beliebtheit.

Frames können auch bildliche Repräsentationen von Werten unterstützen, die so angeordnet sind, dass eine Änderung der Werte auch eine Änderung ihrer bildlichen Darstellung nach sich zieht und umgekehrt. Repräsentationen dieser Art werden typischerweise als *aktive Werte* bezeichnet. Die wichtigsten Überlegungen, die hinter einem Frame-System stehen, kann man demnach wie folgt zusammenfassen: Jedes Objekt besteht aus einer Anzahl von Attributen. Einige Attribute enthalten Eigenschaften, die mit dem Objekt des Frames assoziiert sind. Andere Attribute können Default-Werte, Anzeigeanweisungen oder Regelgruppen beinhalten. Prozedurale Zusätze können in eine sonst deklarative Repräsentation mit eingegliedert sein. Frames können miteinander verknüpft sein und Vererbung zulassen. Schließlich können Frames und O-A-W-Regel-Systeme als Sonderfälle semantischer Netze angesehen werden. Man kann O-A-W-Systeme und Frames jeweils als Ausschnitte eines semantischen Netzes betrachten. In jedem der drei Systeme können dieselben Fakten dargestellt werden.

Auch die Logik stellt eine Methode der Wissensrepräsentation dar. So ist die *Aussagenlogik* ein allgemeines logisches System.<sup>11</sup> Aussagen können entweder wahr oder falsch sein. Aussagen, die durch Aussageverbindungen wie beispielsweise UND, ODER, NICHT, IMPLIZIERT und ÄQUIVALENT miteinander verknüpft sind, werden zusammengesetzte Aussagen oder Aussagenverknüpfungen genannt. Die Aussagenlogik befasst sich mit dem Wahrheitswert von zusammengesetzten Aussagen. Es gibt Regeln, nach denen Wahrheitswerte von Aussagen in Abhängigkeit von der Art der Aussageverbindungen festgestellt werden. Wenn beispielsweise die Aussage X wahr und die Aussage Y falsch ist, dann

---

<sup>11</sup> Vgl. Bauer und Wirsing 1991.

ist die Aussagenverknüpfung „X UND Y“ falsch, während die Verknüpfung „X ODER Y“ wahr ist. Andere Regeln ermöglichen das Ziehen von Schlüssen. Wenn X wahr ist und es gilt: X IMPLIZIERT Y, dann kann man daraus schließen, dass Y ebenfalls wahr ist.

Die *Prädikatenlogik* ist eine Erweiterung der Aussagenlogik. Die Grundelemente der Prädikatenlogik sind Objekte. Aussagen über Objekte werden Prädikate genannt. So ist beispielsweise „ist-grün (Apfel)“ eine Aussage, die besagt, daß ein Apfel grün ist. Dieser Ausdruck ist entweder wahr oder falsch. Prädikate können sich auf mehr als ein Objekt beziehen. Die Aussage: „Sohn-von (Nicolai, Matthias)“ ist ein Beispiel für ein zweistelliges Prädikat. Diese Aussage behauptet, dass Nicolai der Sohn von Matthias ist. Die üblichen Aussageverbindungen können verwendet werden, um Prädikate zu größeren Ausdrücken zu verknüpfen. So ist beispielsweise die Aussage „Sohn-von (Nicolai, Matthias) UND Sohn-von (Nicolai, Christine)“ ein Ausdruck, der entweder wahr oder falsch ist, jenachdem, ob Matthias und Christine tatsächlich die Eltern eines Sohnes namens Nicolai sind.

Logische Formulierungen repräsentieren Wissen auf eine andere Art als die vorher beschriebenen Methoden. Wenn man normalerweise Fakten darstellen möchte, dann will man direkt auf sie zugreifen können. Dabei kann es sich um die Namen von Knoten semantischer Netze handeln, um die Werte, die mit einem Objekt und einem Attribut assoziiert sind, oder um die Eintragungen in den Attributen eines Frames. Die benötigten Werte werden in der Regel durch Suchen ermittelt. Die Logik geht ein wenig anders vor. Wenn man innerhalb der Prädikatenlogik eine Tatsache aussagt, muss ihr Wert entweder *wahr* oder *falsch* sein. Die Logik stellt demnach eine andere Methode zur Verfügung, um Tatsachen über die Welt auszusagen. Fakten nehmen die Form logischer Ausdrücke an, die aus Prädikaten und Werten bestehen. Logische Ausdrücke, die Wissen beschreiben, sind entweder wahr oder falsch. Die Suche nach Werten in einem logisch-basierten System erfolgt nicht so direkt wie die Suche nach Werten in den anderen Systemen. Mit der Logik lässt sich jedoch eine große theoretische Eleganz erreichen, wenn man sie zur Formalisierung einer geeigneten Wissensdomäne einsetzt.

Im Regelfall kommt es vor der Einarbeitung von Wissen in eine bestimmte Repräsentationsform zu einer verbalen bzw. semantischen Repräsentation von Wissen. Speziell für diesen Zwischenschritt lassen sich folgende Forderungen formulieren:

- *Intensionalität*: die Darstellungsmittel müssen es gestatten, die intensionale Bedeutung von natürlichsprachigen Ausdrücken wiederzugeben
- *Extensionalität*: es muss möglich sein, bestimmte Beziehungen von Aussagen oder Begriffen zur Realität (wie Wahrheit und Falschheit von Aussagen, reale Existenz oder nur Gedachtsein von Entitäten) widerzuspiegeln
- *Universalität*: die Darstellungsmittel müssen psycholinguistisch bzw. kognitiv begründbar und logisch adäquat sein (sie dürfen auf keinen Fall nur ad hoc für einen speziellen Diskursbereich ausgewählt werden)
- *Homogenität*: Wortbedeutungen und Satz- bzw. Textbedeutungen müssen mit den gleichen Ausdrucksmitteln darstellbar sein

- *Differenziertheit*: verschiedene Bedeutungen müssen unterschiedlich repräsentierbar sein
- *Vollständigkeit*: es darf keine Bedeutungen natürlichsprachiger Ausdrücke geben, die nicht darstellbar sind
- *Praktikabilität*: die Bedeutungsdarstellungen müssen in den Prozessen der Wissensverarbeitung, insbesondere bei der syntaktisch-semantischen Analyse und bei den Inferenzprozessen, effektiv verwendbar sein; sie sollten auch anschaulich und leicht verständlich sein.

Die Liste der Anforderungen an eine spezielle Wissensrepräsentationsform (Frames, semantische Netze, Produktionsregeln etc.) reduziert sich dann auf die folgenden allgemeinen Anforderungen:

- *Repräsentationszulänglichkeit*: Die Sprache sollte geeignet sein, alles Wissen darzustellen, mit dem ein System arbeiten muss.
- *Schlusszulänglichkeit*: Sie sollte es ermöglichen, dass neues Wissen von elementaren Fakten abgeleitet bzw. gefolgert werden kann.
- *Schlusseffizienz*: Schlussfolgerungen sollten effizient möglich sein.
- *Klare Syntax und Semantik*: Es muss eindeutig klar sein, wie die zulässigen Terme der Sprache aussehen und was sie bedeuten.
- *Natürlichkeit*: Die Sprache sollte ziemlich natürlich und leicht zu verwenden sein.

Es gibt aber keine Repräsentation, die alle diese Anforderungen perfekt erfüllt. In der Praxis hängt die Wahl der Sprache von der Problemstellung ab. Wenn ein bestimmtes Problem gegeben ist, wird es im Allgemeinen notwendig sein, eine geeignete Sprache auszuwählen, die den speziellen Anforderungen der Anwendung entspricht.

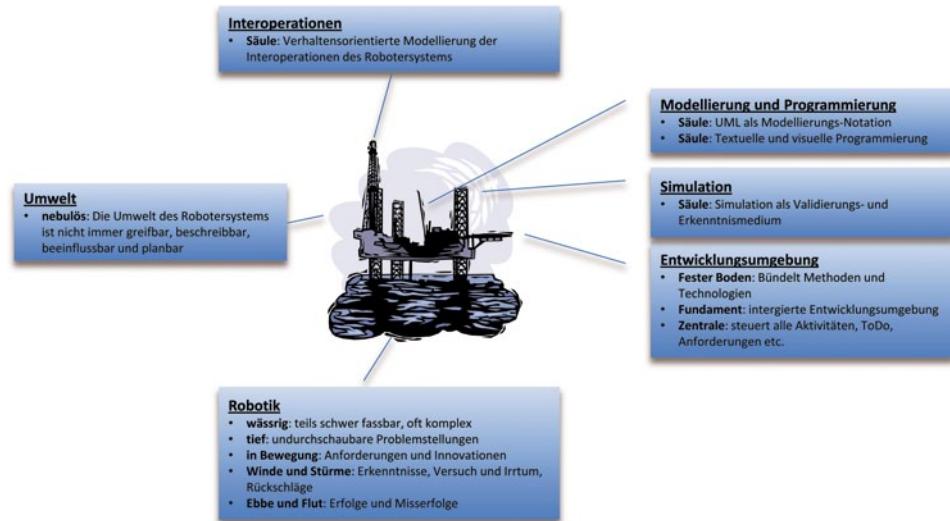
---

## 8.3 Musterlösung: Cognitive Robotic Plattform

In diesem Abschnitt wird eine Plattform für die Entwicklung von Robotersystemen vorgestellt. Dabei wird nicht auf die Anwendung einer speziellen Programmiersprache, sondern auf die nötige Entwicklungsumgebung eingegangen. Einen Überblick über die Programmiersprachen, die innerhalb der Entwicklungsumgebung angewendet werden können, findet der Leser ebenfalls im Anhang dieses Buches und dort in separaten Abschnitten. Die folgende Metapher zeigt die derzeitige Konstitution der Plattform (Abb. 8.12).

### 8.3.1 Entwicklungsumgebung

Die Systemumgebung basiert auf Eclipse und unterstützt dabei sowohl die on-line als auch die off-line Methoden. Bei den *on-line Methoden* wird der Roboter zur Programmierung benötigt. Bei den *off-line Methoden* wird das Roboterprogramm ohne Benutzung des Ro-



**Abb. 8.12** Metapher der Cognitive Robotic Plattform

boters erstellt. Das Robotersystem wird erst zum Test benötigt. Zu den off-line Methoden gehören die grafisch interaktive Programmierung und die textuelle Programmierung auf roboter- und aufgabenorientierter Ebene.

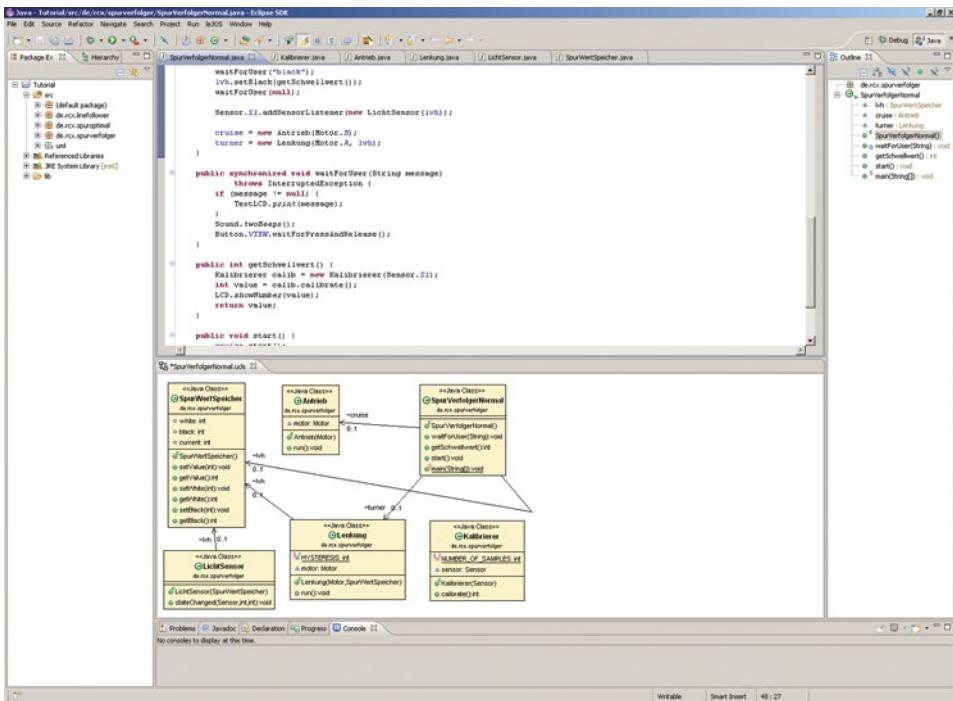
Eclipse ist ein Entwicklungsrahmen zur Integration verschiedenster Anwendungen, die zur komfortablen Entwicklung von komplexen Lösungssystemen benötigt werden. Eine solche Anwendung ist die mitgelieferte Java Entwicklungsumgebung JDT (Java Development Tooling). Ein wesentliches Kennzeichen von Eclipse ist, dass Erweiterungen in Form von *Plugins* zur Verfügung gestellt werden. Plugins werden dabei in das hierfür vorgesehenen Verzeichnis/Plugin Verzeichnis entpackt bzw. gespeichert und von Eclipse beim nächsten Start der Entwicklungsumgebung automatisch erkannt. Eclipse selbst stellt im Prinzip eine wohldefinierte Ansammlung von Plug-Ins dar.<sup>12</sup>

Eclipse verwaltet die zur Lösung benötigten Ressourcen im sogenannten *Workspace*, einem speziellen Verzeichnis innerhalb des Dateisystems. Der Zugriff auf die Ressourcen bzw. die Bearbeitung dieser erfolgt in sogenannten Fenstern, wobei mehrere dieser Fenster des Eclipse Workbench zu sogenannten *Perspektiven* zusammengefasst werden. Eine Perspektive besteht aus Views und Editoren. Ein *View* ist eine Sicht auf Ressourcen und ein *Editor* ermöglicht das komfortable Bearbeiten einer Ressource. Sowohl die Views als auch die Editoren lassen sich beliebig anordnen und kombinieren (Abb. 8.13).

Je nach Wahl der Perspektive lassen sich folgende Kernelemente der Entwicklungsumgebung erkennen:

- Der *Package Explorer* zeigt alle Projekte und Dateien des aktuellen Workspaces an.
- Der *Java-Editor* dient zur Eingabe des Quelltextes. Sind mehrere Dateien geöffnet, werden standardmäßig mehrere Registerkarten angezeigt.

<sup>12</sup> Vgl. Jobst 2001.



**Abb. 8.13** Entwicklungsumgebung

- Das *Outline Fenster* dient zur Navigation im Source-Code im aktuell geöffneten Editorenfenster. Es werden die Typen (Klassen, Interfaces), Methoden usw. angezeigt. Klickt man auf einen Eintrag, wird er im Editor selektiert. Umgekehrt wird der Fensterinhalt mit der aktuellen Position im Editor synchronisiert.
- Das *Navigator Fenster* zeigt alle Dateien des Workspaces an. Über das Kontextmenü lassen sich diese Dateien entsprechend bearbeiten.
- Das *Problem Fenster* bzw. das *Fehler Fenster* zeigt Probleme bzw. Fehler an, sofern diese festgestellt wurden. Durch einen Klick auf die entsprechende Meldung gelangt man direkt zum problem- bzw. fehlerbehafteten Code.

Das eigentliche Arbeiten in der Entwicklungsumgebung findet hauptsächlich in der sogenannten *Workbench* statt. Dieses zentrale Fenster setzt sich aus verschiedenen wichtigen Elementen zusammen, neben der Menüleiste des Hauptmenüs und der Hauptsymbolleiste (*Toolbar*) direkt darunter, sind die wichtigsten Elemente die Perspektiven (*Perspectives*), Ansichten (*Views*) und Editoren (*Editors*) platziert.

Eine *Perspektive* umfasst als Oberbegriff eine Reihe zusammengehöriger Menüs, Ansichten und Editoren. Die Perspektive erleichtert das Entwickeln innerhalb eines Projekt dahingehend, dass darin bestimmte Aufgaben, z. B. das Schreiben von *Java-Code*, sehr komfortabel erledigt werden können. Die jeweiligen Fenster lassen sich dabei beliebig an-

ordnen, neue Ansichten zusammenstellen und somit den projektspezifischen Anforderungen individuell und flexibel anpassen.

Um eine solche Perspektive zu öffnen, muss man in der Menüleiste das *Window*-Menü öffnen und die Option *Open Perspective* auswählen. In der daraufhin erscheinenden Liste wählt man die gewünschte Perspektive aus. Über den Weg der Option *Other* kann man sich eine komplette Liste aller vorhandenen Perspektiven anzeigen lassen. Eclipse verfügt in jeder Variation über diverse vordefinierte Perspektiven, die durch Plug-ins beträchtlich erweitert werden können.

Eine Ansicht (View) ist ein Arbeitsfenster, in dem die verschiedensten Informationen kontextbezogen dargestellt und entsprechend bearbeitet werden können. In der Regel besitzt eine Ansicht eine Titelleiste, in der neben dem Namen der Ansicht und dem *Close*-Symbol, auch eine ansicht-spezifische Symbolleiste (*Toolbar*) mit entsprechenden Bearbeitungssymbolen enthalten ist. Auskunft über die Funktionen, die sich hinter diesen Symbolen verbergen, erhält man, wenn man mit Mauszeiger auf dem Symbol verharrt und die daraufhin erscheinenden *Tooltiptexts* liest. Es ist möglich, mehrere Ansichten übereinander anzurufen, was durch eine entsprechende Anordnung der Registerkartenreiter angezeigt wird. Durch einen Mausklick auf den entsprechenden Reiter, wird diese Ansicht dadurch aktiviert und in den Vordergrund gestellt. Ein kleiner Nachteil dieser Navigationsphilosophie ist, dass stets nur eine Ansicht aktiv im Vordergrund gehalten werden kann.

Auch hier gilt, dass man in der Menüleiste das *Window*-Menü öffnen und die Option *Show View* auswählen muss, um sich die entsprechende Ansicht in den Vordergrund zu holen. In der daraufhin erscheinenden Liste wählt man die gewünschte Ansicht aus, sollte sie nicht angezeigt werden, kann man mit der Option *Other* eine vollständige Liste der Ansichten anzeigen lassen.

Neben diesen Struktur- und Orientierungsfunktionen stellen die sogenannten *Editoren* in Form von Fenstern die Bearbeitungsmöglichkeiten von Dateien zur Verfügung, wenn es gilt, diese anzuzeigen, zu ändern oder zu speichern. Zunächst existieren recht einfache Texteditoren, mit denen man Textdateien erstellen, ändern und speichern kann. Daneben werden aber auch spezielle Editoren angeboten, die über sprachspezifische Funktionen verfügen. So kennt beispielsweise der *Java*-Editor die komplette *Java*-Syntax und kann daher schon bei der Eingabe von *Java*-Code Syntaxfehler markieren, Schlüsselwörter, Anweisungen und Bezeichner farblich hervorheben (*syntax highlighting*) zwecks besserer Lesbarkeit sowie den Code durch Einrückung formatieren. Dabei lassen sich mehrere Editoren des gleichen Typs nebeneinander geöffnet halten.

Um den Inhalt eines Editors zu speichern, öffnet man einfach in der Menüleiste das *File*-Menü und wählt die Option *Save*, alternativ kann man auch die Tastenkombination *Strg + S* drücken. Ein Sternchen in der Titelleiste eines Editors bedeutet, dass der Editor noch ungespeicherte Änderungen enthält und ein Speichern empfohlen wird.

Eine weitere Orientierungshilfe und Auswahlfunktion sellen die zahlreichen unterschiedlichen *Menüs* dar, die Eclipse dem Entwickler anbietet. Eines der zentralen Menüs ist die Haupt-Menüleiste am oberen Rand des Hauptfensters (*Workbench*). Andere Menüs sind an bestimmte Elemente, z. B. Ansichten (*Views*), gebunden. Des weiteren gibt es zahlreiche Kontextmenüs, die geöffnet werden, indem man mit der rechten Maustaste auf ein bestimmtes Element im Hauptfenster klickt. Ergänzend hierzu existieren zahlreiche unterschiedliche Symbolleisten, wobei die wichtigste sich ebenfalls unter dem Hauptmenü befindet und demnach als Hauptsymbolleiste bezeichnet wird. Symbolleisten bestehen aus einer Reihe von Symbolen (*Buttons*), mit denen man häufig genutzte Aktionen „auf einen Klick“ ausführen kann. Wenn man mit dem Mauszeiger über einem Symbol verharrt, erscheint ein Tooltip mit näheren Informationen.

Generell entspricht die Arbeitsweise mit der Entwicklungsumgebung den Anforderungen, die man im Umgang mit komplexen Projekten zu erwarten pflegt. So lassen sich die meisten Anweisungen auf unterschiedliche Arten erteilen, indem man beispielweise mit der Maus einen Menüpunkt auswählen, mit der Maus das entsprechende Symbol in einer *Toolbar* anklicken oder aber einfach die entsprechende Tastenkombination (*Shortcut*) über die Tastatur eingeben kann. *Shortcuts* stellen zweifellos die schnellste Methode zur Bedienung dar, sind aber auch so zahlreich vorhanden, dass die Effizienz dieser Arbeitsweise schlichtweg mit dem Erlernen dieser Tastenkombinationen einhergeht. Die meisten *Shortcuts* stehen in den Menüs rechts neben dem Namen der Aktion. Für gewöhnlich besteht ein *Shortcut* aus einer Kombination von zwei Tasten, die gleichzeitig gedrückt werden müssen – gelegentlich sind es auch einmal drei Tasten. Weiterhin lassen sich die Ansichten und Editoren beliebig umgruppieren, verschieben und stapeln, wobei hier die Einschränkung zu berücksichtigen ist, dass Editoren in ihrem rechteckigen Bereich bleiben müssen und nicht mit Ansichten vermischt werden dürfen. Ebenfalls kann man bei Ansichten und Editoren auch die Größe verändern, indem man den Mauszeiger über eine Kante bewegt und dann, sobald er sich in einen Doppelpfeil verwandelt, damit das Fenster auf die gewünschte Größe einstellt und fixiert. Erfordert die Arbeit die ausschließliche Konzentration auf einen einzelnen Bestandteil der Entwicklungsumgebung, dann kann man dieses Bestandteil ganz einfach durch einen Doppelklick auf die Titelleiste in seiner Größe maximieren und in den Vordergrund stellen. Um diesen Bestandteil wieder auf die ursprüngliche Größe zu bringen, reicht ein erneuter Doppelklick auf die Titelleiste und der Bestandteil wird auf seine ursprüngliche Größe und Position zurückgeführt. Alternativ kann man auch die für Fenster üblichen Symbole zum Maximieren und Minimieren in der oberen rechten Ecke des Element-Fensters klicken. Alle Änderungen und Einstellungen des Entwicklers bleiben erhalten und werden genauso angezeigt, sobald Eclipse, bzw. die entsprechende Perspektive, das nächste Mal erneut geöffnet wird.

### 8.3.2 Simulation

In die Plattform ist auch eine Simulations-Komponente integriert und die als eine Schnittstelle zum Entwickler für das Robotersystem als auch dessen einzelne Einheiten dient.

Diese Komponente kann wie die Plattform auch über verschiedene Schnittstellen mit den Robotersystemen kommunizieren. Folgende Schnittstellen sind vorgesehen:

- *Windows-Socket Ports*: Sie werden von Einheiten verwendet, die innerhalb der Simulationsumgebung sozusagen „virtuell“ simuliert werden.
- *COM-Ports*: Hierbei handelt es sich um RS232-Schnittstellen, die zur Ansteuerung der Einheiten des Robotersystems genutzt werden können.
- *Ethernet/LAN*: Ist der Personal Computer an einem WLAN angeschlossen, so kann die Simulationsumgebung über das Netzwerk mit dem Robotersystem kommunizieren.
- *Bluetooth*: Hier wird ein virtueller COM-Port eines USB-Bluetooth Adapters angesprochen und stellt bei den in diesem Buch zum Einsatz kommenden Roboterbausätzen die zurzeit hauptsächlich zum Einsatz kommende Kommunikationschnittstelle dar.

Zur Überwachung und Steuerung aller angeschlossenen Einheiten sind verschiedene Dialoge vorgesehen, die je nach Anwendungsfall über die Menüführung aktiviert werden können:

- *Logging-Fenster*: Hier werden alle Logging-Nachrichten der Simulation (Fehlermeldungen, Warnungen, Debug-Informationen) aufgezeichnet. Diese Informationen werden auch gleichzeitig in einer Logging-Datei gespeichert, die zusätzlich über eine tabellen-artige Darstellung ausgelesen und eingesehen werden kann. Logging-Daten werden automatisch an die Plattform weitergeleitet, sofern eine Verbindung zwischen Robotersystem und der Simulationsumgebung besteht.
- *History-Fenster*: Hier werden alle Logging-Nachrichten als auch Zustandsinformationen der Simulation aufgezeichnet. Diese Informationen werden auch gleichzeitig in einer History-Datei gespeichert, die ebenfalls auch über eine tabellen-artige Darstellung ausgelesen und eingesehen werden kann. Dieses Fenster bietet also gerade zur Entwicklungszeit wichtige Informationen, um die Historie der Entwicklung verfolgen zu können.
- *Sensor-Monitor-Fenster*: Zeigt die Sensordaten der Einheiten des Robotersystems an. Die Aktualisierung kann dabei entweder explizit (Refreshing) durch Drücken der entsprechenden Schaltfläche, oder aber automatisch erfolgen. In letzterem Fall übernimmt die Simulationsumgebung je nach voreingestellter Frequenz die Datenabholung.
- *Aktor-Monitor-Fenster*: Zeigt die Aktordaten der Einheiten des Robotersystems an. Auch hier kann die Aktualisierung dabei entweder explizit (Refreshing) durch Drücken der entsprechenden Schaltfläche, oder aber automatisch erfolgen.

---

## 8.4 Musterlösung: UML4Robotik

Die Unified Modeling Language (UML) ist ein international (durch OMG und ISO) und durch die Praxis anerkannter Standard in der (objektorientierten) Softwareentwicklung.<sup>13</sup> Sie ist eine Sprache und Notation zur Spezifikation, Konstruktion, Visualisierung und

---

<sup>13</sup> Vgl. Oestreich 2001.

Dokumentation von Modellen für Softwaresysteme.<sup>14</sup> Die UML berücksichtigt die gestiegenen Anforderungen bezüglich der Komplexität heutiger Systeme, deckt ein breites Spektrum von Anwendungsgebieten ab und eignet sich für konkurrierende, verteilte, zeitkritische, sozial eingebettete Systeme u. v. m.<sup>15</sup> Dabei handelt es sich bei der UML um eine Sprache und Notation zur Modellierung, sie ist jedoch bewusst keine Methode.<sup>16</sup>

UML-Modelle umfassen grob zwei Arten von Objektmodellen:

- *Statische Modelle* (Strukturmodelle) beleuchten die Struktur von Objekten in einem System. Dazu gehören ihre Klassen, Schnittstellen, Attribute und Beziehungen zu anderen Objekten.
- *Dynamische Modelle* (Verhaltensmodelle) zeigen das Verhalten der Objekte in einem System. Hierzu zählen ihre Methoden, Zustände und Interoperationen sowie ihre Zusammenarbeit bei der Lösung einer Problemstellung.

Dabei kann man auf die folgenden Diagramme zurückgreifen:

- *Anwendungsfalldiagramm*: Akteure, Anwendungsfälle und ihre Beziehungen.
- *Klassendiagramm*: Klassen und ihre Beziehungen untereinander.
- *Aktivitätsdiagramm*: Aktivitäten, Objektzustände, Zustände, Zustandsübergänge und Ereignisse.
- *Kollaborationsdiagramm*: Objekte und ihre Beziehungen inklusive ihres räumlich geordneten Nachrichtenaustausches.
- *Sequenzdiagramm*: Objekte und ihre Beziehungen inklusive ihres zeitlich geordneten Nachrichtenaustausches.
- *Zustandsdiagramm*: Zustände, Zustandsübergänge und Ereignisse.
- *Komponentendiagramm*: Komponenten und ihre Beziehungen.
- *Verteilungsdiagramm*: Komponenten, Knoten und ihre Beziehungen.

#### 8.4.1 Anwendungsfall (Use Case)

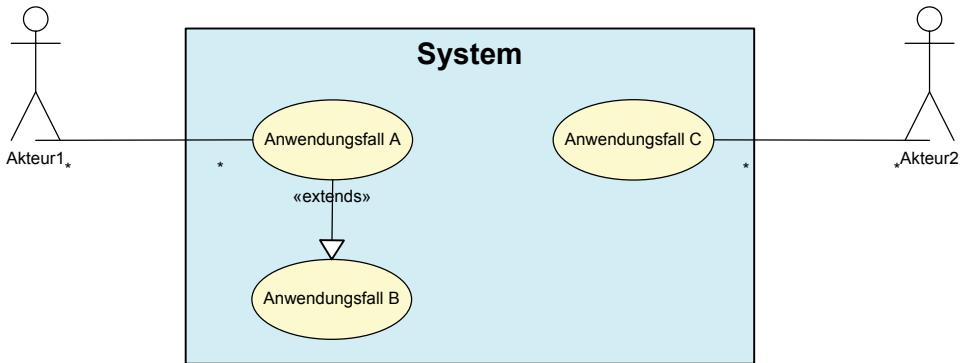
Ein *Anwendungsfall* (use case) beschreibt die Funktionalität eines Prozesses, die ein Akteur ausführen muss, um ein gewünschtes Ergebnis zu erhalten oder um ein bestimmtes Ziel zu erreichen.<sup>17</sup> Anwendungsfälle sollen es ermöglichen, mit den zukünftigen Anwendern bzw. Experten über die Funktionalität des Roboter- und dessen inhärentes Softwaresystem zu sprechen, ohne sich zugleich in Details zu verlieren. Ein *Akteur (actor)* ist eine Rolle, die ein Beteiligter in dem Gesamtvorhaben spielt. Akteure können Menschen oder auch andere automatisierte Systeme, wie beispielsweise Robotersysteme, sein. Sie befin-

<sup>14</sup> Vgl. Rumbaugh et al. 1999.

<sup>15</sup> Vgl. Bengel 2002.

<sup>16</sup> Vgl. Jacobson et al. 1999b, S. 86 ff.

<sup>17</sup> Siehe auch Davenport 1993, S. 73 ff.



**Abb. 8.14** Use Case Diagramm

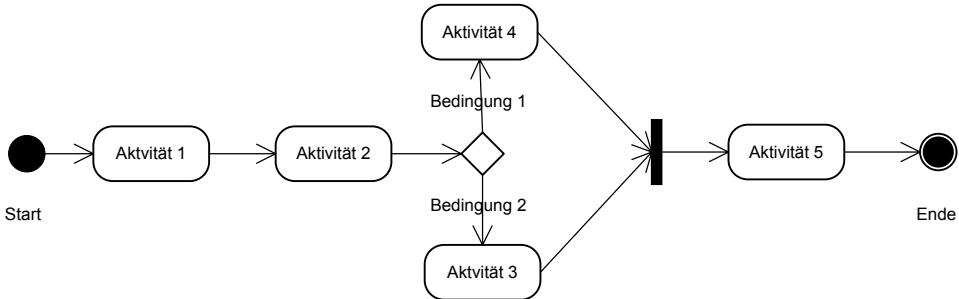
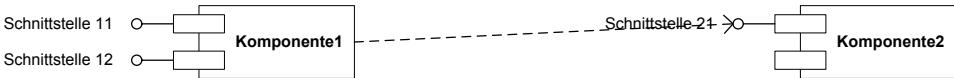
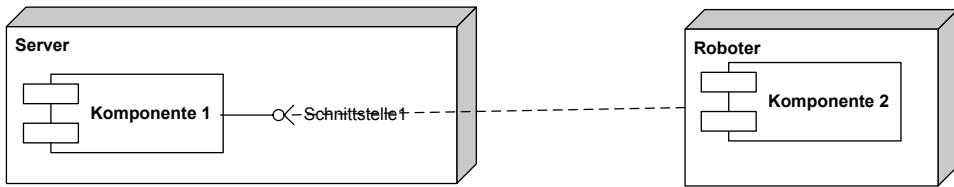
den sich stets außerhalb des Systems. Das *Use Case-Diagramm* (use case diagram) gibt auf hohem Abstraktionsniveau einen guten Überblick über den Problemlösungsprozess und seine Schnittstellen zur Umgebung. Die Akteure werden als Strichmännchen eingetragen, die Use Cases als Ovale. Eine Linie zwischen Akteur und Use Case bedeutet, dass eine Kommunikation stattfindet. Die Use Cases können optional in ein Rechteck eingetragen werden, das die Grenze des betrachteten Systems darstellt (Abb. 8.14).

Mit Hilfe der *extend-Beziehung* (extend relationship) wird ein Anwendungsfall A durch einen Anwendungsfall B erweitert. Der Anwendungsfall A beschreibt die Basisfunktionalität, der Anwendungsfall B spezifiziert Erweiterungen. Der Anwendungsfall A kann alleine oder mit den Erweiterungen von B ausgeführt werden.

Die *include-Beziehung* (include relationship) ermöglicht es, dass die gemeinsame Funktionalität von Use Case A und B durch einen Use Case C beschrieben wird. Der Use Case C kann niemals allein ausgeführt werden, sondern immer nur als Bestandteil von A oder B.

## 8.4.2 Aktivitätsdiagramm

Das *Aktivitätsdiagramm* (activity diagram) ist ein Ablaufdiagramm, mit dem die einzelnen Schritte in einem Anwendungsfall (Use Case) oder allgemein in jedem Arbeitsablauf grafisch modelliert werden können. Es ist besonders gut dafür geeignet, komplexe *Workflows* übersichtlich darzustellen. Außer dem sequentiellen Ablauf und einer Verzweigung des Prozesses entsprechend angegebener Bedingungen (guard condition) kann auch modelliert werden, dass die Reihenfolge einzelner Verarbeitungsschritte, den Aktivitäten, beliebig ist. Bei einer Gabelung (fork) verzweigt der Kontrollfluss in mehrere Pfade. Sie hat immer einen Eingangs- und mehrere Ausgangspfeile. Eine Zusammenführung (join) vereinigt die Kontrollflüsse wieder. Dementsprechend besitzt sie mehrere Eingangspfeile und einen Ausgangspfeil. Gabelungen und Zusammenführungen werden durch den Synchronisationsbalken (synchronization bar) dargestellt. Besitzt ein Synchronisationsbalken mehrere Eingangs- und Ausgangspfeile, dann handelt es sich um eine Kombination von join und fork (Abb. 8.15).

**Abb. 8.15** Aktivitätsdiagramm**Abb. 8.16** Verteilungsdiagramm**Abb. 8.17** Verteilungsdiagramm

### 8.4.3 Komponentendiagramm

Eine *Komponente* (component) ist ein physischer und austauschbarer Teil eines Systems, der bestimmte Schnittstellen (interfaces) realisiert (Abb. 8.16).

Ein *Komponentendiagramm* (component diagram) zeigt die Abhängigkeiten zwischen Komponenten. Beispielsweise wird die Abhängigkeit zwischen Komponente und der Schnittstelle durch einen gestrichelten Pfeil dargestellt.

Zur Darstellung der physischen Verteilung der ausführbaren Komponenten auf Computersystemen enthält die UML das Element *Knoten* (node). Ein Knoten ist ein Betriebsmittel, das Verarbeitungs- und/oder Speicherkapazität zur Verfügung stellt (beispielsweise ein Computer, der als Server arbeitet). Er wird durch einen dreidimensional erscheinenden rechteckigen Kasten repräsentiert. Die auf dem Knoten liegenden Komponenten werden innerhalb des Kastens dargestellt. Ein Diagramm, das zeigt, welche Komponenten auf welchem Knoten liegen und die Abhängigkeiten zwischen den Knoten darstellt, wird in der UML als *Deployment Diagram* bezeichnet (Abb. 8.17).

### 8.4.4 Klassendiagramm

Eine *Klasse* definiert für eine Kollektion von Objekten deren Struktur (Attribute), Verhalten (Operationen) und Beziehungen (Assoziationen und Vererbungsstrukturen). Sie besitzt einen Mechanismus, um neue Objekte zu erzeugen (object factory). Jedes erzeugte Objekt gehört zu genau einer Klasse. Das Verhalten (behavior) einer Klasse wird durch die Botschaften (Nachrichten) beschrieben, auf die diese Klasse bzw. deren Objekte reagieren können. Jede Botschaft aktiviert eine Operation gleichen Namens. Die Klassensymbole werden zusammen mit weiteren Symbolen, beispielsweise Assoziation und Vererbung, in das *Klassendiagramm* eingetragen. Bei großen Systemen ist es im Allgemeinen sinnvoll oder notwendig, mehrere solcher Klassendiagramme zu erstellen.

Der *Klassenname* ist stets ein Substantiv im Singular, dass durch ein Adjektiv ergänzt werden kann. Er beschreibt also ein einzelnes Objekt der Klasse. Beispiele: Mitarbeiter, PKW, Kunde. Der Klassenname muss innerhalb eines Pakets, besser jedoch innerhalb des gesamten Systems, eindeutig sein. Bei Bedarf wird er in der UML wie folgt erweitert; „Paket:Klasse“. Das Namensfeld einer Klasse kann in der UML um einen Stereotypen und eine Liste von Merkmalen erweitert werden. Ein *Stereotyp* (stereotype) klassifiziert Elemente (beispielsweise Klassen, Operationen) des Modells. Die UML enthält einige vordefinierte Stereotype und es können weitere Stereotypen definiert werden. Stereotypen werden in französischen Anführungszeichen (guillemets) mit Spitzen nach außen angegeben, beispielsweise „Stammdaten“. Ein *Merkmal* (property) beschreibt Eigenschaften eines bestimmten Elements des Modells. Mehrere Merkmale können in einer Liste zusammengefasst werden. Sie werden in der folgenden Form beschrieben: {Schlüsselwort = Wert, ...} oder nur {Schlüsselwort}.

Eine *generische Klasse* (parameterized class, template) ist eine Beschreibung einer Klasse mit einem oder mehreren formalen Parametern. Die generische Klasse Queue realisiert das Verhalten einer Warteschlange. Welche und wie viele Elemente die Queue verwalten soll, wird (noch) nicht bestimmt. Der Parameter Element beschreibt einen Typ. Daher sind für diesen Parameter keine weiteren Angaben notwendig. Der Parameter n vom Typ int gibt die maximale Größe der Warteschlange an. Diese generische Klasse bildet die Vorlage für die „normalen“ Klassen Queue <int,100>, in der maximal 100 int-Werte gespeichert werden können und FloatQueue, die maximal 20 Float-Werte enthalten kann. Von diesen beiden Klassen können dann entsprechende Objekte erzeugt werden.

Eine *Schnittstelle* (interface) besteht nur aus den Signaturen von Operationen. Sie ist formal äquivalent zu einer abstrakten Klasse, die ausschließlich abstrakte Operationen besitzt. Für die Darstellung einer Schnittstelle kann das Klassensymbol mit dem Stereotypen „interface“ und der Liste der Operationen, die von der Schnittstelle unterstützt werden, gewählt werden. Die Attributliste entfällt, denn sie ist immer leer. Die Realisierung bzw. Implementierung einer Schnittstelle durch eine Klasse wird durch den gestrichelten „Vererbungspfeil“ gekennzeichnet. Eine Schnittstelle kann alternativ durch einen kleinen Kreis dargestellt werden, unter dem der Name der Schnittstelle steht. Ist dieser Kreis mit einem Klassensymbol verbunden, so bedeutet dies, dass diese Klasse alle Operationen der

Schnittstelle (und eventuell auch mehr) zur Verfügung stellt. Verwendet eine Klasse Client Operationen der Schnittstelle, so wird dies durch einen gestrichelten Pfeil dargestellt.

*Attribute* beschreiben die Daten, die von den Objekten einer Klasse angenommen werden können. Jedes Attribut ist von einem bestimmten Typ. Alle Objekte einer Klasse besitzen dieselben Attribute, jedoch unterschiedliche Attributwerte. Attribute werden durch Angabe von Typ, Anfangswert und Merkmalen spezifiziert. Attributtypen können Standardtypen (einer Programmiersprache), Aufzählungstypen oder selbst wieder Klassen sein. Für die Definition von Aufzählungstypen verwendet UML das Klassensymbol, das mit dem Stereotypen „enumeration“ gekennzeichnet ist und die Aufzählungswerte als Attributnamen enthält. Mit dem Anfangswert wird das Attribut beim Erzeugen des Objekts initialisiert. Ein Merkmal ist beispielsweise {frozen}, das festlegt, dass der Attributwert nicht geändert werden kann. Der *Attributname* muss im Kontext der Klasse eindeutig sein. Er beschreibt die gespeicherten Daten. Im Allgemeinen wird ein Substantiv dafür verwendet. In der UML beginnen Attributnamen generell mit einem Kleinbuchstaben. Bei deutschen Bezeichnern beginnt man wegen der besseren Lesbarkeit jedoch bei Attributnamen mit einem Großbuchstaben, wenn es sich um ein Substantiv handelt. Wird die englische Sprache zur Modellierung verwendet, so sollte die UML-Regel angewendet werden. Da ein Attributname nur innerhalb der Klasse eindeutig ist, verwendet man außerhalb des Klassenkontextes die Bezeichnung Klasse.Attribut.

Ein *Klassenattribut* (class scope attribute) liegt vor, wenn nur ein Attributwert für alle Objekte einer Klasse existiert. Klassenattribute existieren auch dann, wenn es zu einer Klasse noch keine Objekte gibt. Um die Klassenattribute von den (Objekt-) Attributen zu unterscheiden, werden sie in der UML unterstrichen (beispielsweise *Klassenattribut*). Der Wert eines *abgeleiteten Attributs* (derived attribute) kann jederzeit aus anderen Attributwerten berechnet werden. Abgeleitete Attribute werden mit dem Präfix „/“ gekennzeichnet. Ein abgeleitetes Attribut darf nicht geändert werden.

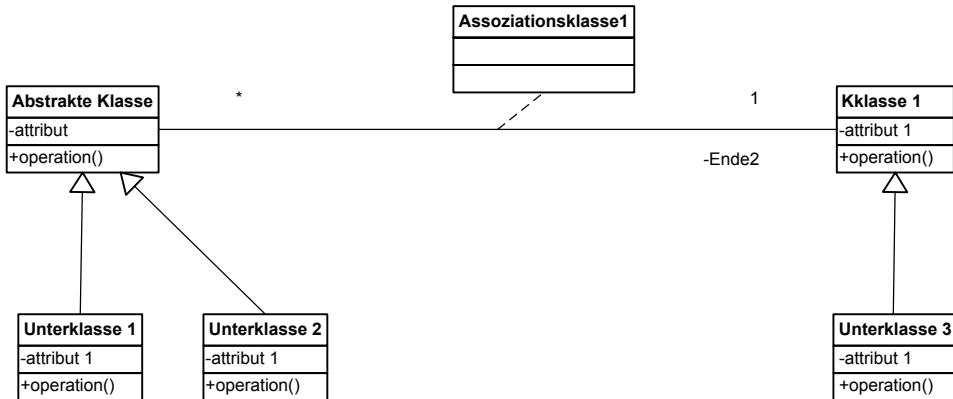
Eine *Operation* ist eine ausführbare Tätigkeit. Alle Objekte einer Klasse verwenden dieselben Operationen. Jede Operation kann auf alle Attribute eines Objekts dieser Klasse direkt zugreifen. Die Menge aller Operationen wird als das Verhalten der Klasse oder als die Schnittstelle der Klasse bezeichnet. Der *Operationsname* soll ausdrücken, was die Operation leistet. Er muß daher im Allgemeinen ein Verb enthalten, beispielsweise *verschiebe()*, *erhöhe Gehalt()*. Der Name einer Operation muss im Kontext der Klasse eindeutig sein. Außerhalb der Klasse wird die Operation mit „Klasse.Operation()“ bezeichnet (Abb. 8.18).

Eine *Klassenoperation* (class scope Operation) ist eine Operation, die der jeweiligen Klasse zugeordnet ist und nicht auf ein einzelnes Objekt der Klasse angewendet werden kann. Sie wird durch Unterstreichen gekennzeichnet.

Eine *abstrakte Operation* besteht nur aus der Signatur. Sie besitzt keine Implementierung. Abstrakte Operationen werden verwendet, um für Unterklassen eine gemeinsame Schnittstelle zu definieren. In der UML werden abstrakte Operationen kursiv eingetragen oder – beispielsweise bei handschriftlicher Modellierung – mittels {abstract} gekennzeichnet.<sup>18</sup> Die *Signatur* (Signature) einer Operation besteht aus dem Namen der Operation,

---

<sup>18</sup> Siehe auch Booch et al. 1999.



**Abb. 8.18** Klassendiagramm

den Namen und Typen aller Parameter und dem Ergebnistyp. Analog zu den Attributen wird auch für Operationen im Entwurf die *Sichtbarkeit* angegeben.

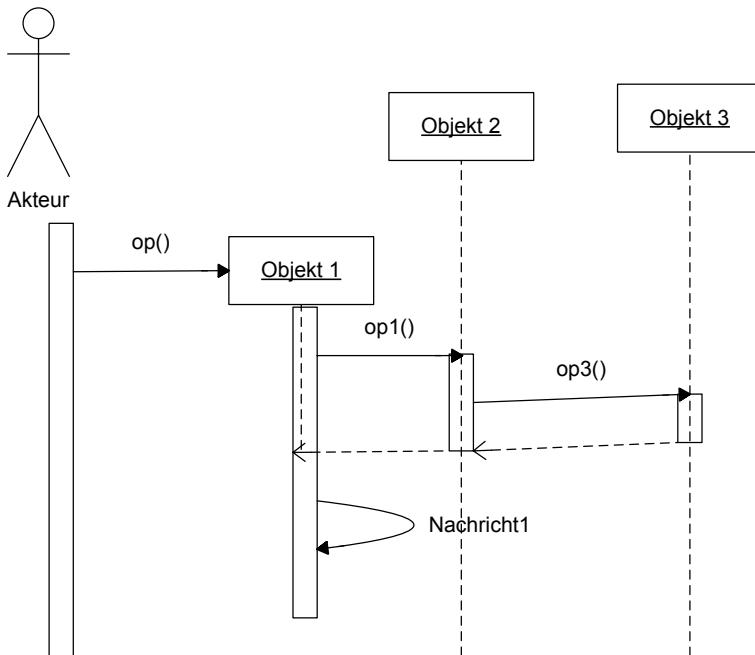
Das *Klassendiagramm* (class diagramm) stellt die Klassen mit Attributen und Operationen, die Vererbung und die Assoziationen zwischen ihnen dar. Außerdem können Pakete und ihre Abhängigkeiten im Klassendiagramm modelliert werden.

#### 8.4.5 Sequenzdiagramm

Ein *Sequenzdiagramm* besitzt zwei Dimensionen: die Vertikale repräsentiert die Zeit, auf der Horizontalen werden die Objekte eingetragen (Abb. 8.19).

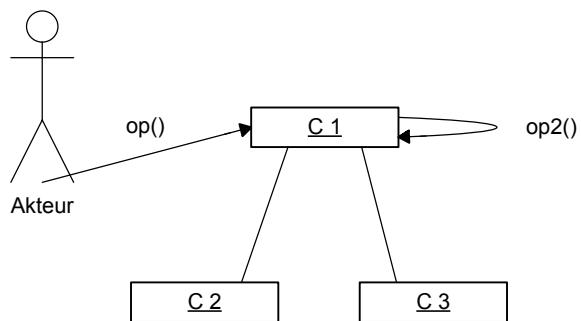
Jedes Objekt wird durch eine gestrichelte Linie im Diagramm, die *Objektlinie*, dargestellt. Sie repräsentiert die Existenz eines Objekts während einer bestimmten Zeit. Die Linie beginnt nach dem Erzeugen des Objekts und endet mit dem Löschen des Objekts. Existiert ein Objekt während der gesamten Ausführungszeit des Szenarios, so ist die Linie von oben nach unten durchgezogen. Am oberen Ende der Linie wird ein *Objektsymbol* angetragen. Wird ein Objekt im Laufe der Ausführung erst erzeugt, dann zeigt eine Botschaft auf dieses Objektsymbol. Das Löschen des Objekts wird durch ein großes „X“ markiert. Die Reihenfolge der Objekte ist beliebig. Sie soll so gewählt werden, dass ein übersichtliches Diagramm entsteht. Bei diesen Objekten handelt es sich im Allgemeinen nicht um spezielle Objekte, sondern um Stellvertreter für beliebige Objekte der angegebenen Klasse. Daher werden sie häufig als anonyme Objekte (d. h. *Klasse*) bezeichnet.

In das Sequenzdiagramm werden die *Botschaften* eingetragen, die zum Aktivieren der Operationen dienen. Jede Botschaft wird als gerichtete Kante (mit gefüllter Pfeilspitze) vom Sender zum Empfänger gezeichnet. Der Pfeil wird mit dem Namen der aktivierte Operation beschriftet. Die Botschaft aktiviert eine Operation gleichen Namens. Diese wird durch ein schmales Rechteck auf der Objektlinie angezeigt. Nach dem Beenden der Operation zeigt eine gestrichelte Linie mit offener Pfeilspitze, dass der Kontrollfluss zur aufrufenden Operation zurückgeht.



**Abb. 8.19** Sequenzdiagramm

**Abb. 8.20** Kollaborationsdiagramm



### 8.4.6 Kollaborationsdiagramm

Ein *Kollaborationsdiagramm* (collaboration diagram) bildet eine Alternative zum Sequenzdiagramm (Abb. 8.20).

Das hier dargestellte Kollaborationsdiagramm modelliert den gleichen Ablauf wie das Sequenzdiagramm der vorherigen Abbildung. Es beschreibt die Objekte, und zusätzlich zum Sequenzdiagramm, die Verbindungen zwischen diesen Objekten. An jede Verbindung (link) kann eine Botschaft in Form eines Pfeiles angetragen werden. Im Kollaborationsdia-

gramm sendet der Akteur die Botschaft `op()`, die ein Objekt der Klasse `C1` erzeugt. Dieses Objekt aktiviert dann zuerst die Operation `op1()` und dann `op2()`. Diese Reihenfolge wird durch die Nummerierung ausgedrückt. Die Operation `op2()` mit der Nr. 2 ruft nun ihrerseits die Operation `op3()` mit der Nr. 2.1 auf. Bei diesen Objekten handelt es sich im Allgemeinen nicht um spezielle Objekte, sondern um Stellvertreter für beliebige Objekte der angegebenen Klasse. Daher werden sie häufig als anonyme Objekte (d. h. *Klasse*) bezeichnet.

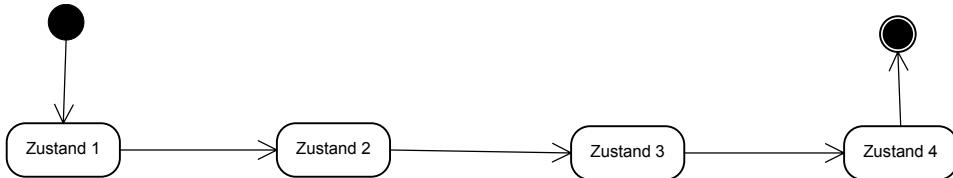
In das Sequenzdiagramm werden die *Botschaften* eingetragen, die zum Aktivieren der Operationen dienen. Jede Botschaft wird als gerichtete Kante (mit gefüllter Pfeilspitze) vom Sender zum Empfänger gezeichnet. Der Pfeil wird mit dem Namen der aktivierte Operation beschriftet. Die Botschaft aktiviert eine Operation gleichen Namens. Diese wird durch ein schmales Rechteck auf der Objektlinie angezeigt. Nach dem Beenden der Operation zeigt eine gestrichelte Linie mit offener Pfeilspitze, dass der Kontrollfluss zur aufrufenden Operation zurückgeht.

### 8.4.7 Zustandsdiagramm

Ein *Zustandsautomat* (finite state machine) besteht aus Zuständen und Zustandsübergängen (Transitionen). Ein Zustand ist eine Zeitspanne, in der ein Objekt auf ein Ereignis wartet, d. h. das Objekt verweilt eine bestimmte Zeit in diesem Zustand. In diesen Zustand gelangt das Objekt durch ein entsprechendes Ereignis. Ein Ereignis tritt immer zu einem Zeitpunkt auf und besitzt keine Dauer. Ein Objekt kann nacheinander mehrere Zustände durchlaufen. Zu einem Zeitpunkt befindet es sich in genau einem Zustand. Tritt in einem beliebigen Zustand ein Ereignis ein, so hängt der nächste Zustand sowohl vom aktuellen Zustand als auch vom jeweiligen Ereignis ab. Der Zustand eines Objekts beinhaltet also implizit Informationen, die sich aus den bisher vorgenommenen Eingaben ergeben haben.

Der Name des *Zustands* ist optional. Zustände ohne Namen heißen *anonyme Zustände* und sind alle voneinander verschieden. Ein benannter Zustand kann dagegen, der besseren Lesbarkeit halber, mehrmals in das Diagramm eingetragen werden. Diese Zustände sind alle identisch. Innerhalb eines Zustandsautomaten muss jeder Zustandsname eindeutig sein. Mit einem Zustand können *Aktionen* oder Aktivitäten verbunden sein. Eine *entry-Aktion* ist atomar. Sie wird beim Eintritt in den Zustand, unabhängig davon, durch welche Transition der Eintritt in diesen Zustand erfolgt, immer ausgeführt und terminiert selbstständig. Eine *exit-Aktion* ist ebenfalls atomar. Sie wird immer ausgeführt, wenn der entsprechende Zustand durch eine beliebige Transition verlassen wird. Eine *Aktivität* beginnt, wenn das Objekt den Zustand einnimmt und endet, wenn es ihn verlässt. Sie kann alternativ durch ein Paar von Aktionen, eine zum Starten und eine zum Beenden der Aktivität, beschrieben oder durch ein weiteres Zustandsdiagramm verfeinert werden. Zusätzlich können weitere interne Aktionen angegeben werden, die durch bestimmte Ereignisse aktiviert werden.

Jeder Zustandsautomat muss einen Anfangszustand und kann einen Endzustand besitzen. Der *Anfangszustand* (initial state) wird durch einen kleinen schwarzen Kreis dar-



**Abb. 8.21** Zustandsdiagramm

gestellt. Es handelt sich um einen Pseudozustand, der mit einem „echten“ Zustand durch eine Transition verbunden ist. Diese Transition kann mit einem Ereignis zum Erzeugen des Objekts beschriftet sein. Der Anfangszustand ist ein grafisches Hilfsmittel. Ein Objekt kann diesen Zustand nicht annehmen, sondern ein neu erzeugtes Objekt befindet sich zunächst in dessen „echten“ Folgezustand. Im *Endzustand* (final state) hört ein Objekt auf zu existieren. Aus diesem Zustand führen keine Transitionen heraus. Der Endzustand, der ebenfalls ein Pseudozustand ist, wird durch ein „Bullauge“ dargestellt und kann optional beschriftet sein.

Eine *Transition* bzw. ein Zustandsübergang verbindet zwei Zustände. Sie wird durch einen Pfeil dargestellt. Eine Transition kann nicht unterbrochen werden, und wird stets durch ein Ereignis ausgelöst. Man sagt: die Transition „feuert“. Tritt ein Ereignis ein und das Objekt befindet sich nicht in einem Zustand, in dem es darauf reagieren kann, wird das Ereignis ignoriert. Meistens ist mit einer Transition ein Zustandswechsel verbunden. Es ist aber auch möglich, dass Ausgangs- und Folgezustand identisch sind. In solch einem Fall werden die entry- und exit-Aktionen bei jedem neuen Eintritt in denselben Zustand ausgeführt.

Ein *Ereignis* kann sein:

- eine Bedingung, die wahr wird,
- ein Signal,
- eine Botschaft (Aufruf einer Operation),
- eine verstrichene Zeit (elapsed time event) oder
- das Eintreten eines bestimmten Zeitpunkts.

Eine Bedingung ist beispielsweise: when (Temperatur > 100 Grad). Die zugehörige Transition feuert, wann immer diese Bedingung wahr wird (Abb. 8.21).

Signale und Botschaften sind durch die Notation nicht unterscheidbar. Sie werden durch einen Namen beschrieben und können Parameter besitzen, beispielsweise rechte Maustaste gedrückt (Mausposition). Eine Transition kann mit einem *Wächter* (guard condition) beschriftet sein. Es handelt sich um einen logischen Ausdruck, der ausgewertet wird, wenn das zugehörige Ereignis eintritt. Nur wenn die spezifizierte Bedingung erfüllt ist, feuert die Transition.

In der Objektorientierung wird das Zustandsdiagramm (state chart diagramm) zur grafischen Darstellung des Zustandsautomaten verwendet.

## 8.5 Musterlösung: Java4Robotic

Ein Robotersystem verarbeitet je nach Typus während seines Lebenszyklus eine Unmenge an Daten. Der Verarbeitung von Daten als Themengebiet widmet sich vor allem die Informatik, indem diese Wissenschaftsdziplin Verfahren und Techniken untersucht, beschreibt und realisiert, mit denen aus Eingabedaten systematisch Ausgabedaten berechnet werden können. Eine solche systematische Berechnung läuft schrittweise ab, wobei eine Verfahrensvorschrift die Abfolge der Berechnungsschritte sowie die Operationen jedes einzelnen Schritts festlegt. Eine derartige Verfahrensvorschrift wird als *Algorithmus* bezeichnet.

### 8.5.1 Vom Algorithmus zum (objektorientierten) Programm

Verfahrensvorschriften gibt es allerdings nicht nur in der Informatik, sondern auch in den meisten anderen Wissenschaften und insbesondere auch im alltäglichen Leben: so liefert ein Kochrezept zunächst die Anweisungen zur Zubereitung einer Mahlzeit, eine Montageanleitung beschreibt, wie man ein Möbelstück zusammenzufügen hat, und ein arithmetisches Verfahren definiert beispielsweise eine Methode zur Addition zweier Zahlen. Allen Verfahrensvorschriften ist gemein, dass sie eine Folge von Schritten festlegen, mit denen aus einer Eingabe (input) oder diversen Eingabewerten bestimmte Ausgaben, (output) oder definierte Ausgabewerte ermittelt werden. Eine Verfahrensvorschrift sollte so präzise formuliert sein, dass stets eindeutig klar ist, was getan werden soll. Die meisten Verfahrensvorschriften des Alltags sind zwar relativ detailliert, legen aber nicht alle Schritte bis ins Letzte fest: so bleibt beispielsweise offen, wie viele Gramm „eine Prise Pfeffer“ sind. Nur manche Verfahren, wie die Methoden für die schriftliche Addition, Multiplikation und so weiter, sind so exakt festgelegt, dass sie keine Freiheiten offen lassen. Während die menschliche Lebenserfahrung und Intelligenz auch mit nicht ganz vollständigen Verfahrensvorschriften zureckkommt (Ausnahmen bestätigen hier die Regel), verlangt die Informatik nach Vorschriften, die bis ins Detail festgelegt sind. Diese Verfahrensvorschriften der Informatik sollen von Maschinen, insbesondere Computern (in Robotersystemen), ausgeführt werden. Insofern wird im Folgenden immer dann von einem Algorithmus gesprochen, wenn damit ein maschinell ausführbares Verfahren beschrieben wird: ein Verfahren, das für die Bearbeitung durch einen Computer hinreichend präzise beschrieben ist.

Ein Algorithmus weist die folgenden charakteristischen Eigenschaften auf:

- *Problemlösungsfähigkeit*: Ein Algorithmus löst eine allgemeine Problemstellung. So führt er Berechnungen nicht nur für bestimmte wenige instanziellen Eingabewerte aus, sondern berechnet für alle Eingabewerte aus einer größeren Wertemenge die zugehörigen Resultate.
- *Korrektheit*: Ein Algorithmus ist korrekt, d. h. er liefert für alle zulässigen Eingabewerte richtige Resultate. Speziell hierzu existieren Verifikationsverfahren, die die Korrektheit eines Algorithmus formal beweisen.

- *Endlichkeit*: Ein Algorithmus besitzt eine endliche Beschreibung. Die Verarbeitungsvorschrift lässt sich durch einen endlich langen Text oder eine begrenzte Grafik eindeutig definieren.
- *Einfachheit*: Ein Algorithmus ist aus Einzelschritten aufgebaut. In jedem dieser Schritte wird eine einfache Operation durchgeführt, wie beispielsweise die Auswertung eines arithmetischen oder logischen Ausdrucks und/oder die Zuweisung eines Werts an eine Variable. Die Operationen sollten so einfach gehalten sein, dass sie auch von einer Maschine unmittelbar ausgeführt werden können.
- *Kontrollfähigkeit*: Ein Algorithmus legt fest, in welcher Abfolge die Einzelschritte ausgeführt werden. Hierfür können Kontrollstrukturen benutzt werden, die den Kontrollfluss bei der Ausführung bestimmen, also die Reihenfolge, in der die einzelnen Schritte bearbeitet werden. Für die schriftliche Addition ist beispielsweise festgelegt, dass man nach Bearbeitung einer Spalte mit der Spalte unmittelbar links davon fortfahren muss. Die einfachste Form einer Kontrollstruktur ist die *Sequenz* mit der Bearbeitung der Schritte „von oben nach unten“, also in der Reihenfolge, in der die Anweisungen im Text der Verarbeitungsvorschrift aufgeführt sind. Weitere häufig benutzte Kontrollstrukturen sind Verzweigungen in Abhängigkeit von logischen Bedingungen, Schleifen zur wiederholten Ausführung von Schrittfolgen oder Sprünge an andere Stellen der Verarbeitungsvorschrift.

Von Algorithmen, die auf Computern eines Robotersystems laufen sollen, verlangt man häufig, daß sie *deterministisch* arbeiten und *terminieren*. Für Robotersysteme, die nur mit einem zentralen Computer und letztere wiederum mit nur einem Prozessor ausgestattet sind, sind zudem *sequentielle* Algorithmen interessant:

- Ein *deterministischer Algorithmus* beginnt in einem festgelegten Anfangszustand mit einem ebenso festgelegten ersten Schritt. Nach jedem Schritt ist eindeutig bestimmt, welcher Schritt als nächster ausgeführt wird, oder dass die Ausführung an dieser Stelle endet.
- Ein *terminierender Algorithmus* liefert sein Ergebnis stets nach Ausführung einer endlichen Anzahl von Schritten, also nach endlich langer Zeit. Das Additionsverfahren hat auch diese Eigenschaft.
- Ein *sequentieller Algorithmus* führt jeweils nur einen Schritt aus, also nie mehrere Schritte gleichzeitig.

Das Wort Algorithmus hat zwei Wurzeln, nämlich den griechischen Begriff *arithmos* (= Zahl) und den arabischen Namen *AI-Chwarismi. Mohammed Ibn Musa AI-Chwansmi* (um 780 – um 850, Nachname auch in anderen Schreibweisen) arbeitete am Hof des Kalifens von Bagdad als Mathematiker, Astronom, Geograph und Historiker. Er veröffentlichte dort Arbeiten zu Arithmetik und Algebra, gestützt auf mathematische Erkenntnisse und Methoden aus den babylonischen, griechischen, iranischen und indischen Kulturkreisen. Der Begriff der *Algebra* ist aus dem Titel einer dieser Werke abgeleitet. Zu AI-Chwarismis größten Verdiensten gehört die Übernahme des indischen Dezimalsystems mit den Zif-

fern 1 bis 9 sowie insbesondere der 0, die Beschreibung von Rechenverfahren in diesem Zahlensystem und die Darstellung von Lösungsmethoden für lineare und quadratische Gleichungen. Da die Arbeiten Al-Chwarismis später ins Lateinische übersetzt wurden, gingen seine Erkenntnisse auch in den europäischen Raum ein. Ein Algorithmus mit diesen drei zusätzlichen Eigenschaften ist also ein Verfahren, das nach der Eingabe von Werten eine eindeutig bestimmte Abfolge von Schritten ausführt und nach einer endlich langen Ausführungsdauer ein eindeutiges, nicht zufallsabhängiges Ergebnis liefert. Es gibt jedoch auch Algorithmen, die eine oder mehrere dieser Eigenschaften nicht besitzen:

- Bei *nichtdeterministischen Algorithmen* sind nach manchen Schritten mehrere Folgeschritte möglich. Wird bei der Ausführung einer der Schritte zufällig ausgewählt, so spricht man von einem zufallsabhängigen oder stochastischen Algorithmus. Man differenziert diese Verfahren dann weiter in determinierte und nichtdeterminierte Algorithmen: Ein determinierter Algorithmus liefert für eine bestimmte Eingabe stets das gleiche Resultat, gelangt aber möglicherweise auf unterschiedlichen Wegen dorthin. Ein nichtdeterminierter Algorithmus kommt dagegen für dieselbe Eingabe bei mehreren aufeinander folgenden Ausführungen möglicherweise zu unterschiedlichen Resultaten. Der praktische Wert solcher Algorithmen liegt darin, dass sie das korrekte (oder ein hinreichend „gutes“) Ergebnis mit hoher Wahrscheinlichkeit liefern, und dass diese Wahrscheinlichkeit durch Mehrfachausführung gesteigert werden kann. Ein Beispiel hierfür sind genetische Algorithmen, die die natürliche Fortpflanzung mit Kreuzung und Mutation von Chromosomen sowie den nachfolgenden Ausleseprozess simulieren. Sie lassen sich gut zur Lösung von Optimierungsproblemen einsetzen.
- *Nichtterminierende Algorithmen* liefern zwar während ihrer Ausführung Zwischenergebnisse, erreichen aber möglicherweise nie einen Endzustand mit einem Endresultat. Sie dienen beispielsweise dazu, Näherungswerte mit ständig wachsender Genauigkeit zu berechnen, wenn das exakte Resultat nicht in endlich vielen Schritten ermittelt werden kann.
- *Nebenläufige Algorithmen* bearbeiten die Schritte nicht notwendigerweise sequentiell (einen nach dem anderen) ab, sondern können Schritte oder Schrittfolgen auch nebenläufig (also gleichzeitig) ausführen. Solche Algorithmen sind vor allem für Multiprozessor-Computer oder Rechnernetze interessant, in denen mehrere Prozesse parallel arbeiten.

Ist man in der Lage, ein vorstelliges Problem verbal bzw. den Algorithmus als Lösung zu beschreiben, so erfüllt man damit zwar eine notwendige, aber noch keine hinreichende Bedingung dafür, gute Programme schreiben zu können. Ähnliches zeigt sich, im übertragenen Sinne, fast täglich in der Zeitung oder in sonstigen Druckwerken: Die Kenntnis der Syntax und Grammatik der deutschen Sprache ist noch lange kein Garant dafür, auch gut strukturierte, verständliche Texte schreiben zu können<sup>19</sup>. Die Entwicklung korrekter, verständlicher und effizienter Algorithmen kann als Kunst aufgefasst werden, denn das

---

<sup>19</sup> Chomsky 1965.

Entwickeln von Algorithmen ist kein rein technischer Vorgang, sondern erfordert zusätzlich eine gewisse Intuition bzw. Inspiration. Am Anfang muss eine Idee dafür vorhanden sein, wie man ein bestimmtes Problem allgemein lösen kann. Hier hilft eine Problem-analyse, also die Überlegung, welche Eingabewerte auftreten können und in welcher Beziehung die jeweils gewünschten Ausgaben dazu stehen. Man muss dann versuchen, aus diesen vergleichsweise abstrakt formulierten Beziehungen konkrete Schritte abzuleiten, mit denen man Eingabewerte in zugehörige Ausgabewerte überführen kann. Bei der Entwicklung dieser Schritte kann man *top-down* vorgehen. Dabei legt man zunächst nur die groben Schritte des Algorithmus fest und verfeinert sie anschließend stufenweise, bis hin zu einfachen Operationen. Die Top-Down-Vorgehensweise ermöglicht zudem unmittelbar die Modularisierung eines Algorithmus, also seine Strukturierung in ein System kleinerer Unteralgorithmen. Solch ein System ist unter Umständen transparenter als ein einzelner großer Algorithmus. Man muss insbesondere beim Entwurf von Unteralgorithmen „das Rad nicht immer wieder neu erfinden“, sondern kann auf bekannte Algorithmen zur Lösung von Standardproblemen zurückzugreifen. Fehlt bereits zu Beginn die zündende Idee zu einer allgemeinen Lösung des Problems, so kann man auch versuchen, zunächst Ansätze für Spezialfälle zu finden und diese dann zu einer Lösung des generellen Problems erweitern. Dies sollte man aber möglichst während der Problemanalyse tun, also *vor* der Entwicklung des Algorithmus mit seinen Schritten. Algorithmen, die lediglich aus einer Sammlung von Speziallösungen bestehen, sind meist nicht sehr übersichtlich und elegant. Abschließend ist anzumerken, dass die hier beschriebene Vorgehensweise nur zur Lösung relativ kleiner Aufgabenstellungen ausreicht. Zur Entwicklung größerer Robotersysteme müssen zusätzlich Methoden des *Robotik Engineerings* herangezogen werden um die Übersicht über Entwicklungsprozesse, Algorithmen, Programme, Komponenten und Hardware zu behalten.

Soll ein Algorithmus auf einem Computer konkret ausgeführt werden, muss er durch ein *Programm* beschrieben werden. Das kann entweder in einer höheren Programmiersprache, wie beispielsweise C, C++ oder Java, geschehen oder in der Maschinensprache eines Prozessors. Programme in Maschinensprache können unmittelbar durch den Prozessor ausgeführt werden, Programme in höheren Sprachen müssen in entsprechende Maschinenbefehle umgesetzt werden. Insofern wird die Software von Computern im Allgemeinen und für Robotersysteme im Speziellen in einer Programmiersprache formuliert. Man unterscheidet zwei große Gruppen von Programmiersprachen, nämlich hardware-nahe Sprachen und höhere Sprachen. Hardwarenahe Sprachen definieren relativ primitive Anweisungen, die sich an den beschränkten Fähigkeiten eines realen Computerprozessors orientieren. Programme in diesen Sprachen sind daher unmittelbar (im Fall von Maschinensprachen) oder nach einer einfachen Umsetzung (im Fall von Assemblersprachen) auf Prozessoren ausführbar. Höhere Programmiersprachen ermöglichen eine Problemlösung in einer anwendungsorientierten, prozessorunabhängigen Form. Programme in diesen Sprachen können nicht unmittelbar auf einem Prozessor ausgeführt werden, sondern müssen dazu übersetzt oder interpretiert werden. Man unterscheidet imperative und nicht-imperative Sprachen. Imperative Programmiersprachen basieren auf dem Algorith-

musbegriff, legen also durch ihre Anweisungen die Schritte fest, die zur Lösung eines Problems ausgeführt werden sollen. Prozedurale Sprachen, die zu dieser Gruppe gehören, definieren entsprechende Kontrollstrukturen (Blöcke, Verzweigungen, Schleifen) sowie ein Unterprogrammkonzept. In objektorientierten Sprachen sind die Unterprogramme den Daten untergeordnet, auf denen sie arbeiten; sie bilden mit ihnen so genannte Objekte. Nichtimperative Sprachen abstrahieren dagegen von den Einzelschritten eines Algorithmus und spezifizieren nur die Eigenschaften, die die gewünschte Problemlösung haben soll. Zu diesen Sprachen gehören die funktionalen Sprachen, die sich am mathematischen Funktionsbegriff orientieren, die Logiksprachen, die auf den Techniken der Aussagenlogik basieren, sowie Anfragesprachen für Datenbanken.<sup>20</sup>

Namensgebend für die *objektorientierte Programmierung* ist der Begriff des Objekts. Ein Objekt ist eine gekapselte, modulare Einheit aus Daten (Attributen) und Operationen (Methoden), auf die nur über eine Schnittstelle zugegriffen werden darf und die Implementierungsdetails in ihrem Körper verstecken kann. Objekte sind typisiert, also bestimmten Klassen zugeordnet. Eine Klasse legt die Eigenschaften einer Gruppe gleichartiger Objekte fest und definiert damit auch einen Bauplan zur Konstruktion neuer Objekte. Unterschiedliche Klassen können gemeinsame Eigenschaften besitzen, was durch Klassenhierarchien ausgedrückt wird. Basisklassen, die in der Hierarchie weiter oben angesiedelt sind, vereinbaren allgemeinere Eigenschaften und vererben diese an spezifischere abgeleitete Klassen, die in der Hierarchie weiter unten stehen. Abstrakte Basisklassen definieren nicht alle ihre Methoden vollständig aus, sondern überlassen dies den von ihnen abgeleiteten Klassen.

Bei der objektorientierten Programmierung ist das Konzept der Polymorphie wichtig, bei dem sich hinter demselben Methodennamen unterschiedliche Methoden verbergen können: Methoden können einander überladen (also denselben Namen, aber unterschiedliche Parameterlisten besitzen) oder sogar überschreiben (also in Name und Parameterliste übereinstimmen). Welche Methode bei einem Aufruf jeweils gemeint ist, wird erst zur Laufzeit aufgrund des dynamischen Typs des betroffenen Objekts festgestellt, d. h. anhand der Klasse, der das Objekt angehört. Es findet also während der Programmausführung ein „spätes Binden“ (late binding) von Namen an Methoden statt. In der heutigen Praxis spielt die objektorientierte Programmiersprache Java gerade im Bereich der Robotik eine wichtige Rolle.

Die grundlegenden Ideen der objektorientierten Programmierung sind schon mehrere Jahrzehnte alt: Die Begriffe des Objekts und der Klasse wurden erstmals in der Programmiersprache *Simula* verwendet, die bereits Mitte der Sechziger Jahre des zwanzigsten Jahrhunderts entwickelt wurde. Simula war eine Sprache, die sich besonders zur Simulation von nebenläufigen, ereignisgesteuerten Systemen eignete. Die erste rein objektorientierte Sprache im heutigen Sinne ist *Smalltalk* aus den Siebziger Jahren, die im Xerox Palo Alto Research Center (Xerox PARC) entwickelt wurde. In Smalltalk werden selbst elementare Daten wie Zahlen oder Zeichen durch Objekte dargestellt. Der Objektbegriff wird hier also durchgehend verwendet, im Unterschied beispielsweise zu C++ und Java, die die skalaren

<sup>20</sup> Siehe auch Abts 2003.

Variablen von C übernommen haben. Smalltalk zeichnet sich durch komfortable Werkzeuge zur Programmentwicklung aus und bietet eine starke Unterstützung für die Programmierung grafischer Benutzeroberflächen. In den Achtziger Jahren erweiterte Björne Stroustrup die Sprache C um objektorientierte Techniken und nannte die resultierende Programmiersprache C++. Da C++ zu C rückwärtskompatibel ist (also alle Konstrukte von C unterstützt), ist C++ keine rein objektorientierte, sondern eine *hybride* Sprache, das heißt eine Mischform. Weitere Sprachen, die in diesem Zeitraum entwickelt wurden, sind *Modula* und *Oberon* sowie *Eiffel*. Sie haben sich allerdings in der Praxis nicht so stark durchgesetzt wie seinerzeit C++ und heutzutage Java. Java ist das Produkt eines Teams der Firma Sun Microsystems aus der Mitte der Neunziger Jahre. Ursprüngliches Ziel war der Entwurf einer Sprache zur Steuerung von Geräten (also zur Programmierung so genannter eingebetteter Systeme), die möglichst plattformunabhängig sein sollte. Mit dem Aufkommen des World Wide Webs zeigte sich dann das Potential Javas zur Programmierung mobiler Codes, also von Programmen, die über das Internet auf eine beliebige Rechnerplattform übertragen und dort ausgeführt werden können.<sup>21</sup> Die Syntax von Java lehnt sich an die von C++ an, um C++ -Programmierern den Umstieg zu erleichtern, vereinfacht aber einige Dinge. So ist Java nicht rückwärtskompatibel zu C, sondern rein objektorientiert. Auch gibt es in Java keine Zeiger. Hierdurch wurde eine große Sicherheitslücke geschlossen, denn in Java kann man somit nur auf definierte Variablen und Objekte zugreifen, nicht jedoch auf beliebige Speicherbereiche, wie in C oder C++ möglich.

### 8.5.2 Struktur eines Java Programms

Ein Java-Programm zeichnet sich durch eine ganz bestimmte Struktur aus elementaren Sprachelementen aus. So besteht ein Java-Programm aus *Klassen*.<sup>22</sup> Meist verwendet man für den Quellcode einer Klasse jeweils eine eigene Datei. Der Compiler erzeugt auf jeden Fall für jede Klasse eine eigene Bytecodedatei. Von den Klassen eines Programms muss mindestens eine startfähig sein. Dazu benötigt sie eine *Methode* mit dem Namen main, dem Rückgabetyp void, eine wohl definierte Parameterliste (String[] args) und den Modifikatoren public und static. Diese main-Methode wird beim Starten der Klasse vom Interpreter ausgeführt. Daher muss sie der Öffentlichkeit zugänglich (Modifikator public) und als Klassenmethode unabhängig von der Existenz konkreter Objekte ausführbar sein (Modifikator static). Auch die Startklasse selbst muss öffentlich sichtbar sein, was aber auch ohne explizite Verwendung des Modifikators public der Fall ist.

Sowohl die Klassen- als die Methodendefinitionen bestehen aus einem Kopf und einem Rumpf. Letzterer besteht aus einem Block mit beliebig vielen Anweisungen, mit denen beispielsweise Variablen definiert oder verändert werden. Ein Klassen- oder Methodenrumpf wird durch geschweifte Klammern begrenzt. Eine *Anweisung* ist die kleinste ausführbare

---

<sup>21</sup> Vgl. Corner 2000.

<sup>22</sup> Vgl. Middendorf et al. 1996.

Einheit eines Programms. In Java sind bis auf wenige Ausnahmen alle Anweisungen mit einem Semikolon abzuschließen. Zur Formatierung von Java-Programmen haben sich Konventionen entwickelt, wenngleich der Compiler hinsichtlich der Formatierung sehr tolerant ist. Lediglich einige wenige Regeln gilt es zu beachten:

- Die einzelnen Bestandteile einer Definition oder Anweisung müssen in der richtigen Reihenfolge stehen.
- Zwischen zwei Sprachbestandteilen muß im Prinzip ein Trennzeichen stehen, wobei das Leerzeichen, das Tabulatorzeichen und der Zeilenumbruch erlaubt sind. Diese Trennzeichen dürfen sogar in beliebigen Anzahlen und Kombinationen auftreten. Zeichen mit festgelegter Bedeutung wie beispielsweise „;“, „(„, „+“, „>“ sind selbstbegrenzend, d. h. vor und nach ihnen sind keine Trennzeichen nötig (aber erlaubt).

### 8.5.3 Strukturelemente eines Java Programms

Diese allgemeine Struktur eines Java Programms gilt es mit „Leben zu füllen“, d. h. mit den verfügbaren Elementen der Programmierprache gemäss der Syntax auszuformulieren. Die wichtigsten Elemente eines Java Programms stehen daher im Fokus der folgenden Abschnitte.<sup>23</sup>

#### Kommentare

Java bietet gleich drei Möglichkeiten, den Quelltext zu kommentieren. So werden alle Zeichen von „//“ bis zum Ende der Zeile als Kommentar interpretiert, wobei kein Terminierungszeichen erforderlich ist, beispielsweise:

```
private int zahler;// wird mit 0 initialisiert
```

Hier wird eine Variablen Deklarationsanweisung in derselben Zeile kommentiert. Zwischen einer Einleitung durch `/*` und einer Terminierung durch `*/` kann sich ein ausführlicher Kommentar auch über mehrere Zeilen erstrecken, beispielsweise:

```
/*
Hier könnte ein Kommentar zur anschließend
definierten Klasse stehen.
*/
public class Beispiel {
...
}
```

<sup>23</sup> Vgl. Schader und Schmidt-Thieme 2000.

Ein mehrzeiliger Kommentar eignet sich u. a. auch dazu, einen Programmteil (vorübergehend) zu deaktivieren, ohne ihn löschen zu müssen. Vor der Definition bzw. Deklaration von Klasse, Interfaces, Methoden oder Variablen darf ein *Dokumentationskommentar* stehen, eingeleitet mit `/**` und beendet mit `*/`, beispielsweise:

```
/**  
Hier könnte ein Dokumentationskommentar zur anschließend  
definierten Klasse stehen.  
*/  
public class Beispiel {  
...  
}
```

Er kann mit dem SDK-Werkzeug javadoc in eine HTML-Datei extrahiert werden. Die systematische Dokumentation wird über Tags für Methoden-Parameter, Rückgabewerte etc. unterstützt. Nähere Informationen findet man in der jeweiligen Dokumentation zum Java-SDK.

## Bezeichner

Für Klassen, Methoden, Variablen, Parameter und sonstige Elemente eines Java-Programms benötigt man eindeutige Namen, für die folgende Regeln gelten:

- Die Länge eines Bezeichners ist nicht begrenzt.
- Das erste Zeichen muß ein Buchstabe, Unterstrich oder Dollar-Zeichen sein, danach dürfen außerdem auch Ziffern auftreten.
- Java-Programme werden intern im *Unicode*-Zeichensatz dargestellt. Daher erlaubt Java im Unterschied zu den meisten anderen Programmiersprachen im Bezeichnern auch Umlaute oder sonstige nationale Sonderzeichen, die als Buchstaben gelten.
- Die Groß-/Kleinschreibung ist signifikant.
- Die folgenden *reservierten Wörter* dürfen nicht als Bezeichner verwendet werden: `abstract`, `boolean`, `break`, `byte`, `case`, `catch`, `char`, `class`, `const`, `continue`, `default`, `do`, `double`, `else`, `extends`, `false`, `final`, `finally`, `float`, `for`, `goto1`, `if`, `implements`, `import`, `instanceof`, `int`, `interface`, `long`, `native`, `new`, `null`, `package`, `private`, `protected`, `public`, `return`, `short`, `static`, `super`, `switch`, `synchronized`, `this`, `throw`, `throws`, `transient`, `true`, `try`, `void`, `volatile`, `while`. Bei Verwendung eines der Java – Schlüsselwörter als Bezeichner, weist der Comiler auf diesen Regelverstoß hin.
- Bezeichner müssen innerhalb ihres Kontextes eindeutig sein.

## Variablen

Ein Programm zur Ansteuerung von Robotersystemen muss in der Regel zahlreiche Daten mehr oder weniger lange in einem begrenzten Arbeitsspeicher ablegen. Zum Speichern eines Wertes (beispielsweise einer Zahl) wird eine so genannte *Variable* verwendet, worunter man sich einen benannten und typisierten Speicherplatz vorstellen kann. Eine Variable erlaubt, über ihren Namen den lesenden oder schreibenden Zugriff auf den zugeordneten Platz im Arbeitsspeicher.

```
public class Variable {  
    public static void main(String[] args) {  
        int variable = 4711; //  
        schreibender Zugriff  
        System.out.println(variable); // lesernder  
        Zugriff  
    }  
}
```

Um die Details bei der Verwaltung der Variablen im Arbeitsspeicher muss sich der Entwickler nicht kümmern. Allerdings verlangt Java beim Umgang mit Variablen im Vergleich zu anderen Programmier- oder Skriptsprachen einige Sorgfalt, letztlich mit dem Ziel, Fehler zu vermeiden:

- Variablen müssen *explizit deklariert* werden. Durch den Deklarationszwang werden beispielsweise Programmfehler wegen falsch geschriebener Variablennamen verhindert.
- Java ist *strengh typisiert*: Jede Variable besitzt einen *Datentyp*, der die erlaubten Werte sowie die zulässigen Operationen festlegt.

In obigem Beispiel wird die Variable variable vom Typ int deklariert, der ganze Zahlen im Bereich von –2147483648 bis 147483647 als Werte aufnehmen kann. Die Variable variable erhält auch gleich den Initialisierungswert 4711. Auf diese oder andere Weise müssen jeder lokal (innerhalb einer Methode) definierten Variable Werte zugewiesen werden, bevor auf diese zum ersten Mal lesend zugegriffen werden kann.

In Java unterscheiden sich Variablen nicht nur hinsichtlich des Datentyps, sondern auch hinsichtlich der Zuordnung zu einer Methode, zu einem Objekt oder zu einer Klasse:

- *Lokale Variablen*: Sie werden innerhalb einer Methode deklariert. Ihre Gültigkeit beschränkt sich auf die Methode bzw. auf einen Block innerhalb der Methode.
- *Instanzvariablen* (Elementvariablen, Eigenschaften): Jedes Objekt (synonym: jede Instanz) einer Klasse verfügt über einen vollständigen Satz der Instanzvariablen (Eigenschaften) der Klasse. Auf Instanzvariablen kann in allen Methoden der Klasse zugegriffen werden. Wenn entsprechende Rechte eingeräumt wurden, ist dies auch in Methoden fremder Klassen möglich.

- *Klassenvariablen*: Diese Variablen beziehen sich auf eine gesamte Klasse, nicht auf einzelne Instanzen. Beispielsweise hält man oft in einer Klassenvariablen fest, wie viele Objekte einer Klasse bereits erzeugt worden sind. Auch für Klassenvariablen gilt, dass neben den klasseneigenen Methoden bei entsprechender Rechtevergabe auch Methoden fremder Klassen zugreifen dürfen.

Im Unterschied zu anderen Programmiersprachen (beispielsweise C++) ist es in Java nicht möglich, so genannte *globale* Variablen außerhalb von Klassen zu definieren. Neben den Datentypen können auch Klassen (aus dem Java-API oder selbst definiert) in einer Variablendeclaration verwendet werden, beispielsweise:

```
int i;
Sensor s1 = new Sensor();
```

Im zweiten Beispiel wird per new-Operator ein Sensor-Objekt erzeugt und dessen Adresse in die Referenzvariable s1 geschrieben.

*Lokale* Variablen, die innerhalb einer Methode existieren, können an beliebiger Stelle im Methoden-Quellcode deklariert werden, aus nahe liegenden Gründen jedoch vor ihrer ersten Verwendung. Dabei hat sich bewährt, Variablennamen mit einem Kleinbuchstaben beginnen zu lassen, um sie im Quelltext gut von den Bezeichnern für Klassen oder Konstanten unterscheiden zu können. Neu deklarierte Variablen kann man optional auch gleich initialisieren, also auf einen gewünschten Wert setzen.

Weil *lokale* Variablen nicht automatisch initialisiert werden, muss man ihnen unbedingt vor dem ersten lesenden Zugriff einen Wert zuweisen. Auch im Umgang des Compilers mit uninitializeden lokalen Variablen zeigt sich das Bemühen der Java-Designer um robuste Programme. Während C++ -Compiler in der Regel nur warnen, produzieren Java-Compiler eine Fehlermeldung und erstellen keinen Bytecode. Um den Wert einer Variablen im weiteren Programmablauf zu verändern, verwendet man eine einfache *Wertzuweisung*, die zu den einfachsten und am häufigsten benötigten Anweisungen gehört:

```
Summe = v1 + v2;
```

Der Rumpf einer Methodendeklaration besteht aus einem Block mit beliebig vielen Anweisungen, abgegrenzt durch geschweifte Klammern. Innerhalb des Methodenrumpfes können weitere Anweisungsblöcke gebildet werden, wiederum durch geschweifte Klammern begrenzt. Man spricht hier auch von einer *Block-* bzw. *Verbundanweisung*, und diese kann überall stehen, wo eine einzelne Anweisung erlaubt ist. Unter den Anweisungen eines Blockes dürfen sich selbstverständlich auch wiederum Blockanweisungen befinden. Einfacher ausgedrückt: Blöcke dürfen geschachtelt werden. Oft treten Blöcke zusammen mit Bedingungen oder Wiederholungen auf.

Anweisungsblöcke haben einen wichtigen Effekt auf die Gültigkeit der darin deklarierten Variablen: eine lokale Variable existiert von der deklarierenden Zeile bis zur schließen-

den Klammer des lokalsten Blockes. Nur in diesem *Gültigkeitsbereich* (engl. scope) kann sie über ihren Namen angesprochen werden. Beim Verlassen des Blockes wird der belegte Speicher frei gegeben, so dass die im folgenden Beispiel auskommentierte Zeile folgende Fehlermeldung produzieren würde: Wert2 cannot resolved.

```
public class Blockanweisung {  
    public static void main(String[] args) {  
        int wert1 = 1;  
        System.out.println(„Wert1 = “ + wert1);  
        if (wert1 == 1) {  
            int wert2 = 2;  
            System.out.println(„Wert2 = “ + wert2);  
        }  
        System.out.println(„Wert2 = “ + wert2);  
    }  
}
```

Bei hierarchisch geschachtelten Blöcken ist es in Java *nicht* erlaubt, auf mehreren Stufen Variablen mit identischem Namen zu deklarieren. Mit dem Verzicht auf diese kaum sinnvolle C++ – Option haben die Java-Designer eine Möglichkeit beseitigt, schwer erkennbare Programmierfehler zu begehen. Bei der übersichtlichen Gestaltung von Java-Programmen empfiehlt sich das Einrücken von Anweisungsblöcken.

## Konstanten

In der Regel können auch konstante Werte (beispielsweise für die Belegung von Sensorwerten) in einer Variablen abgelegt und im Quelltext über ihren Variablennamen angesprochen werden, denn:

- Bei einer späteren Änderung des Wertes ist nur die Quellcode-Zeile mit der Variablen-deklaration und -initialisierung betroffen.
- Der Quellcode ist leichter zu lesen.

```
public class Konstante {  
    public static void main(String[] args) {  
        final double MWST = 1.16;  
        double netto = 100.0, brutto;  
        brutto = netto * MWST;  
        System.out.println(„Brutto: “ + brutto);  
    }  
}
```

Variablen, die nach ihrer Initialisierung im gesamten Programmverlauf garantiert denselben Wert besitzen sollen, deklariert man als `final`.

Die Unterschiede im Vergleich zur gewöhnlichen Variablen-deklaration:

- Am Anfang steht der Typqualifizierer `final`.
- Eine finalisierte Variable muss natürlich initialisiert werden.

Man schreibt üblicherweise die Namen von finalisierten Variablen (Konstanten) komplett in Großbuchstaben, um sie im Quelltext gut von anderen Sprachelementen unterscheiden zu können.

## Primitiva

Als *Primitiva* bezeichnet man in Java die auch in älteren Programmiersprachen bekannten *Standardtypen* zur Aufnahme einzelner Werte (Zeichen, Zahlen oder Wahrheitswerte). Es stehen die folgenden Primitiva zur Verfügung (Tab. 8.5):

Ein Vorteil von Java besteht darin, dass die Wertebereiche der elementaren Datentypen auf allen Plattformen identisch sind, worauf man sich bei anderen Programmiersprachen (beispielsweise C/C++) nicht immer verlassen kann. Im Vergleich zu C/C++ fällt noch auf, dass der Einfachheit halber auf *vorzeichenfreie* Datentypen verzichtet wurde. Neben den primitiven Datentypen existieren noch sogenannte *Referenzdatentypen*, über die sich Objekte aus einer bestimmten Klasse ansprechen lassen. Man kann jede Klasse (aus dem Java-API übernommen oder selbst definiert) als Datentyp verwenden, also Referenzvariablen dieses Typs deklarieren. Auch Zeichenfolgen und Felder (Zusammenfassungen von mehreren gleichartigen Einzelvariablen) sind in Java daher stets Objekte.

## Operatoren

Bereits bei einer Variablen oder einem Methodenaufruf handelt es sich um einen Ausdruck, besitzt dieser doch einen Datentyp und einen Wert. So handelt es sich beispielsweise bei `1.5` um einen Ausdruck mit dem Typ `double` und dem Wert `1.5`. Besteht ein Ausdruck allerdings aus einem Methodenaufruf mit dem Pseudo-Rückgabetyp `void`, dann liegt kein Wert vor. Mit Hilfe diverser Operatoren entstehen komplexe Ausdrücke, wobei Typ und Wert von den Argumenten und den Operatoren abhängen. So entsteht aus `2.0 + 1.5` als Resultat der `double`-Wert `3.5`.

In der Regel beschränken sich Java-Operatoren darauf, aus ihren Argumenten (Operanden) einen Wert zu ermitteln und für die weitere Verarbeitung zur Verfügung zu stellen. Einige Operatoren haben jedoch zusätzlich einen Nebeneffekt auf eine als Argument fungierende Variable. Dann sind in der betreffenden Operandenrolle auch nur Ausdrücke erlaubt, die für eine Variable stehen. Die meisten Java-Operatoren verarbeiten *zwei* Operanden (Argumente) und heißen daher *zweistellig* bzw. *binär*. So erwartet der Additionsoperator „`+`“ zwei numerische Argumente. Manche Operatoren begnügen sich mit einem Argument und heißen daher *einstellig* bzw. *unär*. So erwartet die Vorzeichenum-

**Tab. 8.5** Primitiva

Typ	Beschreibung	Werte	Speicherbedarf (in Bit)
Byte	Dieser Variabtentyp speichert ganze Zahlen	- 128.. + 127	8
Short	Dieser Variabtentyp speichert ganze Zahlen	- 32768 ... + 32767	16
Int	Dieser Variabtentyp speichert ganze Zahlen	- 2147483648 ... + 2147483647	32
Long	Dieser Variabtentyp speichert ganze Zahlen	- 9223372036854775808 ... + 9223372036854775807	64
Float	Dieser Variabtentyp kann Fließkommazahlen mit einer Genauigkeit von mindestens 7 Dezimalstellen speichern	Minimum: $-3.4028235 \cdot 10^{38}$ Maximum: $+3.4028235 \cdot 10^{38}$ Kleinster Betrag: $1.4 \cdot 10^{-45}$	32
Double	Dieser Variabtentyp kann Fließkommazahlen mit einer Genauigkeit von mindestens 15 Dezimalstellen speichern	Minimum: $-1.7976931348623157 \cdot 10^{308}$ Maximum: $+1.7976931348623157 \cdot 10^{308}$ Kleinster Betrag: $4.9 \cdot 10^{-324}$	64
Char	Variablen dieses Typs dienen zum Speichern eines Unicode-Zeichens. Im Speicher landet aber nicht die Gestalt eines Zeichens, sondern seine Nummer im Zeichensatz. Daher zählt char zu den integralen (ganzzahligen) Datentypen. Soll ein Zeichen als Wert zugewiesen werden, so ist es mit <i>einfachen Anführungszeichen einzurahmen</i>	Unicode-Zeichen	16
Boolean	Variablen dieses Typs können Wahrheitswerte annehmen	True, false	1

kehr, bezeichnet mit dem „-“-Symbol, *ein* numerisches Argument. Weil als Argumente einer Operation auch *Ausdrücke* von passendem Ergebnistyp erlaubt sind, können beliebig komplexe Ausdrücke aufgebaut werden.

Da die *arithmetischen Operatoren* für die vertrauten Grundrechenarten der Schulmathematik zuständig sind, müssen ihre Operanden (Argumente) einen Ganzzahl- oder Gleitkomma-Typ haben (byte, short, int, long, char, float oder double). Die resultierenden Ausdrücke haben folglich ebenfalls einen numerischen Ergebnistyp und werden daher gelegentlich als numerische Ausdrücke bezeichnet.

```

public class ArithmetischeOperatoren {
    public static void main(String[] args) {
        int i = -5;
        System.out.println("Vorzeichenenumkehr: " + - i);
        i = 2; int j = 3;
        System.out.println("Addition: " + i + j);
        double a = 2.6; double b = 1.2;
        System.out.println("Subtraktion: " + (a - b));
        i = 4; j = 5;
        System.out.println("Multiplikation: " + i*j);
        System.out.println("Division: " + (8.0/5));
        System.out.println("Division: " + (8/5));
        i = 9; j = 5;
        System.out.println("Modulo (Restwert): " + i%j);
        i = 4;
        a = 0.2;
        System.out.println("Präinkrement: " + (++i) + "\n" +
        "Dekrement: " + (- a));
        i = 4;
        System.out.println("Postinkrement: " + (i++) + "\n" +
        "Postdekrement " + (i--));
    }
}

```

Bei den Prä- und Postinkrement- bzw. Dekrementoperatoren ist zu beachten, dass sie zwei Effekte haben:

- Das Argument wird ausgelesen, um den Wert des Ausdrucks zu ermitteln.
- Der Wert des Argumentes wird verändert.

Wegen dieses „Nebeneffektes“ sind Prä- und Postinkrement- bzw. Dekrementausdrücke im Unterschied zu sonstigen arithmetischen Ausdrücken bereits vollständige Anweisungen. Mit den arithmetischen Operatoren lassen sich nur elementare mathematische Probleme lösen. Darüber hinaus stellt Java eine große Zahl mathematischer Standardfunktionen (beispielsweise Potenzfunktion, Logarithmus, Wurzel, trigonometrische und hyperbolische Funktionen) über Methoden der Klasse Math im API-Paket java.lang zur Verfügung.

```

public class Potenzrechnung {
    public static void main(String[] args) {
        System.out.println(5 * Math.pow(2, 3));
    }
}

```

**Tab. 8.6** Vergleichsoperatoren

Operator	Bedeutung
==	Gleichheit
!=	Ungleichheit
>	Größer
<	Kleiner
<=	Kleiner oder gleich
>=	Größer oder gleich

Alle Methoden der Klasse Math sind static, können also verwendet werden, ohne ein Objekt aus der Klasse Math zu erzeugen. Isofern stellt ein Methodenaufruf durchaus einen vollständigen Ausdruck dar und kann daher auch als Argument komplexerer Ausdrücke verwendet werden, sofern die Methode einen passenden Rückgabewert liefert. Durch Verwendung von *Vergleichsoperatoren* und *logischen Operatoren* entstehen Ausdrücke mit dem Ergebnistyp boolean, die als logische Ausdrücke bezeichnet werden. Sie können die booleschen Werte true (wahr) und false (falsch) annehmen und eignen sich dazu, *Bedingungen* zu formulieren.

```
if (arg > 0)
    System.out.println(Math.log(arg));
```

Ein *Vergleich* ist ein besonders einfacher aufgebauter logischer Ausdruck, bestehend aus zwei Ausdrücken und einem Vergleichsoperator (Tab. 8.6).

Dabei gilt es zu beachten, dass der Identitätsoperator durch zwei „=“-Zeichen ausgedrückt wird. Einer der häufigsten Java-Programmierfehler besteht darin, beim Identitätsoperator nur ein Gleichheitszeichen zu schreiben. Dabei kommt es in der Regel nicht zu einer Compiler-Fehlermeldung, sondern zu einem unerwarteten Verhalten des Programms. Durch Anwendung der *logischen Operatoren* auf bereits vorhandene logische Ausdrücke kann man neue, komplexere logische Ausdrücke erstellen (Tab. 8.7).

Die Wirkungsweise der logischen Operatoren wird üblicherweise in Wahrheitstafeln beschrieben.

```
public class LogischeOperatoren {
    public static void main(String[] args) {
        boolean v1, v2, v3, v4;
        int i = 3;
        char c = ',n';
        v1 = (2 > 3) && (2 == 2) ^ (1 == 1);
        System.out.println(v1);
        v2 = ((2 > 3) && (2 == 2)) ^ (1 == 1);
        System.out.println(v2);
        v3 = i > 0 && i * 4 == 012;
        System.out.println(v3);
        v4 = !(i > 0 || c == ,j);
        System.out.println(v4);
    }
}
```

**Tab. 8.7** Logische Operatoren

Operator	Bedeutung	Beschreibung	Beispiel
NOT (!)	Negation	Umkehrung des Wahrheitswertes	$!0=1$ $!1=0$
AND ( $\wedge$ )	Konjunktion	Beide Bedingungen müssen erfüllt sein, damit das Ergebnis wahr wird	$0 \wedge 0 = 0$ $0 \wedge 1 = 0$ $1 \wedge 0 = 0$ $1 \wedge 1 = 1$
OR (v)	Disjunktion	Mindestens eine der Bedingungen muss erfüllt sein, damit das Ergebnis wahr wird	$0 \vee 0 = 0$ $0 \vee 1 = 1$ $1 \vee 0 = 1$ $1 \vee 1 = 1$
XOR ( $\oplus$ )	Antivalenz (Entweder-oder)	Genau eine der Bedingung muss erfüllt sein, damit das Egebnis wahr wird	$0 \oplus 0 = 0$ $0 \oplus 1 = 1$ $1 \oplus 0 = 1$ $1 \oplus 1 = 0$
NOR ( $\downarrow$ )	Nihilition (Weder-noch)	Keine Bedingung darf erfüllt sein, damit das Ergebnis wahr wird	$0 \downarrow 0 = 1$ $0 \downarrow 1 = 0$ $1 \downarrow 0 = 0$ $1 \downarrow 1 = 0$
NAND ( $\uparrow$ )	Unverträglichkeit	Beide Bedingungen dürfen nicht gleichzeitig erfüllt sein, damit das Ergebnis wahr wird	$0 \uparrow 0 = 1$ $0 \uparrow 1 = 1$ $1 \uparrow 0 = 1$ $1 \uparrow 1 = 0$
EQ ( $\equiv$ )	Äquivalenz	Beide Bedingungen müssen gleich sein, damit das Ergebnis wahr wird	$0 \equiv 0 = 1$ $0 \equiv 1 = 0$ $1 \equiv 0 = 0$ $1 \equiv 1 = 1$
IMPL ( $\rightarrow$ )	Implikation (Wenn-Dann)	Aus einer Bedingung kann ein Ergebnis (wahr oder falsch) gefolgert werden	$0 \rightarrow 0 = 1$ $0 \rightarrow 1 = 1$ $1 \rightarrow 0 = 0$ $0 \rightarrow 0 = 1$

Gerade im Bereich der Robotik kann die Notwendigkeit entstehen, mit Hilfe der Operatoren zur bitweisen Manipulation Variableninhalte zu verändern. Statt einer systematischen Darstellung der verschiedenen Operatoren. Das folgende Programm liefert die Unicode-Kodierung zu einem vom Benutzer als Argument beigegebenes Zeichen.

```
public class Unicode {  
    public static void main(String[] args) {  
        char cbit = args[0].charAt(0);  
        System.out.print("Zeichen: ");  
        System.out.print("Unicode: ");  
        for(int i = 15; i >= 0; i--) {  
            if ((1 << i & cbit) != 0)  
                System.out.print("1");  
            else  
                System.out.print("0");  
        }  
        System.out.println();  
    }  
}
```

Der *Links-Shift-Operator*  $<<$  in:  $1 << i$  verschiebt die Bits in der binären Repräsentation der Ganzzahl 1 um  $i$  Stellen nach links. Von den 32 Bit, die ein int-Wert insgesamt belegt, interessieren im Augenblick nur die rechten 16. Bei der 1 erhalten wir: 0000000000000001 Im 10. Schleifendurchgang ( $i=6$ ) geht dieses Muster beispielsweise über in: 000000001000000. Nach dem Links-Shift- kommt dann der sogenannte bitweise *UND-Operator* zum Einsatz:  $1 << i \& cbit$ . Das Operatorzeichen  $\&$  wird leider in doppelter Bedeutung verwendet: Wenn beide Argumente vom Typ boolean sind, wird  $\&$  als *logischer Operator* interpretiert. Sind jedoch beide Argumente von integralem Typ, was auch für den Typ char zutrifft, dann wird  $\&$  als UND-Operator für Bits aufgefasst. Er erzeugt dann ein Bitmuster, das an der Stelle  $j$  eine 1 enthält, wenn *beide* Argumentmuster an dieser Stelle den Wert 1 haben, anderenfalls eine 0. Bei  $cbit = 'x'$  ist das Unicode-Bitmuster 000000001111000 zu bearbeiten, und  $1 << i \& cbit$  liefert beispielsweise bei  $i=6$  das Muster: 000000001000000. Das von  $1 << i \& cbit$  erzeugte Bitmuster hat den Typ int und kann daher mit der 0 verglichen werden:  $(1 << i \& cbit) != 0$ . Dieser logische Ausdruck wird im  $i$ -ten Schleifendurchgang genau dann wahr, wenn das korrespondierende Bit in der Binärdarstellung des untersuchten Zeichens den Wert 1 hat (Abb. 8.22).

Beim Auswerten des Ausdrucks 2.0/7 trifft der Divisions-Operator auf ein double- und auf ein int-Argument, so dass die Gleitkomma-Arithmetik zur Anwendung kommt. Dabei wird für das int-Argument eine *automatische (implizite) Wandlung* in den Datentyp double vorgenommen. Java nimmt bei Bedarf für primitive Datentypen *Konvertierungen* zu einem allgemeineren Typ automatisch vor. Diese Tatsache gilt es unbedingt zu beachten, da eine Vernachlässigung dieses programmiertechnischen Sachverhalts bei Robotersystemen eklatante Auswirkungen zeigen kann. Gelegentlich zwingen Gründe den Entwickler, über den *Casting-Operator* eine *explizite* Typumwandlung vorzunehmen. Im folgenden Programm wird mit (int) 'm' eine Integer-Darstellung des Bitmusters zum kleinen „m“ erzwungen.

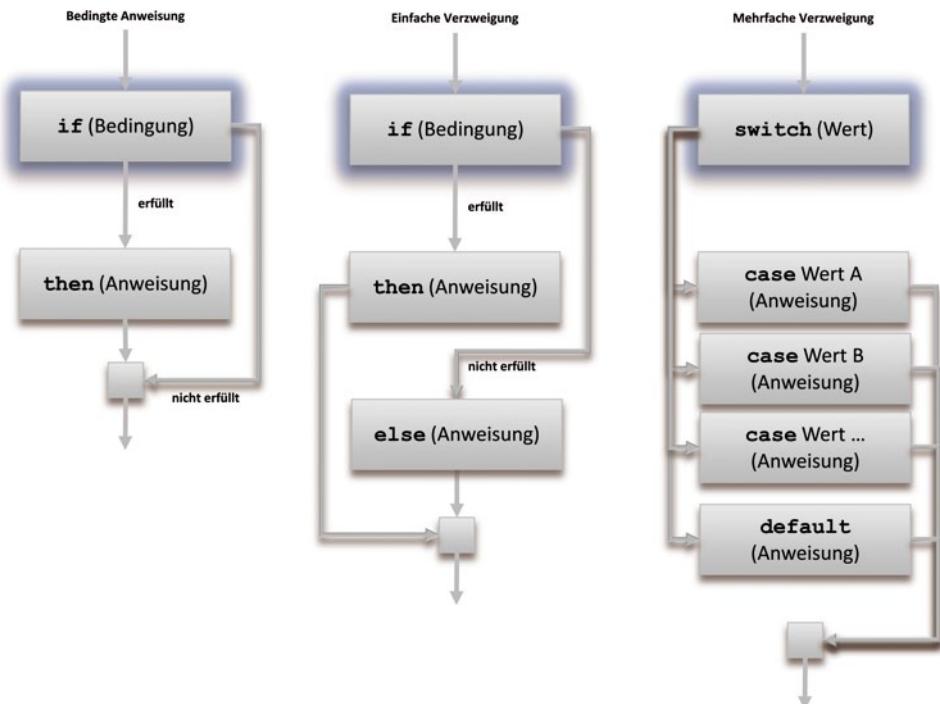


Abb. 8.22 Verzweigungen

```
public class CastingProblem {
    public static void main(String[] args) {
        double a = 3.14159;
        System.out.println((int) ,m');
        System.out.println((int) a);
        System.out.println((int) 1.41268e50);
    }
}
```

In der zweiten Ausgabeanweisung des Beispielprogramms wird per Casting-Operation der ganzzahlige Anteil eines double-Wertes ermittelt. Die dritte Ausgabe zeigt aber auch, dass bei einer explizit angeforderten einschränkenden Konvertierung schwere Programmierfehler möglich sind, wenn die Wertebereiche der beteiligten Variablen bzw. Datentypen nicht beachtet werden.

In Java gelten für *Wertzuweisungen* in Form eines Ausdruckes durch den Operator „=“ folgende Regeln:

- Auf der linken Seite muss eine Variable stehen.
- Auf der rechten Seite muss ein Ausdruck stehen, der einen zum VariablenTyp kompatiblen Wert liefert.
- Der zugewiesene Wert stellt auch den Ergebniswert des Ausdrucks dar.

Analog zum Verhalten des Inkrement- bzw. Dekrementoperators sind auch beim Zuweisungsoperator zwei Effekte zu unterscheiden:

- Die als linkes Argument fungierende Variable wird verändert.
- Es wird ein Wert für den Ausdruck produziert.

In folgendem Beispielprgramm fungiert unter anderem ein Zuweisungsausdruck als Parameter für einen `println()`-Methodenaufruf. Dabei entsteht beim Auswerten des Ausdrucks der an `println()` zu übergebende Wert und die Variable `integervar` wird entsprechend modifiziert.

```
public class Wertzuweisung {  
    public static void main(String[] args) {  
        int integervar = 13;  
        System.out.println(integervar = 4711);  
        int i = 2, j = 4;  
        i = j = j * i;  
        System.out.println(i + " " + j);  
    }  
}
```

Weil bei mehrfachem Auftreten des Zuweisungsoperators eine Abarbeitung von rechts nach links erfolgt, passiert im letzten Beispiel folgendes: Der Ausdruck `j * i` liefert das Ergebnis 8, das der Variablen `j` zugewiesen wird und gleichzeitig den Wert des Ausdrucks `j = j * i` darstellt. In der zweiten Zuweisung, ausgehend von rechts nach links, wird der Wert des Ausdrucks `j = j * i` an die Variable `i` übergeben. Für die häufig benötigten Zuweisungen nach dem Muster `j = j * i`, d. h. eine Variable erhält einen neuen Wert, an dessen Konstruktion sie selbst mitwirkt, bietet Java spezielle Zuweisungsoperatoren. Es wird an dieser Stelle allerdings empfohlen, aus Gründen der Übersichtlichkeit und Transparenz auf die Verwendung dieser sogenannten *Aktualisierungsoperatoren* zu verzichten. Mit der gleichen Argumentation wird der Verzicht auf den sogenannten *Konditionaloperator* begründet, mit dessen Hilfe sich die bedingungsabhängige Entscheidung zwischen zwei Ausdrücken sehr kompakt formulieren lässt. Auch die in konventionellen Programmsequenzen vorteilhafte Besonderheit des Konditionaloperators, die darin besteht, dass er drei Argumente verarbeitet, wiegt den Nachteil der Intransparenz nicht auf. Zum Abschluss dieses Abschnittes zu den einzelnen Operatoren soll die Problematik in Bezug auf den Einsatz von Ausdrücken mit mehreren Operatoren und das damit verbun-

**Tab. 8.8** Operatoren

Prio.	Operator	Bedeutung	Richtung
1	() []. ausdruck ++ ausdruck-	Methodenaufruf	L→R
2	! ~ -unär + unär + + ausdruck -ausdruck	Negation, Prä- oder Postinkrement, Voreichenenumkehr	L→R
3	New (type)	Typumwandlung	L→R
4	* / %	Punktrechnung, Modulo	L→R
5	+ -	Strichrechnung, Stringverkettung	L→R
6	<< >> >>>	Links-, Rechts-Shift	L→R
7	< <= > >= instanceof	Vergleichsoperatoren	L→R
8	== !=	Gleichheit Ungleichheit	L→R
9	&	Bitweises UND	L→R
10	^	Exklusives logisches ODER	L→R
11		Bitweises ODER	L→R
12	&&	Logisches UND	L→R
13		Logisches ODER	L→R
14	? :	Konditionaloperator	R→L
15	= += -= *= /= %= ^=&= ==<<=>>=>>>=	Wertzuweisungen	R→L

dene Problem der *Auswertungsreihenfolge* behandelt werden. Ein Java-Compiler behandelt solche komplexen Ausdrücke unter Berücksichtigung folgender Kriterien:

- *Priorität*: Zunächst entscheidet die Priorität der Operatoren darüber, in welcher Reihenfolge die Auswertung vorgenommen wird. Beispielsweise hält sich Java bei numerischen Ausdrücken an die mathematische Regel „Punkt- vor Strichrechnung“.
- *Auswertungsrichtung*: Bei gleicher Priorität wird ein zweites Attribut der beteiligten Operatoren relevant: ihre Auswertungsrichtung. In der Regel werden gleichrangige Operatoren von Links nach Rechts ausgewertet, doch existieren auch Ausnahmen. Um jede Unklarheit zu vermeiden, ist in Java für Operatoren gleicher Priorität stets auch dieselbe Auswertungsrichtung festgelegt.
- *Klammern*: Wenn die aus obigen Regeln resultierende Auswertungsfolge nicht zum gewünschten Ergebnis führt, kann mit runden Klammern steuernd eingegriffen und eine Auswertungsreihenfolge erzwungen werden: Die eingeklammerten Teilausdrücke werden „von innen nach außen“ ausgewertet.

In der folgenden Tabelle sind die bisher behandelten Operatoren in absteigender Priorität (von oben nach unten) mit Auswertungsrichtung aufgelistet. Gruppen von Java-Operatoren mit gleicher Priorität sind durch fette horizontale Linien begrenzt. Sie verfügen dann stets auch über dieselbe Auswertungsrichtung (Tab. 8.8).

## Kontrollstrukturen

Ein Algorithmus besteht in der Regel aus einfachen *Operationen*, wie beispielsweise arithmetischen Berechnungen, Wertzuweisungen an Variable oder Ein- und Ausgaben von Werten. In welcher Reihenfolge dabei diese Operationen ausgeführt werden, wird durch die Kontrollstrukturen des Algorithmus bestimmt.

Die einfachste Kontrollstruktur ist die *Sequenz*, bei der eine Folge von Aktionen in der Reihenfolge abgearbeitet wird, in der sie in der Beschreibung des Algorithmus aufgeführt sind. Bei den Aktionen kann es sich entweder um einfache Operationen oder wiederum um strukturierte Operationen handeln. Aktionen in Algorithmen werden oftmals in Abhängigkeit von einer bestimmten *Bedingung* ausgeführt.<sup>24</sup> Man spricht in solch einem Fall davon, dass die Ausführung des Algorithmus zu einer von mehreren möglichen Aktionen verzweigt. Entscheidend dabei ist, daß nicht statisch (also bereits beim Codieren des Algorithmenkörpers) festgelegt wird, zu welcher Aktion verzweigt wird: die Aktion ergibt erst dynamisch (also bei der Ausführung des Algorithmus) in Abhängigkeit der Werte, die der Algorithmus aktuell verarbeitet.

Die einfachste Art der *Verzweigung* ist die Alternative, bei der abhängig von einer aussagenlogischen Bedingung entweder ein *Ja-Zweig (true-Zweig)* oder ein *Nein-Zweig (false-Zweig)* ausgeführt wird. Jeder dieser Zweige besteht entweder aus einer einfachen Operation oder einer strukturierten Operation. Mit der folgenden Syntax wird dafür gesorgt, dass eine Anweisung nur dann ausgeführt wird, wenn ein logischer Ausdruck den Wert true annimmt:

In dem folgenden Beispiel wird eine Meldung ausgegeben, wenn die Variable anz den Wert 0 besitzt:

```
if (anz == 0)
    System.out.println(„Die Anzahl muß > 0 sein!“);
```

Der Zeilenumbruch zwischen dem logischen Ausdruck und der (Unter-) Anweisung dient nur der Übersichtlichkeit und ist für den Compiler irrelevant. Soll auch etwas passieren, wenn der logische Ausdruck den Wert false besitzt, erweitert man die if-Anweisung um eine else-Klausel.

Zur Beschreibung der *if-else* - Anweisung wird an Stelle eines Syntaxdiagramms eine alternative Darstellungsform gewählt, die sich am typischen Java-Quellcode-Layout orientiert:

```
if (logischer Ausdruck)
    Anweisung 1
else
    Anweisung 2
```

---

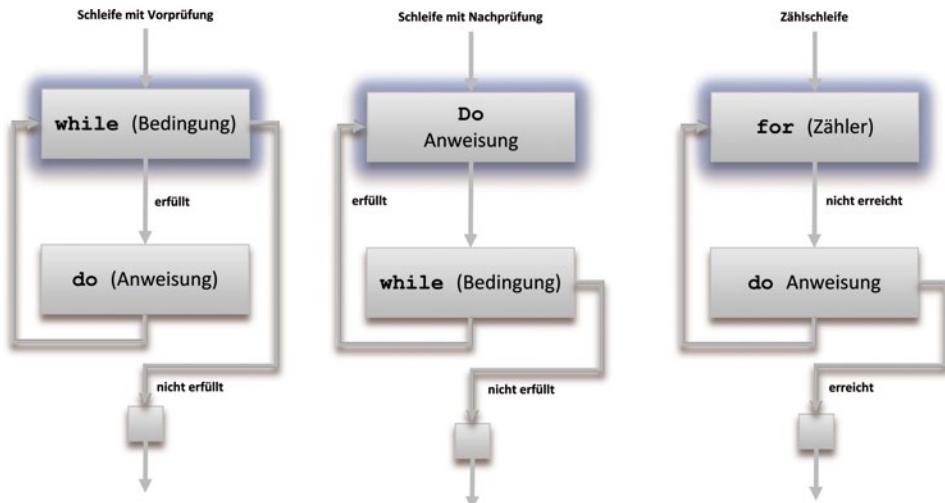
<sup>24</sup> Vgl. Sedgewick 2003.

In manchen Anwendungsfällen gibt es nicht nur zwei, sondern drei oder mehr mögliche Aktionen, wobei der Wert einer Variablen oder eines arithmetischen Ausdrucks bestimmt, welche davon ausgeführt werden soll. Prinzipiell könnte dies durch eine Sequenz oder eine Schachtelung von Alternativen realisiert werden, was aber zu recht unübersichtlichen Strukturen führen kann.

```
if (logischer Ausdruck 1)
    Anweisung 1
else if (logischer Ausdruck 2)
    Anweisung 2
    .
    .
else if (logischer Ausdruck k)
    Anweisung k
else
    Default-Anweisung
```

Enthält die letzte if-Anweisung eine else-Klausel, so wird die zugehörige Anweisung ausgeführt, wenn alle logischen Ausdrücke den Wert false haben. Die Wahl der Bezeichnung *Default-Anweisung* in der Syntaxdarstellung orientierte sich an der im Anschluss vorzustellenden switch-Anweisung. Besser ist die Verwendung einer *Fallauswahl* (Mehrfachverzweigung), bei der alle Aktionen gleichberechtigt nebeneinander dargestellt werden. Wenn eine Fallunterscheidung mit mehr als zwei Alternativen in Abhängigkeit von einem ganzzahligen Ausdruck vom Typ byte, short, char oder int (nicht long!) vorgenommen werden soll, dann ist die Mehrfachauswahl per switch-Anweisung weitaus handlicher als eine verschachtelte if-else-Konstruktion. Stimmt der Wert des Ausdrucks mit der ganzzahligen Konstanten (finalisierte Variable oder Literal) zu einer case-Marke überein, dann wird die zugehörige Anweisung ausgeführt, ansonsten (falls vorhanden) die default-Anweisung. Nach dem Ausführen einer „angesprungenen“ Anweisung wird die switch-Konstruktion jedoch nur dann verlassen, wenn der Fall mit einer break-Anweisung abgeschlossen wird. Ansonsten werden auch noch die Anweisungen der nächsten Fälle (ggf. inkl. default) ausgeführt, bis das Durchreichen nach unten entweder durch eine break-Anweisung gestoppt wird, oder die switch-Anweisung endet. Mit dem etwas gewöhnungsbedürftigen Durchreichen-Prinzip kann man für geeignet angeordnete Fälle sehr elegant kumulative Effekte kodieren, aber auch ärgerliche Programmierfehler durch vergessene break-Anweisungen produzieren. Soll für mehrere Auswahlwerte dieselbe Anweisung ausgeführt werden, setzt man die zugehörigen case-Marken unmittelbar hintereinander und lässt die Anweisung auf die letzte Marke folgen. Leider gibt es keine Möglichkeit, eine Serie von Fällen durch Angabe der Randwerte festzulegen (Abb. 8.23).

Im folgenden Beispielprogramm wird die Persönlichkeit des Robotersystems mit Hilfe seiner Farb- und Zahlpräferenzen analysiert. Während bei einer Vorliebe für Rot oder Schwarz die Diagnose sofort fest steht, wird bei den restlichen Farben auch die Lieblingszahl berücksichtigt:

**Abb. 8.23** Schleifen

```
public class Robotertyp {
    public static void main(String[] args) {
        char farbe = args[0].charAt(0);
        int zahl = Character.getNumericValue(args[1].charAt(0));
        switch (farbe) {
            case 'r': System.out.println("emotionaler Typ."); break;
            case 'g':
            case 'b': {
                System.out.println("sachlicher Typ");
                if (zahl%2 == 0)
                    System.out.println("geradlinigen Charakter");
                else
                    System.out.println("schlechten Charakter");
                }
                break;
            case 's': System.out.println("unzuverlässig");
                break;
            default: System.out.println("mangelnde Disziplin.");
        }
    }
}
```

Das obige Beispielprogramm demonstriert nicht nur die `switch`-Anweisung, sondern auch die Verwendung von Kommandozeilen-Optionen über den `String[]`-Parameter der `main()`-Funktion. Benutzer des Programms sollen beim Start die Farbe des Robotersystems und seine Lieblingszahl (aus dem Wertebereich von 0 bis 9) angeben, wobei die Farb-

gebung folgendermaßen durch einen Buchstaben zu kodieren ist: r für Rot, g für grün, b für Blau und s für Schwarz. Wenn das Robotersystem demnach aus blauem Blech besteht und die Zahl 7 bevorzugt, muss das Programm also folgendermaßen starten:

```
java Robotertyp b 7
```

Man benötigt jeweils nur eine Java-Zeile, um die Kommandozeilen-Optionen in eine char- bzw. int-Variable zu übertragen. Das erste Element des Zeichenketten-Feldes args (Nr. 0) wird (als Objekt der Klasse String) aufgefordert, die Methode charAt() auszuführen, welche das erste Zeichen abliefer. Im Programm wird dieses Zeichen der char-Variablen farbe zugewiesen. Analog wird auch zum zweiten Element des Zeichenketten-Feldes args (Nr. 1) das erste Zeichen ermittelt. Dieses wird mit Hilfe der Klassenmethode getNumericValue() aus der Klasse Character in eine Zahl vom Datentyp int gewandelt und anschließend der Variablen zahl zugewiesen.

Beim Ablauf eines Algorithmus, der nur aus Sequenzen und Verzweigungen besteht, wird jede Aktion nur höchstens einmal ausgeführt. Viele Verfahren verlangen jedoch, dass bestimmte Aktionen *mehrfach* hintereinander durchlaufen werden, wobei oftmals die Anzahl der Durchläufe nicht von vornherein festliegt, sondern sich erst während der Ausführung ergibt. Hierzu benötigt man *Schleifen* (engl. loop). Bei leicht vereinfachender Betrachtung kann man hinsichtlich der Schleifensteuerung unterscheiden:

- *Zählergesteuerte Schleife* (`for`): Die Anzahl der Wiederholungen steht typischerweise schon vor Schleifenbeginn fest. Zur Ablaufsteuerung wird am Anfang eine Zählvariable initialisiert und dann bei jedem Durchlauf aktualisiert (beispielsweise inkrementiert). Die zur Schleife gehörige Anweisung wird ausgeführt, solange die Zählvariable eine festgelegte Grenze nicht überschreitet.
- *Bedingungsabhängige Schleife* (`while`, `do`): Bei jedem Schleifendurchgang wird eine Bedingung überprüft, und das Ergebnis entscheidet über das weitere Vorgehen:
  - `true`: die zur Schleife gehörige Anweisung ein weiteres mal ausführen
  - `false`: Schleife beenden

Bei der so genannten *kopfgesteuerten Schleife* wird bei der Ausführung als Erstes geprüft, ob die Schleifenbedingung wahr ist. Ist das der Fall, so werden die Aktionen des Schleifenkörpers einmal ausgeführt. Anschließend wird die Schleifenbedingung erneut geprüft. Trifft sie immer noch zu, so wird der Körper ein zweites Mal durchlaufen. Dies setzt sich fort, bis die Bedingung falsch ist. Dann läuft der Algorithmus hinter der Schleife weiter. Natürlich sollten die Aktionen des Schleifenkörpers sicherstellen, dass die Bedingung irgendwann einmal falsch wird, denn sonst käme es zu einer sogenannten Endlosschleife und das Verfahren würde nicht terminieren. Entscheidend (und auch namensgebend) bei einer kopfgesteuerten Schleife ist, dass immer *erst die Bedingung* geprüft und dann der Körper ausgeführt wird. Es kann also durchaus vorkommen, dass der Körper keinmal ausgeführt wird, dann nämlich, wenn die Bedingung schon beim Erreichen der Schleife nicht erfüllt ist.

Ein Spezialfall einer kopfgesteuerten Schleife ist die *Zählschleife*, bei der ein Schleifenindex einen bestimmten Wertebereich durchläuft. Der Schleifenindex wird dabei durch

eine so genannte Laufvariable realisiert, die mit einem bestimmten Wert initialisiert und nach jedem Durchlauf durch den Schleifenkörper um einen bestimmten Betrag erhöht oder erniedrigt wird, bis eine festgelegte Grenze erreicht ist.

Die Anweisung einer `for`-Schleife wird ausgeführt, solange eine Bedingung erfüllt ist, die normalerweise auf eine ganzzahlige Indexvariable Bezug nimmt. Auf das Schlüsselwort `for` folgt der von runden Klammern umgebene Schleifenkopf, wo die Initialisierung der Indexvariablen, die Spezifikation der Fortsetzungsbedingung und die Aktualisierungsvorschrift untergebracht werden können. Am Ende steht die wiederholt auszuführende Anweisung:

```
for (Initialisierung; Bedingung; Aktualisierung)
    Anweisung
```

Die drei Bestandteile des Schleifenkopfes gestalten sich wie folgt:

- **Initialisierung:** In der Regel wird man sich auf eine Indexvariable beschränken und dabei einen Ganzzahl-Typ wählen. Diese Variable wird im Schleifenkopf auf einen Startwert gesetzt und nötigenfalls dabei auch deklariert. Deklarationen und Initialisierungen werden vor dem ersten Durchlauf ausgeführt. Eine im Initialisierungsteil deklarierte Variable ist lokal bezüglich der `for`-Schleife, steht also nur in deren Anweisung sblock zur Verfügung.
- **Bedingung:** Üblicherweise wird eine Ober- oder Untergrenze für die Indexvariable gesetzt, doch erlaubt Java beliebige logische Ausdrücke. Die Bedingung wird *zu Beginn* eines Schleifendurchgangs geprüft. Resultiert der Wert true, so findet eine weitere Wiederholung des Anweisungsteils statt, anderenfalls wird die `for`-Anweisung verlassen. Folglich kann es auch passieren, dass überhaupt kein Durchlauf zustande kommt.
- **Aktualisierung:** Die Aktualisierung wird *am Ende* eines Schleifendurchgangs (nach Ausführung der Anweisung) vorgenommen.

```
public class ForSchleife {
    public static void main(String[] args) {
        int n = 10;
        double sq = 0.0;
        for (int i = 1; i <= n; i++) sq += i;
        System.out.println("Quadratsumme = " + sq);
    }
}
```

Zu den (zumindest stilistisch) bedenklichen Konstruktionen, die der Compiler klaglos umsetzt, gehören `for`-Schleifenköpfe ohne Initialisierung oder ohne Aktualisierung, wobei die trennenden Strichpunkte trotzdem zu setzen sind. In solchen Fällen ist die Umlaufzahl einer `for`-Schleife natürlich nicht mehr aus dem Schleifenkopf abzulesen. Dies gelingt auch dann nicht, wenn in der Schleifenanweisung eine Indexvariable modifiziert wird.

Die `while`-Anweisung kann als vereinfachte `for`-Anweisung aufgefasst werden: Wenn man im Kopf einer `for`-Schleife auf Initialisierung und Aktualisierung verzichten kann, ergibt sich daraus die `while`-Schleifenstruktur mit folgender Syntax;

```
while (Bedingung)
    Anweisung
```

Wie bei der `for`-Anweisung wird die Bedingung *zu Beginn* eines Schleifendurchgangs geprüft. Resultiert der Wert `true`, so wird der Anweisungsteil ausgeführt, anderenfalls wird die `while`-Anweisung verlassen, eventuell noch vor dem ersten Durchgang.

```
public class WhileSchleife {
    public static void main(String[] args) {
        int n = 10, i = 1;
        double sq = 0.0;
        while (i <= n) {
            sq += i;
            i++;
        }
        System.out.println(„Quadratsumme = “ + sq);
    }
}
```

Bei *fußgesteuerten Schleifen* wird, im Gegensatz zu kopfgesteuerten, immer erst der Körper ausgeführt und dann die Bedingung überprüft, ob der Körper nochmals durchlaufen werden soll. Der Körper einer fußgesteuerten Schleife wird also mindestens einmal ausgeführt. So wird bei der `do`-Schleife die Fortsetzungsbedingung am Ende der Schleifendurchläufe geprüft, so dass wenigstens ein Durchlauf stattfindet:

```
do
    Anweisung
while (Bedingung);
```

`do`-Schleifen werden seltener benötigt als `while`-Schleifen, sind aber beispielsweise dann von Vorteil, wenn man vom Benutzer eine Eingabe mit bestimmten Eigenschaften einfordern möchte. In folgendem Codesegment kommt die statische Methode `getchar()` aus der Klasse `Simput` zum Einsatz, die ein vom Benutzer eingegebenes und mit Enter quittiertes Zeichen als `char`-Wert ab liefert:

```
char antwort;
do {
    System.out.println(„Beenden (j/n)? “);
    antwort=Simput.getchar();
} while (antwort!=’j’ && antwort!=’n’);
```

Bei einer `do`-Schleife mit Anweisungsblock sollte man die `while`-Klausel unmittelbar hinter die schließende Blockklammer setzen (in derselben Zeile), um sie optisch von einer selbständigen `while`-Anweisung abzuheben. Bei jeder Wiederholungsanweisung (`for`,

while oder do) kann es in Abhängigkeit von der verwendeten Bedingung passieren, daß der Anweisungsblock unendlich oft ausgeführt wird. Endlosschleifen sind als gravierende Programmierfehler unbedingt zu vermeiden. Befindet sich ein Programm in diesem Zustand, muss es mit Hilfe des Betriebssystems abgebrochen werden, beispielsweise bei Konsolenanwendungen unter Windows über die Tastenkombination <Strg>+<C>. Im Falle von Programmen zur Ansteuerung von Robotersystemen kann sich eine solche Endlosschleife natürlich fatal auswirken, zumal sich ein solches System dem direkten Einflussbereich entzieht und daher ein <Strg>+<C> nicht abzusetzen ist.

Mit der break-Anweisung als Bestandteil der switch-Anweisung, kann eine for-, while- oder do-Schleife vorzeitig verlassen werden. Mit der continue-Anweisung veranlasst man Java, den aktuellen Schleifendurchgang zu beenden und sofort mit dem nächsten zu beginnen. In der Regel kommen break und continue im Rahmen einer Auswahlanweisung zum Einsatz, was das folgende Beispielprogramm zur Primzahlendiagnose illustriert.

```
public class PrimzahlenChecker {  
    public static void main(String[] args) {  
        boolean tg;  
        int i, mpk, zahl;  
        System.out.println(„Primzahlen-Checker\n“);  
        while (true) {  
            zahl=Character.getNumericValue(args[0].charAt(0));  
            System.out.print(„Zu untersuchende Zahl lautet: “ + zahl);  
            if (zahl <= 2 && zahl != 0) {  
                System.out.println(„Illegaler Wert!\n“);  
                continue;  
            }  
            if (zahl == 0) break;  
            tg = false;  
            mpk = (int) Math.sqrt(zahl); //maximaler Primzahenkandidat  
            for (i = 2; i <= mpk; i++)  
                if (zahl % i == 0) {  
                    tg = true;  
                    break;  
                }  
            if (tg)  
                System.out.println(zahl+„ist keine Primzahl (Teiler: “ +  
                    i + ”).\n“);  
            else  
                System.out.println(zahl + „ ist eine Primzahl.\n“);  
        }  
        System.out.println(„\nProgrammabbruch!“);  
    }  
}
```

## Vererbungsmechanismen

Bei einer objektorientierten Analyse einer Problemstellung versucht man, alle beteiligten Objekte zu identifizieren und definiert für sie jeweils eine Klasse, die durch Eigenschaften (Instanz- und Klassenvariablen) und Kompetenzen (Instanz- und Klassenmethoden) charakterisiert ist. Beim Modellieren eines Gegenstandsbereiches durch Java-Klassen sollten unbedingt auch die Spezialisierungs- bzw. Generalisierungsbeziehungen zwischen real existierenden Klassen abgebildet werden. In objektorientierten Programmiersprachen wie Java ist es weder sinnvoll noch erforderlich, jede Klasse einer Hierarchie komplett neu zu deklarieren. Stattdessen geht man von der allgemeinsten Klasse aus, leitet durch Spezialisierung neue Klassen ab, nach Bedarf in beliebig vielen Stufen. Eine abgeleitete Klasse erbt dabei alle Variablen und Methoden ihrer *Basis-* oder *Superklasse* und kann nach Bedarf Anpassungen bzw. Erweiterungen zur Lösung spezieller Aufgaben vornehmen:

- zusätzliche Variablen deklarieren
- zusätzliche Methoden definieren
- geerbte Methoden überschreiben, d. h. unter Beibehaltung des Namens umgestalten
- geerbte Variablen überdecken, beispielsweise unter Beibehaltung des Namens den Datentyp ändern

Die Vererbungstechnologie der OOP erlaubt nicht nur die bequeme Deklaration von Klassenhierarchien, sondern bietet auch eine spezielle Flexibilität: Über Variablen vom Basisklassentyp lassen sich auch Objekte aus einer abgeleiteten Klasse verwalten. Zwar können dabei nur Methoden aufgerufen werden, die schon in der Basisklasse bekannt sind, doch kommen ggf. die überschreibenden Realisationen der *abgeleiteten* Klasse zum Einsatz. Es ist also nicht ungewöhnlich, dass beispielsweise ein Array vom Basisklassentyp Objekte aus verschiedenen (abgeleiteten) Klassen aufnimmt, die sich beim Aufruf der gleichnamigen Methode unterschiedlich verhalten. Befragt man ein Objekt beispielsweise über die Methode `getPosition()`, ob es eine bestimmte Position im Raum einnimmt, dann ermittelt es die Antwort je nach Gestalt auf unterschiedliche Weise. Die Fähigkeit, mit Variablen vom Basisklassentyp auch Objekte aus abgeleiteten Klassen zu verwalten, bezeichnet man als *Polymorphie*. Mit ihrem Vererbungsmechanismus bietet die objektorientierte Programmierung ideale Voraussetzungen dafür, Softwaresysteme gemäss der Realität und deren inhärenten Problemstellung zu entwickeln. Dabei können allmählich umfangreiche und dabei doch robuste und wartungsfreundliche Softwaresysteme entstehen. Spätere Verbesserungen bei einer Superklasse kommen allen (direkt oder indirekt) abgeleiteten Klassen zu Gute. Zusätzlich erhöht die Möglichkeit der *Polymorphie* bei richtiger Verwendung die Produktivität für ein anstehendes Software-Recycling. Dank dieser Technik kann ein Programm auch Objekte aus solchen Klassen verarbeiten, die zum Zeitpunkt seiner Übersetzung noch gar nicht existierten. In Java wird die Klassifikation insofern auf die Spitze getrieben, als *alle* Klassen (sowohl die im Java-API mitgelieferten als auch die vom Programmierer definierten) von der Klasse `Object` aus dem Paket `java.lang` abstammen. Wird in der Definition einer Klasse keine Basisklasse angegeben, dann stammt sie auf direktem Wege von `Object` ab, anderenfalls indirekt.

In folgendem Programm wird zunächst die Basisklasse Robotersysteme definiert, die Instanzvariablen für die X- und die Y-Position der linken oberen Ecke des Systems in seiner Umwelt sowie über einen initialisierenden Konstruktor verfügt:

```
class Robotersysteme {  
    int xpos = 100, ypos = 100;  
  
    Robotersysteme(int txpos, int typos) {  
        xpos = txpos;  
        ypos = typos;  
        System.out.println(„Robotersysteme-Konstruktor“);  
    }  
    Robotersysteme() {}  
}
```

Die Klasse MobileRobotersysteme wird mit Hilfe des Schlüsselwortes extends als Spezialisierung der Klasse Robotersysteme definiert. Sie erbt die beiden Positionsvariablen und ergänzt eine zusätzliche Instanzvariable für die Geschwindigkeit.

```
public class MobileRobotersysteme extends Robotersysteme {  
    int geschwindigkeit = 15;  
  
    MobileRobotersysteme(int txpos, int typos, int tgeschwindigkeit) {  
        super(txpos, typos);  
        geschwindigkeit = tgeschwindigkeit;  
        System.out.println(„MobileRobotersysteme-Konstruktor“);  
    }  
    MobileRobotersysteme() {}  
}
```

Es wird ein expliziter MobileRobotersysteme-Konstruktor definiert, der über das Schlüsselwort super den Konstruktor der übergeordneten Klasse aufruft. Das Schlüsselwort legitimiert den oben eingeführten Begriff der *Superklasse*.

```
public class Test {  
    public static void main(String[] args) {  
        Robotersysteme rbs = new Robotersysteme(50, 50);  
        MobileRobotersysteme mrbs =  
            new MobileRobotersysteme(10, 10, 5);  
    }  
}
```

Im obigen Testprogramm werden ein Objekt aus der Basisklasse und ein Objekt aus der abgeleiteten Klasse erzeugt.

In folgender Variante des Beispielprogramms wird der Effekt des Zugriffsmodifikators `protected` demonstriert. Zunächst wird dafür gesorgt, dass die Klassen `Gehmaschinen` und `RadgetriebenSystem` zu verschiedenen Paketen gehören, weil *innerhalb* eines Paketes die abgeleiteten Klassen grundsätzlich dieselben Zugriffsrechte haben wie beliebige andere Klassen. Weil die Basisklasse `Robotersysteme` die Instanzvariablen `xpos` und `ypos` als `protected` deklariert, können ihre Werte in der `RadgetriebenSysteme`-Methode `setzePos()` verändert werden, obwohl `RadgetriebenSysteme` nicht zum selben Paket gehört:

```
import de.ifpe.java.vererbung.Robotersysteme;
public class RadgetriebenSystem extends Robotersysteme {
    int geschwindigkeit = 75;

    public RadgetriebenSystem(int txpos, int typos,
        int tgeschwindigkeit) {
        super(txpos, typos);
        geschwindigkeit = tgeschwindigkeit;
        System.out.println("RadgetriebenSystem-Konstruktor");
    }

    void setzePos(int txpos, int typos) {
        xpos = txpos;
        ypos = typos;
    }
}
```

Es ist zu beachten, dass hier eine *geerbte Instanzvariable* von `RadgetriebenSystem`-Objekten verändert wird. Auf die `xpos`-Eigenschaft von `Robotersystem`-Objekten hat auch die `RadgetriebenSystem` -Klasse keinen Zugriff. Wird beispielsweise im `RadgetriebenSystem` -Konstruktor ein `Robotersysteme`-Objekt erstellt, dann ist *kein* Zugriff auf dessen `xpos` erlaubt. Die folgenden Zeilen

```
Robotersysteme r = new Robotersysteme(20, 20);
r.xpos = 33;
```

führen zur entsprechenden Fehlermeldung. Abgeleitete Klassen erben die Basisklassen-Konstruktoren *nicht*, können sie aber in eigenen Konstruktoren über das Schlüsselwort `super` erreichen und damit verwenden. Dadurch wird es möglich, geerbte Instanzvariablen zu initialisieren, die in der Basisklasse als `private` deklariert wurden. Wird in einem Konstruktor einer abgeleiteten Klasse kein Basisklassen-Konstruktor aufgerufen, dann ruft der Compiler implizit den parameterfreien Konstruktor der Basisklasse auf. Fehlt ein solcher, weil der Programmierer einen eigenen, parametrisierten Konstruktor erstellt und nicht durch einen expliziten parameterfreien Konstruktor ergänzt hat, dann protestiert der Compiler:

RadgetriebenSystem.java:5: cannot resolve symbol

Es existieren zwei Möglichkeiten, ein solches Problem zu lösen:

- Es wird im Konstruktor der abgeleiteten Klasse über das Schlüsselwort super ein parametrisierter Basisklassen-Konstruktor aufgerufen. Dieser Aufruf muss am Anfang des Konstruktors stehen.
- In der Basisklasse wird ein parameterfreier Konstruktor definiert.

Der parameterfreie Basisklassen-Konstruktor wird übrigens auch vom Standard-Konstruktor der abgeleiteten Klasse aufgerufen. Beim Erzeugen eines Unterklassen-Objektes laufen damit folgende Initialisierungs-Maßnahmen ab:

- Alle Elementvariablen werden mit den typspezifischen Nullwerten initialisiert.
- Der Unterklassen-Konstruktor beginnt seine Tätigkeit mit dem (impliziten oder expliziten) Aufruf eines Basisklassen-Konstruktors. Falls in der Vererbungshierarchie der Urahn Object noch nicht erreicht ist, wird am Anfang des Basisklassen-Konstruktors wiederum ein Konstruktor des nächst höheren Vorfahren aufgerufen, bis diese Rekursion schließlich mit dem Aufruf eines Object-Konstruktors endet.

Auf jeder Hierarchieebene (beginnend bei Object) laufen zwei Teilschritte ab:

- Die Instanzvariablen der Klasse werden gemäß der Deklaration initialisiert.
- Der Rumpf des Konstruktors wird ausgeführt.

Betrachtet man das obige Beispiel beim Erzeugen eines RadgetriebenSystem-Objektes mit dem Konstruktor-Aufruf

```
RadgetriebenSystem r = new RadgetriebenSystem(10,10,5);
```

so lässt sich folgender Ablauf aufzeichnen:

- Die Instanzvariablen von RadgetriebenSystem - Objekten (auch die geerbten) werden angelegt und mit Nullen initialisiert.
- Aktionen für die Klasse Object:
  - Die Instanzvariablen der Klasse Object werden initialisiert.
  - Der Rumpf des Object-Konstruktors wird ausgeführt. Weil der Robotersysteme-Konstruktor keinen expliziten Basisklassen-Konstruktor-Aufruf vornimmt, kommt der parameterfreie Object-Konstruktor zum Einsatz.
- Aktionen für die Klasse Robotersysteme:
  - Die Instanzvariablen xpos und ypos erhalten den Initialisierungswert laut Deklaration, was allerdings keine Änderung bewirkt.
  - Der Rumpf des Konstruktor-Aufrufs Robotersysteme(10,10) wird ausgeführt, wobei xpos und ypos die Werte 10 bzw. 10 erhalten.

- Aktionen für die Klasse RadgetriebenSystem:
  - Die Instanzvariable `geschwindigkeit` erhält den Initialisierungswert 75 aus der Deklaration.
  - Der Rumpf des Konstruktor-Aufrufs `RadgetriebenSystem(10, 10, 5)` wird ausgeführt, wobei `geschwindigkeit` den Wert 5 erhält.

Eine Basisklassen-Methode darf in einer Unterklasse durch eine Methode mit gleichem Namen und gleicher Parameterliste überschrieben werden, die dann für die speziellere Unterklassen-Situation ein angepasstes Verhalten realisiert. Es liegt übrigens *keine* Überschreibung vor, wenn in der Unterklasse eine Methode mit gleichem Namen, aber abweichender Parameterliste deklariert wird. In diesem Fall sind die beiden Signaturen verschieden, und es handelt sich um eine *Überladung*.

Im obigen Beispiel wird die MobileRobotersysteme-Basisklasse um eine Methode `Ortsangabe()` erweitert, welche die Position der linken oberen Ecke ausgibt:

```
public void Ortsangabe() {
    System.out.println("\nOben Links: (" + xpos + ", " + ypos + ")");
}
```

In der RadgetriebenSystem-Klasse kann eine bessere Ortsangaben-Methode realisiert werden, weil hier auch die rechte untere Ecke definiert ist:

```
public void Ortsangabe() {
    super.Ortsangabe();
    System.out.println("Unten Rechts: (" +
        (xpos + 2 * geschwindigkeit) +
        ", " + (ypos + 2 * geschwindigkeit) + ")");
}
```

In der überschreibenden Methode kann man sich oft durch Rückgriff auf die überschriebene Methode die Arbeit erleichtern, wobei wieder das Schlüsselwort `super` zum Einsatz kommt. Das folgende Programm schickt an eine MobileRobotersysteme und an einen RadgetriebenSystem jeweils die Nachricht `Ortsangabe()`, und beide zeigen ihr artgerechtes Verhalten:

```
public class Test_Ortsangabe {
    public static void main(String[] args) {
        MobileRobotersysteme mbs =
            new MobileRobotersysteme(10, 20, 5);
        mbs.Ortsangabe();
        RadgetriebenSystem rbs =
            new RadgetriebenSystem(50, 50, 10);
        rbs.Ortsangabe();
    }
}
```

Obwohl die beiden Ortsangabe()-Methodenaufrufe unterschiedliches Verhalten zeigen, liegt *keine* Polymorphie vor, weil die Referenzvariablen mbs und rbs von verschiedenem Typ sind. Polymorphie setzt voraus, dass Referenzvariablen vom *selben* deklarierten Typ auf Objekte aus verschiedenen Klassen zeigen. Wenn dann über die typgleichen Referenzvariablen durch Aufruf einer in den spezialisierten Klassen überschriebenen Basisklassenmethode ein jeweils angepasstes Verhalten ausgelöst wird, spricht man von echter Polymorphie.

Gelegentlich möchte man das Überschreiben einer Methode *verhindern*, damit sich der Anwender einer UnterkLASSE darauf verlassen kann, dass dort die geerbte Methode *nicht* überschrieben worden ist. Dient etwa die Methode `passwort()` einer Klasse A zum Abfragen eines Passwortes, will ihr Entwickler eventuell verhindern, dass `passwort()` in einer von A abstammenden Klasse B überschrieben wird. Damit kann dem Anwender der Klasse B die ursprüngliche Funktionalität von `passwort()` garantiert werden. Um das Überscheiben einer Methode zu verhindern, leitet man ihre Definition mit dem Modifikator `final` ein. So lässt sich in die MobileRobotersysteme-Klasse eine Methode `obereslinkesEck()` zur Ausgabe der oberen linken Ecke implementieren, die von den spezialisierten Klassen nicht geändert werden muss und daher als `final` (endgültig) deklariert werden kann:

```
final public void obereslinkesEck() {  
    System.out.print("\nOben Links: ("+xpost+","+ypos+")  
}
```

Neben der beschriebenen Entwicklungssicherheit bringt das Finalisieren einer Methode noch einen Performanz-Vorteil: Während bei nicht-finalisierten Methoden *das Laufzeit-system* im Dienste der Polymorphie feststellen muß, welche Variante in Abhängigkeit von der Klassenzugehörigkeit des betroffenen Objektes tatsächlich ausgeführt werden soll, steht eine final-Methode schon beim Compilieren fest. Die eben für das Finalisieren von *Methoden* genannten Sicherheitsüberlegungen können auch zum Entschluss führen, eine komplette Klasse mit dem Schlüsselwort `final` als *endgültig* zu deklarieren, so daß sie zwar verwendet, aber nicht mehr beerbt werden kann. Finalisierte Klassen eignen sich auch zum Aufbewahren von Konstanten. Um das unsinnige Erzeugen von Objekten zu einer Konstantenklasse zu verhindern, wird der Konstruktor als `private` definiert.

```
final public class Konstanten {  
    public final static int MAX = 333;  
    public final static int MORITZ = 500;  
    private Konstanten() {}  
}
```

Es lassen sich Elementvariablen auch überdecken. Wird in der abgeleiteten Klasse Sohn für eine Instanzvariable ein Name verwendet, der bereits eine Variable der beerbten Klasse Vater bezeichnet, dann wird die Basis-Variable überdeckt.

```
class Vater {  
    String x = "Vater";  
    void vm() {  
        System.out.println("x in Vater-Methode: "+x);  
    }  
}
```

Sie ist jedoch weiterhin vorhanden und kommt in folgenden Situationen zum Einsatz:

- Vom Vater geerbte Methoden greifen weiterhin auf die Vater-Variable zu, während die zusätzlichen Methoden der Sohn-Klasse auf die Sohn-Variable zugreifen.
- Analog zu einer überschriebenen Methode kann die überdeckte Variable mit Hilfe des Schlüsselwortes `super` weiterhin benutzt werden.

```
class Sohn extends Vater {  
    int x = 333;  
    void sm() {  
        System.out.println("x in Sohn-Methode: "+x);  
        System.out.println("super-x in Sohn-Methode.: "+super.x);  
    }  
}
```

Im folgenden Beispielprogramm führt ein Sohn-Objekt eine Vater- und eine Sohn-Methode aus, um die beschriebenen Zugriffsvarianten zu demonstrieren:

```
public class Test {  
    public static void main(String[] args) {  
        Sohn so = new Sohn();  
        so.vm();  
        so.sm();  
    }  
}
```

Einer Basisklassen-Referenzvariablen kann jederzeit eine Unterklassen-Referenz zugewiesen werden. Dies ermöglicht es, Basisklassen-Referenzvariablen zur flexiblen Verwaltung von Objekten aus beliebigen Unterklassen zu verwenden. Zur Demonstration wird zunächst die Vater-Klasse erweitert:

```
package de.ifpe.java.vatersohn;
class Vater {
    String x = "Vater";
    int iv = 1;

    void vm() {
        System.out.println("x in Vater-Methode: "+x);
    }

    void sageHallo() {
        System.out.println("Hallo, ich bin der Vater...");
    }
}
```

Daraus wird die Klasse namens Sohn abgeleitet, welche die väterliche `sageHallo()`-Methode überschreibt und zusätzliche Elemente ergänzt:

- die Instanzvariable `is`
- einen parameterfreien Konstruktor, der die geerbte Instanzvariable `iv` auf den Wert 2 setzt.
- die Methode `mitteilung()`

```
class Sohn extends Vater {
    int x = 333;
    int is = 3;

    Sohn() {
        iv = 2;
    }

    void sageHallo() {
        System.out.println("Hallo, ich bin der Sohn...");
    }

    void mitteilung() {
        System.out.println("Mitteilung des Sohnes");
    }

    void sm() {
        System.out.println("x in Sohn-Methode: "+x);
        System.out.println("super-x in Sohn-Methode.: "+super.x);
    }
}
```

Ein Array vom Typ Vater kann Referenzen auf Väter und Söhne aufnehmen, wie das folgende Beispielprogramm zeigt:

```
public class ArrayTest {
    public static void main(String[] args) {
        Vater[] varray = new Vater[3];
        varray[0] = new Vater();
        varray[1] = new Sohn();
        System.out.println("iv-Wert von varray[1]: " +
            + varray[1].iv);
        varray[0].sageHallo();
        varray[1].sageHallo();
        System.out.println();
    }
}
```

Hier liegt nun echte *Polymorphie* vor, weil die beiden folgenden Kriterien erfüllt sind:

- Derselbe Methodenaufruf wird von den einzelnen Objekten unterschiedlich ausgeführt.
- Die Objekte werden über Referenzen desselben Basistyps angesprochen, und es wird erst zur Laufzeit festgestellt, welcher Klasse ein Objekt tatsächlich angehört.

Zur Verifikation, dass der Compiler die Klassenzugehörigkeit nicht kennen muss, darf im Beispielprogramm die Klasse des Array-Elementes varray[2] vom Benutzer festgelegt werden. Über eine Vater-Referenzvariable, die auf ein Sohn-Objekt zeigt, sind Erweiterungen der Sohn-Klasse in Form zusätzlicher Elementvariablen und Methoden nicht unmittelbar zugänglich. Wenn eine Basisklassen-Referenz als Unterklassen-Referenz behandelt werden soll, um eine unterklassenspezifische Methode oder Variable anzusprechen, dann muss der cast-Operator verwendet werden, beispielsweise:

```
System.out.println(((Sohn) varray[1]).is);
```

Im Zweifelsfall sollte man sich über den instanceof-Operator vergewissern, ob das referenzierte Objekt tatsächlich zur vermuteten Klasse gehört.

```
if (varray[1] instanceof Sohn)
    ((Sohn) varray[1]).nurso();
```

Um die eben beschriebene gemeinsame Verwaltung von Objekten aus diversen Unterklassen über einen Array vom Basisklassentyp realisieren und dabei Polymorphie nutzen zu können, müssen die betroffenen Methoden in der Basisklasse vorhanden sein. Wenn es für die Basisklasse zu einer Methode keine sinnvolle Implementierung gibt, kann man die Methode dort als abstract deklarieren und auf eine Implementierung an dieser Stelle bewusst verzichten.

---

```
abstract class AbstractVater {
    abstract void hallo();
}
```

Enthält eine Klasse mindestens eine abstrakte Methode, dann handelt es sich um eine *abstrakte Klasse*, und die Klassendefinition muss mit dem Modifikator `abstract` eingeleitet werden. Aus einer abstrakten Klasse kann man zwar keine Objekte erzeugen, aber andere Klassen ableiten. Implementiert eine abgeleitete Klasse die abstrakten Methoden, lassen sich Objekte daraus herstellen. Im folgenden Beispiel wird aus dem abstrakten Vater eine „konkrete“ Tochter-Klasse abgeleitet:

```
public class Tochter extends AbstractVater {
    void sageHallo() {
        System.out.println("Hallo. ich bin die Tochter");
    }
}
```

Aus einer abstrakten Klasse lassen sich zwar keine Objekte erzeugen, doch kann sie als Datentyp verwendet werden. Referenzen dieses Typs sind ja auch unverzichtbar, wenn Objekte diverser Unterklassen gemeinsam verwaltet werden sollen:

```
class AbstractTest {
    public static void main(String[] args) {
        AbstractVater[] varray = new AbstractVater[2];
        varray[0] = new Tochter();
        varray[0].sageHallo();
    }
}
```

## Paketierung

Die Programmiersprache Java geht davon aus, dass jede Klasse einem *Paket* (engl.: *package*) zugeteilt wird.<sup>25</sup> Daher spricht man solche Klassen im Allgemeinen über ein durch Punkt getrenntes Paar aus Paketnamen und Klassennamen an. Obwohl man etliche Klassen auch ohne vorangestellten Paketnamen ansprechen kann, gehören diese doch stets zu einem Paket:

- Alle Klassen in einem Dateiverzeichnis, die keinem Paket zugeordnet wurden, bilden ein *anonymes Paket*. Dies kann dann sinnvoll sein, wenn die Programme nur von sehr geringer Komplexität sind.
- Das Paket `java.lang` mit besonders fundamentalen Klassen (beispielsweise `String`, `System`, `Math`) wird bei jeder Anwendung automatisch importiert, so dass seine Klassen direkt angesprochen werden können.

---

<sup>25</sup> Vgl. Arnold und Gosling 2000.

Vielfach werden Java-Pakete auch als *Klassenbibliotheken* bezeichnet, die neben Klassen auch *Schnittstellen (Interfaces)* enthalten können. Pakete erfüllen in Java viele wichtige Aufgaben, die vor allem in größeren Projekten relevant sind:

- *Strukturierung von Projekten*: Wenn sehr viele Klassen vorhanden sind, kann man mit Paketen Ordnung schaffen. In der Regel befinden sich alle class-Dateien eines Paketes in einem Dateiverzeichnis, dessen Name mit dem Paketnamen übereinstimmt. Es ist auch ein hierarchischer Aufbau über Unterpakete möglich, wobei die Paketstruktur einem Dateiverzeichnisbaum entspricht. Im Namen eines konkreten (Unter-)Paketes folgen dann die Verzeichnissnamen aus dem zugehörigen Pfad durch Punkte getrennt aufeinander, beispielsweise `java.util.zip`. Gerade bei der Weitergabe von Programmen (Deployment) ist es nützlich, eine komplette Paketstruktur in eine Java-Archivdatei (mit Extension `.jar`) zu verpacken.
- *Vermeidung von Namenskonflikten*: Jedes Paket bildet einen eigenen Namensraum. Identische Bezeichner stellen also kein Problem dar, solange sie sich in verschiedenen Paketen befinden.
- *Steuerung der Zugriffskontrolle*: Per Voreinstellung haben die Klassen eines Paketes wechselseitig uneingeschränkte Zugriffsrechte, während Klassen aus anderen Paketen nur auf public-Elemente zugreifen dürfen.

Zur Java-Plattform gehören zahlreiche Pakete, die Klassen für wichtige Aufgaben der Programmierung (beispielsweise Stringverarbeitung, Netzverbindungen, Datenbankzugriff) enthalten. Die Zusammenfassung dieser Pakete wird oft als *Java-API* (Application Programming Interface) bezeichnet.

Um sich den Einsatz von Paketen zu veranschaulichen, wird in dem folgenden Beispielprogramm zunächst ein einfaches Paket namens `demonstrationspackage` mit den Klassen `Demo_A`, `Demo_B` und `Demo_C` erstellt. Hierzu wird an den Anfang jeder einzubehorenden Quellcodedatei die package-Anweisung mit dem Paketnamen gesetzt, der üblicherweise komplett klein geschrieben wird.

```
package de.ifpe.java.demonstrationspackage;
public class Demo_A {
    private static int anzahl;
    private int objnr;

    public Demo_A() {
        objnr = ++ anzahl;
    }

    public void prinr() {
        System.out.println("Klasse Demo_A, Objekt Nr. "+objnr);
    }
}
```

Entsprechend der Klasse `Demo_A` sind die beiden anderen Klassen zu gestalten: `Demo_B` und `Demo_C`.

Müssen aus bestimmten Gründen in einer Quellcodedatei *mehrere* Klassendefinitionen vorhanden sein, was man in Bezug auf die Entwicklung von Java im Allgemeinen und embedded Klassen in Speziellen allerdings vermeiden sollte, so werden *alle* Klassen dem Paket zugeordnet. Vor einer package-Anweisung dürfen höchstens Kommentar- oder Leerzeilen stehen. Die Klassen eines Paketes können von Klassen aus fremden Paketen nur dann verwendet werden, wenn sie als public definiert sind. Pro Quellcodedatei ist nur eine public-Klasse erlaubt. Zusätzlich müssen auch Methoden und Variablen explizit per public-Modifikator für fremde Pakete freigegeben werden. Steht beispielsweise kein public-Konstruktor zur Verfügung, können fremde Pakete eine Klasse zwar „sehen“, aber keine Objekte dieses Typs erzeugen. Die beim Übersetzen der zu einem Paket gehörigen Quellcodedateien entstehenden class-Dateien gehören physikalisch in ein gemeinsames Dateiverzeichnis, dessen Name mit dem Paketnamen identisch ist. Für die Verwendung eines Paketes sind ausschließlich die class-Dateien erforderlich. In dem Beispiel wird also ein Verzeichnis `demopack` angelegt und die Bytecode-Dateien `Demo_A.class`, `Demo_B.class` und `Demo_C.class` dort abgelegt.

Ein Paket kann hierarchisch in *Unterpakete* eingeteilt werden, was bei den Java-API-Paketen in der Regel geschehen ist. Auf jeder Stufe der Pakethierarchie können sowohl Klassen als auch Unterpakete enthalten sein. So enthält beispielsweise das Paket `java.util` u. a. die Klassen `Random`, `Vector`, oder die Unterpakete `jar`, `regex`, `zip`, etc. Soll eine Klasse einem Unterpaket zugeordnet werden, muss in der package-Anweisung am Anfang der Quellcodedatei der gesamte Paketpfad angegeben werden, wobei die Namensbestandteile jeweils durch einen Punkt getrennt werden. Dies ist beispielsweise der Quellcode der Klasse `Demo_A1`, die in das Unterpaket `subpackage1` des `demopackage`-Paketes eingeordnet wird:

```
package de.ifpe.java.demopackage.subpackage1;
public class Demo_A1 {
    private static int anzahl;
    private int objnr;

    public Demo_A1() {
        objnr = ++ anzahl;
    }

    public void prinr() {
        System.out.println("Klasse Demo_A1, Objekt Nr. "+objnr);
    }
}
```

Die class-Dateien einer Pakethierarchie legt man zunächst in einem analog aufgebauten Dateiverzeichnisbaum ab, wobei jedem (Unter)paket ein (Unter)verzeichnis entspricht. In diesem Beispiel müssen die class-Dateien also folgendermaßen angeordnet sein (Abb. 8.24):

Ordner	Name	Größe	Typ	Geändert am
Java-Lessons	bin		Dateiordner	06.05.2006 13:00
bin	de	1 KB	CLASS-Datei	06.05.2006 12:25
de	ifpe	1 KB	CLASS-Datei	06.05.2006 13:02
ifpe	java	1 KB	CLASS-Datei	06.05.2006 13:00
java	deadlock	2 KB	CLASS-Datei	06.05.2006 13:03
deadlock	demopackage	1 KB	CLASS-Datei	06.05.2006 12:49
demopackage	Programm.class			
demopackage	subpackage1			
demopackage\subpackage1	Demo_A.class			
demopackage\subpackage1	Demo_B.class			
demopackage\subpackage1	Demo_C.class			
demopackage\subpackage1	DemoPackageProgramm.class			
demopackage\subpackage1	Programm.class			

**Abb. 8.24** Pakethierarchie

Mit dem SDK-Werkzeug `jar.exe` lassen sich dann ganze Paket- bzw. Verzeichnisbäume in eine einzige Java-Archivdatei verpacken. Damit ein Paket genutzt werden kann, muss es sich in einem Ordner befinden, der vom Compiler bzw. Interpreter bei Bedarf nach Klassen durchsucht wird. Dies kann auf unterschiedliche Weise erfolgen:

- Ablage des Pakets im aktuellen Verzeichnis: Dies ist natürlich keine sinnvolle Option, wenn ein Paket in mehreren Projekten eingesetzt werden soll.
- Bekanntgabe des Pakets in der `CLASSPATH`-Umgebungsvariablen

Angabe der `classpath`-Option bei Aufruf des Compilers bzw. Interpreters: Dies geschieht im Rahmen einer korrekt konfigurierten Entwicklungsumgebung in der Regel automatisch.

Der Ordner mit den Java-API-Paketen wird grundsätzlich durchsucht, falls die Installation nicht beschädigt ist. Für den konkreten Zugriff auf die Klassen eines Paketes bietet Java folgende Möglichkeiten:

- Verwendung des vollqualifizierten Klassennamens: Dem Klassennamen ist der durch Punkt abgetrennte Pakettamen voran zu stellen. Bei einem hierarchischen Paketaufbau ist der gesamte Pfad anzugeben, wobei die Unterpakettamen wiederum durch Punkte zu trennen sind.
- Import einer benötigten Klasse oder eines kompletten Paketes: Um die lästige Angabe von Pakettamen zu vermeiden, kann man einzelne Klassen und/oder Pakete *importieren*. Anschließend sind alle importierten Klassen direkt (ohne Paket-Präfix) ansprechbar.

Die zuständigen `import`-Anweisungen sind an den Anfang einer Quelltextdatei zu setzen, ggf. aber *hinter* einer `package`-Deklaration. So wird im folgenden Programm die Klasse `Random` aus dem API-Paket `java.util` importiert und verwendet:

```
package de.ifpe.java.demopackage;
import java.util.Random;

public class Programm {
    public static void main(String[] args) {
        Random zzg = new Random(System.currentTimeMillis());
        System.out.println(zzg.nextInt(101));
    }
}
```

Um *alle* Klassen aus dem Paket `java.util` zu importieren, schreibt man `import java.util.*;` Dabei gilt es zu beachten, dass Unterpakete durch den Joker-Stern *nicht* einbezogen werden. Für sie ist bei Bedarf eine separate bzw. dedizierte Import-Anweisung fällig. Weil durch die Verwendung des Jokerzeichens *keine* Rechenzeit- oder Speicherressourcen verschwendet werden, ist dieses bequeme Vorgehen im Allgemeinen sinnvoll, falls keine Namenskollisionen (durch identische Klassennamen in verschiedenen Paketen) auftreten. Für das wichtige API-Paket `java.lang` übernimmt der Compiler den Import, so dass seine Klassen stets direkt angesprochen werden können.

```
package de.ifpe.java.demopackage;
import de.ifpe.java.demopackage.subpackage1.Demo_A1;
import de.ifpe.java.demopackage.subpackage1.Demo_B1;
public class DemoPackageProgramm {
    public static void main(String[] args) {
        Demo_A a = new Demo_A(), aa = new Demo_A();
        a.prinr();
        aa.prinr();

        Demo_B b = new Demo_B();
        b.prinr();

        Demo_C c = new Demo_C();
        c.prinr();

        Demo_A1 a1 = new Demo_A1();
        a1.prinr();

        Demo_B1 b1 = new Demo_B1();
        b1.prinr();
    }
}
```

Im Rahmen einer objektorientierten Entwicklung lassen sich Zugriffsrechte auf Klassen, Instanzvariablen und Methoden vergeben. So sind die Klassen eines Paketes für Klassen aus fremden Paketen nur dann sichtbar, wenn bei der Definition der Zugriffsmodifikator `public` angegeben wird. Wird im demopackage-Paket die Klasse `Demo_A` ohne `public`-Zugriffsmodifikator definiert, scheitert das Compilieren. Pro Quellcodedatei darf nur *eine* Klasse als `public` deklariert werden. Eventuell vorhandene zusätzliche Klassen sind also nur paketintern zu verwenden. Diese Regel stellt allerdings keine Einschränkung dar, da man in der Regel ohnehin für jede Klasse eine eigene Quellcodedatei verwendet (mit dem Namen der Klasse plus angehängter Erweiterung.java) (Tab. 8.9).

Bei der Deklaration bzw. Definition von Variablen bzw. Methoden (objekt- oder klassenbezogen) können die Modifikatoren `private`, `protected` und `public` angegeben werden,

**Tab. 8.9** Zugriffe

Modifikator	Der Zugriff ist erlaubt für			
	Eigene Klasse	Abgeleitete Klasse	Klassen im Paket	Sonstige Klassen
Ohne Angabe	Ja	Nein	Ja	Nein
Private	Ja	Nein	Nein	Nein
Protected	Ja	Nur geerbte Elemente	Ja	Nein
Public	Ja	ja	Ja	Ja

um die Zugriffsrechte festzulegen. In der folgenden Tabelle wird die Wirkung bei einer Klasse beschrieben, die selbst als `public` definiert ist. Wenn zu einem Programm zahlreiche class-Dateien und zusätzliche Hilfsdateien gehören, dann sollten diese in einer Java-Archivdatei (Namenserweiterung.jar) übersichtlich und komprimiert zusammengefasst werden. Die wichtigsten Eigenschaften von solchen Java-Archivateien sind:

- *Übersichtlichkeit/Bequemlichkeit:* Im Vergleich zu zahlreichen Einzeldateien ist ein Archiv für den Anwender deutlich bequemer. Ein per Archiv ausgeliefertes Programm kann sogar direkt über die Archivdatei gestartet werden.
- Eine *Archivdatei kann analog zu einem Verzeichnis* in den Suchpfad für class-Dateien aufgenommen werden.
- *Verkürzte Zugriffs- bzw. Übertragungszeiten:* Eine einzelne Archivdatei reduziert im Vergleich zu zahlreichen einzelnen Dateien die Zugriffszeiten beim Laden von Klassen und Ressourcen.
- *Kompression:* Java-Archivdateien können komprimiert werden, was für embedded Software durch den beschleunigten Transport sinnvoll ist, bei lokal installierten Anwendungen jedoch wegen der erforderlichen Dekomprimierung eher nachteilig. Die Archivdateien mit den Java-API-Klassen (beispielsweise `rt.jar`) sind daher *nicht* komprimiert. Weil Java-Archive das von PKWARE definierte ZIP-Dateiformat besitzen, können sie von diversen (De-)Komprimierungsprogrammen geöffnet werden. Das Erzeugen von Java-Archiven sollte man aber dem speziell für diesen Zweck entworfenen SDK-Werkzeug `jar.exe` überlassen.
- *Sicherheit:* Bei *signierten* JAR-Dateien kann sich der Anwender Gewissheit über den Urheber verschaffen und der Software entsprechende Rechte einräumen.

Zum Erstellen und Verändern von Java-Archivdateien dient das Werkzeug `jar.exe` aus dem Java-SDK. Es wird dazu verwendet, um aus dem demopackage-Paket eine Archivdatei zu erzeugen. Dies lässt sich beispielsweise über ein Konsolenfenster durchführen, indem man dort mit folgendem jar-Aufruf das Archiv `demopackage.jar` mit der gesamten Paket-Hierarchie erstellt:

```
jar cf0 demopack.jar demopack
```

Die Parametrisierung des Aufrufs bedeutet dabei:

- 1. Parameter (*Optionen*): Die Optionen bestehen aus jeweils einem Buchstaben und müssen unmittelbar hintereinander geschrieben werden. C steht für (create) das Erstellen eines Archivs. Mit f (für: file) wird die Ausgabe in eine Datei geleitet, wobei der Dateiname als zweiter Kommandozeilenparameter anzugeben ist. Mit der Ziffer 0 wird die ZIP-Kompression abgeschaltet.
- 2. Parameter (*Archivdatei*): Der Archivdateiname muss einschließlich Extension (üblicherweise.jar) angegeben werden.
- 3. Parameter (zu archivierende Dateien und Ordner): Bei einem Ordner wird rekursiv der gesamte Verzeichnis-Ast einbezogen. Dabei werden nicht nur Bytecode-, sondern beispielsweise auch Multimediadateien berücksichtigt. Selbstverständlich kann eine Archivdatei auch mehrere Pakete bzw. Ordner aufnehmen.<sup>26</sup>

Weitere Informationen über das Archivierungswerkzeug findet man in der SDK-Dokumentation unter *Java Archive (JAR) Files*. Um ein Archiv mit seinen Paketklassen nutzen zu können, muss es in den Klassen-Suchpfad des Compilers bzw. Interpreters aufgenommen werden.<sup>27</sup> Befindet sich beispielsweise die Archivdatei demopackage.jar im aktuellen Verzeichnis zusammen mit der Quellcodedatei der Klasse PackageDemo, die das Paket demopackage importiert, dann kann das Übersetzen und Ausführen dieser Klasse mit folgenden Aufrufen der SDK-Werkzeuge javac und java durchgeführt werden:

```
javac -classpath demopack.jar;. PackDemo.java  
java -classpath demopack.jar;. PackDemo
```

Um eine als Applikation ausführbare JAR-Datei zu erhalten, kann man folgendermaßen vorgehen: Man nimmt eine ausführbare Klasse in das Archiv auf, die bekanntlich eine Methode main() mit folgender Signatur besitzen muss:

```
public static void main(String[] args)
```

Diese Klasse wird im *Manifest* des Archivs als Hauptklasse eingetragen. Im Manifest eines Archivs wird sein Inhalt beschrieben. Es befindet sich in der Textdatei MANIFEST.MF, die das jar-Werkzeug im Ordner META-INF eines Archivs anlegt. Um dort beispielsweise die Hauptklasse DemoPackageProgramm auszuzeichnen, legt man eine Textdatei an, die folgende Zeile und eine anschließende Zeilenschaltung enthält:

```
Main-Class: PackDemo
```

Im jar-Aufruf zum Erstellen des Archivs wird über die Option m eine Datei mit Manifest-Informationen angekündigt, beispielsweise mit dem Namen PDApp.txt:

<sup>26</sup> Vgl. Steinmetz 2000.

<sup>27</sup> Vgl. Aho et al. 1986.

```
jar cmf0 PDApp.txt PDApp.jar PackDemo.class demopack
```

Dabei gilt es zu beachten, dass die Namen der Manifest- und der Archivdatei in derselben Reihenfolge wie die zugehörigen Optionen stehen. Dann kann die Applikation über die Archivdatei gestartet werden:

```
java -jar PDApp.jar
```

Dabei muß auf dem Zielrechner natürlich die Java-Laufzeitumgebung installiert sein. Wenn mit dem Betriebssystem noch ein jar-Dateityp passend verknüpft wird, klappt der Start sogar per Mausklick auf die Archivdatei.

## Threads

Moderne Betriebssysteme können mehrere Programme (Prozesse) parallel betreiben. Sofern nur *ein* Prozessor vorhanden ist, der den einzelnen Programmen bzw. Prozessen reihum vom Betriebssystem zur Verfügung gestellt wird, reduziert sich zwar die Ausführungs geschwindigkeit jedes Programms im Vergleich zum Solobetrieb, doch ist in den meisten Anwendungen ein flüssiges Arbeiten möglich. Als Ergänzung zum *Multitasking*, das ohne Zutun der Programmierer vom Betriebssystem bewerkstelligt wird, ist es oft sinnvoll oder gar unumgänglich, auch *innerhalb* einer Anwendung nebenläufige *Ausführungspfade* zu realisieren, wobei man hier dann vom *Multithreading* spricht. Mit Nebenläufigkeit bezeichnet man die Fähigkeit eines Systems, zwei oder mehrere Vorgänge gleichzeitig oder quasi-gleichzeitig ausführen zu können. So muss man beispielsweise bei einem Internet-Browser nicht untätig den quälend langsamen Aufbau einer Seite abwarten, sondern kann in einem anderen Browser-Fenster Suchbegriffe eingeben etc. Während jeder *Prozess* einen eigenen Adressraum besitzt, laufen die *Threads* eines Programms im selben Adressraum ab, so dass sie gelegentlich auch als *leichtgewichtige Prozesse* bezeichnet werden. Sie haben beispielsweise einen gemeinsamen Heap, wohingegen jeder Thread als selbständiger Kontrollfluss bzw. Ausführungspfad aber einen eigenen Stack benötigt.<sup>28</sup>

In Java ist die Multithread-Unterstützung in die Sprache (genauer: in das API) eingebaut und von jedem Entwickler ohne große Probleme zu nutzen. Ein Thread ist in Java als Objekt der gleichnamigen Klasse bzw. einer Unterklasse realisiert. Im folgenden Beispiel werden die Klassen ErzeugerThread und BenutzerThread aus der Basisklasse Thread abgeleitet. Sie sollen einen Produzenten und einen Konsumenten modellieren, die gemeinsam auf einen Lagerbestand einwirken, der von einem Objekt der Klasse Lager gehütet wird. Die main()-Methode beschränkt sich im Wesentlichen darauf, ein Stack-Objekt zu erzeugen. Im Konstruktor werden ein ErzeugerThread- und ein BenutzerThread-Objekt generiert:

```
ErzeugerThread pt = new ErzeugerThread(this);  
BenutzerThread kt = new BenutzerThread(this);
```

---

<sup>28</sup> Siehe auch Oechsle 2001.

Weil beide Threads mit dem Stack-Objekt kooperieren sollen, erhalten sie als Konstruktor-Parameter eine entsprechende Referenz über das Schlüsselwort `this`. Anschließend werden die beiden Threads vom Zustand `new` durch Aufruf ihrer `start()`-Methode in der Zustand `ready` gebracht:

```
pt.start();  
kt.start();
```

Von der `start()`-Methode eines Threads wird seine `run()`-Methode aufgerufen, welche die im Thread auszuführenden Anweisungen enthält. Eine aus Thread abgeleitete Klasse muss also vor allem die `run()`-Methode überschreiben, beispielsweise

```
public class ErzeugerThread extends Thread {  
    private Stack stack;  
    public ErzeugerThread(Stack tstack) {  
        super("Erzeuger");  
        stack = tstack;  
    }  
  
    public void run() {  
        while (stack.offen()) {  
            stack.add((int) (Math.random()*100));  
            try {  
                Thread.sleep((int) (1000 + Math.random()*3000));  
            } catch(InterruptedException ie) {  
                System.err.println(ie.toString());}  
        }  
    }  
}
```

Im Beispiel enthält die `run()`-Methode eine `while`-Schleife, die bis zum Eintreten einer Terminierungsbedingung läuft. Ein Thread im Zustand `ready` wartet auf die Zuteilung der (bzw. einer) CPU durch das Laufzeitsystem und erreicht dann den Zustand `running`. Die VJM arbeitet *preemptiv*, d. h. sie kann den Threads die Rechnerlaubnis jederzeit entziehen. Damit wechseln die Threads in Abhängigkeit von den Vergaberichtlinien des Laufzeitsystems zwischen den Zuständen `ready` und `running`. Sobald seine `run()`-Methode abgearbeitet ist, endet ein Thread. Er befindet sich dann im Zustand `dead` und kann nicht erneut gestartet werden. Im Beispiel ergänzt der `ErzeugerThread` innerhalb einer `while`-Schleife den Speicher (`Stack`) um eine zufallsbestimmte Menge. Er spricht über die per Konstruktor beschaffte Referenz das `Stack`-Objekt an und ruft dessen `add()`-Methode auf:

```
pl.add((int) (Math.random()*100));
```

Anschließend legt er sich durch Aufruf der statischen Thread-Methode sleep() ein zufallsabhängiges Weilchen zur Ruhe:

```
Thread.sleep((int) (1000 + Math.random()*3000));
```

Durch Ausführen dieser Methode wechselt der Thread vom Zustand running zum Zustand sleeping. Weil die Methode sleep() potentiell die obligatorische Ausnahme InterruptedException wirft, wenn ein anderer Thread oder der Benutzer den Schlafenden per Unterbrechungs-Aufforderung aufweckt, ist eine Ausnahmebehandlung erforderlich. Im Beispiel wird die Ausnahme auf der Standardfehlerausgabe protokolliert. Schläft ein Thread, während der Benutzer das Programm abbricht, erscheint dort beispielsweise:

```
java.lang.InterruptedException: sleep interrupted
```

Die Thread-Methode interrupt() wird oft dazu eingesetzt, einen per sleep() in den Schlaf oder per wait() in den Wartezustand geschickten Thread über das beschriebene Exception-Verfahren wieder aufzuwecken. In der Ausnahmebehandlung muss der Entwickler dann geeignet reagieren. Beim ErzeugerThread-Konstruktor ist noch zu beachten, dass im Aufruf des Superklassen-Konstruktors ein Thread-Name festgelegt wird.

Der Benutzer-Thread ist weitgehend analog definiert:

```
public class BenutzerThread extends Thread {  
    private Stack stack;  
  
    public BenutzerThread(Stack tstack) {  
        super("Benutzer");  
        stack = tstack;  
    }  
  
    public void run() {  
        while (stack.offen()) {  
            stack.get((int) (5 + Math.random()*100));  
            try {  
                Thread.sleep((int) (1000 + Math.random()*3000));  
            } catch(InterruptedException ie){  
                System.err.println(ie.toString());}  
        }  
    }  
}
```

In beiden run()-Methoden wird vor jedem Schleifendurchgang geprüft, ob der Stack noch aktiv, d. h. noch offen ist. Nach Deaktivierung (im Beispiel: nach 20 Arbeitsgängen) enden beide run()-Methoden und damit auch die Threads. Der automatisch zur Startklasse kreierte Thread main ist zu diesem Zeitpunkt ebenfalls bereits Geschichte, weil er mit der Stack-Methode main()

seine Tätigkeit einstellt, so dass die gesamte Anwendung endet. In den beiden Ausführungspfaden Erzeuger bzw. Benutzer, die von einem ErzeugerThread bzw. einem Benutzer-Thread-Objekt begründet werden, kommt das Stack-Objekt wesentlich zum Einsatz:

- In seiner Methode `offen()` entscheidet es auf Anfrage, ob weitere Veränderungen des Stack möglich sind.
- Die Methoden `add()` und `get()` erhöhen oder reduzieren den Zählerstand, aktualisieren die Anzahl der Werteveränderungen und protokollieren jede Maßnahme. Dazu bessern Sie sich mit der stationären Thread-Methode `currentThread()` eine Referenz auf den aktuell ausgeführten Thread und stellen per `getName()` dessen Namen fest.
- Mit Hilfe der privaten Stack-Methode `formZeit()` erhält das Ereignis-Protokoll bequem formatierte Zeitangaben.

In einem typischen Ablaufprotokoll des Programms zeigen sich einige Ungereimtheiten, verursacht durch das unkoordinierte Agieren der beiden Threads:

Der Stack ist offen (Stand = 100)

```
Nr. 1: Benutzer entnimmt 43 um 14:14:39 Uhr. Stand: 156
Nr. 2: Erzeuger ergänzt 99 um 14:14:39 Uhr. Stand: 156
Nr. 3: Erzeuger ergänzt 64 um 14:14:41 Uhr. Stand: 220
Nr. 4: Benutzer entnimmt 28 um 14:14:43 Uhr. Stand: 192
Nr. 5: Erzeuger ergänzt 26 um 14:14:43 Uhr. Stand: 218
Nr. 6: Benutzer entnimmt 35 um 14:14:45 Uhr. Stand: 183
Nr. 7: Erzeuger ergänzt 60 um 14:14:47 Uhr. Stand: 243
Nr. 8: Benutzer entnimmt 89 um 14:14:47 Uhr. Stand: 154
Nr. 9: Benutzer entnimmt 7 um 14:14:50 Uhr. Stand: 147
Nr. 10:Erzeuger ergänzt 50 um 14:14:50 Uhr. Stand: 197
Nr. 11:Benutzer entnimmt 68 um 14:14:53 Uhr. Stand: 129
Nr. 12:Erzeuger ergänzt 51 um 14:14:54 Uhr. Stand: 180
Nr. 13:Benutzer entnimmt 52 um 14:14:54 Uhr. Stand: 128
Nr. 14:Erzeuger ergänzt 84 um 14:14:55 Uhr. Stand: 212
Nr. 15:Benutzer entnimmt 46 um 14:14:56 Uhr. Stand: 166
Nr. 16:Benutzer entnimmt 79 um 14:14:58 Uhr. Stand: 87
Nr. 17:Erzeuger ergänzt 70 um 14:14:58 Uhr. Stand: 157
Nr. 18:Benutzer entnimmt 101 um 14:15:00 Uhr. Stand: 56
Nr. 19:Erzeuger ergänzt 86 um 14:15:02 Uhr. Stand: 142
Nr. 20:Benutzer entnimmt 23 um 14:15:04 Uhr. Stand: 119
Achtung Erzeuger, der Stack ist gesperrt!
Achtung Benutzer, der Stack ist gesperrt!
```

Offenbar muss verhindert werden, dass zwei Threads simultan auf den Stack zugreifen. Genau dies ist mit dem von Java unterstützten *Monitor*-Konzept leicht zu realisieren. Zu einem *Monitor* kann jedes Objekt werden, sobald eine seiner Methoden den Modifikator `synchronized` erhält. Sobald ein Thread eine als `synchronized` deklarierte Methode betritt,

wird er zum Besitzer dieses Monitors. Man kann sich auch vorstellen, dass er den (einzigen) Schlüssel zu den synchronisierten Bereichen des Monitors an sich nimmt. In der englischen Literatur wird der Vorgang als *obtaining the lock* beschrieben. Jedem anderen Thread wird der Zutritt zum Monitor verwehrt und er muss warten. Sobald der Monitor-Besitzer die synchronized-Methode beendet, kann ein wartender Thread den Monitor übernehmen und seine Arbeit fortsetzen. Insofern werden in dem Beispiel die Stack-Methoden `offen()`,

```

public synchronized boolean offen() {
    if (anzahl < MAXANZAHL)
        return true;
    else {
        System.out.println("\nAchtung
"+Thread.currentThread().getName()+" , der Stack ist ge-
sperrt!");
        return false;
    }
}
add()
public synchronized void add(int add) {
    summe += add;
    anzahl++;
    System.out.println("Nr.
"+anzahl+":\t"+Thread.currentThread().getName() +
    " ergänzt\t"+add+"\tum "+formZeit()+" Uhr. Stand:
"+summe);
}
und get()
public synchronized void get(int sub) {
    summe -= sub;
    anzahl++;
    System.out.println("Nr.
"+anzahl+":\t"+Thread.currentThread().getName() +
    " entnimmt\t"+sub+"\tum "+formZeit()+" Uhr. Stand:
"+summe);
}

```

als `synchronized` deklariert. Neben dem `synchronized`-Modifikator für Methoden bietet Java auch den `synchronized`-*Block*, wobei statt einer kompletten Methode nur eine einzelne Blockanweisung in den synchronisierten Bereich aufgenommen und ein beliebiges Objekt als Monitor angegeben wird. Nach der kleinen Modifikation des Programms zeigt sich die große Wirkung:

```
Der Stack ist offen (Stand = 100)
Nr. 1: Benutzer entnimmt 31 um 14:52:29 Uhr. Stand: 69
Nr. 2: Erzeuger ergänzt 77 um 14:52:29 Uhr. Stand: 146
Nr. 3: Erzeuger ergänzt 69 um 14:52:31 Uhr. Stand: 215
Nr. 4: Benutzer entnimmt 78 um 14:52:32 Uhr. Stand: 137
Nr. 5: Erzeuger ergänzt 76 um 14:52:33 Uhr. Stand: 213
Nr. 6: Benutzer entnimmt 69 um 14:52:34 Uhr. Stand: 144
Nr. 7: Erzeuger ergänzt 62 um 14:52:37 Uhr. Stand: 206
Nr. 8: Benutzer entnimmt 101 um 14:52:38 Uhr. Stand: 105
Nr. 9: Erzeuger ergänzt 32 um 14:52:40 Uhr. Stand: 137
Nr. 10: Benutzer entnimmt 22 um 14:52:41 Uhr. Stand: 115
Nr. 11: Benutzer entnimmt 70 um 14:52:43 Uhr. Stand: 45
Nr. 12: Erzeuger ergänzt 32 um 14:52:44 Uhr. Stand: 77
Nr. 13: Benutzer entnimmt 82 um 14:52:46 Uhr. Stand: -5
Nr. 14: Erzeuger ergänzt 86 um 14:52:46 Uhr. Stand: 81
Nr. 15: Benutzer entnimmt 93 um 14:52:48 Uhr. Stand: -12
Nr. 16: Erzeuger ergänzt 72 um 14:52:48 Uhr. Stand: 60
Nr. 17: Benutzer entnimmt 20 um 14:52:49 Uhr. Stand: 40
Nr. 18: Erzeuger ergänzt 12 um 14:52:51 Uhr. Stand: 52
Nr. 19: Benutzer entnimmt 78 um 14:52:52 Uhr. Stand: -26
Nr. 20: Erzeuger ergänzt 13 um 14:52:54 Uhr. Stand: -13
Achtung Benutzer, der Stack ist gesperrt!
Achtung Erzeuger, der Stack ist gesperrt!
```

Mit Hilfe der Object-Methoden `wait()` und `notify()` können negative Summen verhindert werden: Trifft eine Benutzer-Anfrage auf einen unzureichenden Bestand, wird der Thread mit der Methode `wait()` in den Zustand `waiting` versetzt. Die Methode `wait()` ist bereits in der Klasse `Object` definiert und kann nur in einem Monitor, also im synchronisierten Bereich, aufgerufen werden, beispielsweise:

```
public synchronized void get(int sub) {
    while (summe < sub)
        try {
            System.out.println(Thread.currentThread().getName() +
                " muß warten: Keine "+sub+" Einheiten vorhanden.");
            wait();
        } catch (InterruptedException ie) {
            System.err.println(ie.toString());
        }
    summe -= sub;
    anzahl++;
    System.out.println("Nr.
"+anzahl+":\t"+Thread.currentThread().getName()+
    " entnimmt\t"+sub+"\tum "+formZeit()+" Uhr. Stand:
"+summe);
}
```

Mit den Object-Methoden `notify()` bzw. `notifyAll()` kann ein Thread aus dem Zustand `waiting` in den Zustand `ready` versetzt werden:

- `notify()` versetzt den Thread mit der längsten Wartezeit in den Zustand `ready`.
- `notifyAll()` versetzt alle wartenden Threads in den Zustand `ready`.

```
Der Stack ist offen (Stand = 100)
Nr. 1: Benutzer entnimmt 68 um 15:25:50 Uhr. Stand: 32
Nr. 2: Erzeuger ergänzt 24 um 15:25:50 Uhr. Stand: 56
Nr. 3: Benutzer entnimmt 21 um 15:25:52 Uhr. Stand: 35
Nr. 4: Erzeuger ergänzt 36 um 15:25:52 Uhr. Stand: 71
Nr. 5: Erzeuger ergänzt 63 um 15:25:54 Uhr. Stand: 134
Nr. 6: Benutzer entnimmt 86 um 15:25:55 Uhr. Stand: 48
Nr. 7: Erzeuger ergänzt 95 um 15:25:58 Uhr. Stand: 143
Nr. 8: Benutzer entnimmt 78 um 15:25:58 Uhr. Stand: 65
Nr. 9: Erzeuger ergänzt 5 um 15:26:00 Uhr. Stand: 70
Nr. 10: Benutzer entnimmt 19 um 15:26:00 Uhr. Stand: 51
Benutzer muß warten: Keine 74 Einheiten vorhanden.
Nr. 11: Erzeuger ergänzt 11 um 15:26:03 Uhr. Stand: 62
Nr. 12: Erzeuger ergänzt 71 um 15:26:06 Uhr. Stand: 133
Nr. 13: Erzeuger ergänzt 35 um 15:26:08 Uhr. Stand: 168
Nr. 14: Erzeuger ergänzt 36 um 15:26:12 Uhr. Stand: 204
Nr. 15: Erzeuger ergänzt 36 um 15:26:14 Uhr. Stand: 240
Nr. 16: Erzeuger ergänzt 50 um 15:26:17 Uhr. Stand: 290
Nr. 17: Erzeuger ergänzt 5 um 15:26:20 Uhr. Stand: 295
Nr. 18: Erzeuger ergänzt 16 um 15:26:23 Uhr. Stand: 311
Nr. 19: Erzeuger ergänzt 35 um 15:26:25 Uhr. Stand: 346
Nr. 20: Erzeuger ergänzt 81 um 15:26:28 Uhr. Stand: 427
Achtung Erzeuger, der Stack ist gesperrt!
```

Wie `wait()` können auch `notify()` und `notifyAll()` nur in einem synchronisierten Bereich aufgerufen werden, beispielsweise:

```
public synchronized void add(int add) {
    summe += add;
    anzahl++;
    System.out.println("Nr." + anzahl + ":" + t
        "+Thread.currentThread().getName() +
        "ergänzt\t" + add + "\tum " + formZeit() + " Uhr. Stand:
        "+summe);
    notify();
}
```

Zum Unterbrechen und Reaktivieren eines Threads sollten an Stelle der Thread-Methoden `suspend()` und `resume()`, die als missbilligt (deprecated) eingestuft sind, die Object-Methoden `wait()` und `notify()` eingesetzt werden. Zum Stoppen eines Threads sollte man nicht mehr die unerwünschte Thread-Methode `stop()` verwenden, sondern ein behutsames, kommunikatives Prozedere anwenden:

- Mit der Thread-Methode `interrupt()` wird signalisiert, dass ein Thread seine Tätigkeit einstellen soll. Der betroffene Thread wird nicht abgebrochen, sondern sein Interrupt-Signal wird auf den Wert `true` gesetzt.
- Ein Thread, der mit einem Interrupt-Signal rechnen muss, prüft in seiner `run()`-Methode regelmäßig durch Aufruf der Thread-Methode `isInterrupted()`, ob er sein Wirken einstellen soll. Falls ja, verlässt er einfach die `run()`-Methode und erreicht damit den Zustand `dead`.

Als Beispiel soll das bisherige Stack-Programm dahingehend erweitert werden, indem die Verwaltungsinstanz des Stacks den Benutzer-Thread zum Terminieren auffordert, sobald dieser mit einer Transaktion in den negativen Wertebereich gelangen würde.

```
public void get(int sub) {  
    if (summe - sub < 0)  
        Thread.currentThread().interrupt();  
    else {  
        summe -= sub;  
        anzahl++;  
        System.out.println("Nr. " + anzahl + " entnimmt " +  
            sub + " \tum " + formZeit() + " Uhr. Stand: " + summe);  
    }  
}
```

Trifft der Benutzer-Thread auf ein Interrupt-Signal, stellt er seine Tätigkeit ein:

```
public void run() {  
    while (stack.offen()) {  
        if (isInterrupted()) {  
            System.out.println("Der Benutzer-Thread wurde gestoppt!");  
            return;  
        }  
        stack.get((int) (5+Math.random()*100));  
        int limit = (int) (1000 + Math.random()*3000);  
        long time = System.currentTimeMillis();  
        while (System.currentTimeMillis() - time < limit);  
    }  
}
```

In dieser `run()`-Methode wird bewusst ein Aufruf von `Thread.sleep()` vermieden, weil ein `interrupt()`-Aufruf in der Schlafenszeit zu einer `InterruptedException`-Exception führt. In dem Anwendungsszenario muss der Benutzer frühzeitig aussteigen:

```
Der Stack ist offen (Stand = 100)
Nr. 2: Erzeuger ergänzt 2 um 15:53:19 Uhr. Stand: 33
Nr. 2: Benutzer entnimmt 69 um 15:53:19 Uhr. Stand: 33
Nr. 3: Erzeuger ergänzt 57 um 15:53:22 Uhr. Stand: 90
Der Benutzer-Thread wurde gestopp!
Nr. 4: Erzeuger ergänzt 56 um 15:53:25 Uhr. Stand: 146
Nr. 5: Erzeuger ergänzt 42 um 15:53:27 Uhr. Stand: 188
Nr. 6: Erzeuger ergänzt 1 um 15:53:29 Uhr. Stand: 189
Nr. 7: Erzeuger ergänzt 80 um 15:53:32 Uhr. Stand: 269
Nr. 8: Erzeuger ergänzt 30 um 15:53:34 Uhr. Stand: 299
Nr. 9: Erzeuger ergänzt 10 um 15:53:37 Uhr. Stand: 309
Nr. 10:Erzeuger ergänzt 24 um 15:53:38 Uhr. Stand: 333
Nr. 11:Erzeuger ergänzt 23 um 15:53:40 Uhr. Stand: 356
Nr. 12:Erzeuger ergänzt 75 um 15:53:42 Uhr. Stand: 431
Nr. 13:Erzeuger ergänzt 54 um 15:53:43 Uhr. Stand: 485
Nr. 14:Erzeuger ergänzt 61 um 15:53:46 Uhr. Stand: 546
Nr. 15:Erzeuger ergänzt 95 um 15:53:48 Uhr. Stand: 641
Nr. 16:Erzeuger ergänzt 44 um 15:53:51 Uhr. Stand: 685
Nr. 17:Erzeuger ergänzt 58 um 15:53:52 Uhr. Stand: 743
Nr. 18:Erzeuger ergänzt 95 um 15:53:53 Uhr. Stand: 838
Nr. 19:Erzeuger ergänzt 41 um 15:53:55 Uhr. Stand: 879
Nr. 20:Erzeuger ergänzt 1 um 15:53:58 Uhr. Stand: 880
Achtung Erzeuger, der Stack ist gesperrt!
```

Angewandt auf einen per `wait()` in den Wartezustand versetzten oder per `sleep()` ruhig gestellten Thread hat `interrupt()` folgende Effekte:

- Der Thread wird sofort in den Zustand ready versetzt, auch wenn die `sleep()`-Zeit noch nicht abgelaufen ist.
- Es wird eine `InterruptedException` geworfen, die vom `wait()`- bzw. `sleep()`-Aufrufer abgefangen werden muss.
- Das Interrupt-Signal wird ggf. *aufgehoben* (auf `false` gesetzt).

Es kann daher sinnvoll sein, `interrupt()` in der `catch`-Klausel der `InterruptedException`-Behandlung erneut aufzurufen, um das Interrupt-Signal wieder auf `true` zu setzen, damit die `run()`-Methode bei nächster Gelegenheit passend reagiert. Im Beispiel findet die `InterruptedException`-Behandlung allerdings in der `run()`-Methode statt, so dass es sich anbietet, durch eine `return`-Anweisung in der `catch`-Klausel die `run()`-Methode und damit den Thread sofort zu beenden.

**Tab. 8.10** Thread-Prioritäten

Konstante in der Klasse Thread	Wert
Thread.MAX_PRIORITY	10
Thread.NORM_PRIORITY	5
Thread.MIN_PRIORITY	1

```

public void run() {
    while (stack.offen()) {
        if (isInterrupted()) {
            System.out.println("Der Benutzer-Thread wurde
gestoppt!");
            return;
        }
        stack.get((int) (5 + Math.random()*100));
        try {
            Thread.sleep((int) (1000 + Math.random()*3000));
        } catch(InterruptedException ie) {
            //interrupt();
            return;
        }
    }
}

```

Den Bestandteil der virtuellen Maschine, der die verfügbare Rechenzeit auf die arbeitswilligen und -fähigen Threads (Zustand ready) verteilt, bezeichnet man als *Scheduler*. Er orientiert sich u. a. an den *Prioritäten* der Threads, die in Java Werte von 1 bis 10 annehmen können (Tab. 8.10):

Es hängt allerdings zum Teil von der Plattform ab, wie viele Prioritätsstufen wirklich unterschieden werden. Der in einer Java-Anwendung automatisch gestartete Thread main hat beispielsweise die Priorität 5, was man unter Windows in einer Java-Konsolenanwendung über die Tastenkombination <Strg>+<Pause> in Erfahrung bringen kann:

```
"main"    prio    =  5    tid=0x00034D28    nid=0x1464    runnable
[7f000..7fc3c]
```

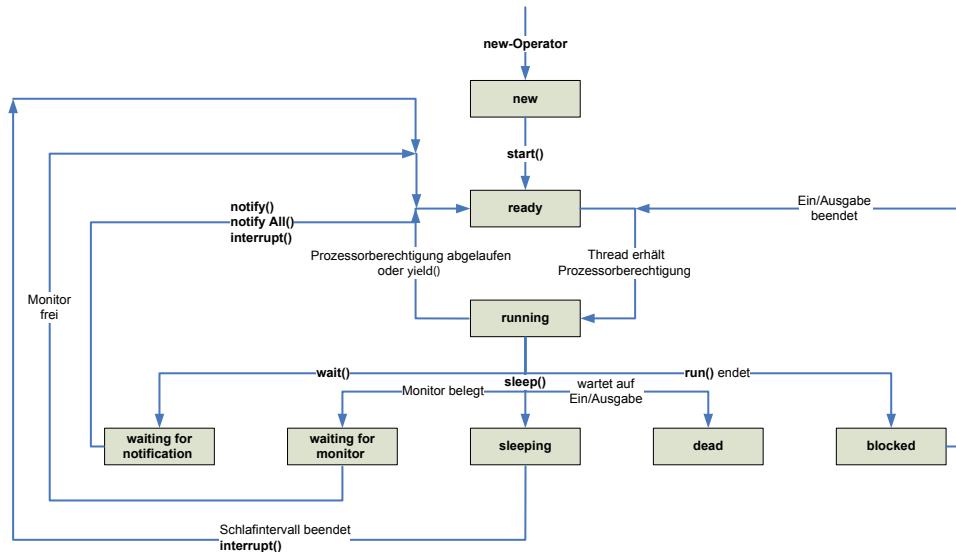
Ein Thread (beispielsweise main) überträgt seine aktuelle Priorität auf die bei seiner Ausführung gestarteten Threads:

```
"Benutzer" prio=5 tid=0x00A33DF0 nid=0x220 waiting on condi-
tion [ac9f000..ac9fd88]
...
"Erzeuger" prio=5 tid=0x00A33C98 nid=0x136c waiting on con-
dition [ac5f000..ac5fd88]
```

Mit den Thread-Methoden `getPriority()` bzw. `setPriority()` lässt sich die Priorität eines Threads feststellen bzw. ändern. In der Spezifikation für die virtuelle Java-Maschine wird das Verhalten des Schedulers bei der Rechenzeit-Vergabe an die Threads nicht sehr präzise beschrieben. Er muß lediglich sicherstellen, dass die einem Thread zugeteilte Rechenzeit mit der Priorität ansteigt. In der Regel läuft der Thread mit der höchsten Priorität, jedoch kann der Scheduler Ausnahmen von dieser Regel machen, beispielsweise um das *Verhungern* (engl. *starvation*) eines anderen Threads zu verhindern, der permanent auf Konkurrenten mit höherer Priorität trifft. Daher darf keinesfalls der korrekte Ablauf eines Programms davon abhängig sein, dass sich die Rechenzeit-Vergabe an Threads in einem strengen Sinn an den Prioritäten orientiert. Leider zeigt das Verhalten des Schedulers bei Threads gleicher Priorität nicht unerhebliche Unterschiede zwischen den JVMs, so dass diesbezüglich *keine vollständige Plattformunabhängigkeit* besteht. Auf einigen Plattformen, wie beispielsweise unter MS-Windows, benutzt die JVM das von modernen Betriebssystemen zur Prozesserverwaltung verwendete *preemptive Zeitscheibenverfahren* für die Thread-Verwaltung:

- Die Threads gleicher Priorität werden reihum (*Round-Robin*) jeweils für eine festgelegte Zeitspanne ausgeführt.
- Ist die Zeitscheibe eines Threads verbraucht, wird er vom Scheduler in den Zustand `ready` versetzt, und der Nachfolger erhält Zugang zum Prozessor.

Auf anderen Plattformen, beispielsweise Solaris, benutzt die JVM ein rein prioritätenbasiertes Multithreading, so dass ein Thread in der Regel nur von Konkurrenten mit höherer Priorität verdrängt werden kann. Diesen Aspekt gilt es bei der Entwicklung von Softwareprogrammen für Robotersysteme unbedingt zu beachten, damit es bei der späteren Ausführung in Echtzeit nicht zu unangenehmen Überraschungen kommt. Freundliche Threads können aber auch ohne Round-Robin-Unterstützung der JVM den gleichrangigen Kollegen eine Chance geben, indem sie bei passender Gelegenheit die Thread-Methode `yield()` aufrufen, um den Prozessor abzugeben und sich wieder in die Warteschlange der rechenwilligen Threads einzureihen. Mit Hilfe dieser Methode können also die Plattformunterschiede in der Thread-Verwaltung kompensiert werden. Threads können als zustandsbasierte Automaten aufgefasst werden, um damit die wichtigen Thread-Zustände und Anlässe für Zustandsübergänge transparent zu machen. Neben den eigenentwickelten Threads kennt Java noch so genannte Daemon-Threads, die im Hintergrund zur Unterstützung anderer Threads tätig sind und dabei nur aktiv werden, wenn ungenutzte Rechenzeit vorhanden ist. Der Garbage Collector ist ein typisches Beispiel für sich einen Daemon-Thread. Mit der Thread-Methode `setDaemon()` lässt sich auch ein Benutzer-Thread dämonisieren, was allerdings vor dem Aufruf seiner `start()`-Methode geschehen muss. Um das Terminieren von Daemon-Threads braucht man sich in der Regel nicht zu kümmern, denn ein Java-Programm oder Applet endet, sobald ausschließlich Daemon-Threads vorhanden sind (Abb. 8.25).



**Abb. 8.25** Zustände und Zustandsübergänge von Threads

Gerade bei der Entwicklung von Programmen zur Ansteuerung von Robotersystemen unter Verwendung von Monitoren zur Thread-Synchronisation ist sicher zustellen, dass es nicht zu einer sogenannten *Deadlock*-Situationen kommt.

Im Falle eines Deadlocks blockieren sich Threads gegenseitig.

```

class DeadLockOne {

    synchronized static void monitor() {
        System.out.println(Thread.currentThread().getName() + " in
DeadLockOne.Monitor");
        try {
            Thread.sleep(100);
        } catch (InterruptedException e) {}
        System.out.println(Thread.currentThread().getName() +
        " wartet darauf, daß DeadLockTwo.Monitor frei wird . . .");
        DeadLockTwo.monitor();
    }
    public static void main(String[] args) {
        ThreadOne t1 = new ThreadOne();
        ThreadTwo t2 = new ThreadTwo();
        t1.start();
        t2.start();
    }
}
  
```

Im folgenden Beispiel begeben sich ThreadOne und ThreadTwo jeweils in einen Monitor, der durch das Synchronisieren von Klassenmethoden entsteht, so dass man der jeweiligen Klasse (DeadLockOne bzw. DeadLockTwo) die Rolle des Aufpassers zuschreiben kann:

```
class DeadLockTwo {  
    synchronized static void monitor() {  
        System.out.println(Thread.currentThread().getName()+" in  
DeadLockTwo.Monitor");  
        try {  
            Thread.sleep(100);  
        } catch (InterruptedException e) {}  
        System.out.println(Thread.currentThread().getName()+"  
        \" wartet darauf, daß DeadLockOne.Monitor frei wird...\"");  
        DeadLockOne.monitor();  
    }  
}
```

Nach einem kurzen Schläfchen versuchen die beiden Threads, eine Methode aus dem jeweils anderen (blockierten) Monitor aufzurufen.

```
class ThreadOne extends Thread {  
    ThreadOne() {super("ThreadOne");}  
    public void run() {  
        DeadLockOne.monitor();  
    }  
}
```

Entsprechend gestaltet sich der Aufbau von ThreadTwo:

```
package de.ifpe.java.deadlock;  
class ThreadTwo extends Thread {  
    ThreadTwo() {super("ThreadTwo");}  
    public void run() {  
        DeadLockTwo.monitor();  
    }  
}
```

Weil kein Thread seinen Monitor frei gibt, bevor er den anderen betreten darf, kommt es zur Blockade, und das Programm hängt fest:

```
ThreadOne in DeadLockOne.Monitor  
ThreadTwo in DeadLockTwo.Monitor  
ThreadOne wartet darauf, daß DeadLockTwo.Monitor frei wird.  
...  
ThreadTwo wartet darauf, daß DeadLockOne.Monitor frei wird.  
...
```

Insgesamt ist ein Thread ein selbständiger Ausführungspfad (Kontrollfluss) innerhalb eines Java-Programms. Alle Threads eines Programms laufen im selben Adressraum. Sie teilen sich den Heap (mit den Objekten des Programms), haben jedoch jeweils einen eigenen Stack. Bei der Vergabe von Rechenzeit an die Threads berücksichtigt der Scheduler der JVM deren Prioritäten und wendet meist für Threads gleicher Priorität ein Zeitscheibenverfahren an.

Zum Erzeugen eines Threads bietet Java zwei Optionen:

- Eine eigene Klasse aus `java.lang.Thread` ableiten und ein Objekt dieser Klasse erzeugen
- In einer eigenen Klasse das Interface `java.lang.Runnable` implementieren und ein Objekt dieser Klasse an einen Thread-Konstruktor übergeben

Über *Monitore* kann man verhindern, dass ein Code-Bereich von mehreren Threads parallel durchlaufen wird:

- Sobald ein Thread eine als `synchronized` definierte Instanzmethode für ein Objekt aufruft, ignoriert dieses Objekt jeden Aufruf einer `synchronized`-Instanzmethode durch einen anderen Thread, bis der erste Thread den Monitor verlassen hat.
- Sobald ein Thread eine als `synchronized` definierte statische Methode einer Klasse aufruft, sind alle statischen `synchronized`-Methoden der Klasse für alle anderen Threads gesperrt, bis der erste Thread den Monitor verlassen hat.
- Sobald ein Thread einen `synchronized`-Block betreten hat, der durch ein Lock-Objekt geschützt wird, sind *alle* von diesem Objekt geschützten `synchronized`-Blöcke für andere Threads gesperrt, bis der erste Thread das Lock-Objekt frei gibt.
- Ein Lock-Objekt kann gleichzeitig synchronisierte Instanzmethoden und synchronisierte Blöcke überwachen.

Die wichtigsten Methoden im Zusammenhang mit Threads sind:

---

java.lang.Thread	<i>currentThread()</i>
	Liefert eine Referenz auf das aktuelle Thread-Objekt
	<i>getName()</i>
	Liefert den Namen des jeweiligen Threads
	<i>getPriority()</i>
	Ermittelt die Priorität des jeweiligen Threads
	<i>interrupt()</i>
	Ein via wait() deaktivierter oder via sleep() ruhig gestellter Thread wird durch eine InterruptedException wieder aufgeweckt. Für andere Threads wird das Interrupted-Signal auf true gesetzt
	<i>isAlive()</i>
	Test, ob der Thread noch am Leben ist, d. h. gestartet wurde und noch nicht beendet ist
	<i>Run()</i>
	Kapselt die vom Thread auszuführenden Anweisungen
	<i>setPriority()</i>
	Setzt die Priorität des jeweiligen Threads
	<i>sleep()</i>
	Legt den Thread eine bestimmte Zeit schlafen
	<i>start()</i>
	Initiiert den Thread und ruft seine run()-Methode auf
	<i>yield()</i>
	Gibt Rechenzeit an Threads gleicher Priorität ab.
java.lang.Object	<i>notify()</i>
	Von den auf einen Monitor wartenden Threads wird derjenige mit der längsten Wartezeit in den Zustand ready versetzt
	<i>notifyAll()</i>
	Alle auf einen Monitor wartenden Threads werden in den Zustand ready gesetzt
	<i>wait()</i>
	Der aktuelle Thread wird in den Wartezustand versetzt

---

## Literatur

- Abts, D.: Aufbaukurs Java. vieweg, Braunschweig (2003)
- Aho, A.V., Sethi, R., Ullman, J.D.: Compilers. Addison-Wesley, Reading (1986) (deutsch: Compilerbau, 2. Aufl. Oldenbourg, München (1999))
- Alexander, C. et al.: A Pattern Language. Oxford University Press, New York (1998)
- Arnold, K., Gosling, I. et al.: The Java Programming Language, 3rd ed. Addison-Wesley, Boston (2000) (deutsch: Die Programmiersprache Java. Addison-Wesley, München (2001))
- Bauer, F.L., Wirsing, M.: Elementare Aussagenlogik. Springer, Berlin (1991)

- Bengel, G.: Verteilte Systeme, 2. Aufl. Vieweg, Braunschweig (2002)
- Bittner, U., Hesse, W., Schnath, J.: Praxis der Software-Entwicklung, Methoden, Werkzeuge, Projektmanagement – eine Bestandsaufnahme. Oldenbourg, München (1995)
- Booch, G., Rumbaugh, J., Jacobson I.: Unified Modeling Language User Guide. Addison Wesley Longman, Reading (1999)
- Bunke, H., Mey, H. (Hrsg.): Künstliche Intelligenz und Expertensysteme. Informatik und Nachrichtentechnik Bd. 8. Bern (1987)
- Chomsky, N.: Aspects of the Theory of Syntax. M.I.T. Press, Cambridge (1965)
- Corner, D.E.: Computer Networks and Internets, 3rd ed. Prentice Hall, Upper Saddle River (2000)
- Davenport, T.H.: Process Innovation: Reengineering Work through Information Technology. Harvard Business School Press, Boston (1993)
- Harrington, H.J.: Business Process Improvement: The Breakthrough Strategy for Total Quality, Productivity, and Competitiveness. McGraw-Hill, New York (1991)
- Hitz, M., Kappel, G.: Von der Analyse zur Realisierung, 2. Aufl. dpunkt.verlag, Heidelberg (2003)
- Jacobson, I., Ericson, M., Jacobson, A.: The Object Advantage: Business Process Reengineering with Object Technology. Addison-Wesley (1999a)
- Jacobson, I., Booch, G., Rumbaugh, J.: Unified Software Development Process. Addison Wesley Longman, (1999b)
- Jobst, F.: Einführung in Java, 2. Aufl. Fachbuchverlag Leipzig (2001)
- Middendorf, S., Singer, R., Strobel, S.: Java: Programmierhandbuch und Referenz. d-punkt- Verlag, Heidelberg (1996)
- Nof, Sh. Y. (Hrsg.): Handbook of industrial robotics. Wiley, Chichester (1999)
- Oechsle, R.: Parallele Programmierung mit Java Threads. Fachbuchverlag Leipzig (2001)
- Oestereich, B.: Objektorientierte Softwareentwicklung: Analyse und Design mit der UML, 5. Aufl. Oldenbourg Wissenschaftsverlag, München (2001)
- Österle, H.: Business Engineering, Prozess- und Systementwicklung, Bd. 1, Entwurfstechniken. Springer-Verlag, Berlin (1995)
- Rumbaugh, J., Jacobson, I., Booch, G.: Unified Modeling Language Reference Manual. Addison Wesley Longman, Reading (1999)
- Rupp, C. et al.: Requirements Engineering. Hanser, München (2001)
- Schader, M., Schmidt-Thieme, L.: Java – Eine Einführung, 3. Aufl. Springer, Berlin (2000)
- Scheer, A.-W.: ARIS – Modellierungsmethoden. Metamodelle, Anwendungen. 4. Aufl. Springer-Verlag, Berlin, (2001)
- Sedgewick, R. Algorithms in Java, 3. Aufl. Addison-Wesley, Boston (2003)
- Seidimeier, H.: Prozessmodellierung mit ARIS. Eine beispielorientierte Einführung für Studium und Praxis. Vieweg, (2002)
- Steinmetz, R.: Multimedia-Technologie, 3. Aufl. Springer, Berlin (2000)
- Summerville, I., Sayer, P.: Requirements Engineering, A good practice guide. Wiley, Chichester (1997)
- Tanenbaum, A.S., Stehen, M. van: Distributed Systems. Prentice Hall, Upper Saddle River (2002) (deutsch: Verteilte Systeme. Pearson Studium, München (2003))Literatur

---

# Sachverzeichnis

## A

Ablaufsteuerung 25  
Abstandssensor 422  
Abstraktion 58, 298  
Achsregelung 222  
Ackermannlenkung 424  
Adaption 287  
    proaktive 290  
Adäquatheit 167  
Agenten 331  
    interoperative 361  
Agenten-Architektur 195  
Agentensystem 198  
Ähnlichkeitsmodell 79  
Aktionsschritte, interoperative 116  
Aktionssteuerung 25  
Aktivierung 374  
Aktivierungsarten 388  
Aktivierungsfunktion 378  
Aktivitäten 116, 143  
    automatisierte 78  
Aktivitätsdiagramm 527  
Aktoren 28, 80, 147, 186, 188, 192, 231, 406,  
    414, 423  
    hydraulische 233  
    pneumatische 232  
ALGOL 250  
Algorithmen 94, 456, 535  
    deterministische 536  
    evolutionäre 272, 352, 354  
    genetische 293, 351, 354  
    nebenläufige 537  
    nichtdeterministische 537  
    nichtterminierende 537  
    rekursive 351  
    sequentielle 536

symbolische 193  
terminierende 536  
Allgemeinwissen 101  
Analogiebildung 121  
Analogiemodell 86  
Android 15  
Anforderungsanalyse 502  
Anforderungsliste 502  
Angleichung 59  
Animat 14  
Annotation, kognitive 218  
Antrieb 23, 222  
    elektrischer 222  
    elektromotorischer 85  
    hydraulischer 222  
    pneumatischer 222  
Antriebsart  
    elektrische 23  
    hydraulische 23  
    pneumatische 23  
Anweisungen 6, 26, 248, 540  
    rechnerinterne 241  
    roboterorientierte 480  
Anwendungsfall 161, 526  
Anwendungsgebiet 40, 328  
Architektur 28, 185, 436  
    hybride 369  
    intrinsische topologische 385  
    reaktiv-reflexive 426  
    symbolisch orientierte 185  
    verhaltensbasierte 185, 427  
Architekturmodell 196  
Architekturmuster 433, 434  
Archivierungswerkzeug 577  
Attraktoren 106  
Attribut-Wert-Paare 516

- Attribute 515  
eines Systems 53
- Attributklassen 53
- Aufbau eines klassischen Robotersystems 21
- Ausführungsbedingungen 77
- Automat, mechanischer 1
- Automatenbau 2
- Automatismen 119
- Autonomer Raum Explorer (ARE) 405
- B**
- Backpropagation  
Netz 396  
Regel 393
- Backtracking 317
- Bausätze 406
- BDI-Architektur 367
- Bedingungen 114  
situative 151
- Begriffsbildung 460
- Beobachtung 117
- Beratungssystem 336
- Berechnungsmodell 92
- Berührungssensor 412
- Best-First-Suche 318
- Bewegungssensoren 230
- Bewegungssteuerung 25
- Bewusstsein, artifizielles 199, 216
- Bifurkation 107
- Bildmodell, ikonisch-grafisches 61
- Bildsensoren 231
- Bioroboter 14
- Bitstring-Kodierung 358
- Black-Box 187, 196
- Blueprint-Kodierung 359
- Brainware 27, 188, 271, 426
- Building Block Hypothese 353
- C**
- Casting-Operator 551
- Chunkboard-Muster 434
- Client-Server-Prinzip 157, 268
- Cluster 513
- Code, genetischer 356
- Cognitive  
Computing 27, 149, 199, 211, 282, 332, 465,  
479
- Robotic Plattform 520
- Computer Aided Software Engineering  
(CASE) 158
- Computermodell 62
- Constraintpropagierung 310, 312
- Constraints 350
- Crossover 353  
Operatoren 358
- Crosstalk 228
- D**
- Darstellungsmodell, graphisches 60
- Daten 33, 52, 95, 96
- Datenflussdiagramm 156
- Datenflussmodell 241
- Deadlock-Situation 589
- Default-Anweisung 556
- Definition, operationale 39
- Delegation 16
- Delta-Regel 393  
generalisierte 397
- Dendriten 379
- Denken  
artifizielles 444  
erfassendes 207  
planendes 217
- Depth of cognitive processing 148
- Diagnosesystem 335
- Differentialantrieb 423
- Diskurs-Plattform 479
- Dispositionswissen 99
- Dokumentation 178
- Dokumentationskommentar 542
- Double-Loop-Lernen 287
- Drei-Schichten-Architektur 268
- Dreiradantrieb 424
- Dualismus 274
- Durchführungsplanung 127
- E**
- Eclipse 520
- Effektor 24, 219
- Effizienz 57, 168
- Eingabe-Verarbeitung-Ausgabe-Architektur  
187
- Einsichtshypothese 71
- Einstellverfahren 236
- Elektromotor 231
- Elite 357
- Emergenz, artifizielle 481
- Emotion 209  
artifizielle 210, 445
- Emotionalität 445

- Empfänger 52  
Entity-Relationship-Modell 156, 267  
Entscheidungsalgorithmen 72  
Entscheidungsarchitektur 480  
Entscheidungsmodell 88  
Entwicklungshandbuch 505  
Entwicklungsprojekt 155, 164, 176, 499  
Entwicklungsprozess 60, 157, 162, 171, 197, 499  
Entwurfsmuster 271, 433  
Entwurfssystem 336  
Erfahrung 457  
Erfahrungswissen 95, 98  
Erfolgswahrscheinlichkeit 125  
Erinnerungen 210  
Erkenntnis 55  
Erkenntnisgewinnung 472  
Erkenntnisstruktur 485  
Erkenntnistheorie 206  
  evolutionäre 56  
Erklärungskomponente 344  
Ersetzungsstrategie 357  
Erwartung-mal-Wert-Modell 447  
Evolutionsstrategie 351  
Expertensystem 278, 333, 366  
Exterozeption 443  
Extreme Programming 163
- F**  
Fachwissen 101  
Faktenwissen 120  
Feed-forward-Netzwerk 385  
Feedback-Netz 386  
Finite-Element-Methode 84  
Firmware 408  
Fixpunktattraktoren 107  
Folgeprogrammierung 236  
Formalisierung 89, 277  
FORTRAN 250  
Frame Problem 325  
Frames 517  
Freiheitsgrade 22  
Freizeitroboter 12  
Funktionen 24, 36  
  heuristische 315, 320  
  kognitive 208  
  kommunikative 208  
  logische 87  
  referentielle 208  
  sigmoide 382
- vegetative 140  
Fuzzy-Logik 345, 430
- G**  
Gebrauchswissen, nicht-propositionales 98  
Gedächtnis  
  autobiographisches 213  
  episodisches 213  
  prozedurales 213  
  semantisches 213  
Gedächtnishypothese 71  
Gedächtnisschemata 70  
Gedächtnissystem 211, 452, 453  
Gehirn, menschliches 139  
Gehmaschine 19, 20  
Genetic Repair-Effekt 353  
Gestaltungsmodell 88  
Glass box 93  
Gliazellen 142  
Glieder 514  
Graphentheorie 385, 477
- H**  
Hall-Sensor 229  
Handeln  
  intelligentes 150  
  konkretes 78  
  sensorgestütztes 480  
  zielgerichtetes 368  
Handlungsziele 77  
Hardware 221  
Hebb'sche Lernregel 392  
Heuristik 314  
  des nächsten Nachbarn 315  
Hill Climbing 316  
Hirnforschung 212  
Historie 279  
Holographie 61  
Humanoid 9, 13  
Hypothesen 33
- I**  
Implementierung 170, 499  
Industrieroboter 9, 83, 233  
Inferenz  
  analogiebasierte 218  
  deduktive 218  
  induktive 218  
Inferenzkomponente 342  
Inferenzprozess 121

- Inferenzverfahren 278  
 Informatiksystem, artifizielles 62  
 Informationen 97, 189  
 Informationsreduktion 207  
 Informationsverarbeitung 102, 283  
 Infrarot-Schnittstelle 233, 407, 409  
 Infrarotsensoren 227  
 Inhaltsmodell 446  
 Initialisierung 173  
 Innovation 287  
     punktuelle 290  
 Integrationsmodell 196  
 Intelligenz  
     artifizielle 16, 27, 199, 271, 273  
     systemische 202, 282, 461  
 Intelligenzkriterien 286  
 Intelligenzprofil 30, 203  
 Intelligenzquotient 108, 203, 283  
     systemischer 30  
 Intention, systemische 123  
 Intentional Programming 240  
 Intentionalität 217  
 Interaktion 48, 49, 103, 209  
     extrinische 51  
     intrinsische 51  
 Interaktionstheorie 483  
 Interoperation 51, 103, 109, 287  
     aktive 291  
     passive 291  
 Interoperationsanalyse 117, 126, 504  
 Interoperationsmanagement 364  
 Interoperationsplanung 127  
 Interoperationstheorie 78, 104, 108  
     systemische 111  
 Interozeption 443  
 Interpolationsproblem 118  
 Interpretationssystem 335  
 Interpreter 26  
 Interview, strukturiertes 511  
 Introspektion 450  
 Intuition 151, 217  
 Iteration 509  
 Iterationsplanung 507
- J**  
 Java 242, 244, 253, 255, 408, 540  
 Java -Editor 523  
 Java-Programm 256, 263, 264  
 Java4Robotic 535
- K**  
 Kapselung 263  
 Karten, neuronale 144  
 Kartenerstellung 112, 133  
 Kausalmodell 74  
 Kellerspeicher 342  
 Kinematik 23, 82, 221  
 Klassen 263  
 Klassendiagramm 529  
 Klassifizierungswissen 99  
 Knoten 514  
 Knowledge Engineering 200, 215  
 Knowledge Systems Hypothesis (KSH) 149  
 Kognogenese 295  
 Kognition 138, 286  
     agierende 289  
     artifizielle 107, 110, 145, 159, 428, 465  
 Kognitionsbegriff 147  
 Kognitionssystem 441  
 Kognitionstheorie 144  
 Kognitionsverarbeitung 204  
 Kognitivismus 41  
 Kohonen-Netz 398  
 Kollaborationsdiagramm 532  
 Kollisionsvermeidung 224  
 Kommentare 541  
 Kommunikation 28, 48, 115, 188, 486  
 Kommunikationsfähigkeit 16  
 Kommunikationskomponente 344  
 Kompasssensoren 229  
 Komponenten 49–51, 192, 528  
     kognitive 110  
     mentale 216  
 Komponentendiagramm 528  
 Komponentengruppe 17  
 Kompositionalität 168  
 Konditionaloperator 553  
 Konfidenzfaktor 516  
 Konfiguration 66  
 Konnektionismus 149, 330, 372, 379, 470  
 Konnektionsmatrix 384  
 Konsequenzen 305  
 Konstanten 122, 255, 544, 567  
     arithmetische 87  
 Konstruktgitter-Technik 511  
 Kontrollstruktur 555  
 Kontrollwissen 340  
 Konzeptionalisierung 170, 173, 274, 499  
 Kooperationsstrategie 116

- Koppelnavigation 112  
Kopplung 52  
Kräfte 23, 43, 83  
Künstliche Intelligenz 190  
Künstliches Leben 190  
Kurzschluss-Verbindung 387
- L**  
Laser 228, 234  
Laserentfernungsmeßgeräte 228  
Leben, artifizielles 16, 110, 271  
Lebenswelt 6, 12, 486  
Lebenszyklus 106, 121, 180, 283, 485  
Leib-Seele-Problem 41  
Leistungsmerkmal 29, 368  
Lernalgorithmen 384  
Lernen 376, 455  
    artifizielles 215, 458  
    bestärkendes 391, 457  
    durch Analogiebildung 460  
    erklärungsbasiertes 460  
    in neuronalen Netzen 460  
    induktives 459  
    maschinelles 456  
    überwachtes 390, 457  
    unüberwachtes 391, 457  
Lernerfahrung 461  
Lernsystem 455  
Lokalisierung 112
- M**  
Makro-Operatoren 459  
Map-Matching 430  
Maschinennatur 5  
Maschinenmodell 267  
Maschinensprache 249  
Maßstabsmodell 86  
Master-Slave-Programmierung 236, 237  
Mechanik 221  
Mehrkörpermodellsystem 83  
Metakognition 445, 449  
Metamodell 63  
Metatheorie 40  
Metawissen 99  
Methodendeklaration 544  
Mimesis 65  
Mindstorms- Motor 414  
Mitteilbarkeit 167  
Mittel-zum-Zweck-Analyse 328  
Mittel-zum-Zweck-Tabelle 327  
Modalaspekte 101  
Model Driven Architecture 240  
Model-View-Controller-Muster (MVC) 271  
Modell 41, 54  
    analytisches 79  
    bestätigtes 137  
    des Schlussfolgerns 70  
    des Verstehens 74  
    deterministisches 136  
    diskretes 136  
    dynamisches 87  
    elektromechanisches 62  
    enaktives 61  
    formales 86  
    geometrisches 134  
    graphisches 59  
    inneres 77  
    kognitives 64  
    kontinuierliches 136  
    logisches 87  
    lokales mentales 72  
    materielles 86  
    mathematisches 79, 86  
    mechanisches 62  
    mentales 64, 68  
    nicht-lineares 93  
    nichtparametrisches 82  
    parametrisches 81  
    physikalisches 78  
    physischotechnisches 61  
    probabilistisches mentales 73  
    rasterbasiertes 134  
    semantisches 59, 62  
    statisches 87  
    symbolisches 86  
    technisches 59  
    topologisches 134  
    zyklisches 159  
Modellbegriff 36, 57, 85  
Modellgültigkeit 57, 80  
Modellierung 46, 56, 84, 267, 476  
    analytische 136  
    kognitive 65, 169, 277  
    mimetische 65, 67, 136  
Modellklassen 156  
Modelltheorie 33, 57  
Modellvarianten 58, 134  
Modularisierung 268, 270

Modulkopplung 269

Momente 23, 222

Momentumterm 398

Monitormodell 88

Monte-Carlo-Methode 136

Motivation 210

  artifizielle 446

Multiagentensystem 467

Multitasking 251, 408, 578

Multithreading 578

Mutation 352, 358

Mutationsoperatoren 358

## N

Näherungssensor 421

Navigation 112, 133, 429

Nervenzelle 143

Netz

  autoassoziatives 398

  neuronales 144, 147

  semantisches 514

Netzwerkarchitektur 395

Netzwerkstruktur 377

Neuro-Fuzzy-Modell 350

Neurobiologie 205

Neuron 142, 381, 468

Neuronale Netze 144, 147, 330, 378

  artifizielle 231

  nach Architektur 399

  nach Lernparadigmen 397

Neuronales Netzwerk 373

Neuronen-Modell 373

Neuropsychologie 476

NQC 408

Nullplanung 132

## O

Oberflächensensoren 25, 223

Objekt-Attribut-Wert-Tripel 514

Objektorientierte Analyse (OoA) 157

Off-Road Navigation 21

Offlineprogrammierung 425

Ohm'sche Last 416

Onlineprogrammierung 424

Operatoren 118, 546

  arithmetische 87, 547

Operatorwissen 120

Orchestrierung 448

## P

Pair Programming 163

Paketierung 571

Paradigma 64, 164

  der Objektorientierung 441

  der Symbolverarbeitung 329

Paradigmenwechsel 473

Parameter-Kodierung 359

PASCAL 250

Persistenceproblem 130

Persönlichkeitsmerkmal 151

Perzeptionsmodell 74

Perzeptron 386

Pfad(weg) planung 133

Pflichtenheft 157

Phasenraum 106

Philosophie 54, 55, 105, 170, 274, 476, 497

Physical Symbol Systems Hypothesis 376

Piloting 114

Pilotmodell 79

Planung 111, 127, 327

  autonome 17

  bedingte 132

  reaktives 132

Planungsaufwand 129

Planungskomponente 192

Planungsschritte 130

Planungssystem 336

Planungsverfahren

  deduktives 131

  mengenbasiertes 131

  planbasiertes 131

  suchbasiertes 131

Plattform 26, 195

Polymorphie 567, 570

Potentiometer 230

Prädikatenlogik 519

Präfiguration 66

Pragmatismus 58

Predictionproblem 130

Primitiva 546

Problemdomäne 362

Problemformulierung 296

Problemlösen 119, 150, 296, 466

  systemisches 117

Problemlösermodell 241

Problemlösungsstrategie 100

Problemmodellierung 171, 177, 299

- Produktionsregelsystem 333  
Produktlebenszyklus 474  
Prognosemodell 88  
Programmiersprache 5, 55, 94, 239  
    allgemeine 248  
    aufgabenorientierte 247  
    dedizierte 242  
    deklarative 242  
    der Wissensverarbeitung 256  
    imperative 241  
    implizite 246  
    logik-orientierte 242  
    objektorientierte 157, 242, 259, 262, 539  
    prädiktive 258  
    problemorientierte 250  
    RCX-kompatible 408  
    regel-orientierte 242  
    selbstreflexive 241  
    simulationsorientierte 253  
Programmiertechnik 265  
Programmierung 28, 156, 160, 235  
    aufgabenorientierte 239  
    durch Beispiele 237  
    durch Training 239  
    ereignisgesteuerte 157  
    evolutionäre 351  
    genetische 351  
    komponentenbasierte 270  
    manuelle 235, 237  
    objektorientierte 539  
    problemorientierte 240  
    roboterorientierte 239  
Programmmuster 433  
Projektion 360  
Projektmanagement 155, 177, 179  
Projektplanung 180  
Projektsteuerung 181  
PROLOG 257  
Propagierung 311  
Propagierungsfunktion 382  
Protokollanalyse 512  
Prototyp 160  
Prototypmodell 79  
Prozessanalyse 505  
Prozesse 504  
Prozessmodell 171, 446, 499  
Prozessorientierung 93  
Psychologie 476  
Psychometrie 283  
Pulsweitenmodulation 415
- Q**  
Qualifikationsproblem 129
- R**  
Radarsensor 422  
Rahmenproblem 129  
Randbedingungen 164, 177  
Real world 91  
Realismus  
    kritischer 56  
    naiver 55  
Realität 91  
Realtime-Navigation 114  
Rechnerarchitektur 61  
Reflex 428  
Regel-Kodierung 359  
Regelwissen 339  
Reihenfolgeplanung 127  
Rekombination 353  
Relationen 36  
Relationenwissen 99  
Relaxationsnetzwerk 394  
Remote-Control-Programmierung 237  
Requirements Engineering 56  
Ressourcenplanung 127  
Rezeptoren 204  
Roadmap 507  
Roboter 14  
    autonomer 14  
    Gesetze 15  
    humanoider 13  
    medizinischer 10  
Roboterbausätze 406  
Roboterbetriebssystem 26  
Roboterkonstruktion 420  
Roboterprogrammiersprache 26, 239  
    explizite 245  
    implizite 245  
Robotersimulationssystem 90  
Robotersteuerung 25  
Robotersystem 6, 34, 104  
    artifizielles 8  
    Aufbau 21  
    intelligentes 16, 28, 106  
    interoperationsbasiertes 192  
    kognitives 30, 204  
    Lebenszyklus 179  
    lernfähiges 45  
    mobiles 18, 112  
    stationäres 18

- Robotersystemprogrammierung 243  
Robotic Controller Explorer (RCX) 406  
Robotic Invention System 407  
Robotik 15  
  Engineering 266  
  interoperationsbasierte 105  
  klassische 14  
  kognitive 27, 199, 473  
Robotikforschung 490  
  multidisziplinäre 491  
Rotations-Sensor Watcher Block 413  
Rotationssensor 413  
Rückkopplung 52, 159, 387, 398
- S**
- Sachwissen, propositionales 98  
Schedulerkomponente 192  
Scheduling 128  
Schemabegriff 70  
Schleifensteuerung 558  
Schlüsselwörter 172, 255  
Schlussfolgerungskomponente 192  
Schlussfolgerungsprozess 217  
Schlussweise 218  
Schrittmotor 232  
Schwellenwert 372  
Selbstlokalisierung 133, 134  
  globale 135  
  lokale 135  
Selbstorganisation 45  
Selektionsstrategie 356  
Semantik, duale 518  
Sensordaten 25  
Sensoren 188, 205, 222  
  externe 24, 223  
  haptische 225  
  interne 24, 223  
  optischer 412  
  taktile 421  
  visuelle 422  
Sensorik 419, 421  
Sequenzdiagramm 531  
Sequenzen 114, 362  
Serviceroboter 7  
Servomotor 232  
SIMULA 260  
Simulation 40, 74, 88, 470, 524  
Simulationsbegriff 91  
Simulationsmethode 286  
Simulationsmodell 79  
Simulationssystem 337  
Simulationstheorie 90  
Situiertheit 108  
Skalierungstechnik 513  
Slots 517  
SMALLTALK 260  
Software 54, 235, 424  
  Factories 240  
Softwareagenten 361  
Softwaretechnik 266  
Sonarsensoren 227, 228  
Spielzeugroboter 13  
Spiralmodell 159  
Sprachelement 540  
Steepest-Ascent-Hill-Climbing-Verfahren 316  
Steuerstruktur 342  
Steuerung 234  
Steuerungs-Architektur 185  
Struktur 36  
Strukturdeterminiertheit 45  
Strukturlege-Verfahren 513  
Strukturmuster 436  
Subsumptionsarchitektur 368  
Subsumtionsarchitektur 151  
Subsymbolismus 372  
Suchbaum 305  
Suche, heuristische 314  
Suchstrategie 301  
Symbolismus 146  
Symbolmanipulation 285  
Symbolverarbeitung 375  
Synapsen 379  
Synchronantrieb 423  
Syntaxdiagramm 267  
Synthese-Problem 118  
System 34  
  adaptives 45  
  allopoietisches 46  
  artifizielles 67, 72  
  autopoietisches 47  
  determiniertes 44  
  effektorisches 218, 451  
  emotionales 209  
  epigenetisches 214, 449  
  Freiheitsgrad 44  
  geschlossenes 43, 52  
  intentionales 217, 444, 451  
  kognitives 199, 443

- lernfähiges 45  
motivationales 210, 446  
offenes 43, 51  
perzeptives 204, 443  
probabilistisches 44  
psychisches 48  
selbstorganisierendes 44  
semantisches 208, 444  
situatives 211, 450  
strukturdeterminiertes 44
- Systemanalyse 40  
Systemarchitektur 158  
Systembegriff 49  
  hierarchischer 50  
Systemheurismen 120  
Systemklasse 39  
Systemkompetenz 121  
Systemmodell  
  zeitinvariantes 82  
  zeitvarianter 82  
Systemmodellierung 178  
Systemrealisierung 178  
Systemsynthese 40  
Systemtheorie 37  
Systemvalidierung 178  
Systemvarianten 41
- T**  
Teach-in 236  
Technologiephilosophie 478  
Terme, linguistische 345  
Textanalyse 513  
Theoriodynamik 470  
Threads 578  
Tiefensuche 307  
Top-Down-Vorgehensweise 156  
Topologie 386  
Trainingssystem 336  
Trajektorie 21  
Transduktion, kognitive 206  
Transfiguration 66  
Transformationsmaßnahmen 77  
Transparenz 57
- U**  
Überlebenswahrscheinlichkeit 176  
Überwachungssystem 336  
Uhrwerke 2  
Umgebungsmodellierung 133
- UML4Robotik 525  
Umwelt 109  
Umweltkarte 112  
Umweltdmodell 429  
Unified Modeling Language (UML) 60, 157, 267  
Unified Process 160  
Unwissen 101
- V**  
V-Modell 159  
Valenz 448  
Validierung 137, 170, 500  
Variablen 543  
Variationsoperatoren 353  
Verarbeitungsmodell  
  konventionelles 241  
  relationales 241  
Vererbung 514  
Vererbungsmechanismen 562  
Verhalten 452  
  intelligentes 329  
Verhaltensgedächtnis 453  
Verifikation 137  
Vernunftwissen 95  
Versagenswahrscheinlichkeit 176  
Verständlichkeit 167  
Verstärkungslernen 460  
Versuch-Irrtum 121  
Verzweigung 555  
Verzweigungsproblem 130  
Virtualisierung 89  
Vorgehensmodell 155  
  kognitives 171  
Vorwärtsverkettung 340
- W**  
Wahrnehmung, artifizielle 205, 207  
Wahrnehmungsmodell 74  
Wahrscheinlichkeitshinweis 73  
Wasserfallmodell 159  
Wechselbeziehung 115  
Wechselbild 482  
Weltwissen 452  
Wiederverwendung 270  
Wirkwahrscheinlichkeiten 305  
Wissen 95, 189  
  definitorisches/kontingentes 100  
  deklaratives, faktisches 98

- explizites 95  
heuristisches 98  
implizites 95  
kausales 98 operatives 95  
prototypisches 100  
prozedurales 98  
statistisches 98  
systemisch-heuristisches 120  
ungenaues 100  
unsicheres 100  
unvollständiges 100  
widersprüchliches 100  
Wissensakquisition 511  
Wissensbasis 338  
Wissensbegriff 101  
Wissenserhebung 511  
Wissenserwerbskomponente 344  
Wissensgedächtnis 453  
Wissensojekt 194
- Wissensqualitäten 100  
Wissensrepräsentation 455  
Wissenstheorie 96  
Wissensvarianten 97  
Wissensverarbeitung, automatische 277  
Wissensverarbeitungssystem 149  
Workbench 522
- Z**
- Zeichen 97  
Zeitplanung 127  
Zielreduktionsregel 370  
Zustands-Problem 297  
Zustandsautomat 533  
Zustandsdiagramm 533  
Zustandsgraph 303  
Zustandsmengenraum 298  
Zustandsraum 106  
Zustandsspeicher 52