Prof. Dr. Lars Mönch

Kurs 01772

E-Buisness Management

LESEPROBE

Fakultät für
Mathematik und Informatik

FernUniversität in Hagen

Lernziele

Die vorliegende Kurseinheit beschäftigt sich mit Multi-Agenten-Systemen. Softwareagenten sind Programme, die in der Lage sind, selbständig Entscheidungen zu treffen, um ihre Entwurfsziele zu erreichen. Sie sind für das E-Business Management wichtig, da sie unter anderem erlauben, Verhandlungen zwischen Nachfragern und Anbietern zu automatisieren.

Nach dem Bearbeiten von Abschnitt 3.1 können Sie die Begriffe Softwareagent und Multi-Agenten-System erläutern. Sie sind in der Lage, die Unterschiede zwischen dem objekt- und dem agentenorientierten Paradigma zu erläutern. Außerdem können Sie wichtige Architekturen für Softwareagenten beschreiben.

Fragen der Kommunikation in Multi-Agenten-Systemen werden in Abschnitt 3.2 diskutiert. Sie sind nach dem Studium dieses Abschnitts dazu befähigt, die Notwendigkeit einer ausdrucksstarken Kommunikation für ein Zusammenwirken von Agenten in Multi-Agenten-Systemen zu begründen. Sie können erläutern, warum Ontologien wichtig sind und wie sie entworfen werden. Mit den wesentlichen Prinzipien von Kommunikations- und Inhaltssprachen sind Sie vertraut. Die Bedeutung von Interaktionsprotokollen können Sie beschreiben.

Nach dem Studium von Anschnitt 3.3 sind Sie in der Lage, die Prinzipien des verteilten Problemlösens in Multi-Agenten-Systemen zu erläutern. Sie können wesentliche Ansätze zur Aufgaben- und Ergebnisaufteilung in Multi-Agenten-Systemen beschreiben. Außerdem können Sie typische Koordinationsansätze in Multi-Agenten-Systemen darstellen.

Die softwaretechnische Umsetzung von Multi-Agenten-Systemen wird im Übungsbetrieb behandelt. Nach der Bearbeitung der Einsendeaufgaben können Sie erläutern, wie die Kommunikation in Multi-Agenten-Systemen softwaretechnisch realisiert wird. Das gilt auch für bestimmte Koordinationsansätze.

Die Übungsaufgaben dienen der Überprüfung des erreichten Kenntnisstandes. Erst durch das selbständige Lösen von Übungsaufgaben werden Sie eine große Sicherheit im Umgang mit den Begriffen, die in der Kurseinheit eingeführt wurden, erlangen. Außerdem soll Sie die eigenständige Beschäftigung mit den Aufgaben zu einer weiteren Durchdringung des erarbeiteten Stoffes anregen. Die vorgeschlagenen Lösungen werden Ihnen dabei helfen.

3.1 Softwareagenten und Multi-Agenten-Systeme

Nach einer Motivation und einem ausführlichen Beispiel werden in diesem Abschnitt einige der für diese Kurseinheit zentralen Begriffsbildungen vorgenommen. Wir führen dazu die Begriffe Softwareagent und Multi-Agenten-System ein. Anschließend werden verschiedene Architekturen für Softwareagenten diskutiert.

3.1.1 Motivation und einführendes Beispiel

Die zunehmende Verbreitung des Internets und dessen große Bedeutung als neue Plattform für den Austausch von Gütern hat zu einer Dematerialisierung von betrieblichen Prozessen und damit zu grundlegend veränderten Anforderungen an die betriebliche Informationsverarbeitung geführt [12]. Als neue Anforderung ergibt sich für Anwendungssysteme ein erhöhter Bedarf an Flexibilität, um auf dynamische Veränderungen in ihrer Umwelt reagieren zu können.

Der Einsatz von Computerprogrammen verlangt typischerweise, dass jede Aktion des Programms geplant, vorgedacht und programmiert ist [12, 26]. Die in Kurseinheit 2 behandelten Client-Server-Architekturen führen zu Anwendungssystemen, die zu dieser Systemklasse gehören. Wenn Situationen auftreten, die bei der Systemerstellung nicht bedacht worden sind, kann das zum Fehlverhalten des Systems bis hin zum Systemabsturz führen.

Diese Situation tritt insbesondere immer dann auf, wenn Leistungsprozesse eng verzahnt sind, die Ziele lokaler Akteure mit den Zielen übergeordneter Akteure in Einklang gebracht werden müssen und zur selben Zeit schnell auf unvorhergesehene Ereignisse reagiert werden muss [12]. Es werden somit Anwendungssysteme erforderlich, die online-fähig sind, d.h. auf eine veränderte Situation zeitnah reagieren können. Die geforderte Online-Fähigkeit kann aber nur durch Anwendungssysteme erreicht werden, die über die folgenden Eigenschaften verfügen:

- Das Verhalten der Systeme ist zielorientiert.
- Die Systeme betreiben auch ohne menschliche Eingriffe permanent die ihnen zugewiesenen Aufgaben.
- Diese Systeme sind in der Lage, Ausnahmesituationen selbständig zu erkennen sowie zu bewerten.
- Den Systemen zugeordnete Pläne für bestimmte Abläufe können in Echtzeit an veränderte Bedingungen angepasst werden.

Wir werden in dieser Kurseinheit sehen, dass Softwareagenten geeignete Artefakte sind, um die geforderten online-fähigen Anwendungssysteme zu konzipieren und umzusetzen. Softwareagenten haben ihre Wurzeln in der Künstlichen Intelligenz. Gleichzeitig ist aber festzustellen, dass sie zu einer neuen Softwaretechnologie, der Agententechnologie, geführt haben.

Bevor wir Softwareagenten weiter untersuchen, soll zunächst ein erstes Beispiel aus der E-Business-Domäne angegeben werden, das den Einsatz von Softwareagenten motiviert. Wir betrachten in Anlehnung an [12] ein spezielles dienstorientiertes Anwendungssystem, das börsenbezogene Dienstleistungen für Privatkunden bereitstellt. Dienstorientierte Anwendungssysteme werden durch den Anwender als Portal wahrgenommen. Ein Portal dient dazu, dass die vom Benutzer benötigten Dienste, zu deren Ausführung er berechtigt ist, zusammengesucht und durch das Portal in einer einheitlichen Oberfläche angeboten werden.

Beispiel 3.1.1 (Web-Portal) Das Portal soll zusätzlich zu den Handelsfunktionen, auch als Brokerage bezeichnet, Informationsdienste und Community-Funktionen anbieten sowie eine individualisierbare Bannerwerbung ermöglichen. Die Architektur des Portals für Finanzdienstleistungen ist in Abbildung 3.1 dargestellt. Für jeden Nutzer des Portals werden eigene Agenten zur Verfügung gestellt. Die vier Agententypen bieten die folgende Funktionalität an:

- Der **Informationsagent** ermittelt das vom Benutzer angefragte Wissen. Dazu können je nach Implementierung Datenbankabfragen, aber auch eine selbständige Suche im Internet, zählen.

- Der **Handelsagent** stellt die für die Durchführung von Börsentransaktionen erforderliche Funktionalität zur Verfügung. Typische Funktionen sind die Bereitstellung eines Auftragsformulars, die Durchführung von Konto- und Depotabfragen zur Sicherstellung der Auftragsvoraussetzungen, die Bereitstellung von Möglichkeiten zum Aufruf der einzelnen Buchungsroutinen sowie das Ermöglichen eines alle Einzelaktivitäten steuernden Handelsmanagements. Es besteht auch die Möglichkeit, dass ein Handelsagent bei entsprechendem Bedarf des Anwenders zusätzliche Funktionalität, zum Beispiel zur Abwicklung von Arbitragegeschäften, zur Verfügung stellt. Ein Arbitragegeschäft erlaubt Transaktionen, die mehrere zusammenhängende Geschäfte auf unterschiedlichen Märkten umfassen.

- Ein **Avatar** vertritt den Benutzer in den von den Community-Foren angebotenen Chatforen. Avatare müssen in der Lage sein, mit dem Benutzer zu kommunizieren. Sie benötigen dazu ein bestimmtes Wissen über den Benutzer. Der Portalbetreiber kann einen Avatar dazu benutzen, Wissen über den jeweiligen Benutzer zu sammeln und in der

Kundenprofildatenbank abzulegen.

- Der **Konfigurationsagent** ist für eine benutzerabhängige Gestaltung der Bannerwerbung verantwortlich. Die Konfiguration basiert auf den Kundenprofilen sowie auf Wissen, das eine Zuordnung zwischen Eigenschaften des Kundenprofils und Bannern ermöglicht.

Unter Architektur-Gesichtspunkten wird zwischen einer Datenhaltungsschicht, in der sich verschiedene relationale Datenbanken befinden, sowie einer Anwendungsschicht unterschieden (vergleiche Unterabschnitt 2.3.2). Die Anwendungsschicht beinhaltet eine Teilschicht, welche die Anwendungsfunktionalität gegenüber der Präsentationsschicht abgrenzt. Außerdem dient ein benutzerspezifischer Informationsraum dazu, dynamisch und inhaltsgetrieben die erforderlichen Informationen zusammenzustellen. Die Agenten sind in der Lage, andere Agenten in ihrer Umgebung zu identifizieren und mit ihnen zu kommunizieren. Außerdem können die Agenten in einem bestimmten Umfang selbständig Aktionen durchführen. Als Ergebnis verändert sich der Zustand des Informationsraums. Dadurch beeinflussen die Agenten ihre eigene Umgebung und die der anderen Agenten.

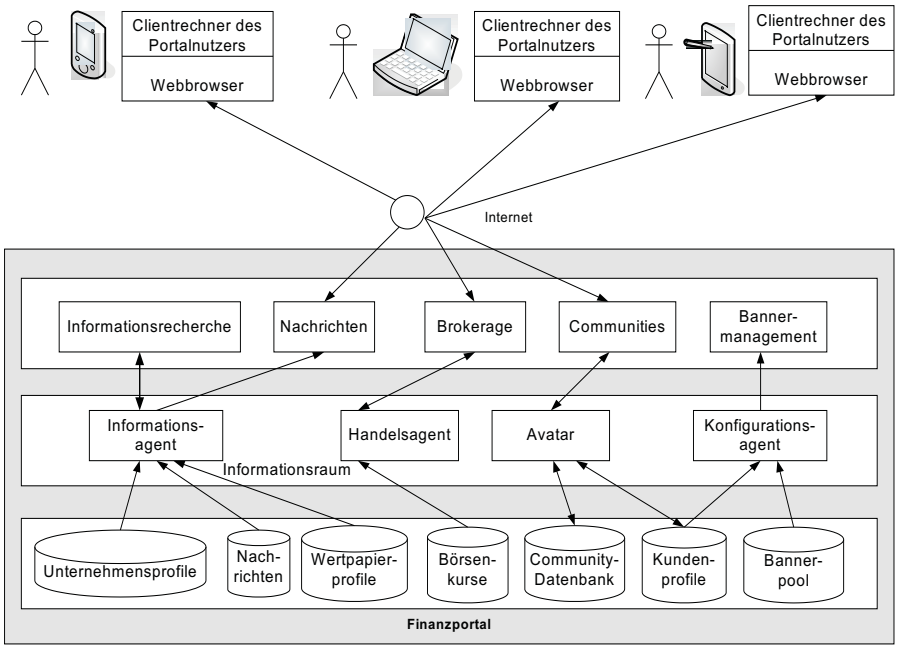

Abbildung 3.1: Portal für Finanzdienstleistungen

Wir erkennen, dass sich die Portaldienste des Web-Portals aus Beispiel 3.1.1 dynamisch an die Bedürfnisse der Benutzer anpassen, die sich in deren Interaktionsverhalten widerspiegeln.

Übungsaufgabe 3.1 (Agenten) Geben Sie ein eigenes Beispiel analog zu Beispiel 3.1.1 an. Stellen Sie dar, wodurch in Ihrem Beispiel die Online-Fähigkeit der Agenten erreicht wird.

Softwareagenten und Multi-Agenten-Systeme sind Gegenstand des Forschungsgebiets der Verteilten Künstlichen Intelligenz. Es gibt verschiedene Anwendungen im E-Business Management. Beispielsweise werden Softwareagenten dazu verwendet, menschliche Aufgabenträger im Rahmen von Verhandlungen zu vertreten [5, 22]. Entsprechende Anwendungen im Lieferkettenmanagement existieren. Softwareagenten werden weiterhin zur Prozesskoordination in Produktionsnetzwerken eingesetzt [28]. Man kann aber feststellen, dass die hohen Erwartungen an Softwareagenten zum Teil nicht erfüllt wurden.

3.1.2 Softwareagenten

Umgangssprachlich versteht man unter einem Agenten eine Person, die im Auftrag anderer handelt [13]. Wir versuchen in diesem Unterabschnitt, diesen umgangssprachlichen Agentenbegriff für unsere Zwecke zu schärfen und geeignet zu erweitern. Wir definieren dazu zunächst den Begriff eines Agenten in Anlehnung an [6] wie folgt.

Definition 3.1.1 (Agent) Ein Agent ist eine physische oder virtuelle Einheit,

- die in ihrer Umgebung handeln kann,
- die mit anderen Agenten direkt kommunizieren kann,
- die durch eine Menge von Tendenzen getrieben ist,
- die eigene Ressourcen besitzt,
- die in beschränktem Umfang ihre Umgebung wahrnehmen kann,
- die entweder über gar keine oder nur über eine partielle Repräsentation der Umgebung verfügt,
- die Fähigkeiten besitzt und diese in Form von Diensten anderen Agenten zur Verfügung stellen kann,
- die sich eventuell selber reproduzieren kann,
- deren Verhalten auf die Erfüllung ihrer Ziele abzielt, wobei die zur Verfügung stehenden Ressourcen und Fähigkeiten sowie die Wahrnehmung der Umwelt berücksichtigt werden.

Eine Tendenz in Definition 3.1.1 wird dabei entweder durch individuelle Ziele oder durch eine zu optimierende Befriedigungs-/Überlebensfunktion repräsentiert. Die Definition verdeutlicht, dass der Umgebungsbegriff für Agenten bedeutsam ist. Unter der Umgebung bzw. Umwelt verstehen wir alles außerhalb des Agenten. Das bedeutet insbesondere, dass andere Agenten sich in der Umwelt eines Agenten befinden können. Ein Agent nimmt seine Umwelt über Sensoren wahr und agiert in ihr. Umgekehrt wird über Aktuatoren die Umwelt beeinflusst. Das wird in Abbildung 3.2 dargestellt.

Abbildung 3.2: Agent und Umgebung

In Anlehnung an Russel und Norvig [25] existieren die folgenden Umgebungseigenschaften:

- **vollständig beobachtbar versus teilweise beobachtbar** : Wenn der Zustand der Umgebung zu jedem Zeitpunkt vollständig durch den Agenten bestimmt werden kann, liegt eine vollständig beobachtbare Umgebung vor. Die meisten Umgebungen sind nur teilweise beobachtbar. Alternativ wird auch von vollständig einsehbaren und teilweise einsehbaren Umgebungen gesprochen.

- **deterministisch versus stochastisch** : Wenn der nächste Zustand der Umgebung vollständig durch den aktuellen Zustand und die Aktion des Agenten bestimmt wird, sprechen wir von einer deterministischen

Umgebung. Andernfalls ist die Umgebung stochastisch. Vollständig beobachtbare Umgebungen sind deterministisch. Teilweise beobachtbare Umgebungen hingegen können stochastisch sein.

- **diskret versus kontinuierlich** : Eine Umgebung wird als diskret bezeichnet, wenn die Anzahl möglicher Zustände diskret ist, d.h. abzählbar. Das bedeutet insbesondere, dass Umgebungen mit endlich vielen möglichen Stati diskret sind. Eine Umgebung, die nicht diskret ist, heißt kontinuierlich.

- **statisch versus dynamisch** : Eine statische Umgebung ist dadurch charakterisiert, dass sie nur durch Aktionen des Agenten verändert werden kann. In einer dynamischen Umgebung kann sich deren Zustand ändern, ohne dass das vom Agenten beeinflusst werden kann. Die physische Welt ist typischerweise hochgradig dynamisch.

Eine alternative Agentendefinition, die stärker eine Softwareperspektive berücksichtigt, ist in [13, 27] gegeben.

Definition 3.1.2 (Softwareagent) Ein Softwareagent ist ein Computerprogramm, das die folgenden Eigenschaften hat:

1. **Eigenschaft**: Es befindet sich in einer bestimmten Umgebung.
2. **Eigenschaft**: Es bietet nützliche Dienste an.
3. **Eigenschaft**: Es ist in der Lage, autonome Aktionen in seiner Umwelt durchzuführen, um seine Entwurfsziele zu erfüllen.
4. **Eigenschaft**: Die Autonomie wird geleitet durch eigene Ziele. Bestandteile des Entscheidungsprozesses sind Überlegungen zur Zielerreichung sowie eine Bewertung von Ziel-Mittel-Beziehungen.

Wir erläutern zunächst Definition 3.1.2 genauer. Die erste Eigenschaft bedeutet, dass der Agent die Umgebung wahrnimmt und in ihr agiert. Dabei ist wesentlich, dass die Interaktion mit der Umgebung nicht nach der Erfüllung einer bestimmten Aufgabe beendet wird, sondern permanent stattfindet. Die zweite Eigenschaft kann dahingehend interpretiert werden, dass das Verhalten eines Agenten auf Basis der Qualität seiner Ergebnisse bewertet wird. Wir bemerken, dass Computerprogramme, welche die Eigenschaft 1 und 2 besitzen, nicht notwendigerweise Softwareagenten darstellen. Wir betrachten dazu das folgende Beispiel.

Beispiel 3.1.2 (Computerprogramm, das kein Agent ist) Eine Software zur Maschinensteuerung interagiert ebenfalls mit physischen Objekten aus einer bestimmten Umgebung, wird aber aufgrund der fehlenden Autonomie nicht als Softwareagent im Sinne von Definition 3.1.2 aufgefasst. Ein

Webservice bietet, wie in Eigenschaft 2 beschrieben, Dienste an. Gleichzeitig wird aber ein Webservice nicht als Softwareagent betrachtet.

Die dritte Eigenschaft ist für unsere Einführung von Softwareagenten wesentlich, denn sie führt den Begriff der Autonomie ein. Der Autonomiebegriff wird nun in Anlehnung an [24] wie folgt definiert.

Definition 3.1.3 (Autonomie eines Computerprogramms) Ein Computerprogramm wird als autonom bezeichnet, wenn es seine Funktion ohne Steuereingriffe von außen erbringt.

Autonomes Verhalten ist eine nicht-funktionale Eigenschaft. Ein Softwareagent besitzt im Gegensatz zu konventioneller Software bestimmte Freiheitsgrade bezüglich der Erfüllung von Aufgaben. Der Grad des Handlungsspielraums wird durch die Ziele des Agenten bestimmt. Ein Agent kann sich dafür entscheiden, eine bestimmte Aufgabe nicht weiter zu bearbeiten, sie zu unterbrechen oder den Startzeitpunkt für die Bearbeitung der Aufgabe festzulegen. Ein Agent kann gleichzeitig mehrere Ziele verfolgen, die in Konflikt zueinander stehen. Der Agent hat zu entscheiden, welche Ziele er verfolgen will und wie er diese erreicht. Das ist die Kernaussage von Eigenschaft 4.

Die Umgebung, in der Softwareagenten operieren, ist typischerweise nicht vollständig einsehbar, d.h., der Agent ist nicht in der Lage, vollständige, aktuelle Informationen zu erhalten oder diese sind mit Unsicherheit behaftet [27]. Ein Softwareagent kann deshalb eine solche Umgebung nicht vollständig wahrnehmen. In der Folge kann der Agent deshalb seine Umwelt nicht vollständig steuern, sondern nur zum Teil beeinflussen. Eine nur teilweise beobachtbare und stochastische Umgebung wird als nicht-deterministisch bezeichnet. Das führt zum Begriff eines intelligenten Softwareagenten.

Definition 3.1.4 (Intelligenter Softwareagent) Ein intelligenter Softwareagent ist ein Softwareagent, der in der Lage ist, in einer nicht-deterministischen Umgebung zu operieren. Im Vergleich zu einem herkömmlichen Softwareagenten besitzt ein intelligenter Softwareagent die folgenden zusätzlichen Eigenschaften:

5. **Eigenschaft**: Ein Agent ist reaktiv, d.h., er beobachtet ständig seine Umgebung und reagiert auf Veränderungen dieser zeitnah.

6. **Eigenschaft**: Ein Agent findet eine geeignete Balance zwischen zielgerichtetem und reaktivem Verhalten.

7. **Eigenschaft**: Ein Agent ist proaktiv. Das bedeutet, dass er die Initiative ergreift, um seine Ziele zu erreichen.

8. **Eigenschaft**: Ein Agent verfügt über soziale Fähigkeiten, d.h., er ist in der Lage, mit anderen Agenten zu interagieren, um seine Dienste zur Verfügung zu stellen.

9. **Eigenschaft**: Ein Agent ist in der Lage, zu lernen.

Wir diskutieren nun die Eigenschaften intelligenter Softwareagenten. Eigenschaft 5 besagt, dass der Agent seine Umgebung ununterbrochen beobachtet. Diese Beobachtung ist umfassender als ein ausschließliches Reagieren auf Anfragen. Die Art und der Zeitpunkt einer zeitnahen Reaktion wird dabei durch den Softwareagenten selber bestimmt, die Umwelt kann das nicht festlegen. Eigenschaft 6 besagt, dass ein intelligenter Softwareagent auf der einen Seite im reaktiven Modus in einer zielgerichteten Art und Weise auf Ereignisse in der Umgebung reagiert. Im Normalfall wird sich ein Agent im reaktiven Modus befinden. Auf der anderen Seite können die Ziele das Verhalten des Agenten bestimmen. In diesem Fall muss der Agent selber die Initiative ergreifen und die Umgebung verändern, um seine Ziele zu erreichen. Das ist die Aussage von Eigenschaft 7. Eigenschaft 8 besagt, dass Agenten bestimmte Problemlösungsschritte an andere Agenten delegieren können, wenn sie nicht in der Lage sind, das Problem allein zu lösen. Die sozialen Fähigkeiten von Agenten erlauben es, dass Agenten Konflikte im Zusammenhang mit der Verfolgung ihrer Ziele lösen können. Eigenschaft 9 schließlich besagt, dass die Zielerreichung und die dazu erforderliche Auswahl von Plänen durch Erfahrungen beeinflusst werden. Es ist deshalb erforderlich, dass intelligente Softwareagenten aus Erfahrungen lernen können, wie sie ihre zukünftigen Aktivitäten verbessern können.

Softwareagenten stellen eine natürliche Erweiterung des objektorientierten Paradigmas dar [27]. Objekte sind Entitäten, die einen Status kapseln und in der Lage sind, Aktionen auszuführen. Sie können über Botschaften miteinander kommunizieren. Die möglichen Aktionen eines Objektes werden durch seine Methoden repräsentiert. Kommunikation über Botschaften bedeutet im Wesentlichen den Aufruf einer Methode m_{1k} eines bestimmten Objekts o_1 durch ein Objekt o_2. Objekte unterscheiden sich von Agenten in den folgenden Punkten:

1. **Grad der Autonomie**: Ein Objekt o_i besitzt die Methoden m_{ik}. Diese können von anderen Objekten o_j zu einem beliebigen Zeitpunkt aufgerufen werden. Die Entscheidung, ob eine bestimmte Aktion ausgeführt wird, liegt in einem objektorientierten Softwaresystem beim aufrufenden Objekt. Objekt o_i hat keinen Einfluss auf diesen Vorgang. In agentenbasierten Systemen entscheidet der Agent, der zu einer bestimmten Aktion aufgefordert wird, ob er diese tatsächlich ausführt.

2. **Realisierung von autonomem Verhalten**: Das objektorientierte Standardmodell sieht keine Möglichkeiten vor, derartiges Verhalten abzubilden.

3. **Kontrollfluss**: In objektorientierten Softwaresystemen liegt eine einzige Kontrollinstanz zur Ablaufsteuerung vor. In agentenbasierten Systemen hingegen besitzt jeder Agent einen eigenen Kontrollfaden. Der

Kontrollfluss ist somit auf die unterschiedlichen Agenten verteilt. In modernen objektorientierten Programmiersprachen wie Java wird zwar das Konzept aktiver Objekte durch die Möglichkeiten für Multi-Threaded-Programming unterstützt, die Fähigkeit zur Proaktivität fehlt aber nach wie vor.

Dem Begriff der abstrakten Klasse in der Objektorientierung entspricht in der agentenbasierten Sichtweise der Begriff der generischen Rolle. Wir definieren zunächst in Anlehnung an [16] den Rollenbegriff.

Definition 3.1.5 (Agentenrolle) Das normative Verhaltensrepertoire eines Agenten wird als Rolle bezeichnet.

Während in objektorientierten Systemen von abstrakten Klassen konkrete Klassen abgeleitet werden, können in agentenbasierten Systemen anstelle von generischen Rollen domänenspezifische Rollen von Agenten eingenommen werden. Den Attributen eines Objekts entsprechen Wissen und Überzeugungen eines Agenten. Dabei können, falls Agenten objektorientiert implementiert werden, auch Attribute zur Abbildung von Wissen und Überzeugungen herangezogen werden. In Objekten werden Methoden dazu verwendet, den Zustand eines Objektes zu verändern. In der agentenorientierten Sicht entsprechen den Methoden die Fähigkeiten der Agenten. Objekte interagieren durch Methodenaufrufe. Agenten verhandeln nachrichtenbasiert miteinander. Objekte können durch Komposition zusammengesetzt werden. Das entsprechende agentenbasierte Gegenstück zum Kompositionskonzept bilden holonische Agenten, bei denen ebenfalls Agenten zu neuen Agenten aggregiert werden können. Im Rahmen einer Mehrfachvererbung können Klassen von mehreren Elternklassen erben. Agenten können gleichzeitig mehrere Rollen einnehmen.

Von Klassen können entsprechende Instanzen gebildet werden. Dem entspricht bei einer agentenbasierten Betrachtungsweise die Erzeugung von Agenten, die mit domänenspezifischen Rollen und speziellem Wissen ausgestattet sind.

Unter Polymorphie wird in der Objektorientierung die Fähigkeit mehrerer Klassen von Objekten verstanden, auf die gleiche Nachricht in unterschiedlicher Art und Weise zu reagieren. Der Empfänger einer Nachricht entscheidet, wie diese zu interpretieren ist, nicht der Absender. Der Sender der Nachricht braucht dabei nicht zu wissen, zu welcher Klasse die Empfänger-Instanz gehört. Der Polymorphie entspricht in der agentenbasierten Sichtweise das Matchmaking von Diensten. Beim Matchmaking wird bei bestimmten Agenten nachgefragt, welche Dienste durch welche Agenten angeboten werden. In Tabelle 3.1 werden zusammenfassend Objekte und Agenten sowohl unter elementaren als auch strukturellen Aspekten gegenübergestellt.

Wir möchten an dieser Stelle bemerken, dass es häufig nicht klar ist, ob es sich um ein Objekt oder um einen Softwareagenten handelt [9]. Der Wissen-

Tabelle 3.1: Gegenüberstellung von Objekten und Agenten

	Objekte	Agenten
elementare Aspekte	abstrakte Klasse	generische Rolle
	Klassen	domänenspezifische Rollen
	Attribute	Wissen, Überzeugungen
	Methoden	Fähigkeiten
strukturelle Aspekte	Interaktion von Objekten	Verhandlungen
	Komposition	holonische Agenten
	Vererbung	mehrfache Rollen
	Erzeugung von Instanzen	domänenspezifische Rollen, individuelles Wissen
	Polymorphie	Dienste-„Matchmaking“

schaftler Hewitt bemerkte dazu ironisch, dass die Beantwortung der Frage, was ein Agent sei, für einen Forscher der Verteilten Künstlichen Intelligenz zu ähnlichen Problemen führt, wie die Frage an einen Forscher auf dem Gebiet der Künstlichen Intelligenz, was man unter Intelligenz versteht.

Übungsaufgabe 3.2 (Unterscheidung Objekt/Agent) Untersuchen Sie, ob ein Thermostat als Agent im Sinne dieser Kurseinheit aufgefasst werden kann. Berücksichtigen Sie dabei insbesondere auch die Eigenschaften der Umgebung.

3.1.3 Multi-Agenten-Systeme

In vielen praktischen Situationen liegen keine einzelnen Agenten, sondern ganze Gesellschaften von Agenten vor. Das kann damit begründet werden, dass viele zu lösende Probleme für einen einzelnen Agenten zu komplex sind. Außerdem sind viele praktische Probleme auf natürliche Art und Weise verteilt.

Beispiel 3.1.3 (Verteiltes Problem) An Verhandlungen sind stets mindestens zwei Verhandlungspartner beteiligt. Diese verfügen über eigene Ziele und sind autonom. Wenn eine Verhandlung durch Softwareagenten unterstützt werden soll, ist somit ein Agent nicht ausreichend. Maschinenbelegungsprobleme können durch Verhandlungen gelöst werden. Dabei wird jeder Job (Fertigungsauftrag) durch einen Softwareagenten repräsentiert. Jobagenten verhandeln mit einem Maschinenagenten um die knappe Bearbeitungskapazität auf der Maschine, die durch den Maschinenagenten vertreten wird.

Es ist eine Infrastruktur erforderlich, welche die Agenten bei der Organisa-

tion und der Durchführung ihres Zusammenwirkens unterstützt. Wir nehmen dazu vereinfachend an, dass ein Agent-Directory-Service (ADS) existiert, der es den Agenten ermöglicht, herauszufinden, wie sie andere Agenten kontaktieren können, die ebenfalls Bestandteil des Multi-Agenten-Systems sind. Der ADS ist auch ein Agent. Die typische Funktionalität eines ADS besteht darin, die Adressen aller am Multi-Agenten-System beteiligten Agenten anzubieten. Außerdem können Beschreibungen der von den Agenten angebotenen Dienste durch den ADS zur Verfügung gestellt werden. Die erste Funktionalität wird als Weiße-Seiten-Funktionalität, die zweite als Gelbe-Seiten-Funktionalität bezeichnet.

Wir definieren den Begriff eines Multi-Agenten-Systems in Anlehnung an [7] wie folgt.

Definition 3.1.6 (Multi-Agenten-System) Ein aus Agenten bestehendes System heißt Multi-Agenten-System, wenn es die drei folgenden Eigenschaften besitzt:

1. Es existiert eine statische Problembeschreibung, die aus den Agentenklassen A_i und einem Adressendienst, dem Agent-Directory-Service ADS, besteht. Dabei gilt für das prototypische Multi-Agenten-System AS_{prot}:
$$MAS_{prot} := (\{A_1, \ldots, A_n\}, ADS), \quad n \in \mathbb{N}. \tag{3.1}$$

2. Das Multi-Agenten-System enthält instanziierte Agenten zum Zeitpunkt $t = 0$. Es gilt:
$$A_{init} := \{A_{110}, \ldots, A_{1k_10}, \ldots, A_{n10}, \ldots, A_{nk_n0}\} \tag{3.2}$$
mit $A_i \triangleright A_{ij0}, \ 1 \leq j \leq k_i, \ k_i \in \mathbb{N}, \quad 1 \leq i \leq n$. Das Multi-Agenten-System sieht dann zum Zeitpunkt $t = 0$ wie folgt aus:
$$MAS_{init} = (A_{init}, ADS_{init}), \tag{3.3}$$
wobei ADS_{init} den ADS darstellt, der zum Zeitpunkt $t = 0$ bereits existiert.

3. Es besitzt weiterhin instanziierte Agenten zu einem beliebigen Zeitpunkt $t > 0$:
$$MAS_t := (A_t, ADS_t) \tag{3.4}$$
mit
$$A_t := \{A_{11t}, \ldots, A_{1l_1t}, \ldots, A_{n1t}, \ldots, A_{nl_nt}\} \tag{3.5}$$
und $A_i \blacktriangleright A_{ijt}, \ 1 \leq j \leq l_i, \ l_i \in \mathbb{N}, \quad 1 \leq i \leq n$.

Einige der Agenten in A_{init} sind nicht durch den Entwickler des Multi-Agenten-Systems entworfen, sondern können sich bereits in der Umgebung

befinden. In Definition 3.1.6 haben wir dabei die Notation $A \triangleright A_k$ verwendet, um auszudrücken, dass die Instanz A_k eines Agententyps A gebildet wird. Bei der Instanziierung erbt die Instanz A_k das Verhalten und das initiale Wissen von A. Gleichzeitig kann weiteres Wissen zum Initialisierungszeitpunkt vorliegen. Beispielsweise ist für jeden instanziierten Agenten eine eindeutige Identifizierung innerhalb des Multi-Agenten-Systems vorzusehen. Die Bezeichnung $A_i \blacktriangleright A_{ijt}$ wird verwendet, um auszudrücken, dass zum einen $A_i \triangleright A_{ijt}$ gilt, gleichzeitig aber auch $A_{ij0} \rightsquigarrow A_{ijt}$. Durch die Bezeichnung $x_s \rightsquigarrow x_t$ für zwei Zeitpunkte s, t mit $s < t$ wird die Veränderung des Agenten x im zeitlichen Verlauf bezeichnet. Beispielsweise kann sich der interne Zustand des Agenten x zum Zeitpunkt s vom internen Zustand des Agenten zum Zeitpunkt t unterscheiden, d.h., der Agent hat zum Zeitpunkt t andere Ziele als zum Zeitpunkt s.

Die Entwicklung des Multi-Agenten-Systems wird durch das Versenden und Empfangen von Nachrichten durch die Agenten vorangetrieben. Neue Agenten können im Multi-Agenten-System gestartet werden. Andere Agenten können ihre Arbeit beenden. Jeder Agent verfügt über eine systemweit eindeutige Adresse, die durch den ADS den anderen Agenten zur Verfügung gestellt wird.

Im weiteren Verlauf des Kurses werden wir besonderes Augenmerk darauf legen, die Kommunikationsfähigkeiten von Softwareagenten und die darauf aufbauenden Möglichkeiten zur Kooperation darzustellen.

3.1.4 Agentenarchitekturen

Wir geben zunächst eine abstrakte Architektur für einen intelligenten Softwareagenten an, bevor wir drei konkrete Architekturen diskutieren.

3.1.4.1 Abstrakte Architektur für intelligente Softwareagenten

Wir wollen nun die bisher vorgenommenen Begriffsbildungen in Anlehnung an [27] formalisieren. Dazu nehmen wir zunächst an, dass der Status s der Umgebung eines Agenten durch $s \in S$ angegeben wird, wobei

$$S := \{s_1, s_2, \ldots, \} \tag{3.6}$$

die Menge der möglichen Zustände der Umgebung darstellt. Die Fähigkeiten eines Agenten werden durch die Menge der möglichen Aktionen aus dem Aktionsraum A beschrieben. Es gilt:

$$A := \{a_1, a_2, \ldots\}. \tag{3.7}$$

Ein Agent wird durch eine Funktion *action* beschrieben:

$$action : S^* \longrightarrow A, \tag{3.8}$$

wobei S^* eine Menge von Folgen von Umgebungsstati darstellt. Der Übergang von S zu S^* ist notwendig, weil ein Agent im Allgemeinen über die nächste auszuführende Aktion auf der Basis seiner Erfahrungen entscheidet. Die Funktion *action* wird als Standardagent bezeichnet. Nachfolgend werden verschiedene Erweiterungen eines Standardagenten diskutiert, da dieser für praktische Zwecke zu allgemein ist.

Wir modellieren nun das nicht-deterministische Verhalten der Umwelt. Dazu verwenden wir die Funktion *env*, die wie folgt angegeben werden kann:

$$\begin{aligned} env: \quad S \times A &\longrightarrow P(S) \\ (s,a) &\longmapsto \{s_{k_1}, s_{k_2}, \ldots\}, \end{aligned} \tag{3.9}$$

wobei $P(S)$ die Potenzmenge von S bezeichnet. Die Potenzmenge einer Menge S ist definiert als die Menge der Teilmengen von S, wobei die leere Menge auch als Teilmenge aufgefasst wird. Die Funktion *env* ordnet einer Aktion a und einem Umweltstatus s eine Menge von Stati zu, die als Ergebnis der Ausführung von a unter der Bedingung des Vorliegens von s möglich sind. Wir bemerken, dass die einzelnen s_{k_i} mit Eintrittswahrscheinlichkeiten versehen werden können. Weiterhin wird der Spezialfall einer deterministischen Umgebung durch $|\{s_{k_1}, s_{k_2}, \ldots\}| = 1$ abgebildet.

Wir beschreiben nun die Interaktion von Agent und Umwelt durch die Einführung einer Historie h. Eine Historie h ist eine Folge von Zustandsübergängen der Form

$$h: s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \ldots, \tag{3.10}$$

wobei wir mit s_0 den Status der Umgebung bezeichnen, mit dem der Agent mit der Ausführung von Aktionen beginnt. Die Notation $s_i \xrightarrow{a_i} s_{i+1}$ gibt an, dass der Umweltstatus s_i als Ergebnis der Ausführung von Aktion a_i in den Umweltstatus s_{i+1} überführt wurde.

Wir versuchen nachfolgend, das Verhalten eines Agenten durch die Betrachtung aller möglicher Historien zu charakterisieren. Dazu nehmen wir zunächst an:

1. $action: S^* \to A$ ist ein Agent.
2. $env: S \times A \to P(S)$ ist eine Umgebung.
3. s_0 ist ein initialer Zustand von *env*.

Dann ist die Folge

$$h: s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \ldots, \tag{3.11}$$

eine mögliche Historie des Agenten *action* in der Umgebung *env*, wenn die folgenden beiden Bedingungen erfüllt sind:

1. $a_n = action((s_0, \ldots, s_n))$ für alle $n \in \mathbb{N}$.

2. $s_n \in env(s_{n-1}, a_{n-1})$ für alle $n \in \mathbb{N}^+$.

Das Verhalten eines Agenten kann durch die Angabe aller Historien beschrieben werden.

Wir betrachten als Nächstes Agenten, die gedächtnislos sind, d.h., die Aktionen des Agenten werden ohne Bezug zur Vergangenheit gewählt. Wir erhalten die folgende, im Vergleich zur allgemeinen Agentenfunktion (3.8) vereinfachte Funktion:

$$action : S \to A. \tag{3.12}$$

Wir bemerken, dass dazu in der Funktion (3.8) der Definitionsbereich S^* durch S ersetzt wird.

Wir verfeinern nun die Modellierung der Entscheidungsfähigkeit eines Agenten. Dazu führen wir ein Wahrnehmungs- und ein Handlungssubsystem ein. Wesentlicher Bestandteil des Wahrnehmungssubsystems ist die Funktion *see*, die dazu dient, die Umgebung zu beobachten. Die Menge der möglichen Wahrnehmungen eines Agenten wird mit P bezeichnet. Wir erhalten:

$$see : S \longrightarrow P. \tag{3.13}$$

Übungsaufgabe 3.3 (Umsetzung der see-Funktion) Beschreiben Sie jeweils anhand eines Beispiels wie die see-Funktion im Falle eines physischen Agenten und im Falle eines Softwareagenten realisiert werden kann.

Die Funktion *action* ist der wesentliche Bestandteil des Handlungssubsystems. Da der Agent nun seine Aktionen auf Basis seiner Wahrnehmungen auswählt, ist die Funktion *action* zu modifizieren. Wir erhalten:

$$action : P^* \longrightarrow A, \tag{3.14}$$

wobei P^* die Menge von Folgen von Wahrnehmungen darstellt. Offensichtlich ergibt sich dann eine konkrete Aktion a durch Anwenden der Funktion *action* auf eine Folge $(p_1, \ldots, p_n)$ mit $p_i \in P, i = 1, \ldots, n$. Der Zusammenhang zwischen Wahrnehmungs- und Handlungssubsystem ist in Abbildung 3.3 gezeigt.

Wir haben bisher Agenten beschrieben, deren Entscheidungsfindung durch eine Historie beeinflusst wird. Das ist kein besonders intuitiver Ansatz. Wir betrachten deshalb Agenten, die einen internen Status pflegen. Derartige Agenten besitzen typischerweise Datenstrukturen, die Informationen über den Status der Umgebung und die Historie aufzeichnen. Solche Agenten heißen zustandsbasiert. Wir bezeichnen die Menge der möglichen internen Stati des Agenten mit I.

Die Entscheidungsfindung des Agenten kann zumindest teilweise unter Verwendung von I erfolgen. Dazu modifizieren wir erneut die *action*-Funktion. Wir erhalten:

$$action : I \longrightarrow A. \tag{3.15}$$

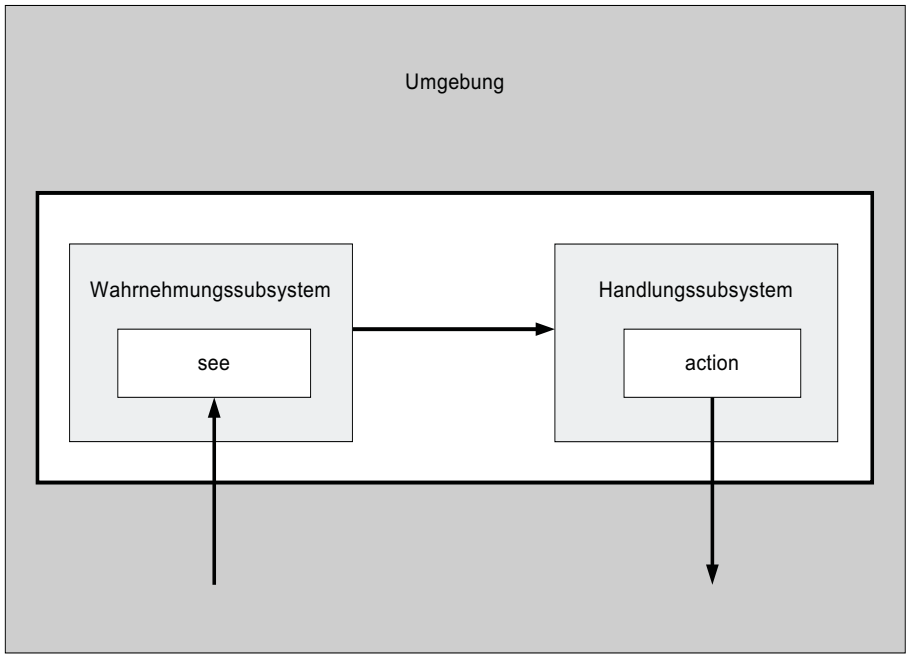

Abbildung 3.3: Wahrnehmungs- und Handlungssubsystem

Das bedeutet, dass die als Nächstes auszuführende Aktion des Agenten von seinem internen Status abhängig ist. Im Gegensatz zur *action*-Funktion in Beziehung (3.14) wird eine Folge von Wahrnehmungen nicht mehr länger benötigt. Es ist aber eine zusätzliche Funktion *next* erforderlich, die den internen Zustand des Agenten in Abhängigkeit von der Wahrnehmung der Umgebung verändert. Diese Funktion sieht wie folgt aus:

$$next : I \times P \longrightarrow I. \tag{3.16}$$

Die Funktion *next* ordnet einem internen Zustand und einer Wahrnehmung einen neuen internen Zustand zu. Auf diese Art und Weise erfolgt implizit eine Speicherung der Historie bzw. der Folge der Wahrnehmungen.

Die nachfolgende Schrittfolge wird von einem zustandsbasierten Agenten durchlaufen, um eine neue Aktion auszuwählen und auszuführen.

1. Lege den initialen internen Zustand i_0 des Agenten fest. Bestimme außerdem den initialen Zustand der Umgebung s_0.

2. Der Agent kann jetzt seine Umgebung durch Auswerten von $see(s_0)$ beobachten. Das Ergebnis ist eine Wahrnehmung p_1.

3. Durch Ausführen von $next(i_0, p_1)$ kann der Agent seinen internen Zustand anpassen. Der aktuelle Zustand ist i_1.

4. Die als nächste auszuführende Aktion ist durch $a_1 = action(i_1)$ gegeben.

5. Der Agent verändert seine Umgebung durch Ausführen der Aktion a_1.

Der Algorithmus zur Berechnung einer neuen Aktion ist in Abbildung 3.4 veranschaulicht.

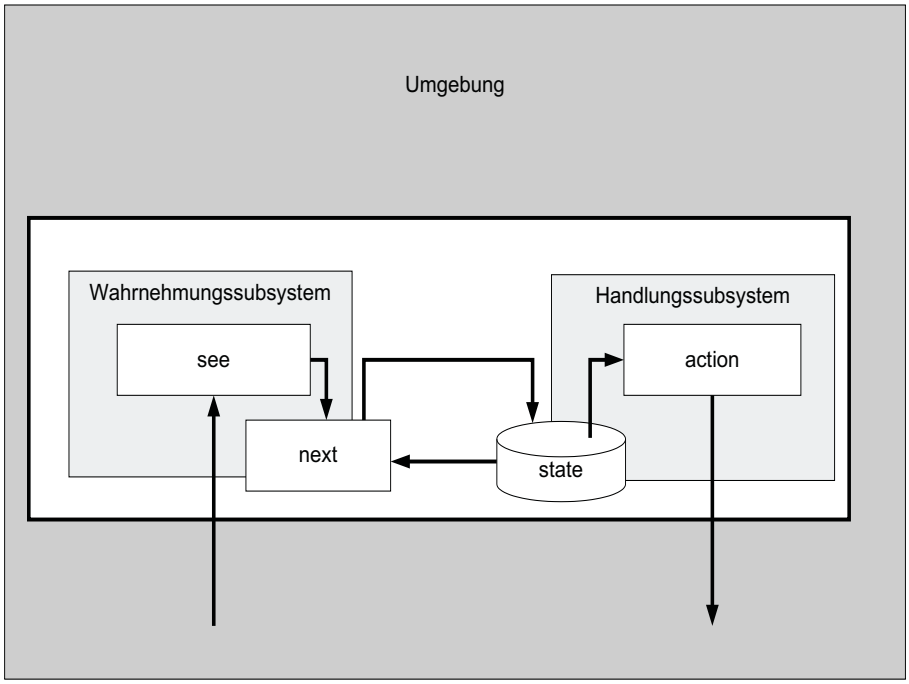

Abbildung 3.4: Zustandsbasierter Agent

Der interne Zustand des Agenten wird in einer Wissensbasis abgelegt. Darunter kann man sich im einfachsten Fall eine Textdatei vorstellen, es sind aber auch Datenbanken oder sogar Expertensysteme zur Abbildung von Regeln möglich.

3.1.4.2 Reaktive Agenten

Die reaktive Agentenarchitektur ist die einfachste mögliche Architektur. Wir diskutieren zunächst grundlegende Eigenschaften reaktiver Systeme. Reaktive Systeme besitzen die folgenden Eigenschaften:

- Es fehlt ein symbolisches Modell der Umwelt, eine explizite Wissensrepräsentation ist nicht vorhanden.

- Abstrakte Schlussfolgerungsmöglichkeiten sind nicht vorhanden.

- Es liegt ein reflexartiges Stimulus-Response-Reaktionsmuster vor, d.h., Reize führen unmittelbar zu Reaktionen, wobei nicht interessant ist, wie die Reaktionen intern zustande kommen.

Die gedächtnislose *action*-Funktion (3.13) ist für reaktive Agenten charakteristisch. Reaktive Agenten beziehen ihre Intelligenz aus der Interaktion mit der Umgebung. Insbesondere eine ständige Interaktion mit anderen Agenten ermöglicht und erhöht die Intelligenz. An dieser Stelle wird von emergentem Verhalten gesprochen. Unter dem Begriff der Emergenz versteht man das sinnvolle Zusammenwirken unterschiedlicher Agenten, so dass die Fähigkeiten des Gesamtsystems weit über die Fähigkeiten eines einzelnen Agenten hinausgehen. Für reaktive Agenten ist typisch, dass ihr Verhalten durch Regeln dargestellt wird, die bei Vorliegen einer gewissen Situation zu einer bestimmten Aktion führen.

Wir betrachten nachfolgend die Subsumption Architecture für reaktive Agenten, die von Brooks [4] vorgeschlagen wurde. Subsumption bedeutet Unterordnung. Die Architektur besteht aus unterschiedlichen Kompetenzmodulen. Jedes dieser Module ist für die Realisierung einer bestimmten Verhaltensweise des Agenten zuständig. Grundlegende, primitive Aufgaben sind in der Hierarchie weiter unten angesiedelt, während höherliegende Module komplexere Verhaltensweisen realisieren. Die einzelnen Module arbeiten autonom, aber übergeordnete Module schließen die Aufgaben untergeordneter Module mit ein.

Beispiel 3.1.4 (Robotersteuerung entsprechend der Subsumption Architecture) Ein entfernter Planet soll erkundet werden. Unter Verwendung eines Roboters sollen Gesteinsproben gesammelt werden. Wir betrachten dazu eine Robotersteuerung, die aus drei Kompetenzmodulen besteht.

- **Kompetenzmodul 0**: Das Modul dient dazu, Zusammenstöße mit anderen Objekten zu verhindern. Durch Sensoren können potentielle Hindernisse erkannt werden.

- **Kompetenzmodul 1**: Die Fähigkeiten des Roboters werden erweitert. Der Roboter ist nun in der Lage, beliebig in der Gegend umherzuwandern. Dazu muss er allerdings Hindernissen ausweichen können. Modul 1 kann die Ein- und Ausgaben von Modul 0 überwachen und beeinflussen. Auf diese Art und Weise wird die Funktionalität von Modul 0 beeinflusst.

- **Kompetenzmodul 2**: Dieses Modul setzt auf Modul 1 auf, indem der Roboter dazu befähigt wird, seine Umwelt zu erkunden. Dazu muss er in der Lage sein, sich zielgerichtet zu bewegen. Das Kompetenzmodul 2 beinhaltet Fähigkeiten von Modul 1, da zur Erkundung der Umgebung nach entfernten Gegenständen Ausschau gehalten und danach dorthin gewandert werden muss.

Wir beschreiben nachfolgend, wie die Einflussnahme auf die Funktionalität eines untergeordneten Moduls durch ein übergeordnetes erfolgt:

1. Einführung von Unterdrückungsknoten, die Eingangssignale abhören und diese gegebenenfalls modifizieren

2. Einführung von Verbotsknoten, welche die Ausgabe eines bestimmten Signals für einen bestimmten Zeitraum verbieten.

Die Subsumption Architecture kann in einer dynamischen und schwer einsehbaren Umgebung eingesetzt werden, die schnelle Reaktionen zwingend erfordert. Die verteilte Kontrollstruktur setzt aber der Planungs- und Lernfähigkeit enge Grenzen. Eine Abbildung komplexer Modelle ist in einer solchen einfachen Modulstruktur nicht möglich. Falls ein komplexes Verhalten erzeugt werden soll, ist es erforderlich, eine große Anzahl von Modulen bereitzustellen. Ein solches Vorgehen erfordert aber einen hohen Verwaltungs- und Synchronisationsaufwand.

Wir stellen im folgenden Unterabschnitt zunächst Belief-Desire-Intention-Architekturen vor. Diese Architekturen stellen das Gegenstück zu den reaktiven Agentenarchitekturen dar. Da Belief-Desire-Intention-Architekturen ähnlich wie die reaktiven Architekturen über eine Reihe von Nachteilen verfügen, werden anschließend Schichtenarchitekturen als Kompromiss zwischen reaktiven und Belief-Desire-Intention-Architekturen in Unterabschnitt 3.1.4.4 diskutiert.

3.1.4.3 Belief-Desire-Intention-Architekturen

Wir beschreiben nun Belief-Desire-Intention-(BDI)-Architekturen in Anlehnung an [3]. BDI-Agenten werden auch als deliberative Agenten bezeichnet. Sie verfügen über ein explizites symbolisches Modell der Umgebung und können logische Schlussfolgerungen ziehen.

Die Umwelt des Agenten ist in einem internen Modell geeignet abzubilden. Da das interne Modell häufig komplex ist, ist es typischerweise nicht möglich, das symbolische Modell der Umgebung zur Ausführungszeit des Agenten zu verändern. Aus diesem Grund sind Agenten, denen eine BDI-Architektur zugrundeliegt, nur bedingt für einen Einsatz in einer dynamischen Umgebung geeignet.

Wir untersuchen nachfolgend, wie erreicht werden kann, dass BDI-Agenten zu logischen Schlussfolgerungen fähig sind. Dazu ist es erforderlich, den internen Zustand eines Agenten näher zu beschreiben. Der interne Zustand eines Agenten wird auch als mentaler Zustand des Agenten bezeichnet. Wir folgen [23] und geben zunächst die drei Bestandteile an, die zur Beschreibung des internen Zustands verwendet werden:

1. **Überzeugungen (Beliefs)**

2. **Wünsche (Desires)**

3. **Intentionen (Intentions)**.

Außerdem werden diese drei Grundbestandteile noch durch die beiden zusätzlichen Bestandteile **Ziele (Goals)** und **Pläne (Plans)** ergänzt [14]. Wir beschreiben nun in Anlehnung an [3] diese fünf Bestandteile der Reihe nach.

Überzeugungen dienen dazu, die wesentlichen Ansichten eines Agenten bezüglich seiner Umgebung zu modellieren. Mit Hilfe von Überzeugungen kann der Agent auch seine Erwartung in Bezug auf zukünftige Zustände der Umgebung ausdrücken.

Wünsche werden aus den Überzeugungen des Agenten abgeleitet. Sie dienen dazu, zukünftige Situationen der Umgebung aus Sicht des Agenten zu beurteilen. Die Wünsche, die ein Agent besitzt, können unter Umständen aber nicht umgesetzt werden. Außerdem ist es möglich, dass Wünsche konfliktär sind. Wir betrachten dazu das folgende Beispiel.

Beispiel 3.1.5 (Konfliktäre Wünsche) Ein Agent, der im Rahmen der Produktionssteuerung für die Maschinenbelegung einer bestimmten Maschine verantwortlich ist, kann als Wunsch formulieren, die Wartezeiten der Jobs auf Bearbeitung auf der Maschine zu minimieren. Gleichzeitig kann er daran interessiert sein, die Maschine möglichst hoch auszulasten. Der Konflikt zwischen diesen beiden Wünschen ist in der Betriebswirtschaftslehre als Dilemma der Ablaufplanung bekannt.

Ziele sind eine Teilmenge der Wünsche eines Agenten. Ein Agent kann daran arbeiten, seine Ziele zu erfüllen. Der potentielle Handlungsspielraum eines Agenten wird durch seine Ziele eingegrenzt.

Intentionen wiederum stellen in gewisser Weise eine Konkretisierung von Zielen dar. Typischerweise kann ein Agent nicht gleichzeitig alle Ziele verfolgen, deshalb sind die Ziele zu priorisieren und entsprechend ihrer so festgelegten Wichtigkeit zu behandeln. Wenn ein Agent sich dafür entscheidet, ein bestimmtes Ziel zu verfolgen, wird aus dem Ziel eine Intention.

Schließlich dienen Pläne dazu, Intentionen eines Agenten zu Einheiten einer bestimmten Granularität zusammenzufassen. Ein Plan besteht dabei aus verschiedenen Aktionen, die in einer bestimmten Reihenfolge oder auch parallel ausgeführt werden können. Eine Intention eines Agenten stellt einen Teilplan des Gesamtplans für diesen Agenten dar. Dabei ist zu beachten, dass mögliche Inkonsistenzen zwischen einzelnen Intentionen bei der Erstellung des Gesamtplans beseitigt werden.

Dabei wird wie folgt vorgegangen. Die Wissensbasis eines Agenten enthält das symbolische Umweltmodell dieses Agenten. Aus dem Umweltmodell werden Wünsche, Ziele und Intentionen abgeleitet. Die unterschiedlichen Intentionen werden von einer Planungskomponente, als Planer bezeichnet, zu einem konsistenten Gesamtplan zusammengefügt. Die Erstellung der Plä-

ne erfolgt inkrementell, d.h., wenn eine neue Intention dem Planer bekannt gegeben wird, untersucht dieser, ob die Intention Abhängigkeiten zu einem bestehenden Plan aufweist. Existierende Abhängigkeiten werden dann bei der Erstellung eines neuen Gesamtplans berücksichtigt.

Die Pläne werden anschließend an einen Scheduler übergeben, der den Aktionen eines Planes Startzeiten zuordnet und diese an eine Ausführungskomponente übergibt. Diese Komponente führt die Aktion aus, überwacht ihren Fortschritt und beendet die Ausführung gegebenenfalls vorzeitig, falls die zur Ausführung der Aktion zur Verfügung stehende Zeit überschritten wird. Außerdem kann die Ausführungskomponente dafür sorgen, dass die Ausführung der Aktion zu einem späteren Zeitpunkt fortgesetzt wird.

Wir diskutieren nun die BDI-Architektur von Rao und Georgeff [23]. Der Ausgangspunkt für diesen Ansatz liegt in den folgenden zwei Erkenntnissen:

1. Zu einem Zeitpunkt $t \geq 0$ existieren viele Möglichkeiten, wie sich die Umwelt des Agenten entwickeln kann.

2. Es existieren gleichzeitig viele Möglichkeiten für die Aktionen, die ein Agent zum Zeitpunkt $t \geq 0$ durchführen kann.

Diese beiden Erkenntnisse sagen aus, dass die Agenten und die Umgebung ein nicht-deterministisches System bilden. Eine geeignete Struktur zur Darstellung derartiger Systeme bilden Entscheidungsbäume. Ein solcher Baum enthält Knoten, die Umweltzustände und die möglichen Zustandsübergänge durch Agentenaktionen oder Umweltereignisse repräsentieren. Jeder Zweig des Baumes stellt einen möglichen Ausführungspfad dar. Zwei Arten von Knoten existieren. Wahlknoten (Choice Nodes) dienen dazu, Zustandsübergänge infolge von Agentenaktionen zu modellieren, während Chancen-Knoten Zustandsübergänge infolge von Ereignissen der Umwelt abbilden. Die Chancen-Knoten spiegeln die Tatsache wider, dass Umweltereignisse nicht vorhersehbar sind und nicht beeinflusst werden können.

Ein konkreter Pfad von der Wurzel des Baumes zu einem Blatt, d.h. einem Knoten ohne Nachfolger, stellt gleichzeitig die Erreichung eines bestimmten Ziels des Agenten dar. Durch die Auswahl der Aktionen, die zu Knoten gehören, die auf dem Pfad liegen, erhält man eine Folge von auszuführenden Aktionen. Probleme bereiten aber die Chancen-Knoten, da der Agent nicht beeinflussen kann, welches mit einem Chancenknoten assoziierte Ereignis tatsächlich auftritt.

Rao und Georgeff [23] schlagen deshalb den Begriff „mögliche Welten“ als Ansatz vor, um die Chancen-Knoten zu entfernen. Jede „mögliche Welt“ kann als Realisierung eines Entscheidungsbaums angesehen werden, der durch die Existenz von Chancen-Knoten als Zufallsvariable interpretiert werden kann. Der nachfolgende Algorithmus dient dazu, die „möglichen Welten“ zu erzeugen.

Algorithmus 3.1.1 (Ermittlung aller „möglichen Welten“): Gegeben ist ein Entscheidungsbaum T, der Wahlknoten und Chancen-Knoten enthält.

1. Durchlaufe von der Wurzel beginnend den Baum T, bis ein Chancen-Knoten CK erreicht wird.

2. Erzeuge für alle von diesen Chancen-Knoten ausgehenden Kanten jeweils einen neuen Entscheidungsbaum T', indem der Chancen-Knoten CK aus T entfernt wird und der Vaterknoten von CK mit der Wurzel eines der Teilbäume, die einen Kindknoten von CK als Wurzel haben, durch eine neue Kante verbunden ist.

3. Wiederhole die Schritte 1 und 2 so lange, bis alle Chancen-Knoten aus dem Baum T entfernt sind.

Als Ergebnis von Algorithmus 3.1.1 erhält man eine Menge von Bäumen, die keine Chancen-Knoten mehr enthalten. Jeder dieser Bäume stellt eine „mögliche Welt“ dar. Ein konkreter Baum ist mit einer bestimmten Eintrittswahrscheinlichkeit verbunden. Außerdem kann man jedem Pfad in einem der so erhaltenen Bäume einen Nutzenwert für die damit verbundene Zielerreichung zuordnen. Auf Basis der Wahrscheinlichkeiten und der Nutzenwerte kann ein Agent entscheiden, welche Aktionen auszuführen sind.

Das Vorgehen in Algorithmus 3.1.1 ist beispielhaft in Abbildung 3.5 veranschaulicht. Dabei stellen die grau gefärbten Knoten Wahlknoten dar. Die weiß gefärbten Knoten sind Chancen-Knoten.

In Teil a) von Abbildung 3.5 wird zunächst der Entscheidungsbaum dargestellt. Aus diesem Baum entstehen in den Teilabbildungen b), c) und d) die drei „möglichen Welten“ durch Entfernen von Knoten 3.

In [23] wird eine formale Logik entwickelt, die für die Schlussfolgerungsprozesse eines Agenten verwendet werden kann. Dieser Ansatz wird im Rahmen dieses Kurses nicht weiterverfolgt. Wir diskutieren aber kurz einen BDI-Interpreter, der besser als die rein logik-basierten Ansätze für praktische Zwecke geeignet ist. Das gilt insbesondere für die Zeit, die für die Auswahl der nächsten auszuführenden Aktion zur Verfügung steht, da Methoden des maschinengestützten Beweisens in den logik-basierten Ansätzen typischerweise sehr zeitaufwendig sind.

Der Interpreter besteht aus jeweils einer Datenstruktur für Überzeugungen, Wünsche und Intentionen. Außerdem wird eine Warteschlange für Ereignisse verwendet. Der dem Interpreter zugrundeliegende Algorithmus [23] wird nachfolgend angegeben.

Algorithmus 3.1.2 (BDI-Interpreter): Die folgenden Schritte werden im Rahmen eines iterativen Vorgehens ausgeführt:

initialize_state();

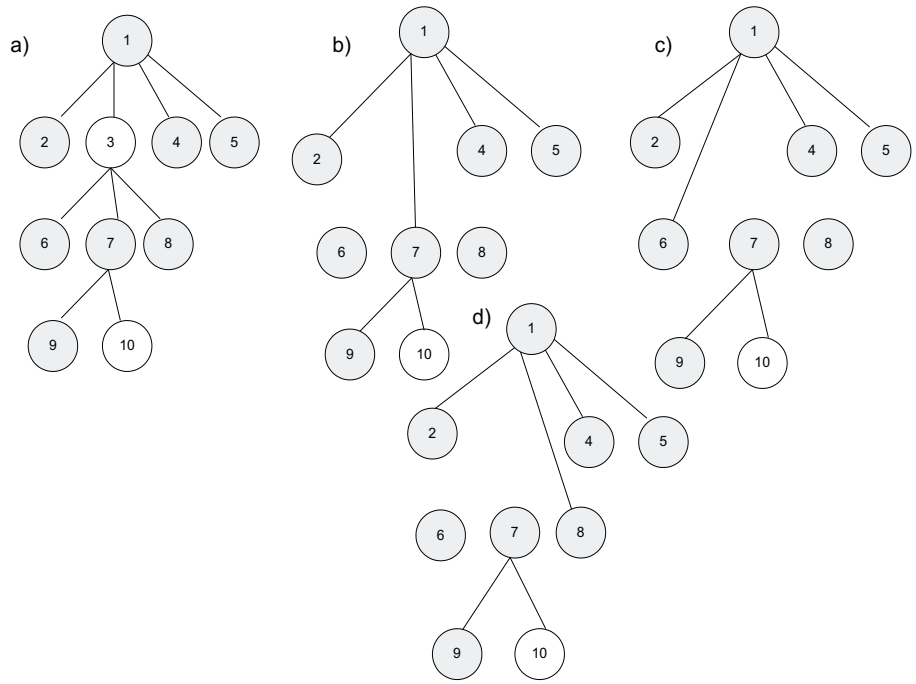

Abbildung 3.5: „Mögliche Welten“ in Anlehnung an [3]

```
Repeat until
        options := option_generator(event-queue);
        selected_options := deliberator(options);
        update_intentions(selected_options);
        execute();
        get_new_external_events();
        drop_successfull_attitudes();
        drop_impossible_attitudes();
end repeat;
```

In Algorithmus 3.1.2 wird zunächst eine Initialisierung vorgenommen, indem der Zustand der Umwelt gelesen wird. Dadurch werden die Überzeugungen des Agenten bestimmt. Anschließend werden verschiedene Schritte im Rahmen eines Schleifendurchlaufs wiederholt. Zunächst wird die Warteschlange auf Ereignisse überprüft. Die so ermittelten Ereignisse werden dazu verwendet, eine Menge von Optionen zu bestimmen. Optionen stellen im Sinne der BDI-Begrifflichkeiten Wünsche, also prinzipiell mögliche Ziele, dar. Diese hängen dabei von der konkreten Umweltsituation ab. Im nächsten Schritt werden vom Interpreter diejenigen Optionen ausgewählt, die für Agenten unter Berücksichtigung der aktuell zur Verfügung stehenden Ressourcen den größten Nutzen versprechen. Diese Menge wird dann durch den Interpreter zu den bereits vorhandenen Intentionen hinzugefügt,

denn der Agent möchte die ausgewählten Optionen erreichen. Außerdem ist es eventuell wegen neuer Umweltinformationen erforderlich, bereits bestehende Intentionen zu verändern. Anschließend werden im nächsten Schritt die aktuell anstehenden Aktionen durch den Interpreter ausgeführt, um die Intentionen zu erreichen. Dann wird die Ereigniswarteschlange aktualisiert. Das wird dadurch erreicht, indem der Interpreter die Umwelt auf neue Ereignisse untersucht und diese anschließend in die Warteschlange einfügt. Am Ende eines Schleifendurchlaufs werden alle erreichten Wünsche und Intentionen aus der jeweiligen Datenstruktur entfernt. Das gleiche gilt für Wünsche und Intentionen, die derzeit nicht erreicht werden können. Anschließend beginnt der Interpreter erneut mit einem weiteren Schleifendurchlauf.

Wir bemerken, dass die konkrete Ausgestaltung der Erzeugung von Optionen sowie der Ermittlung der Optionen offen bleibt. Hier besteht in praktischen Anwendungen Konkrektisierungsbedarf.

Wir diskutieren abschließend kurz die Nachteile von BDI-Architekturen. Ein wesentlicher Nachteil besteht darin, dass BDI-Agenten in dynamischen Umgebungen Probleme haben, ihr symbolisches Umweltmodell schnell genug an die aktuelle Umweltsituation anzupassen. Dieser Nachteil wird durch das planbasierte Vorgehen von BDI-Agenten noch verschärft. Der Umweltzustand kann sich während der Erstellung und Ausführung des Planes bereits verändern, da die Erstellung von Plänen, die einem bestimmten Optimalitätskriterium genügen, typischerweise sehr zeitaufwendig ist. In vielen praktischen Situationen wird es günstiger sein, den Optimalitätsanspruch zu Gunsten schneller berechenbarer Pläne aufzugeben, um damit auf einen veränderten Zustand der Umgebung besser reagieren zu können.

3.1.4.4 Schichtenarchitekturen

Typischerweise werden Agenten benötigt, die gleichzeitig reaktive und proaktive Eigenschaften besitzen [27]. Die Idee besteht deshalb darin, Subsysteme bereitzustellen, die über beide Eigenschaften verfügen. Das führt zu einer Klasse von Architekturen, in denen solche Subsysteme als eine Hierarchie interagierender Schichten angeordnet sind. Es wird wenigstens eine Schicht zur Bereitstellung reaktiven Verhaltens und eine weitere Schicht für proaktives Verhalten benötigt.

Wir unterscheiden zwischen zwei prinzipiellen Architekturen:

1. Architekturen mit horizontalen Schichten

2. Architekturen mit vertikalen Schichten.

Architekturen mit horizontalen Schichten sind dadurch charakterisiert, dass die einzelnen Schichten direkt mit Sensoren ausgestattet sind und auch Aktionen ausführen können. Jede Schicht agiert dabei wie ein eigenständiger

Agent, der Empfehlungen für bestimmte Handlungen produziert. Bei vertikalen Schichten ist hingegen höchstens eine Schicht mit Sensoren ausgestattet und in der Lage, Aktionen vorzuschlagen.

Architekturen mit horizontalen Schichten basieren auf einem einfachen Konzept. Falls ein Agent, der diesem Architekturparadigma folgt, n Verhaltensausprägungen unterstützen soll, ist die Implementierung n unterschiedlicher Schichten erforderlich. Gleichzeitig findet aber ein Wettbewerb zwischen den Schichten bezüglich der durchzusetzenden Aktionen statt. Zur Sicherstellung des gewünschten Agentenverhaltens ist es deshalb zweckmäßig, dass ein Mediator verwendet wird. Dieser steuert, welche Schichten zu einem bestimmten Zeitpunkt das Verhalten des Agenten durch Aktionen bestimmen. Wenn n Schichten vorhanden sind, von denen jede m Aktionen zur Verfügung stellt, dann gibt es m^n mögliche Kombinationen von Aktionen, die vom Mediator zu berücksichtigen sind. Der Mediator kann aus diesem Grund zum Engpass werden.

Für Architekturen mit vertikalen Schichten liegen unterschiedliche Gestaltungsoptionen vor. Wir unterscheiden an dieser Stelle zwischen:

1. Achitekturen mit einem Durchlauf (One-Pass-Architekturen)

2. Architekturen mit zwei Durchläufen (Two-Pass-Architekturen).

Bei One-Pass-Architekturen geht der Kontrollfluss von einer Schicht zur nächsten über, wobei erst die letzte Schicht die eigentliche Aktion erzeugt. Bei Two-Pass-Architekturen hingegen fließen zunächst Informationen von den unteren zu den oberen Schichten. Das wird als erster Durchlauf (First Pass) bezeichnet. Im zweiten Durchlauf, auch als Second Pass bezeichnet, wird dann die Kontrolle schrittweise von den oberen an die unteren Schichten abgegeben. Ähnliche Vorgehensweisen lassen sich auch in betrieblichen Organisationen beobachten.

Als vorteilhaft ist der geringere Aufwand im Vergleich zu horizontalen Schichten anzusehen. Um das einzusehen, nehmen wir wiederum an, dass n Schichten vorhanden sind, die jeweils m mögliche Aktionen zur Verfügung stellen. Da $n-1$ Schnittstellen zwischen den Schichten existieren, sind insgesamt lediglich

$$\sum_{i=1}^{n-1} m^2 = (n-1)m^2 \tag{3.17}$$

Interaktionen zwischen den Schichten möglich. Nachteilig ist, dass für eine Entscheidungsfindung jede Schicht zu durchlaufen ist. Wir stellen die verschiedenen Gestaltungsmöglichkeiten für Schichtenarchitekturen zusammenfassend in Abbildung 3.6 dar.

Wir diskutieren nachfolgend die **Inte**gration of **R**eactive Behaviour and **Ra**tional **P**lanning (InteRRaP)-Architektur [14] als Beispiel für eine vertikale Schichtenarchitektur. Die Architektur umfasst die folgenden drei Schichten:

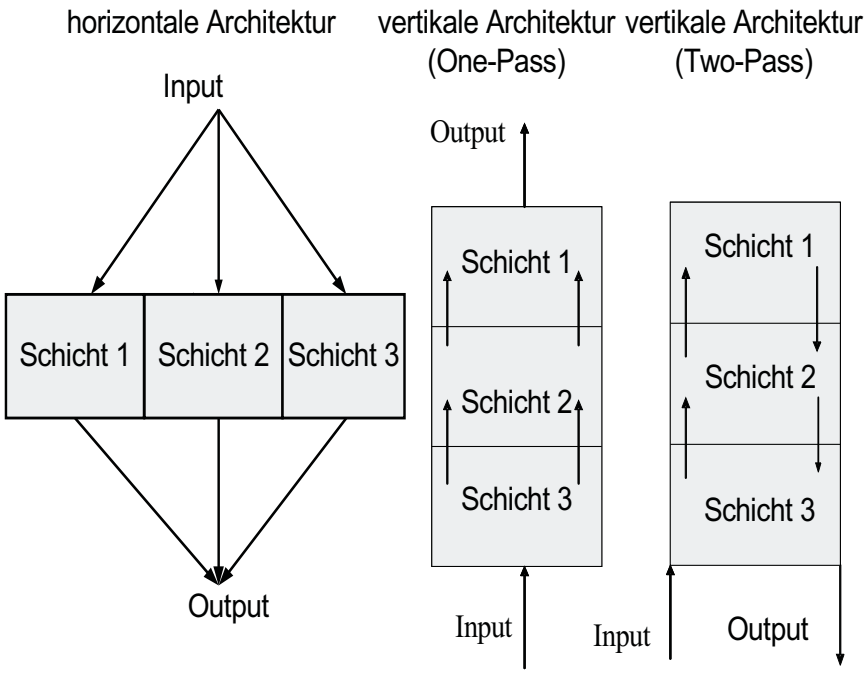

Abbildung 3.6: Schichtenarchitekturen

1. verhaltensbasierte Schicht
2. lokale Planungsschicht
3. kooperative Planungsschicht.

Die verhaltensbasierte Schicht stellt die unterste Schicht dar. Sie ermöglicht ein reaktives Verhalten des Agenten. Die lokale Planungsschicht bildet die mittlere Schicht. Sie ermöglicht Planungsaktivitäten, um die Ziele des Agenten zu erreichen. Die kooperative Planungsschicht ist die oberste Schicht von Agenten, die der InteRRaP-Architektur folgen. Sie dient dazu, soziale Interaktionen des Agenten mit der Umgebung, insbesondere mit anderen Agenten, zu ermöglichen. Dadurch können gemeinsame Ziele der beteiligten Agenten erreicht werden.

Jeder Agent in InteRRaP besteht aus einer Wissensbasis, einer Kontrolleinheit sowie aus einer sogenannten Weltschnittstelle. Diese Schnittstelle ermöglicht den Kontakt zwischen Agent und Umwelt. Die Wissensbasis enthält die Überzeugungsmodelle. Die Kontrolleinheit besteht aus drei Schichten. Die verhaltensbasierte Schicht (VBS) bildet die unterste Schicht. Sie bildet die reaktiven Fähigkeiten des Agenten ab. Die lokale Planungsschicht (LPS) und die kooperative Planungsschicht (KPS) werden dann angewendet, wenn die vorliegende Situation von der verhaltensbasierten Schicht nicht behandelt werden kann. Das bedeutet, dass eine längerfristige Zielfindung und Planung

notwendig ist. Jede der drei Schichten enthält ein Situationserkennungs-/Zielaktivierungsmodul, das alle Aufgaben bis zur Optionserzeugung behandelt. Dieses Modul wird mit SZ abgekürzt. Außerdem besitzt jede der drei Schichten ein Planungs-/Schedulingmodul. Für dieses Modul verwenden wir die Abkürzung PS. Es wird zum Ermitteln von Intentionen und der eigentlichen Ablaufplanung verwendet.

Jede Schicht der Kontrolleinheit kann stets nur auf die ihr zugeordnete Schicht der Wissensbasis und auf darunterliegende Schichten zugreifen. Der Kontrollfluss fließt von den unteren zu den übergeordneten Schichten. Daraus folgt, dass lediglich die verhaltensbasierte Schicht auf die Weltschnittstelle zugreifen kann. Die kooperative Schicht hingegen hat Zugriff auf alle Teilmodelle der Wissensbasis. Die InteRRaP-Architektur ist in Abbildung 3.7 dargestellt.

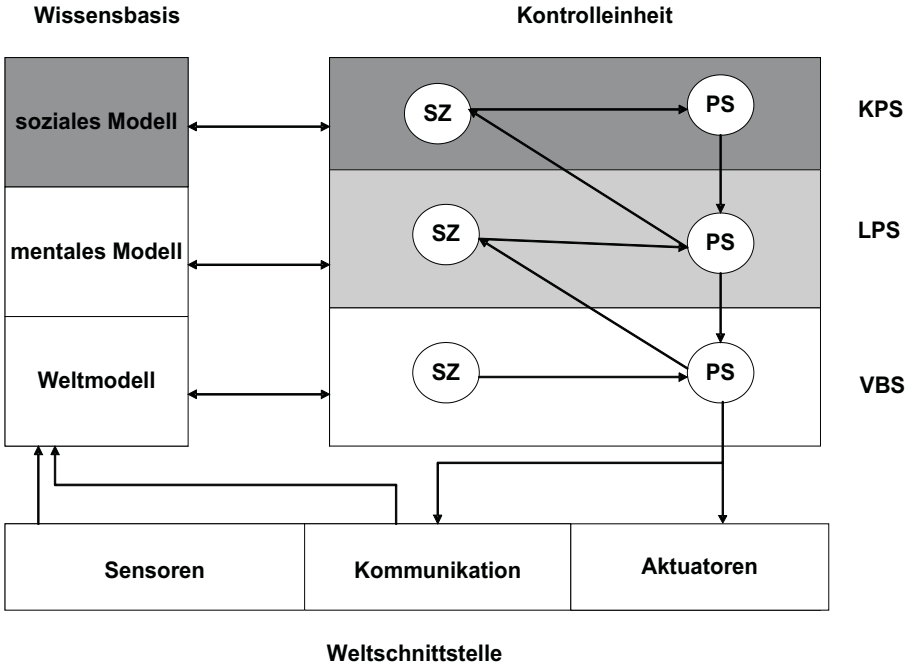

Abbildung 3.7: InteRRaP-Architektur in Anlehnung an [3]

Das Zusammenwirken der drei Schichten ist in InteRRaP wie folgt organisiert. Falls eine Schicht nicht in der Lage ist, auf eine aktuelle Situation reagieren zu können, gibt diese Schicht die Kontrolle an die nächsthöhere Schicht ab. Dieser Vorgang wird als Bottom-Up-Aktivierung bezeichnet. Dem gegenüber steht eine Top-Down-Aktivierung, bei der eine Schicht die Dienste der direkt untergeordneten Schicht benutzt, um ihre eigenen Ziele zu erreichen. Der grundsätzliche Kontrollfluss sieht wie folgt aus:

1. Über Sensoren nimmt die verhaltensbasierte Schicht die Umgebung

wahr. Falls eine reaktive Behandlung durch diese Schicht möglich ist, wird diese durchgeführt.

2. Falls das nicht möglich ist, geht im Rahmen einer Bottom-Up-Aktivierung die Kontrolle an die lokale Planungsschicht über.

3. Wenn eine Behandlung durch die lokale Planungsschicht möglich ist, findet diese im Rahmen einer Top-Down-Ausführung statt.

4. Ansonsten geht die Kontrolle im Rahmen einer Bottom-Up-Aktivierung an die kooperative Planungsschicht über.

Wie bereits beschrieben, verläuft der Ausführungsprozess in InterRRaP von oben nach unten. Lediglich die verhaltensbasierte Schicht kann eine Aktion starten, da nur sie auf die Aktuatoren zugreifen kann. Intentionen, die in der lokalen und kooperativen Planungsschicht ermittelt werden, werden jeweils über die direkt untergeordnete Schicht bis zur verhaltensbasierten Schicht weitergeleitet. Damit stellt InteRRaP ein Beispiel für eine vertikale Two-Pass-Architektur dar.

Wir skizzieren nun, wie Aktionen in InteRRaP ermittelt werden. Der Agent nimmt die Umwelt unter Verwendung von Sensoren wahr. Der Umweltzustand wird dann in Überzeugungen (Beliefs) des Agenten über den Zustand der Umwelt umgewandelt. Die Überzeugungen sind in drei unterschiedliche Modelle aufgeteilt (siehe Abbildung 3.7):

1. Weltmodell

2. mentales Modell

3. soziales Modell.

Das Weltmodell enthält Überzeugungen bezüglich der Umwelt des Agenten. Das mentale Modell umfasst die Überzeugungen, die der Agent über sich selber besitzt. Das soziale Modell enthält schließlich die Überzeugungen, die der Agent über andere Agenten aus seiner Umgebung hat. Die im Weltmodell enthaltenen Überzeugungen werden für reaktive Zwecke verwendet. Überzeugungen, die im mentalen Modell beinhaltet sind, werden für deliberative Zwecke benutzt, während das soziale Modell dazu dient, eine Kooperation mit anderen Agenten zu ermöglichen.

Diese Überzeugungen, die noch nicht sehr konkret sind, werden dann im nächsten Schritt in Situationen umgewandelt. Situationen sind diejenigen Zustände der Umgebung, die für den Agenten zum aktuellen Zeitpunkt interessant sind. Die Situationen bilden offensichtlich eine Untermenge der Überzeugungen. In Analogie zu den Überzeugungen teilen sich Situationen in

1. Routine- bzw. Notfallsituationen

2. lokale Planungssituationen
3. Situationen, in denen eine kooperative Planung mehrerer Agenten erforderlich ist

auf. Zur Beschreibung von Situationen dienen die bereits dargestellten Überzeugungsmodelle. Wir betrachten dazu das nachfolgende Beispiel.

Beispiel 3.1.6 (Situationen in InteRRaP) Die Routine- und Notfallsituation basieren jeweils auf dem Weltmodell. Zur Beschreibung von lokalen Planungssituationen wird sowohl das Welt- als auch das mentale Modell herangezogen, da der Agent sowohl den Zustand der Umgebung als auch seine eigenen Fähigkeiten kennen muss.

Die ermittelten eingetretenen Situationen dienen dazu, eines oder mehrere Ziele zu aktivieren. Die Ziele lassen sich wiederum in

1. sehr kurzfristige Ziele
2. lokale Ziele
3. kooperative Ziele

unterteilen. Die sehr kurzfristigen Ziele sind Reaktionen auf einen bestimmten Zustand der Umgebung. Beim Eintreten einer bestimmten Situation erfolgt eine Aktivierung von einem oder mehreren Zielen des Agenten. Unter Optionen werden in InteRRaP diejenigen Ziele verstanden, die zu einer bestimmten Situation passen. Aus der Menge der Optionen werden ausführbare Intentionen abgeleitet. Die Intentionen werden unter Verwendung operationaler Primitive bestimmt. Diese sind vom jeweiligen Ziel bzw. der jeweiligen Option abhängig. Die sehr kurzfristigen Ziele werden durch Verhaltensmuster repräsentiert. Lokale und kooperative Ziele erfordern hingegen die explizite Ermittlung von Zielzuständen, bevor geplant werden kann. Die auf diese Weise ermittelten Intentionen werden dann ausgeführt.

Wir betrachten nachfolgend ein Beispiel aus der agentenbasierten Maschinenbelegungsplanung, um die verschiedenen Arten von Zielen zu erläutern.

Beispiel 3.1.7 (Agentenbasierte Maschinenbelegungsplanung) Ein Softwareagent ist für die Belegung einer Maschine M_1 mit Jobs (Fertigungsaufträgen) verantwortlich. Dazu berechnet der Softwareagent Ablaufpläne für die Jobs, die vor der Maschine auf Bearbeitung warten. Diese Berechnung kann zeitlich aufwendig sein. Dabei werden gewünschte Fertigstellungstermine für die Jobs berücksichtigt. Wenn der Jobagent darüber informiert wird, dass der ihm zugeordnete Job in einer Woche statt erst in zwei Wochen an einen Kunden zu liefern ist, wird situationsbedingt vom Ablaufplan abgewichen und der Job als Nächster auf der Maschine M_1 bearbeitet. Es liegt somit eine Notfallsituation vor.

Ein Schichtbeginn im Unternehmen stellt hingegen eine Situation dar, die eine Berechnung neuer Ablaufpläne für die Jobs vor der Maschine erfordert. Es liegt somit eine lokale Planungssituation dar.

Falls die Bearbeitung eines Teils der Jobs auf einer Maschine M_2 fortgesetzt wird, die gerade zum Engpass geworden ist, ist es sinnvoll, diese Information bei der Ablaufplanung für Maschine M_1 zu berücksichtigen. Jobs, die nicht auf M_2 bearbeitet werden, sind dann weniger wichtig als die Jobs, die für den Engpass vorgesehen sind. Eine kooperative Planung findet statt.