3 Aspekte

In diesem Kapitel werden Ausprägungsvarianten der herausgearbeiteten Konzepte diskutiert und für das zu entwickelnde Framework zugänglich gemacht. Es nutzt dafür Erfahrungen aus den Gebieten der Wissensrepräsentation, der verteilten objektorientierten Programmierung und der Literatur zur Agenten-Technik. Dabei geht es um die Repräsentation von Informationen innerhalb des Agenten, die Repräsentation von Aktionen, die Aktionsauswahl, die Ablaufsteuerung, die Kooperation zwischen Agenten, die Sicherheit und das Management von Agenten. Jedes der folgenden Unterkapitel nennt zunächst Informationsquellen und Anforderungen, stellt für die Konzepte relevante Arbeiten vor und beschreibt die Einfügung unterschiedlicher Ausprägungen in das Framework. Ein besonderes Augenmerk liegt auf der Diskussion der Flexibilität und der Effizienz der benutzten Ausprägungen sowie dem Abwägen zwischen beiden.

3.1 Wissensrepräsentation und Aktualisierung

3.1.1 Aufgaben und Quellen

Die angemessene Repräsentation der Informationen, die einem Agenten über seine Umgebung und über sich selbst zur Verfügung stehen und auf die der Agent sein Verhalten einstellt, spielt eine entscheidende Rolle beim Entwurf eines Agenten. Die richtige Wahl der zu speichernden Informationen und die Art ihrer Darstellung beeinflussen im Besonderen die Effizienz und die Flexibilität des Agenten. Sind wichtige Informationen schnell verfügbar, so kann der Agent entsprechend schnell reagieren. Durch die symbolische Repräsentation von Fähigkeiten ist ein Agent zum Beispiel in der Lage, diese zur Laufzeit einzuschätzen, anzupassen oder zu kombinieren. Er kann so im Gegensatz zu fest hineinkompilierten Fähigkeiten flexibel auf die aktuellen Informationen reagieren. Die Flexibilität umfasst dann mehr als eine Fallunterscheidung.

Die Repräsentation der Informationen ist eng mit der Art ihrer Nutzung (Ableiten neuer Informationen, Entscheidungsfindung) verknüpft. Hier soll es aber zunächst nur um die Repräsentation (was wird repräsentiert und wie) gehen. Die Verarbeitung der Informationen wird in späteren Kapiteln besprochen.

Eine weitere Frage bezüglich der gespeicherten Informationen ist die ihrer Aktualisierung. Welche Informationen können wann und unter welchen Umständen ergänzt, verändert oder gelöscht werden? Die hier vorgesehenen Varianten beeinflussen ebenfalls die Flexibilität eines Agenten.

Eine wichtige Quelle für dieses Kapitel sind die Arbeiten auf dem Gebiet der Wissensrepräsentation. Viele verschiedene Repräsentationsformen wurden in

diesem grundlegenden Bereich der Künstlichen Intelligenz vorgeschlagen und untersucht ([Ginsberg 88, Sowa 91, Bibel et al. 93, Aamodt/Plaza 94, Messing 96]). Diese Untersuchungen wurden in ihren unterschiedlichen Formen auch in die entsprechenden Arbeiten über Agenten einbezogen. Es werden hier deshalb nur Arbeiten in Betracht gezogen, die Arten der Wissensrepräsentation in Bezug auf Agenten vorstellen. Es wird jeweils untersucht, welche Informationen wie repräsentiert sind und aktualisiert werden, und das Verhältnis zwischen Flexibilität und Effizienz beleuchtet.

Wissensrepräsentation in Agent0

In der von Yoav Shoham vorgeschlagenen Sprache für die agentenorientierte Programmierung ([Shoham 93]) werden für einen Agenten vier Arten von Informationen gespeichert, auf die sich die Abarbeitung des Agenten stützt:

- die Fähigkeiten des Agenten,
- die Annahmen des Agenten über die Umgebung,
- die Reaktionsregeln des Agenten und
- die eingegangenen Verpflichtungen

Die Fähigkeiten sind zunächst nur durch ihren Namen und ihre Argumente beschrieben. Die Folge von Aktionen, die sich hinter ihnen verbirgt, muss man in der Wirtssprache (Lisp oder Prolog) programmieren. Es sind allerdings einige Standardaktionen wie das Informieren eines anderen Agenten oder das Stellen einer Anfrage vorgegeben. Abhängig von der Wirtssprache können Fähigkeiten und zugehörige Aktionen auch zur Laufzeit erzeugt und ergänzt werden.

Die Annahmen eines Agenten betreffen die Gültigkeit von Fakten zu einer bestimmten Zeit oder in einer bestimmten Zeitspanne. Sie sind deshalb nicht für die Darstellung von Gesetzmäßigkeiten, sondern für eine Zustandsbeschreibung gedacht. Gesetzmäßigkeiten müsste man in der Wirtssprache ausdrücken. Für die Fakten sieht Agent0 keine Spezialisierungen vor: Ein Agent kann Annahmen über andere Agenten oder Annahmen über die Umgebung speichern, er wird aber nicht speziell unterstützt.

Im Sinne der bisher in diesem Text verwendeten Begriffswelt beschreiben die Reaktionsregeln die Handlungsvarianten eines Agenten: Auslösebedingungen und auszuführende Aktionen. Aktionen werden durch eintreffende Nachrichten ausgelöst. Durch die Nachrichten und aufgestellte Bedingungen ist festgelegt, welche Verpflichtungen der Agent eingehen wird. Die Reaktionsregeln eines Agenten werden üblicherweise vor dessen Start programmiert, es spricht aber nichts dagegen, sie während der Laufzeit zu ändern oder zu ergänzen.

Verpflichtungen sind mit einer Ausführungszeit und Ausführungsbedingungen versehene Aktionen. Sie werden im Allgemeinen als ein Ergebnis einer Reaktionsregel in die Wissensbasis aufgenommen.

Sowohl die Reaktionsregeln als auch die Verpflichtungen sind an Bedingungen gebunden. Diese können davon abhängen, ob der Agent die Wahrheit eines Faktes zu einer bestimmten Zeit annimmt, ob der Agent eine bestimmte Fähigkeit besitzt oder ob der Agent schon bestimmte Verpflichtungen eingegangen ist. Weitere Bedingungen können in der Wirtssprache beschrieben werden.

Es ist in Agent0 nicht von vornherein vorgesehen, Informationen zur momentanen Absicht und zur momentanen Aktivität zu speichern. Speichert man diese Informationen allerdings nicht, dann führt ein eventueller Fehler in der momentanen Aktionsfolge (die die Absicht umsetzen soll) dazu, dass der Agent die Absicht danach nicht weiterverfolgt. Es ist ja dann nicht klar, dass die Aktionsfolge die Absicht umsetzen sollte. Mit den Verpflichtungen speichert der Agent zwar zukünftige Handlungen, die Absicht dahinter ist als solche jedoch nicht gespeichert.

Es gibt in Agent0 ebenfalls keine explizite Darstellungsform für Informationen über den Nachrichtenaustausch und die Kooperation zwischen Agenten. Einige Nachrichtenfolgen und die Reaktionen auf bestimmte Nachrichten sind fest einprogrammiert.

Agent0 sieht einen fest verankerten Weg der Aktualisierung von Informationen vor. Diese Aktualisierung betrifft nur die Annahmen eines Agenten. Sie findet als Reaktion auf das Eintreffen einer Nachricht statt. Entsprechend der erhaltenen Information wird der zeitliche Wahrheits- oder Werteverlauf einer Annahme verändert.

Die Flexibilität eines Agenten ist von der Wirtssprache abhängig. Ein Agent0-Agent kann sehr flexibel reagieren: Fähigkeiten und Handlungsvarianten könnten zur Laufzeit ergänzt oder verändert werden. Die dazu notwendigen Mechanismen müssten allerdings in der Wirtssprache implementiert werden, Agent0 unterstützt sie nicht selbst, verhindert sie aber auch nicht. Es gibt Teile der Implementierung eines Agenten, die außerhalb der eigentlichen Sprache Agent0 liegen. Das betrifft insbesondere die bei der Anwendung von Fähigkeiten auszuführenden Aktionen und Gesetzmäßigkeiten der Umgebung, die dargestellt werden sollen. Abhängig von der Wirtssprache sind diese für den Agenten transparent oder gekapselt.

Wissensrepräsentation in Concurrent MetateM

In Concurrent MetateM ([Fisher 94]) ist das Verhalten eines Agenten vollständig in einer temporalen Logik beschrieben. Eine Menge von Regeln gibt vor, unter welchen jetzt oder früher geltenden Bedingungen welche jetzigen oder zukünftigen Bedingungen gelten werden. Die Sprache verzichtet vollständig auf eine weitere Untergliederung. Alle gespeicherten Informationen werden demnach gleich behandelt. Man kann allerdings ein Äquivalent für Fähigkeiten ausmachen: ein Agent gibt über seine Schnittstelle bekannt, welche Nachrichten er bearbeiten kann und welche Nachrichten er versenden wird. Diese

Schnittstelle ist aber nicht explizit als Menge von Fähigkeiten repräsentiert. Es ist nicht klar, ob die aufgestellten Verhaltensregeln zur Laufzeit ergänzt oder verändert werden können.

Diese Repräsentation besitzt eine formale Semantik. Hat man das Verhalten des Agenten mittels einer temporalen Logik spezifiziert, so kann Concurrent MetateM diese ausführen: Spezifikation und Implementierung fallen zusammen. Concurrent MetateM ist ein Beispiel dafür, wie die hinter den Fähigkeiten eines Agenten stehenden Aktionsfolgen vollständig für den Agenten transparent in der Repräsentationssprache, formal und mit gegenseitigen zeitlichen Bezügen beschrieben werden können.

Wissensrepräsentation in VIVA

Gerd Wagner stellt in seiner Habilitationsschrift und weiteren Arbeiten das Konzept der von ihm so genannten vividen Agenten und deren Programmierung mit VIVA vor ([Wagner 97, Wagner 98]). VIVA steht in der Tradition von Agent0.

Ein vivider Agent besitzt ein vivides Wissenssystem und kann das eigene Handlungsrepertoire explizit repräsentieren. Ein vivides Wissenssystem ist durch eine Wissensrepräsentationssprache, eine Eingabe- und eine Anfragesprache, durch eine Inferenzrelation und durch eine Aktualisierungsoperation definiert. Das Wissenssystem nutzt der Agent, um Annahmen (dort als Überzeugungen bezeichnet) über die Umgebung und den eigenen inneren Zustand (dort als mentaler Zustand bezeichnet) zu speichern. Im Gegensatz zu Agent0 kann das Wissenssystem hier sowohl die geltenden Zusammenhänge der Umgebung als auch den aktuellen Zustand speichern. Zu dem inneren Zustand gehört eine Menge von Aufträgen (Zielen) und eine Menge von momentanen Absichten. Eine Absicht besteht aus einem Ziel-Plan-Paar.

Neben diesem Wissenssystem besitzt ein vivider Agent (Wagner unterscheidet dabei zwischen rein reaktiven und proaktiven Agenten) Reaktionsregeln, die sein Verhalten beim Eintreffen von Nachrichten und der Gültigkeit bestimmter Bedingungen beschreiben, und Aktionsregeln, die Aktionen mit ihren Vorbedingungen und Auswirkungen beschreiben. Aktionen werden in Plänen verwendet, um ein bestimmtes Ziel zu erreichen. Beide Arten von Regeln besitzen zwar die für die Wissensrepräsentation klassische Form von Regeln, sind also explizit repräsentiert, sind aber selbst nicht Gegenstand von Änderungen. Reaktionsregeln beschreiben reaktive Handlungsvarianten: Wie und unter welchen Bedingungen reagiert der Agent auf Ereignisse. Aktionsregeln beschreiben angepasst an die in dieser Arbeit verwendete Terminologie die Fähigkeiten des Agenten. Aus diesen Fähigkeiten ergeben sich weitere Handlungsvarianten durch Planung, mit denen der Agent die gestellten Aufträge bearbeiten kann.

Während der Inhalt des Wissenssystems über dessen Aktualisierungsoperation verändert werden kann, sind die Handlungsvarianten und Fähigkeiten des

Agenten fest vorgegeben. Der Agent kann jedoch die für einen Auftrag notwendigen Schritte mit einem Planungsalgorithmus planen. Dies ist möglich, da die Fähigkeiten explizit durch Aktionsregeln repräsentiert sind. Allerdings sind Regeln zwar für die Formulierung von Vor- und Nachbedingungen adäquat, stellen aber keine natürliche Beschreibung sequentieller Programme dar. Sie sind ein Mittel zur Beschreibung der Auswirkung einer Aktion, nicht aber ein Mittel zur Ablaufbeschreibung der Aktion.

Die Sprache sieht nicht explizit die Darstellung von Informationen über die Kooperation zwischen Agenten und über andere Agenten vor.

Wissensrepräsentation in MAGSY

Im System MAGSY von Klaus Fischer ([Fischer 93]) speichert ein Agent Fakten (die Annahmen über den aktuellen Zustand), eine Menge von Aktionsregeln (dort als Verhalten bezeichnet) und eine Menge von Diensten (Fähigkeiten). MAGSY basiert auf OPS-5, einer regelbasierten Programmiersprache, und übernimmt dessen Ausdrucksfähigkeit für die Wissensrepräsentation im Agenten. Die Aktionsregeln werden hier aus Petrinetzen erzeugt, die das Verhalten des Agenten beschreiben.

In OPS-5 ist nur die Aktualisierung der Fakten, nicht jedoch die der Regeln möglich. Der Agent ist demnach nur in der Auswahl der angewendeten Regel oder in der unterschiedlichen Kombination der Regeln (Planung) flexibel. Wieder ist zu sagen, dass Regeln für die Aktionsbeschreibung nicht gut geeignet sind, wenn die Aktionen eigentlich sequentielle Anweisungsfolgen sind.

Auch MAGSY sieht keine explizite Darstellung von Informationen zur Kooperation zwischen Agenten vor. Diese ist vielmehr durch Verbindungen zwischen den Petrinetzen (Aktionsregeln) fest vorgegeben.

Wissensrepräsentation in GPGP

Victor R. Lesser diskutiert in [Lesser 98] das GPGP-Framework (Generic Partial Global Planning) für die Programmierung von Agenten. In der dort vorgeschlagenen Agenten-Architektur gibt es eine Datenbasis für die Annahmen des Agenten; eine netzförmige Aufgabenstruktur (Task Structure), in der verschiedene Fähigkeiten und ihre Beziehungen untereinander abgebildet sind; eine Menge von Verpflichtungen; einen aktuellen Abarbeitungsplan (Schedule); Informationen über die Organisationsstruktur des Agentensystems, über die Netzwerkressourcen und andere Agenten, über die aktuellen Ziele der Organisation und über die Statistik typischer Aufgaben in der Organisation.

Die zentrale Repräsentationssprache ist TAEMS. In TAEMS kann man Aufgabenstrukturen mit folgenden Informationen darstellen: Zielen (Aufträge, abstrakte Aufgaben), die der Agent erreichen will oder soll; ein oder mehreren Möglichkeiten (Handlungsvarianten), wie der Agent ein entsprechendes Ziel erreichen

kann (Baum, dessen Blätter atomare Aktionen, Methodenaufrufe mit Wahrscheinlichkeitsangaben zu Qualität, Kosten und Dauer beschreiben: Fähigkeiten); messbaren Angaben zum Nutzen von Aufgaben für den Agenten; den Zusammenhängen zwischen verschiedenen Aufgaben (ihre gegenseitige Beeinflussung) und den Beziehungen von Aufgaben zu benötigten Ressourcen. Eine Aufgabenstruktur ist als ein vorkompiliertes Netz von Aufträgen, Handlungsvarianten und Fähigkeiten zu verstehen. Allerdings können die Bewertungen von Aufgaben und atomaren Aktionen zur Laufzeit geändert werden.

Interessant ist hier die Bewertung. Mit diesem Mittel ist es möglich, Handlungsvarianten und Fähigkeiten entsprechend der bisher mit ihnen erreichten Wirkungen zu justieren. Der in Handlungsvarianten ausgedrückte Zusammenhang zwischen Umgebungssituationen und anwendbaren Fähigkeiten ist dadurch nicht starr, sondern ein Ansatzpunkt für eine Anpassung.

Wissensrepräsentation in GRATE und GRATE*

In [Jennings 94] stellt Nick Jennings das System GRATE (Generic Rule and Agent model Testbed Environment) und seine Anwendung in zwei industriellen Umfeldern vor. Aus den gesammelten Erfahrungen leitet er Forderungen an die Wissensrepräsentation ab, die in GRATE selbst noch nicht berücksichtigt sind und die zur Entwicklung von GRATE* führten.

GRATE ist ein in Lisp und CLOS implementiertes System zur Programmierung von Agenten. In GRATE kann ein Agent Informationen folgender Kategorien speichern:

- Wissen über den inneren Zustand
- Wissen über die Fähigkeiten
- Wissen über die Absichten
- Annahmen über die Umgebung und ihre internen Gesetzmäßigkeiten

Der innere Zustand enthält Angaben darüber, welche Aktivitäten gerade aktiv sind, wieweit sie bearbeitet sind und wann sie wahrscheinlich beendet werden. Die Aktivitäten entsprechen dem in dieser Arbeit verwendeten Begriff der Aktivität.

Fähigkeiten (dort auch als Rezepte benannt) sind durch ihren Namen, ihre Ausführungsbedingung, ihre Priorität, ihre Ausführungsdauer, ihr Ergebnis und eine Liste von Aktionen charakterisiert. Die Aktionen geben die Schritte vor, die der Agent ausführen muss, wenn er die Fähigkeit anwenden will. Die Fähigkeiten enthalten Informationen (z. B. Dauer) anhand derer der Agent entscheiden kann, welches von verschiedenen Mitteln und Wegen, dasselbe Ziel zu erreichen, er anwendet.

Absichten betreffen sowohl die aktuellen Absichten (Aktivitäten) als auch die zukünftigen Absichten (Verpflichtungen). Sie besitzen ebenfalls einen Namen, eine Motivation (das übergeordnete Ziel), die anzuwendende Fähigkeit, eine Priorität und einen Zeitrahmen. Absichten müssen untereinander und zu den Annahmen konsistent sein. Zwei Absichten sind in GRATE konsistent, wenn sie sich nicht zeitlich überschneiden. Zukünftige Absichten werden zu ihrem Termin nur verfolgt, wenn die Ausführungsbedingung der zugehörigen Fähigkeit dann erfüllt ist.

Zu den Annahmen über die Umgebung zählen die Informationen, die der Agent über andere Agenten der Umgebung besitzt. Der Agent speichert dazu Informationen zu denselben Kategorien, zu denen er auch Informationen über sich selbst speichert.

Die Verbindung all dieser gespeicherten Informationen zu den zu verfolgenden Absichten wird durch Regelsysteme in drei verschiedenen Modulen realisiert. Diese Regelsysteme repräsentieren Wissen über die Steuerung der Aktionen im zugrunde liegenden Anwendungssystem; die Erkennung bestimmter Situationen und die daraufhin notwendigen Reaktionen und über die Kooperation mit anderen Agenten. In der bisherigen Begriffswelt entsprechen diese Regeln den Handlungsvarianten.

Als Repräsentationsformen wählt GRATE zusammengefasst eine Frame-artige Beschreibung (CLOS) für alle Aspekte des aktuellen Zustandes; die Programmiersprache Lisp für die Fähigkeiten und Gesetzmäßigkeiten sowie ein Regelsystem für die Handlungsvarianten. Die Fähigkeiten und Handlungsvarianten eines GRATE-Agenten sind fest einprogrammiert.

In GRATE* ist zusätzlich explizites Wissen über die Kooperation zwischen Agenten repräsentiert. Das betrifft die Vereinbarung und Speicherung eines Kooperationsziels (joint goal); eines gemeinsamen Mittels (joint plan), dieses Ziel zu erreichen; und der Absichten lokale Fähigkeiten anzuwenden (joint actions), um damit Teile des gemeinsamen Plans zu übernehmen. Konventionen beschreiben, unter welchen Umständen ein Agent Absichten, die aus einem Kooperationsziel resultierten, neu festlegt oder fallen lässt. Von Jennings als soziale Konventionen benannte Regeln legen fest, wie und wann der Agent andere am Kooperationsziel beteiligte Agenten informieren soll, wenn der lokale Anteil gelungen ist, fehlerhaft unterbrochen wurde oder nicht zu dem erwarteten Resultat führen wird.

Konventionen und soziale Konventionen sind wiederum als Regeln repräsentiert: Treten bestimmte Bedingungen ein, führt das zu einer bestimmten Reaktion des Agenten.

Der wesentliche Gesichtspunkt von GRATE* ist die explizite Repräsentation von Kooperationswissen. Dadurch sind GRATE*-Agenten in der Lage, flexibel zu kooperieren. Auch die Kooperation muss dann nicht fest vorprogrammiert sein.

Wissensrepräsentation in INTERRAP

Jörg Müller stellt in seiner Arbeit ([Müller 96]) mit INTERRAP eine Architektur für dynamisch interagierende Agenten vor. Der innere (mentale) Zustand eines Agenten besteht in INTERRAP aus der aktuellen Beobachtung, einer Menge von Annahmen, einer Menge von Situationen, einer Menge von potentiellen Zielen, einer Menge von momentan erfüllbaren Zielen (Teilmenge der potentiellen Ziele: dort Option genannt), einer Menge von effektiven (momentan verfolgten) Zielen (Teilmenge der erfüllbaren Ziele: dort Absicht genannt) und einer Menge von operationalen Primitiven (Fähigkeiten). Einige dieser Bestandteile existieren in drei Schichten: einer Reaktionsschicht, einer lokalen Planungsschicht und einer Kooperationsschicht (kooperative Planung). Die Menge von Annahmen enthält dementsprechend in der untersten Schicht Informationen über die Umgebung, in der mittleren Schicht Informationen über den Agenten selbst und in der dritten Schicht Informationen über andere Agenten.

Situationen dienen als ein Bindeglied zwischen den gesammelten Annahmen und auszuführenden Aktionen. Sie fassen bestimmte relevante Konstellationen in den Annahmen zusammen. Entsprechend der Dreischichtung gibt es Situationen, die eine Reaktion hervorrufen (solche Situationen enthalten nur Annahmen über die Umgebung); Situationen, die eine lokale Planung anstoßen (Situationen mit Annahmen über die Umgebung und den inneren Zustand des Agenten); und Situationen, die eine kooperative Planung benötigen (Annahmen aus allen drei Schichten).

Ziele sind ebenfalls dreigeteilt in Reaktionen, lokale Ziele und Kooperationsziele. Eine Reaktion bezeichnet dabei die Ausführung eines Verhaltensmusters (siehe unten). Eine Menge von Agenten kann dasselbe Kooperationsziel verfolgen. Ziele können genauer charakterisiert werden als Ziele, einen Zustand zu erreichen, und als Ziele, einen Zustand beizubehalten.

Operationale Primitive sind eine Generalisierung der Operatoren in STRIPS und der Pläne in BDI-Architekturen. Sie stellen die Fähigkeiten, die primitiven Bestandteile dar, die der Agent einsetzen kann, um ein Ziel zu erfüllen. Die operationalen Primitive umfassen entsprechend der Dreiteilung gekapselte Verhaltensmuster, lokale Pläne und Kooperationspläne mehrerer Agenten. Dabei bauen lokale Pläne hierarchisch auf Verhaltensmustern und Kooperationspläne auf lokalen Plänen auf. Die operationalen Primitive sind die Grundlage für die Berechnung von Absichten aus tatsächlich verfolgten Zielen.

Die Menge der Absichten (in der dort verwendeten Begriffswelt) ist nicht explizit repräsentiert, sondern ist im aktuellen Abarbeitungszustand angestoßener operationaler Primitive abzulesen. Allerdings kann ein Agent durch die Kenntnis der effektiven Ziele rekapitulieren, aus welchem Grund er gerade eine Aktion ausführt.

Mit den Kooperationszielen, den Kooperationsplänen und den Annahmen über andere Agenten sind in INTERRAP Informationen über die Kooperation mit anderen Agenten explizit repräsentiert.

INTERRAP verwendet verschiedene Arten der Wissensrepräsentation. Die Annahmen eines Agenten speichert INTERRAP in einer assertionalen Wissensbasis, die die Funktionalität eines semantischen Netzes (Konzepte, Attribute, Relationen; Hinzufügen, Abfragen und Löschen; [Sowa 91]) bietet. Diese Wissensbasis lässt ebenfalls die Installierung von aktiven Triggern zu, die bestimmte Änderungen in den Daten signalisieren. Ein semantisches Netz ist geeignet, ein Modell der Umgebung zu speichern und anhand aktueller Informationen weitere Informationen abzuleiten.

Reaktionsregeln findet man in INTERRAP in der Form von Situation-Verhaltensmuster-Paaren wieder. Verhaltensmuster selbst sind aufgeteilt in eine Beschreibung (mit benötigten Ressourcen, Aktivierungsbedingung etc.) und einen Anweisungsteil (Aktion). Dieser ist in einer speziell zugeschnittenen Programmiersprache festgelegt. Für Pläne ist eine eigene Planrepräsentationssprache vorgesehen. Ein Agent besitzt eine Bibliothek von Plänen, die zur Laufzeit instantiiert werden können.

Aus der verwendeten Repräsentation geht hervor, dass Annahmen zur Laufzeit aktualisiert werden können. Die Verhaltensmuster sind fest vorgegeben, lokale und Kooperationspläne entstehen durch Planungsprozesse aus einer fest vorgegebenen Planbibliothek. Auch die Reaktionsregeln aller drei Schichten sind statisch.

Die zentrale Idee von INTERRAP ist die Schichtenarchitektur. Reichen die Informationen einer unteren Schicht nicht aus, zieht der Agent Informationen aus höheren Schichten zu Rate.

Wissensrepräsentation durch Fälle

Durch Fälle kann der Aspekt der Erfahrung in Agenten modelliert werden ([Burkhard 97]). Ein Fall könnte die Anwendung einer bestimmten Fähigkeit in einer bestimmten Situation und ihren Erfolg beschreiben. Ein Fall bietet damit ein alternatives Konzept für eine Reaktionsregel an. Fälle könnten während der Laufzeit des Agenten ergänzt werden. Sie beschreiben nicht das Verhalten, das der Programmierer vorher festgelegt hat, sondern ein Verhalten, das der Agent erfolgreich oder nicht erfolgreich »ausprobiert« hat. Die Fallbasis kann dann als Grundlage für die Bestimmung der nächsten Aktion in einer neuen Situation dienen: Ein Fall mit einer ähnlichen Situation verweist unter Umständen auf eine Fähigkeit, die nach bestimmten Anpassungen auch in der neuen Situation hilfreich ist.

Für diese Art der Repräsentation ist allerdings entweder eine initiale Fallbasis oder zumindest die Spezifikation des initialen Verhaltens notwendig. Zu klären

ist, woher der Agent neue Fälle, neue Handlungsvarianten bekommt. Die bloße Nutzung einer fest vorgegebenen Fallbasis würde kaum über ein Regelsystem hinausgehen (ein Unterschied wäre, dass man im Prinzip für jede Situation einen am besten passenden Fall, nicht jedoch eine anwendbare Regel finden kann).

Zusammenfassung

Aus der Aufzählung der vorgestellten Projekte wird zunächst deutlich, dass die unterschiedlichsten Begriffe für gleiche oder ähnliche Konzepte verwendet werden. Weitere Projekte (z.B: [Kirn 91], [Bechtolsheim 93]) repräsentieren dieselben Konzepte auf ähnliche Weise.

Allgemein lässt sich feststellen, dass für die Repräsentation der Umgebungsinformationen eher Frames oder semantische Netze ([Sowa 91]) benutzt wurden. Die Informationen zur Umgebung betreffen die dort wirkenden Gesetzmäßigkeiten (T-Box eines semantischen Netzes) und die aktuelle Situation (A-Box eines semantischen Netzes). In manchen Anwendungen ist es interessant, auch den zeitlichen Verlauf der Umgebung zu speichern.

Für die Beschreibung der Handlungsvarianten kamen meist Regeln zum Einsatz. Sie stellen eine natürliche Verknüpfung zwischen Umgebungssituation und anwendbaren Fähigkeiten dar. Parallel zu diesen explizit repräsentierten Handlungsvarianten besteht in einigen Systemen die Möglichkeit der Planung auf der Basis der vorhandenen Fähigkeiten oder auf der Basis einer vordefinierten Planbibliothek. Durch eine Berechnung des Nutzens bzw. Wirkungsgrads einer Handlungsvariante kann die Auswahl der Handlungsvariante von früheren Erfolgen oder Misserfolgen abhängig gemacht werden. Noch einen Schritt weiter geht die Repräsentation von Handlungsvarianten in Fällen, die die Wirkung durchgeführter Aktionen dokumentieren.

Fähigkeiten und die mit ihnen verknüpften Aktionen bzw. Pläne wurden ebenfalls durch Regeln oder in eigenen prozeduralen Sprachen dargestellt. Zu unterscheiden sind hier Mittel zur Repräsentation der Fähigkeiten, bei denen das Augenmerk auf den Auswirkungen und den Voraussetzungen einer Aktion liegt, von Mitteln zur Repräsentation der Aktion selbst. Hier muss beschrieben werden, wie die Auswirkungen der Aktion erreicht werden.

In unterschiedlicher Weise sind die Handlungsmotive der Agenten dargestellt. Die Skala reicht von verhandelten Zielen (GRATE*) über fest vorgegebenen Ziele, aus denen sich der Agent aktuelle aussucht (INTERRAP), bis zur Speicherung von Verpflichtungen (Agent0) ohne explizite Ziele. Auch die Verknüpfung und die tatsächliche Bedeutung der Begriffe »Ziel«, »Absicht«, »Verpflichtung« und »Aktivität« ist nicht einheitlich.

Im Bezug auf die Kooperation zwischen Agenten zeigen die vorgestellten Arbeiten das Spektrum zwischen fest vorgegebenem Verhalten ohne explizite

Repräsentation und flexiblem, Situations-angepasstem Verhalten mit expliziter Repräsentation.

3.1.2 Ausprägungsvarianten

In der Diskussion, welche Informationen (Was) auf welche Art und Weise (Wie) repräsentiert werden können, versucht die folgende Ausarbeitung eine in Hinblick auf die in dieser Arbeit entwickelte Architektur einheitliche Begriffsbildung zu finden.

Annahmen über die Umgebung

Das Verhalten eines Agenten soll sich zur Laufzeit an die gegebene Umgebungssituation anpassen. Das macht gerade seine Flexibilität aus. Dabei soll die Flexibilität über eine bloße Fallunterscheidung hinausgehen. Der Agent soll die Umgebung aktiv wahrnehmen und die Aktionsauswahl darauf abstimmen. Er muss dazu in der Lage sein, bestimmte zu Handlungen zwingende Situationen zu erkennen. Der Agent muss auch in vom Programmierer nicht vorausberechneten Situationen Entscheidungen treffen können. Zur Berücksichtigung dieser Forderungen ist es zunächst nötig, dass der Agent Umgebungsinformationen adäquat repräsentiert. Die Daten müssen in einer Form vorliegen, die die oben genannten Prozesse unterstützt.

Nach der Analyse der Beispielanwendungen und der zuvor aufgeführten Systeme sind Angaben zu folgenden Kategorien zu repräsentieren:

- Gesetzmäßigkeiten der Umgebung
- die aktuelle Situation

Die Annahmen über die aktuelle Situation fließen direkt in den Entscheidungsprozess ein. Wirkende Gesetzmäßigkeiten sind notwendig, wenn man weitere Annahmen aus der aktuellen, eventuell unvollständig wahrgenommenen Situation herleiten will (dafür ist ein Ableitungsalgorithmus, z. B. Vorwärtsverkettung bei Regeln, notwendig). So kann der Fußballer aus der Kenntnis des Spielfeldaufbaus und den aktuell wahrgenommenen Spielfeld-Daten seines Blicksektors die relative Position des gegnerischen Tores bestimmen, ohne dass diese Angabe tatsächlich in den aktuellen Daten enthalten war. Der Agent kann Gesetzmäßigkeiten ebenfalls benutzen, um Fehler in den wahrgenommenen oder gespeicherten Daten zu bemerken.

In der bisher vorgestellten Architektur spielen in Bezug auf die Umgebung die Begriffe »Situation«, »Information« und »Auftrag« eine Rolle. Die in der Umgebung wirkenden Gesetzmäßigkeiten wurden nur implizit in der Aktualisierungsstrategie der Wissensbasis verwendet. Soll dieses Wissen explizit repräsentiert werden, so bieten sich dafür verschiedene Formen der Wissensrepräsentation an: die T-Box eines semantischen Netzes, Constraints, Regeln. Die hier

zu treffende Wahl ist stark von der konkreten Anwendung abhängig (siehe Abbildung 3.1).

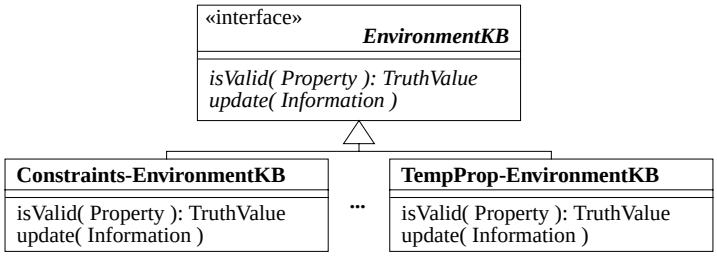

Abbildung 3.1: Diagramm für die Repräsentation von Umgebungsdaten

Die aktuelle Situation ist im Allgemeinen als eine Menge von Attribut-Wert- bzw. Konzept-Individuum-Paaren mit einem zeitlichen Gültigkeitsintervall (z. B. in der A-Box eines semantischen Netzes) darstellbar. Die gewählte Darstellungsform muss auf jeden Fall die Formulierung und den Test von Bedingungen gestatten. Dieser Test ist gleichzusetzen mit einer Anfrage an eine klassische Wissensbasis. Die Art der Wissensbasis bestimmt die Art der möglichen Anfragen (vgl. [Wagner 97]). Varianten sind hier die Frage

- nach der Gültigkeit von Fakten,
- nach der Gültigkeit von Relationen
  (im Sinne einer SQL-Datenbankabfrage),
- nach der Gültigkeit von Aussagen (im Sinne der Aussagenlogik),
- nach der Gültigkeit von Zeit-behafteten Aussagen
  (temporale Aussagenlogik),
- nach der Gültigkeit von Prädikaten (im Sinne der Prädikatenlogik erster Ordnung).

Im Kontext von unvollständiger und fehlerbehafteter Information spielen auch mehrwertige Logiken (z. B. ein Wahrheitswert mit der Bedeutung »unbekannt«) und solche Logiken eine Rolle, die Aussagen einen Wahrscheinlichkeitswert oder einen Zeitstempel zuordnen. Beim Fußballer wurden Umgebungsinformationen mit dem Zeitpunkt der letzten Aktualisierung versehen. Eine lange nicht aktualisierte Information wurde als wahrscheinlich fehlerhaft eingestuft (siehe Abbildung 3.2).

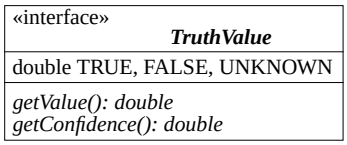

Abbildung 3.2: Diagramm für einen Wahrheitswert

Bei den Daten zur aktuellen Situation muss man berücksichtigen, dass es sich hier um Annahmen des Agenten über die tatsächliche Situation handelt. Annahmen unterliegen anderen logischen Gesetzen als Fakten. Sie können sich zum Beispiel widersprechen, ohne dass dies zu einem Konflikt im Verhalten führen muss. Diese Eigenschaften von Annahmen müssen in die Aktualisierungs- und der Abfrageoperation der Wissensbasis einfließen. Sie bestimmen ja erst die Bedeutung der gespeicherten Daten.

Eine wichtige Frage bei den untersuchten Systemen war die Frage der Aktualisierung: Welche Daten können verändert werden, welche sind konstant? Offensichtlich unterliegen die Daten zur aktuellen Situation häufigen Änderungen, das muss also in der verwendeten Wissensrepräsentationsform vorgesehen sein. Abzuwägen ist dabei zwischen dem Überschreiben alter Daten, dem Verschmelzen oder der Markierung mit Zeitstempeln. Sind die wirkenden Gesetzmäßigkeiten explizit gespeichert, so können sie prinzipiell ebenfalls Gegenstand von Änderungen zur Laufzeit sein. Diese Alternative hat jedoch mehrere Bedingungen und Konsequenzen. Zunächst muss die verwendete Wissensrepräsentationsform die Änderungen zulassen (z. B. Prolog in Form des Prädikats *assert*). Außerdem kann die Änderung dieser Daten das Verhalten des Agenten grundlegend verändern. Sinnvoll erscheint hier eine Einteilung in unveränderliche Zusammenhänge (analog den Naturgesetzen) und Zusammenhänge, die Änderungen unterliegen können (z. B. die Struktur einer Organisation). Man muss auch unterscheiden zwischen den Gesetzmäßigkeiten und dem Algorithmus, der die Ableitung neuer Annahmen aus vorhandenen Annahmen und Gesetzmäßigkeiten gestattet. Dieser Inferenz-Algorithmus wird im Allgemeinen stabil sein. Allerdings könnte ein Agent neue Ableitungsfähigkeiten aufnehmen (siehe letzter Absatz zur Expertise).

Zuletzt soll untersucht werden, woher die hier gespeicherten Daten kommen. Den Großteil der wirkenden Gesetzmäßigkeiten muss der Programmierer des Agenten angeben. Er muss festlegen, welche Umgebungsinformationen gespeichert werden sollen, welche also später einen Einfluss auf die Aktionsauswahl haben. Ist dieser Rahmen festgelegt, so gelangen weitere Daten über eines der Interaktionsmodule in die Wissensbasis. Die tatsächliche Aktualisierung nimmt der später betrachtete Aktualisierungsalgorithmus vor. Der Agent kann auch nicht beobachtete Umgebungsdaten aktualisieren: Wendet er Fähigkeiten an, so kennt er deren Konsequenzen und kann diese im Erfolgsfall als gültig annehmen. Auf diese Weise aktualisierte Daten müssen jedoch speziell gekennzeichnet sein, um sie von tatsächlich beobachteten zu unterscheiden. Die automatische Übernahme dieser Daten ist gegenüber einer selektiven vom Anwendungsprogrammierer vorgegebenen Aktualisierung abzuwägen.

Expertise

Soll ein Agent Entscheidungen über die nächsten Schritte treffen, die er ausführen wird, so müssen diese Schritte so repräsentiert sein, dass der Agent Vor- und

Nachteile abwägen kann und dass er die Auswirkungen der Handlungen mit seinen Zielen vergleichen kann. Es reicht im Allgemeinen nicht aus, die Namen und Parameter von aufzurufenden Methoden zu kennen. Es ist ja gerade ein wichtiger Unterschied zu einfachen Objekten (Objekte im Sinne der objektorientierten Programmierung), dass ein an einen Agenten gestellter Auftrag nicht direkt zu einem Methodenaufruf führt, sondern stattdessen der Agent die notwendigen Schritte einleitet. Das hier notwendige Wissen, zu welchen Schritten der Agent unter welchen Bedingungen in der Lage ist und wie sie auszuführen sind, nennt man die Expertise des Agenten.

Die bisher vorgestellte Architektur führte für die Kategorie Expertise die Begriffe »Aktion«, »Plan«, »Fähigkeit« und »Handlungsvariante« ein. Diese ließen sich auch in den in diesem Kapitel untersuchten Systemen in unterschiedlichen Ausprägungen wiederfinden. Entscheidend ist hier die Zweiteilung der Darstellung: einerseits die explizite Repräsentation von Voraussetzungen, Konsequenzen und anderen Eigenschaften als Grundlage der Aktionsauswahl und andererseits die Repräsentation der operationalen Semantik (das »Wie« der Ausführung). Diesen zweiten Teil sehen die meisten Systeme als nicht transparent für den Agenten an. Das bedeutet, dass durch die Angabe eines Namens und von Parametern eine Aktion gestartet wird (analog zu einem Methodenaufruf), aber die Einzelschritte dieser Aktion nicht symbolisch repräsentiert und damit nicht für Entscheidungsprozesse des Agenten zugänglich sind.

Andere Systeme (z. B. INTERRAP) verbinden beide Aspekte in einer Datenstruktur (operationales Primitiv). Neben der Beschreibung der Auswirkung steht dort eine Anweisungsfolge, die bei der Ausführung durch einen Interpreter abgearbeitet wird.

Die hier vorgestellte Architektur verwendet die Abstraktionen »Aktion« und »Plan« für die operationale Semantik und die Abstraktion »Fähigkeit« für die symbolische Darstellung der Eigenschaften von Aktionen und Plänen. Durch die Angabe seiner Fähigkeiten kann ein Agent anderen Agenten gegenüber sein Know-how darstellen. Das ist die zweite Bedeutung der Abstraktion »Fähigkeit«. Aktionen sind durch einen eindeutigen Namen adressierbar. Dieser Name bildet das Verbindungsglied zwischen Fähigkeit und Aktion. Einfache Aktionen sind gegenüber dem Agenten nicht transparent. Pläne sind Aktionen, die eine Folge von Aktionen enthalten. Pläne sind für den Agenten insofern transparent, als die Folge von Aktionen symbolisch repräsentiert ist.

Die Handlungsvarianten eines Agenten sind in den meisten vorgestellten Systemen durch Reaktionsregeln repräsentiert. Es geht, wie gesagt, um die Verknüpfung von Umgebungssituationen und anwendbaren Fähigkeiten. Handlungsvarianten sind damit auch ein Mittel, den Suchraum bei der Aktionsauswahl einzuschränken. Varianten sind hier

- die Repräsentation der Handlungsvariante in Fällen eines fallbasierten Systems,

- die Verwendung eines Entscheidungsbaums,
- die Berechnung einer Bewertung der Handlungsvariante in Abhängigkeit zur aktuellen Situation (diese Bewertung kann dem Erfolg entsprechend justiert werden).

Eine spezielle Handlungsvariante besteht darin, mit einem Planungsalgorithmus Fähigkeiten so zu kombinieren, dass ihre Anwendung zum gewählten Ziel führen; oder einen passenden Plan aus einer vorgefertigten Planbibliothek zu instantiieren.

In den meisten der vorgestellten Systeme sind die Expertisedaten statisch. Lediglich in GPGP lässt sich die Bewertung in der TAEMS-Aufgabenstruktur verändern. Wenn aber ein Agent auf vorher nicht vorgesehene Situationen ebenfalls reagieren soll, kann es notwendig sein, ihm zur Laufzeit neue Fähigkeiten zu geben. Dazu ist es notwendig, dass der Agent die Fähigkeit und die zugehörige Aktion zur Laufzeit laden kann. Eine andere Quelle »neuer« Fähigkeiten ist die Abspeicherung von zur Laufzeit erzeugten Plänen. Eine offene Frage ist, wie der Agent selbst zu echten neuen Aktionen kommen kann. Auf jeden Fall müssen die neu eingeführten Fähigkeiten auch in neuen Handlungsvarianten benutzt werden. Die entsprechende Kombination muss entweder der Programmierer liefern, oder sie muss sich in einem Lernprozess bilden (siehe Abbildung 3.3).

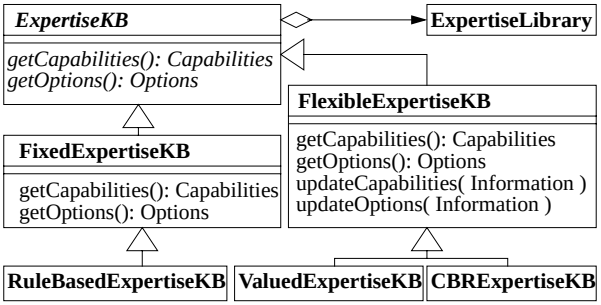

Abbildung 3.3: Diagramm für die Repräsentation von Expertisedaten

Damit ist wieder die Frage zu klären, woher die Expertisedaten für den Agenten kommen. Die initialen Aktionen, Pläne, Fähigkeiten und Handlungsvarianten muss der Programmierer des Agenten festlegen. Hier kann er durch vorgefertigte Expertisedaten unterstützt werden. So enthält das System GRATE* zum Beispiel vorgefertigte Fähigkeiten und Handlungsvarianten für die Kooperation mit anderen Agenten. Fähigkeiten, die nicht direkt zum eigentlichen Anwendungsgebiet gehören, sondern allgemein in Agentensystemen zu lösende Problemstellungen betreffen, können demnach bereits in einer Bibliothek vorliegen. Ein anderes Beispiel dafür ist die vorher erwähnte Planung zur Laufzeit. Während der Laufzeit kann der Agent weitere Expertisedaten vom Benutzer oder von anderen Agenten übernehmen, wenn die benutzte Repräsentationsform das zulässt.

Zu diskutieren ist der Ansatz, nachdem alle Abläufe im Agenten (also auch die Aktionsauswahl, die Aufnahme von Umgebungsdaten, die Ableitung neuer Annahmen etc.) Fähigkeiten des Agenten darstellen. Diese Vorstellung korrespondiert mit der Idee, bei den Umgebungsdaten auch alle gespeicherten Gesetzmäßigkeiten zur Laufzeit ändern zu können. Ist nämlich zum Beispiel die Aktionsauswahl als Fähigkeit repräsentiert, und ist die Änderung von Fähigkeiten möglich, so ist hier wie auch bei geänderten Gesetzmäßigkeiten eine Quelle für radikale Verhaltensänderungen im Gegensatz zur Anpassung an die aktuelle Situation gegeben. Wie bei der Diskussion zur Aktualisierung von Umgebungsdaten vorgeschlagen, sollte auch hier ein Kompromiss zwischen beiden Extremen gewählt werden. Manche der internen Abläufe kann man durchaus als Fähigkeit repräsentieren. Zum Beispiel könnte der Agent zur Laufzeit eine neue Fähigkeit zur Aufnahme von Umgebungsdaten oder zur Ableitung neuer Annahmen aus vorhandenen bekommen. Diese Fähigkeiten sind nicht anwendungsspezifisch und könnten wie oben erwähnt in einer Bibliothek von Fähigkeiten bereits vorhanden sein.

Motivation

Wenn der Antrieb zu auszuführenden Aktionen nicht im Aufruf von Methoden besteht, der Agent stattdessen selbst zu einem Auftrag passende Aktionen auswählen soll, so benötigt man eine Kategorie von Informationen, die diesen Antrieb schafft. Diese Art von gespeicherten Daten kann man als Motivation des Agenten bezeichnen. Der Agent soll wissen, warum er eine Aktion ausgeführt hat bzw. gerade ausführt. Mit diesem Wissen kann er auf Änderungen in der Umgebung reagieren, Aktivitäten abbrechen und neue, dann passende Aktionen starten. Schlägt der Methodenaufruf bei einem Objekt fehl, muss der Aufrufer reagieren. Ein Agent kann eine fehlgeschlagene Aktion in Erfüllung eines Auftrages selbst ersetzen, wenn er weiß, dass die fehlgeschlagene Aktion zu dem Auftrag gehörte. Entsprechend der expliziten Repräsentation der Motive kann ein Agent flexibel reagieren oder nicht.

Die hier vorgestellte Architektur verwendete in dieser Kategorie die Begriffe »Ziel«, »Absicht«, »Verpflichtung« und »Aktivität«. In den am Anfang des Kapitels vorgestellten Systemen finden sich gerade in diesem Bereich große Unterschiede. In Agent0 werden zum Beispiel nur Verpflichtungen gespeichert, in GRATE* können Agenten über ein gemeinsames Ziel verhandeln.

Ziele bilden für einen Agenten die oberste Stufe des Antriebs zum Handeln. Sie können von vornherein vorgegeben sein (z. B. INTERRAP) oder zur Laufzeit entstehen. Sie beziehen sich entweder auf die Erfüllung eines Auftrages, auf das Erreichen einer bevorzugten Umgebungssituation oder das Beibehalten einer Situation (siehe Abbildung 3.4). Die Ziele eines Agenten können sich widersprechen, was bedeutet, dass er sie nicht tatsächlich gleichzeitig verfolgen kann.

Absichten stehen konzeptuell zwischen Zielen und Handlungen. Sie verkörpern die momentan gewählten Handlungsvarianten, mit deren Ausführung

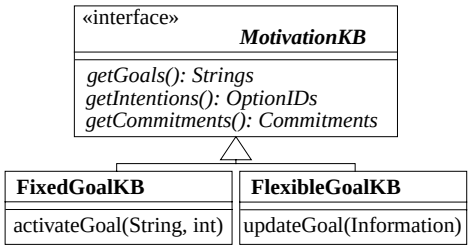

Abbildung 3.4: Diagramm für die Repräsentation von Motivationsdaten

der Agent die aktuellen Ziele umsetzen kann. Statt also die zu der gewählten Handlungsvariante gehörenden Aktionen direkt an das Ausführungsmodul zu geben, wird die aktivierte Handlungsvariante zusätzlich als Absicht des Agenten repräsentiert. Eine Absicht kann demnach als aktivierter Plan angesehen werden. Die im Plan vorgesehenen Einzelaktivitäten können die übergeordnete Absicht informieren, falls ihre Ausführung fehlschlägt. Eine Absicht besitzt wiederum eine Referenz auf das übergeordnete Ziel und kann den Erfolg oder Misserfolg melden. Die Absichten eines Agenten dürfen sich nicht widersprechen, insofern durch sie ausgelöste Aktionen auch wirklich nebenläufig ausführbar sein müssen. Absichten dürfen auch den gespeicherten Annahmen nicht widersprechen, insofern notwendige Ausführungsbedingungen enthaltener Aktionen tatsächlich erfüllt sein müssen.

Die explizite Repräsentation von Absichten soll neben gezielten Fehlerreaktionen auch ein über gewisse Zeitintervalle stabiles Verhalten des Agenten ermöglichen. Er kann die momentan verfolgte Absicht in die nächste Aktionsauswahl einbeziehen. Für eine Absicht könnte man eine zusätzliche Bedingung speichern, bei deren Eintreten das weitere Verfolgen der Absicht spätestens neu geprüft werden soll. Eine solche Bedingung wird im System GRATE* Konvention genannt. Diese Bedingung gehört jedoch konzeptionell schon zur zugehörenden Handlungsvariante.

Eine Absicht verkörpert also eine Folge von zur Ausführung vorgesehenen Aktionen. Diese lassen sich unterscheiden in Aktionen, die wirklich im Moment ausgeführt werden (Aktivitäten), und Aktionen, die zu einem späteren Zeitpunkt ausgeführt werden sollen (Verpflichtungen). Dementsprechend gibt es eine Liste von Aktivitäten, die direkt im Ausführungsmodul gehalten wird, und einen Zeitplan von Verpflichtungen. Verpflichtungen und Aktivitäten besitzen eine Priorität. Verpflichtungen enthalten darüber hinaus den Zeitpunkt ihrer Ausführung und eventuell notwendige Ausführungsbedingungen.

Zu klären ist nun die Frage, welche der Motivationsdaten aktualisierbar sind. Ziele sind in einigen der vorgestellten Systeme nicht explizit repräsentiert. Man besitzt in diesen Systemen also auch keine Möglichkeit, sie explizit zu ändern. In INTERRAP sind die potentiellen Ziele fest vorgegeben, man kann jedoch die Menge der effektiven Ziele ändern. Prinzipiell soll es möglich sein, dem Agenten zur Laufzeit auch neue Ziele vorzugeben. Die Frage ist dann, wie er zu neuen

Zielen passende Aktionen auswählen kann. Absichten entstehen in einem Auswahlprozess, sind also kontinuierlichen Änderungen unterlegen. Dasselbe gilt für Aktivitäten und Verpflichtungen.

Wie bereits gesagt, Absichten, Verpflichtungen und Aktivitäten sind Ergebnisse von Auswahlprozessen. Sie entstehen also zur Laufzeit und müssen nicht vom Programmierer festgelegt werden (stattdessen bestimmt er den Auswahlalgorithmus). Ziele dagegen sind vom Programmierer vorzugeben oder entstehen als Ergebnis der Kommunikation mit anderen Agenten.

Für die Repräsentation der Motivationsdaten ist im Gegensatz zu den Umgebungs- bzw. Expertisedaten keine besondere Form der Wissensrepräsentation notwendig. Es genügen entsprechende Listen.

Aktueller Stand

Viele der am Anfang besprochenen Systeme speichern Daten zum aktuellen Abarbeitungsstand (z. B. INTERRAP und GPGP). In der hier vorgestellten Architektur speichert der Agent eine Liste von Aktivitäten. Ein Parallelisierungsalgorithmus steuert die gleichzeitige Ausführung. In einer Aktivität ist die bisher benötigte Zeit, die erwartete Fertigstellungszeit, die Priorität und der Grund der Aktivität gespeichert. Auch für den aktuellen Abarbeitungsstand benötigt man keine speziellen Formen der Wissensrepräsentation.

Annahmen über andere Agenten

Soll ein Agent in einem Multiagentensystem agieren, so betreffen die Umgebungsinformationen auch die Informationen zu anderen Agenten. Seine Handlungen werden in einem solchen System offensichtlich durch die Handlungen anderer Agenten beeinflusst und beeinflussen ihrerseits die Handlungen anderer Agenten. Der Agent kann versuchen, die Unterstützung anderer Agenten zu bekommen oder selbst andere Agenten unterstützen. Manche Aufgaben sind überhaupt nur lösbar, wenn mehrere Agenten zusammenwirken. Es geht dann nicht um die Erfüllung eines eigenen Ziels, sondern um die Erfüllung eines Kooperationsziels. Der Nutzen für die beteiligten Agenten mag dabei gering sein.

In dieser Kategorie kann ein Agent prinzipiell alle Angaben speichern, die er auch über sich selbst speichert. Es ist jedoch abzuwägen zwischen dem dadurch erzielbaren Nutzen und dem entstehenden Speicher- und Berechnungsaufwand. Für einen guten Kompromiss muss der Agent Angaben zu anderen Agenten selektieren und abstrahieren. So könnten etwa die momentan verfolgten Ziele, nicht jedoch die daraus abgeleiteten Absichten anderer Agenten gespeichert werden. Der wichtigste Punkt zu anderen Agenten scheinen jedoch Angaben zu dessen Fähigkeiten zu sein. Auf der Basis dieser Information kann

der Agent Kooperationspartner auswählen. Um bei weiteren Wahlentscheidungen bisherige Erfahrungen berücksichtigen zu können, kann die Kooperation bewertet und mit geeigneten Attributen im Datensatz zu dem entsprechenden Agenten abgelegt werden. Als Attribute kommen der Grad der Aufgabenerfüllung, die benötigte Zeit und andere Leistungsmaße in Frage. Alle Angaben zu einem anderen Agenten dienen letztlich zur Einschätzung von dessen Kompetenz. Für die Repräsentation solcher Erfahrungen bietet sich eine Fallbasis an.

Eine andere Art von Annahmen sind Annahmen, die gleichzeitig und gemeinsam in einer Gruppe von Agenten etabliert sind. Solche Annahmen sind Voraussetzungen für kooperatives Handeln. In einer Gruppe kooperierender Agenten kann zum Beispiel jeder Agent annehmen, dass jedes Gruppenmitglied das gemeinsame Kooperationsziel verfolgt. Für diese Art von Annahmen muss der Agent also Annahmen anderer Agenten speichern. Das würde letztlich zu einer Endlosschleife führen, in der der eine Agent annimmt, dass der andere Agent annimmt, dass der eine Agent annimmt etc., dass der andere Agent das Kooperationsziel verfolgt. In der Praxis behilft man sich hier mit der Beschränkung auf eine endliche Zahl von Zyklen.

Effizienz vs. Flexibilität

Ein Ziel der in diesem Kapitel aufgeführten Betrachtungen zur Wissensrepräsentation ist das Abwägen zwischen Effizienz und Flexibilität. Sind alle der bisher herausgearbeiteten Abstraktionen explizit repräsentiert, kann der Agent flexibel auf neue Situationen eingehen. Allerdings sind dafür sicher zeitaufwendigere Berechnungen notwendig, als sie bei fest eingestellten Reaktionen notwendig wären. Sind andererseits alle Wahlmöglichkeiten weitgehend eingeschränkt, dann verringert das die Flexibilität des Agenten.

Ein in anderen Gebieten anzutreffendes Prinzip (z. B. im Verhältnis von Interpretation und Kompilation) lässt sich auch für die Balance zwischen den Extremen hier anwenden. Der Agent sollte flexibel starten, während der Laufzeit sein Verhalten an die Charakteristik der vorgefundenen Umgebung anpassen, die Flexibilität zugunsten der Effizienz einschränken. Erst wenn die Umgebung ihre Charakteristik ändert, müsste der Agent das Gewicht wieder mehr in Richtung Flexibilität verschieben. Für dieses Prinzip sind unterschiedliche Ausprägungen möglich. Das Verhalten des Agenten könnte insofern flexibel sein, als es nicht in sein ausführbares Programm hineinkompiliert ist, sondern beim Start aus Konfigurationsdateien eingelesen wird. Das ist jedoch die Art von Flexibilität, wie sie bei jeder Art von Programmen heute Usus ist. Für Agenten ist zu fordern, dass sie auch zur Laufzeit flexibel sind. Das genannte Prinzip kann dann so ausgelegt sein, dass der Agent in einer ersten Phase die Umgebungscharakteristika lernt und in der zweiten Phase effizienter handeln kann.

Wie auch immer das Prinzip angewendet wird, es muss in den verwendeten Datenstrukturen und also in der Repräsentation vorgesehen sein. Zur Unterstützung der Flexibilität ist die explizite symbolische Repräsentation wichtig, für

die Effizienz eine für die Ausführung optimierte Darstellung. Dieses Verhältnis kann man gut am Unterschied zwischen einer Aktion und einem Plan festmachen. Der Inhalt einer Aktion ist gegenüber dem Agenten nicht transparent, sie ist vorkompiliert, sofort ausführbar. Ein Plan enthält als symbolische Repräsentation die Folge von zugehörigen Aktionen. Den Plan auszuführen, bedeutet, wie ein Interpreter die Aktionen der Reihe nach abzuarbeiten. Noch länger würde es dauern, wenn der Plan vorher noch erzeugt werden müsste. Durch Kompilierung von Plänen zu Aktionen und das Merken einmal berechneter Pläne lässt sich hier ein Effizienzgewinn erzielen. Das gleiche Verhältnis besteht im Prinzip zwischen einer Fähigkeit und einer Handlungsvariante. Hätte der Agent durch die Handlungsvarianten nicht schon eine Verknüpfung zwischen Umgebungs- und Expertisedaten gegeben, so müsste er bei jeder Entscheidung die gesamte Liste an Fähigkeiten untersuchen. Die Abspeicherung von Fällen, die die erfolgreiche Anwendung von Fähigkeiten dokumentieren, unterstützt also das Prinzip. Fälle müssten gelöscht werden, wenn sich die Umgebungscharakteristik ändert.

Ein anderes Prinzip zur Abwägung zwischen Effizienz und Flexibilität ist die Einführung einer Schichtenarchitektur (z. B. in INTERRAP). Eine untere Schicht wäre für Reaktionen zuständig, eine obere Schicht für Situationen, in denen keine Reaktion passt. Auch für die Anwendung dieses Prinzips müssen die Daten entsprechend repräsentiert sein. Die Daten müssten entweder mit einer Schichtenkennung versehen werden oder zumindest in den Auswahlalgorithmen entsprechend behandelt werden. Der Kompilation würde hier eine Verlagerung von erzeugten Plänen in die Reaktionsschicht entsprechen.

Als drittes Prinzip kommt der Einsatz von Any-Time-Algorithmen in Frage. Zu einem bestimmten Zeitpunkt wird die bis dahin gefundene Wahl als endgültige Wahl festgelegt. Die Güte der Wahl wird dabei mit dem Zeitbedarf abgewogen, sie zu berechnen. Die Anwendung dieses Prinzips hängt im wesentlichen davon ab, ob ein Auswahlmechanismus mit Any-Time-Eigenschaft konstruiert werden kann. Das ist nicht Hauptsache der Wissensrepräsentation.

3.1.3 Zusammenfassung

Für die unterschiedlichen Arten zu speichernder Informationen eignen sich unterschiedliche Formen der Wissensrepräsentation. Die bisher einheitlich aufgefasste Wissensbasis besteht demnach aus folgenden entsprechenden Teilen: der Wissensbasis für Umgebungsdaten, der Wissensbasis für Expertisedaten und einer Datensammlung für die Motivationsdaten bzw. für den Abarbeitungszustand (siehe Abbildung 3.5).

Als ein wichtiges Kriterium für die zu speichernden Informationen wurde das Verhältnis zwischen Flexibilität und Effizienz herausgestellt. Für die einzelnen Kategorien wurde herausgearbeitet, welche Informationen der Programmierer zu liefern hat. Er hat außerdem die Aufgabe, die jeweils passende Repräsentationsform auszuwählen.

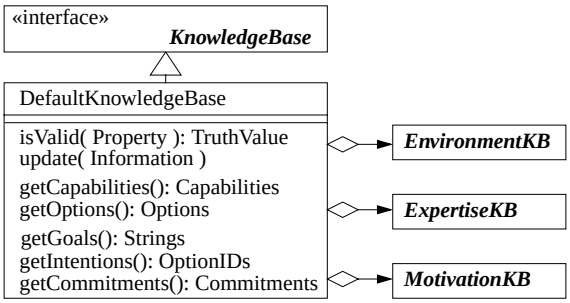

Abbildung 3.5: Diagramm für die Wissensbasis

3.2 Aktionen und Pläne

Nach dem allgemeinen Überblick zur Wissensrepräsentation in Bezug auf Agenten, beleuchten die nächsten Abschnitte einige Details genauer. Zunächst geht es um die Darstellung der Fähigkeiten eines Agenten und die für sie notwendigen Aktionen und Pläne.

3.2.1 Aufgaben und Quellen

Im Bereich der KI beschäftigen sich Forscher schon seit langem mit der Darstellung von Aktionen, mit Aktionstheorien und dem Planungsproblem. In diesen Forschungen geht es darum, Aktionen in einer Weise zu repräsentieren, dass sie Inferenz- und Planungsmechanismen zugänglich werden. Man will die zielgerichtete Ausführung von Aktionen unterstützen, will die zeitliche Ausdehnung von Aktionen berücksichtigen, will Konflikte zwischen Aktionen erkennen, will Pläne aus Aktionen erzeugen, will Aktionen parallelisieren, will Konsequenzen aus Aktionsfolgen abschätzen. In den nun vorgestellten KI-Arbeiten ist allerdings der Begriff der Aktion im Allgemeinen in einem anderen Sinn verwendet worden, als bisher in dieser Arbeit, nämlich synonym zu dem hier benutzten Begriff der Fähigkeit. Aktionen werden dort auch nicht in Bezug auf einen Handelnden beschrieben, sondern als Eigenschaft einer Domäne: in einer Domäne gibt es bestimmte Attribute (Fluents), die im Zeitverlauf ihren Wert ändern können (z. B. von wahr auf falsch), man kann in ihr unter bestimmten Bedingungen (Attribute haben einen bestimmten Wert) Aktionen ausführen, die dann gewisse Konsequenzen (Änderung von Attributwerten) haben.

In der Forschung innerhalb der KI zur Repräsentation von Aktionen und Veränderung spielten vor allem drei Probleme eine Rolle, die sich direkt aus den vorgeschlagenen Formalismen ergaben (insbesondere aus dem Situationskalkül: [McCarthy/Hayes 69]): das *Frame-Problem*, das *Qualification-Problem* und das *Ramification-Problem* (vgl. [Thielscher 97]).

Das Frame-Problem bezieht sich auf die Frage, wie man in einer Logik beschreibt, dass alle Werte von Attributen der Domäne, die durch eine Aktion

nicht berührt werden, nach Ausführung der Aktion unverändert bleiben. Dieses Problem entsteht, wenn man Aktionen ansieht als eine Überführung aus einer Situation, in der die Attribute gewisse Werte besitzen, in eine andere Situation, in der sie andere Werte besitzen; wenn der Situationsparameter also explizit repräsentiert ist. Das Problem hat zwei Aspekte: wie beschreibt man die Nichtänderung effizient und wie leitet ein Inferenzmechanismus die Nichtänderung effizient her (eigentlich sollte für eine Nichtänderung kein Zeitaufwand nötig sein). Im Bezug auf das Frame-Problem wurden Lösungen vorgeschlagen, die das Repräsentationsproblem, jedoch nicht das Inferenzproblem lösten. Andere Lösungen (z. B. STRIPS, siehe unten) umgehen das Inferenzproblem, schränkten jedoch die Ausdrucksfähigkeit ein. Eine Lösung für beide Aspekte bietet der Fluent-Kalkül, wie er zum Beispiel in der unten vorgestellten Sprache A_C^+ benutzt wird.

Das Ramification-Problem liegt in der Tatsache, dass es im Allgemeinen nicht möglich sein wird, wirklich alle Effekte einer Aktion aufzulisten. Das betrifft insbesondere Folgeeffekte. Um dieses Problem zu lösen, muss die gerade aufgestellte Forderung nach der Stabilität nicht direkt aufgeführter Attribute abgeschwächt werden. Nach dem Berechnen der direkten Effekte einer Aktion muss ein ergänzender Inferenzschritt für indirekte Effekte erfolgen. Damit können zumindest solche Folgeeffekte berücksichtigt werden, die in zusätzlichem domänenspezifischen Wissen kodiert sind. Dieses muss allerdings in einer Weise dargestellt sein, dass man aus ihm nicht Folgeeffekte ableiten kann, die keine Entsprechung in der Wirklichkeit haben. Solche unintuitiven Ableitungen entstehen zum Beispiel dadurch, dass klassische Inferenzmechanismen nicht gerichtet sind: Ein Beispiel ist die Aussage $a \wedge b \rightarrow c$ mit der beabsichtigten Bedeutung, dass man aus dem Eintreten von a bei gleichzeitiger Gültigkeit von b auf die Gültigkeit von c in der dann erreichte Situation schließen kann. Tritt nun a tatsächlich in einer Situation ein, in der b gilt, so kann in der dann erreichten Situation im Sinne der klassischen Logik jedoch c oder $\neg b$ wahr sein. Einen Ausweg bietet hier die Einführung einer speziellen gerichteten Kausalitätsbeziehung.

Das Qualifikations-Problem beschreibt die Unmöglichkeit, alle Voraussetzungen einer Aktion aufzulisten. Das betrifft insbesondere Voraussetzungen, die man als für gewöhnlich erfüllt ansieht. Für den Aktionsformalismus soll es deshalb nicht notwendig sein, alle diese Voraussetzungen mit aufzulisten, aber möglich sein, Ausnahmen zu formulieren. Sind für eine Aktion alle angegebenen Voraussetzungen erfüllt, so führt ihre Ausführung zu den genannten Konsequenzen, solange nicht das Gegenteil gezeigt werden kann. Ein Mittel zur Lösung dieses Problems sind nichtmonotone Logiken (Bevorzugung von Modellen mit minimalen Ausnahmen) mit Kausalitätsbeziehungen.

Die im Bereich der Formalismen zur Beschreibung von Aktionen und Veränderungen gemachten Erfahrungen sollten für die Programmierung von Agenten genutzt werden. Aus dem letzten Abschnitt ist klar geworden, dass die explizite Repräsentation der Fähigkeiten eine wichtige Grundlage der Aktionsauswahl

ist. Es geht um den Zusammenhang zwischen Aktionen und Plänen, um die Auswirkungen von Aktionen, allgemein um das zielgerichtete Handeln. Ein zweiter wichtiger Grund für die explizite Repräsentation von Fähigkeiten ist die dadurch mögliche Formulierung von Angeboten. Ein Angebot muss so beschrieben sein, dass andere Agenten erkennen können, ob es ihnen einen Nutzen bringt. In der anderen Richtung müssen Aufträge oder Aufgaben so formuliert sein, dass der Auftragnehmer erkennt, ob und wie er den Auftrag erfüllen kann.

Zustandsbasiertes Planen

Der bekannteste Modellierungsansatz für Aktionen und Pläne ist das zustandsbasierte Planen (z. B. STRIPS, [Fikes/Nilsson 90]). Die Welt wird durch eine Menge von Zuständen (Menge von wahren und falschen Aussagen (Fluents)) beschrieben, eine Aktion überführt das System von einem Zustand in einen anderen.

Eine Aktion besteht hier aus einer Menge von Voraussetzungen, einem Körper und einer Menge von Effekten. Sind in einem Zustand die Voraussetzungen einer Aktion erfüllt, so kann diese ausgeführt werden. Der Körper enthält eine Menge von Teilaktionen oder nähere Angaben zur Ausführung der Aktion (Seiteneffekte). Die Effektmenge beschreibt, welche Aussagen im erreichten Zustand gelten (ADD-Menge) und welche nicht mehr gelten (DELETE-Menge).

Planen geschieht hier durch einen Suchalgorithmus. Es wird ein »Pfad« von Aktionen gesucht, so dass jede Aktion auf dem Pfad im jeweils erreichten Zustand ausführbar ist, und der zuletzt erreichte Zustand die Zielbedingung enthält.

Es gibt einige Probleme mit diesem Ansatz. Die Aktionen haben hier keine zeitliche Ausdehnung, werden sequentiell ausgeführt. Die Zeit ist nur implizit über die Reihenfolge von Aktionen repräsentiert. Ein weiteres Problem ist die Zielbedingung. Optimalitätsbedingungen sind in einem solchen Formalismus schwer darzustellen. STRIPS ist adäquat in einer statischen, vollständig bekannten und modellierten Umgebung.

A_C^+: Parallele Aktionen und Konflikte

In [Bornscheuer/Thielscher 94] wird ein logikbasierter Ansatz zur Repräsentation von Aktionen vorgestellt. A_C^+ ist eine Erweiterung der Aktionsbeschreibungssprache A von M. Gelfond und V. Lifschitz ([Gelfond/Lifschitz 93]) und deren Dialekt A_C. Dieser Ansatz sei hier den eher pragmatischen Ansätzen im Bereich der Agentensysteme entgegengesetzt. Der benutzte Fluent-Kalkül stellt Mittel zur Lösung der oben genannten Probleme bereit.

Situationen werden hier nicht als Listen von Attributen, sondern als Terme repräsentiert, die aus Attributen zusammengesetzt sind (z. B. $atr_1 \circ atr_2$, Verknüpfungsoperator: $\circ$). Zwei Situationsterme kann man unifizieren, wenn die

enthaltenen Attribute übereinstimmen oder an freie Variablen gebunden werden können. Die Konsequenzen von einzeln oder gleichzeitig ausgeführten Aktionen werden in so genannten Effekt-Aussagen beschrieben. Diese enthalten einen Situationsterm (s_1), der die geforderte Ausgangssituation festlegt; einen Aktionsterm (a), der die gleichzeitig ausführbaren Aktionen benennt; und einen zweiten Situationsterm (s_2), der die Konsequenzen der Aktionsausführung aufzählt. Eine Einzelaktion a ist in der aktuellen Situation s ausführbar, wenn es eine Effekt-Aussage der Form (s_1, a, s_2) gibt und der Situationsterm $s_1 \circ V$ (V ist eine freie Variable, die die nicht in s_1 enthaltenen Attribute der aktuellen Situation aufnehmen soll) mit der Situation s unifizierbar ist (V wird durch die Unifikation an einen Wert gebunden). Als Ergebnis der Ausführung wird der Situationsterm $s_2 \circ V$ zur aktuellen Situation.

Das gilt aber nur für den speziellen Fall der Einzelaktionen. Betrachtet man auch gleichzeitig ausgeführte Aktionen, so können sich Effekt-Aussagen widersprechen. Sie widersprechen sich, wenn eine einzeln ausgeführte Aktion andere Effekte hat, als wenn sie in Kombination mit einer zweiten Aktion ausgeführt wird. Die Berechnung des Effekts gleichzeitig ausgeführter Aktionen besteht also im allgemeinen Fall darin, die Effekte der Einzelaktionen zu sammeln und Effekt-Aussagen über Verbundaktionen einzurechnen.

In dem Kalkül ist allerdings nicht verlangt, dass man für jede Kombination von Aktionen eine Effekt-Aussage formuliert. Das wäre auch angesichts der Komplexität nicht sinnvoll. Eine andere Art des Widerspruchs lässt sich deshalb aber nicht auflösen: Ist der Effekt einer Verbundaktion nicht beschrieben und widersprechen sich die Effekt-Aussagen der Einzelaktionen, so ist nicht klar, zu welcher Situation die Verbundaktion führt. A_C betrachtet die Verbundaktion in diesem Fall als nicht ausführbar, A_c^+ verfolgt dagegen alle Varianten weiter.

Als zweite Art von Aussagen kann man so genannte Wert-Aussagen formulieren. Das sind Aussagen über den Wert eines Attributs nach der Ausführung einer Folge von Aktionen. Ziel dieses Ansatzes ist es nun herauszufinden, ob es für eine Menge solcher Effekt- und Wert-Aussagen ein Modell gibt. Das ist die Frage, ob es eine Anfangssituation gibt, von der ausgehend alle Wert-Aussagen unter Beachtung der Effekt-Aussagen erfüllt sind.

Der Fluent-Kalkül sieht darüber hinaus die Formulierung von Kausalitätsbeziehungen zur Berechnung von Folgeeffekten vor. Die Arbeitsgruppe um S. E. Bornscheuer und M. Thielscher veröffentlichte weitere Erweiterungen der Aktionsbeschreibungssprache A_C. Alle bauen auf dem Fluent-Kalkül auf.

Der Ansatz besticht durch seine Formulierung in der Prädikatenlogik erster Ordnung. Die aufgestellten Sprachen wurden jeweils in Prolog implementiert. Allerdings abstrahiert auch er von Eigenschaften, die man in einem Softwareagenten aber antrifft: die Aktionen besitzen keine zeitliche Ausdehnung wie die Aktionen eines Agenten; sie sind zwar parallel ausführbar, aber nicht überlappend; Änderungen der Situation entstehen nur durch Aktionen und nicht durch andere Einflüsse.

Wie kann der Ansatz aber für die Aktionsauswahl verwendet werden? Betrachtet man die Annahmen eines Agenten über die aktuelle Situation als ein Modell der aktuellen Situation, fasst man also das Verhalten eines Agenten als ein fortlaufendes Generieren eines Modells auf, so kann der Agent unter Berücksichtigung der einschränkenden Bedingungen Wert-Aussagen für bestimmte Aktionsfolgen testen. Was letztlich heißt, dass er prüfen kann, ob eine bestimmte Aktionsfolge zu einer gewünschten Situation führt. Was der Ansatz allerdings nicht bietet, ist ein Hinweis in der anderen Richtung: Welche Aktionsfolgen kommen überhaupt für einen Test in Frage? Hier wären also zusätzlich Handlungsvarianten aufzulisten.

Explizite Zeitmodellierung beim Planen

Allen führt in [Allen/Kautz 91] folgende Begriffsbildung ein: Ein Plan ist eine Handlungsanweisung zum Erreichen eines bestimmten Ziels unter gewissen Ausgangsbedingungen. Eine Handlungsanweisung ist eine Menge von zeitlich verknüpften Aktionen. Aktionen selbst haben eine zeitliche Ausdehnung. Sie können gleichzeitig, zeitlich überlappend oder müssen zu unterschiedliche Zeitpunkten stattfinden. Sie können von äußeren Ereignissen abhängen. Allen ordnet deshalb der Aktionsausführung ein Zeitintervall zu, in dem sie stattfindet. Er bezeichnet die Aktionsausführung als Ereignis. Der Begriff »Zeitintervall« bezieht sich dabei auf Allens Intervallalgebra ([Allen 83, Allen 84]).

Zwischen Zeitintervallen können Beziehungen dargestellt werden. Die 13 atomaren Beziehungen ergeben sich aus der Kombination der Beziehungen zwischen Anfangs- bzw. Endpunkten der Intervalle. Liegt zum Beispiel der Endpunkt eines Intervalls v_1 vor dem Anfangspunkt eines Intervalls v_2, so liegt v_1 vor v_2, beginnt dagegen v_2 vor v_1 und stimmen die Endpunkte der Intervalle überein, so sprechen wir davon, dass v_1 v_2 beendet.

Die Menge $\{p, pi, m, mi, d, di, s, si, f, fi, o, oi, e\}$ enthält die Symbole für die 13 atomaren Beziehungen, die zwischen 2 Intervallen möglich sind. Dabei stehen die Buchstaben jeweils für die Anfangsbuchstaben der entsprechenden Beziehung, also **p***recedes*, **m***eets*, **d***uring*, **s***tarts*, **f***inishes*, **o***verlaps*, **e***quals*. **i** steht für die inverse Einschränkung. Der Begriff »invers« ist dabei zunächst ein bisschen irreführend, wird aber von Allen so verwendet. Die Bedeutung wird klar, wenn man sich die Beziehung zwischen zwei Intervallen als beschriftete gerichtete Kante vorstellt. Die invertierte Kante müsste mit der inversen Beziehung beschriftet werden, um denselben Sachverhalt zu beschreiben.

Die Beziehung zwischen den Zeitintervallen zweier Aktionsausführungen oder zwischen einer Aktionsausführung und dem Gültigkeitsintervall einer Aussage kann dann durch eine Disjunktion einer Auswahl von atomaren Beziehungen eingeschränkt werden. Hat man so die zeitlichen Beziehungen in einer Menge von Aktionsausführungen und gültigen Aussagen festgelegt, so kann man durch Anwendung der Transitivitätstabelle (siehe Abbildung 3.6) und

entsprechender Algorithmen (siehe z. B. [Kühnel 94]) berechnen, welche Ausführungsreihenfolge den Einschränkungen entspricht.

1 2 3 2	p	m	o	s	d	f	e	fi	di	si	oi	mi	pi
p	p	p	p	p	p,m,o s,d	p,m,o s,d	p	p	p	p	p,m,o s,d	p,m,o s,d	R_a
m	p	p	p	m	o,s,d	o,s,d	m	p	p	m	o,s,d	f,e,fi	pi,mi oi,si di
o	p	p	p,m,o	o	o,s,d	o,s,d	o	p,m,o	p,m,o fi,di	o,fi,di	o,s,d,f e,fi,di si,oi	di,si oi	pi,mi oi,si di
fi	p	m	o	o	o,s,d	f,e,fi	fi	fi	di	di	di,si oi	di,si oi	pi,mi oi,si di
di	p,m,o fi,di	o,fi,di	o,fi,di	o,fi,di	o,s,d,f e,fi,di si,oi	di,si oi	di	di	di	di	di,si oi	di,si oi	pi,mi oi,si di
si	p,m,o fi,di	o,fi,di	o,fi,di	si,e,s	d,f,oi	oi	si	di	di	si	oi	mi	pi
e	p	m	o	s	d	f	e	fi	di	si	oi	mi	pi
s	p	p	p,m,o	s	d	d	s	p,m,o	p,m,o fi,di	si,e,s	d,f,oi	mi	pi
d	p	p	p,m,o s,d	d	d	d	d	p,m,o s,d	R_a	pi,mi oi,f,d	pi,mi oi,f,d	pi	pi
f	p	m	o,s,d	d	d	f	f	f,e,fi	pi,mi oi,si di	oi,mi pi	oi,mi pi	pi	pi
oi	p,m,o fi,di	o,fi,di	o,fi,di si,e,s d,f,oi	d,f,oi	d,f,oi	oi	oi	di,si oi	pi,mi oi,si di	oi,mi pi	oi,mi pi	pi	pi
mi	p,m,o fi,di	si,e,s	d,f,oi	d,f,oi	d,f,oi	mi	mi	mi	pi	pi	pi	pi	pi
pi	R_a	pi,mi oi,f,d	pi,mi oi,f,d	pi,mi oi,f,d	pi,mi oi,f,d	pi	pi	pi	pi	pi	pi	pi	pi

Abbildung 3.6: Die Transitivitätstabelle

Im Gegensatz zu den beiden ersten vorgestellten Ansätzen unterscheidet Allen zwischen Aktionsausführungen (d.h. dem tatsächlichen Ausführen einer Aktion) und der Fähigkeit, eine Aktion auszuführen. Aktionsausführungen werden in Axiomen (Regeln) durch zeitliche Beziehungen zu ihren Vor-, Nach- und während der Ausführung geltenden Bedingungen repräsentiert. Ein Agent kann dadurch das Wissen repräsentieren, welche Konsequenzen eine Aktion hat, ohne zu wissen, wie die Aktion auszuführen ist. Wenn ein Agent aber die

Fähigkeit besitzt, eine Aktion auszuführen, dann kann er auch versuchen, sie auszuführen. Dieser Versuch wird bei Allen in weiteren Regeln beschrieben. So kann der Ausführungsversuch zur erfolgreichen Aktionsausführung führen, wenn die Vorbedingungen zur Aktionsausführung erfüllt sind.

Planen besteht laut Allen dann darin, Annahmen über die aktuelle und zukünftige Situationen und über das eigene Verhalten (welche Aktionen der Agent versuchen soll) zu treffen, um eine Zielsituation zu erreichen. Dazu müssen auch die Gesetze der Domäne, in der die Aktionen stattfinden sollen, als Axiome mit zeitlichen gerichteten Beziehungen formuliert werden. Das entspricht dem Ansatz der Kausalitätsbeziehungen im Fluent-Kalkül. Der Planungsprozess (ereignisbasiertes Planen) besteht in einem Versuch, das Ziel des Plans aus der momentanen Situation und den Axiomen der Domäne herzuleiten. In diesem Beweisprozess werden Lücken auftreten, sonst wären keine Aktionen seitens des Agenten notwendig. Wenn eine solche Lücke durch eine Aktionsausführung geschlossen werden kann, so setzt der Planer den Versuch, diese Aktion auszuführen, in den Plan ein. Der Planer benutzt also zunächst die Regeln für einen rückwärts schließenden Prozess (vom Ziel zur momentanen Situation), um notwendige Aktionsausführungen zu finden. Und er benutzt dann die Regeln und die gefundenen Ausführungsversuche, um die dann erreichte Situation vorherzusagen. Bei beiden Richtungen werden jeweils die zeitlichen Einschränkungen, die in den Regeln formuliert sind, entsprechend den Mechanismen der Intervallalgebra berücksichtigt.

Der Planer trifft dabei Hypothesen, dass gewisse Intervallvariablen das gleiche Zeitintervall beschreiben. Zum Beispiel könnte eine Aussage in einem Zeitintervall gelten, das direkt an eine Aktionsausführung anschließt. Ist diese Aussage nun für eine spätere Aktionsausführung als Voraussetzung gefordert (die Aussage muss also in einem Zeitintervall gelten, das direkt vor der Aktionsausführung liegt), so kann der Planer annehmen, dass sich das Gültigkeitsintervall der Aussage von der ersten Aktionsausführung bis zur zweiten erstreckt. Diese als Persistenzhypothesen bezeichneten Gleichsetzungen werden in einem nachgeschalteten Algorithmus verifiziert. Im Fall von Widersprüchen wird der Planer zurückgesetzt.

Mit dem Ansatz von Allen sind einige der in den ersten beiden Ansätzen vorhandenen Beschränkungen aufgehoben. Zu dem Planungsalgorithmus ist jedoch zu sagen, dass er je nach Umsetzung des Constraint-Lösers für die zeitlichen Einschränkungen entweder unvollständig oder NP-schwer ist (siehe z. B. [Vilain/Kautz 86]). Das Konzept eignet sich sowohl für das Planen gleichzeitiger Aktionen, als auch für das hierarchische Planen (im Gegensatz zum deduktiven Planen), wie es im folgenden Ansatz von F. v. Martial benutzt wird.

Hierarchisches Planen

Während es beim deduktiven Planen darum geht, aus einer Menge von Aktionen eine Aktionsfolge zu bestimmen, die zu einem Ziel führt; bestimmen

hierarchische Planer Instantiierungen abstrakter, vorgefertigter Pläne aus einer Planbibliothek, die dann für den Agenten ausführbar sind. Es geht also um eine Verfeinerung, Konkretisierung des Plans, eine genauere Anordnung der Einzelaktionen.

Eine Anwendung des hierarchischen ereignisbasierten Planens zeigt die Arbeit [v. Martial 92]. Der Schwerpunkt dieser Arbeit liegt in der Repräsentation von Aktionen und Plänen für das koordinierte Zusammenwirken mehrerer Agenten. Auch v. Martial verbindet Aktionen mit ihren zeitlichen Beziehungen. Er führt zunächst Sorten für Objekte, Ressourcen, Agenten, Aktionen, Pläne, Zeitpunkte, Zeitintervalle, Zeitdauern und Mengen ein und baut darauf eine algebraische Beschreibung seines Aktions- und Planungsformalismus auf.

Für die Definition der lokalen Ausführbarkeit einer Aktion benutzt der Ansatz eine Menge von vorbereitenden und verhindernden Aktionen. Im Gegensatz zu den vorher vorgestellten Arbeiten sind Voraussetzungen also nicht durch Aussagen bzw. Attribute beschrieben, sondern ebenfalls durch Aktionen. Eine Aktion ist lokal ausführbar, wenn vorher eine Menge von vorbereitenden Aktionen ausgeführt und keine dieser Aktionen rückgängig gemacht wurde und wenn außerdem Mengen von verhindernden Aktionen nicht vorher ausgeführt oder wieder rückgängig gemacht wurden.

Zur genauen Beschreibung von Aktionen führt v. Martial Funktionen ein, die den Aktionen benötigte Ressourcen, planende Agenten (konkretisieren notwendige Schritte der Aktion), ausführende Agenten (führen Schritte aus) und die Dauer ihrer Ausführung zuordnen. Im Unterschied zu den zuvor vorgestellten Arbeiten kann eine Aktion in Teilaktionen zerlegt werden. In der bisher in dieser Arbeit verwendeten Terminologie heißen solche zusammengesetzten Aktionen bereits Plan. Bei v. Martial steht der Begriff »Plan« allerdings für zur Ausführung ausgesuchte Aktionen: Plan also im Sinne der aktuell nächsten Schritte und nicht als allgemeine Schrittfolge. So entspricht die von v. Martial unterstellte Bedeutung eher dem in dieser Arbeit verwendeten Begriff der Absicht.

Als Primitive der Zeitrepräsentation benutzt er Zeitdauerangaben und Zeitpunkte. Zeitintervalle (wie bei Allen verwendet) ergeben sich aus Paaren von Zeitpunkten. Zur zeitlichen Verknüpfung dienen absolute Beziehungen, die Aktionen in Bezug auf eine Zeitachse festlegen, und relative Beziehungen (analog den von Allen eingeführten), die Aktionen untereinander anordnen. Ein Plan besteht schließlich aus einer Menge von Aktionen und einer Relation, die den Aktionen gerade expandierte (bearbeitete) Teilaktionen zuweist. Diese Relation stellt den momentanen Bearbeitungsstand des Planes dar. Daraus wird ersichtlich, dass v. Martial mit dem Begriff »Plan« einen schon instantiierten Plan meint. Teilaktionen von Aktionen entstehen in einem Verfeinerungsprozess, bis schließlich atomare oder noch nicht zerlegte Aktionen erreicht sind. Einen Plan kann man sich also als gerichteten Graphen vorstellen, der von der allgemeinen Aufgabe bis zu den atomaren oder noch nicht zerlegten Aktionen

führt. Die Aktionen sind dabei nicht als geordnet zu verstehen, sondern sind zeitlich durch relative Einschränkungen in Beziehung gesetzt.

Die Einordnung einer Aktion als nicht teilbare Aktion ist eine Frage der Fähigkeiten eines Agenten. Ein Agent muss nicht in der Lage dazu sein, eine Aktion weiter zu verfeinern, obwohl es in der Domäne nötig wäre. Das bedeutet, dass der Agent diese Aktion nicht selbst ausführen kann. Blätter in einem Plangraphen müssen also nicht atomaren Aktionen der Domäne entsprechen. Durch die Kommunikation von Plänen oder der Blätter von Plänen können andere Agenten weitere Verfeinerungen vornehmen.

Was die koordinierte Ausführung von Plänen betrifft, so gibt es mehrere Beziehungen, die berücksichtigt werden müssen. So können Pläne nicht zusammenpassen, weil in ihnen Aktionen enthalten sind, die die gleichen Ressourcen benötigen. Diese Pläne müssen synchronisiert werden. Pläne sind inkompatibel, wenn sie Aktionen mit entgegengesetzten Effekten enthalten. Positive Beziehungen zwischen Plänen bestehen, wenn sie identische Aktionen enthalten, ein Agent also auf die Ausführung verzichten könnte; oder wenn eine Aktion eine andere subsumiert. Eine weitere Beziehung zwischen zwei Plänen besteht, wenn eine Aktion von einem Agenten geplant, von einem anderen aber ausgeführt werden soll. Vor der Ausführung der Pläne übernimmt ein Agent deshalb deren Koordination. Er schränkt also die zeitliche Anordnung der Einzelaktionen weiter ein.

Auch in dem Ansatz von v. Martial sind demnach Algorithmen gefordert, die entweder NP-schwer oder unvollständig sind, da sich die Konfliktauflösung auf die Propagierung in einem Allenschen Constraintnetz bezieht. Trotzdem ist der Aspekt interessant, Aktionen aufeinander aufzubauen, notwendige Vorbedingungen auf das Vorhandensein von Ressourcen einzuschränken.

Aktionen und Pläne in dynamischen Umgebungen

Pläne in dynamischen Umgebungen müssen außerdem damit zurechtkommen, dass sich die Umgebung ohne Zutun des Agenten, also außerhalb seines Einflusses, verändert. Das ständige Anpassen des verfolgten Planes oder seine Neuplanung wird notwendig. S. Wood widmet sich in ihrer Arbeit ([Wood 93]) der speziellen Repräsentation von Plänen für diese Anforderung. Der erste Unterschied ist, dass die Agenten hier beobachtete Daten über die sich permanent ändernde Umgebung in der Planung berücksichtigen müssen. Allerdings können durch die Beobachtung auch erwartete Effekte überprüft werden.

Der vorgeschlagene Ansatz geht davon aus, dass es in einer dynamischen Umgebung nicht ausreicht, Pläne aus vorher existierendem Wissen abzuleiten. Stattdessen geht es beim Planen hier darum, wie man mit einfachen Aktionen dynamisch erzeugte Absichten erreichen kann, wie man auf Einschränkungen und Veränderungen der Umgebung reagiert. Das Problem besteht hauptsächlich darin, dass man Aktionen für eine Situation festlegen muss, die man noch

nicht wahrgenommen hat. Während die Aktionsauswahl stattfindet, ändert sich die Situation bereits wieder. Diese Situationsänderungen muss man deshalb durch Simulation mit berücksichtigen. Dazu benötigt man zusätzliches Wissen über die Dynamik des Systems (z. B. entsprechend den Vorgaben der qualitativen oder quantitativen Simulation). Außerdem kann es notwendig sein, die Absichten anderer Agenten in der Umgebung aus seinen Aktionen abzuleiten, um auf dessen weitere Aktionen zu schließen.

Aktionen beziehen sich in dieser Arbeit auf die Steuerung gewisser Attribute. Aktionen können den Wert des Attributs erhöhen oder verringern (z. B. die Geschwindigkeit eines Autos). Planen besteht dann in der Auswahl der zu ändernden Attribute und dem Berechnen der passenden aktuellen Parameter für die Aktionen. Nach der Ausführung der berechneten Aktionen oder beim Eintreten von Ereignissen, die die Abarbeitung unterbrechen, berechnet der Agent den nächsten Plan, ohne noch den alten zu berücksichtigen. Lediglich die hinter dem letzten Plan liegende Absicht kann auch im nächsten Zyklus noch gelten. Absichten entstehen durch hierarchische Ableitung aus vorgegebenen Zielen. Dieser Ansatz ähnelt in vielem der Umsetzung der Fußballspieler im zweiten Anwendungsbeispiel. Im Unterschied zu den Fußballspielern kann der Agent hier jedoch aktiv Informationen von der Umgebung abfragen.

Bei der Berechnung der nächsten Aktionen werden hier zwar die Absichten anderer Agenten mit eingerechnet, es ist jedoch keine Kooperation zwischen Agenten vorgesehen. Die hier betrachteten Aktionen sind vorher festgelegt und sehr einfach (Werte erhöhen und verringern). Aktionen werden sequentiell ausgeführt.

Operationale Primitive in INTERRAP

Die bisher vorgestellten Arbeiten berücksichtigten jeweils unterschiedliche Aspekte des Planungs- und Repräsentationsproblems. Wenn es darum geht, die Mechanismen einem Softwareagenten zugänglich zu machen, müssen diese unterschiedlichen Aspekte gegeneinander abgewogen werden. Die folgende Arbeit, die bereits im letzten Kapitel vorgestellt wurde, soll nun noch spezieller als Beispiel dafür dienen, wie dieses Abwägen realisiert werden kann.

Die unterschiedlichen Aspekte werden in INTERRAP ([Müller 96]) durch eine Schichtenarchitektur berücksichtigt. Müller verwendet den Begriff »Operationales Primitiv« zur Kennzeichnung von Handlungen, die der Agent durchführen kann. In der Reaktionsschicht sind das vordefinierte so genannte Verhaltensmuster, die durch einen Namen, einen Typ (Reaktion oder Prozedur), benötigte Argumente, einen Zustand (aktiviert, nicht aktiviert, zur Ausführung bestimmt), eine Liste benötigter Ressourcen, eine Priorität, eine Aktivierungsbedingung für Reaktionen, Bedingungen zum Test der erfolgreichen oder fehlerhaften Abarbeitung, eine Liste von möglichen Ausnahmesituationen und schließlich einen Anweisungsteil gekennzeichnet sind. Sie speichern allerdings nicht den Grund (Ziel, Absicht) ihrer Ausführung.

Ein Agent besitzt zu jedem Zeitpunkt eine Menge von aktivierten Verhaltensmustern, die potentiell abgearbeitet werden. Eine Reaktion wird dabei aktiv, wenn die Aktivierungsbedingung erfüllt ist; eine Prozedur wird dagegen durch die lokale Planungsschicht oder durch andere Verhaltensmuster aktiviert. Ein anderer Unterschied zwischen Prozeduren und Reaktionen ist ihr Abschluss: Reaktionen enden, wenn ihre positive oder negative Terminierungsbedingung erfüllt ist; Prozeduren können zusätzlich von der lokalen Planungsschicht abgebrochen werden. Terminierungsbedingungen können zur Laufzeit hinzugefügt werden. Von den aktivierten Verhaltensmustern werden nur die abgearbeitet, die nicht dieselben Ressourcen benötigen. INTERRAP sucht dafür die größte Menge (Größe zum Beispiel als Summe der Prioritäten) sich nicht widersprechender Verhaltensmuster aus. Der Anweisungsteil eines Verhaltensmusters ist in seiner Komplexität nicht begrenzt. Die Ausführung kann demnach Zeit kosten. Verhaltensmuster sind deshalb in Einzelschritte gegliedert. Nach jedem Einzelschritt kann sich der Agent entscheiden, das Muster fortzusetzen oder abzubrechen. Zur Formulierung des Anweisungsteils benutzt INTERRAP eine eigene Programmiersprache.

In der Reaktionsschicht entfällt das Planen: ist ein Verhaltensmuster aktiviert, wird sein Anweisungsteil ausgeführt. Für echtes hierarchisches Planen ist die lokale Planungsschicht zuständig, die über dem reaktiven Verhalten ein zielgerichtetes Verhalten realisiert. Die Planungsschicht besteht aus einem Planerzeuger, der Pläne einer Planbibliothek instantiiert; einer Planauswertung, um den besten passenden Plan auszusuchen; einem Planinterpreter, der den gewählten Plan entsprechend der Plansprache abarbeitet; einem Terminplaner, der Ausführungszeitpunkte der in den Plänen enthaltenen Verhaltensmuster festlegt, und schließlich einem Ausführungsmodul, das die Verhaltensmuster fristgerecht aktiviert. Die Planbibliothek besteht aus Einträgen mit Anwendungsbedingung-Plan-Paaren. Im allgemeinen geschieht das Instantiieren eines Planes durch Unifizieren eines Ziels mit der Anwendungsbedingung.

Wie beim hierarchischen Planen üblich, muss ein Plan noch nicht vollständig durch Aktionen festgelegt sein, sondern kann weitere Ziele enthalten, zu deren Bearbeitung ein erneuter Zugriff auf die Planbibliothek notwendig wird. Diese Teilziele können eventuell erst während der Abarbeitung des Plans durch die dann erreichte Situation genauer bestimmt werden. Ein Plan besteht also aus der sequentiellen oder disjunktiven Verknüpfung von Subplänen. Auch Wiederholungen und bedingte Ausführungen sind zugelassen. Die Primitive der Planbildung sind Aufrufe der Verhaltensmuster aus der Reaktionsschicht.

In einer weiteren Schicht repräsentiert INTERRAP Kooperationspläne, die wiederum auf lokalen Plänen aufbauen. Kooperationspläne werden nach einem Verhandlungsprozess aktiviert, jeder beteiligte Agent arbeitet dann seinen Teil ab (siehe Abschnitt 3.5).

Formalisierung von Angeboten und Aufträgen

Ein ganz anderer Aspekt der Repräsentation der Fähigkeiten (Aktionen, Pläne) eines Agenten ist deren Rolle bei der Auftragsvergabe und -übernahme in einem System von Agenten. Die Angebots- und Auftragsbeschreibung hängt eng mit der Repräsentation von Aktionen zusammen. Auch hier ist es möglich, Ausgangspunkt und Ziel des Auftrags (Angebots) anzugeben oder aber die genaue Schrittfolge aufzulisten. Es wird im Allgemeinen nicht reichen, Argumente und Namen der anzuwendenden Fähigkeit anzugeben.

Die folgende Arbeit zeigt einen Ansatz für diese Problemstellung. In [Kirn 91] stellt S. Kirn eine Task Description Language (operationale Beschreibung des Auftrags) vor. Sie dient insbesondere dazu, Aufträge für andere Agenten zu formulieren und die Kompetenz eines Agenten zur Auftragsbearbeitung einzuschätzen. Die Arbeit behandelt insgesamt das Problem der Kooperation zwischen föderativen wissensbasierten Systemen. Damit ist die Art der ausführbaren Aktionen eingegrenzt. Es geht im Allgemeinen um die Beantwortung von Anfragen auf der Basis abgespeicherten Expertenwissens.

Im Unterschied zu allen anderen hier vorgestellten Arbeiten benutzt die Auftragsbeschreibungssprache attributierte eindeutige kontextfreie Grammatiken und durch sie definierte Syntaxbäume als Ausdrucksmittel. Eine Auftragsbeschreibungssprache ist also die durch eine eindeutige kontextfreie Grammatik beschriebene kontextfreie Sprache. Aufträge sind Worte dieser Sprache. In einer eindeutigen kontextfreien Grammatik gibt es zu jedem Wort der durch die Grammatik beschriebenen Sprache genau einen Syntaxbaum, dessen Wurzel das Startsymbol der Grammatik repräsentiert. Mit einem solchen durch Parsen des Auftragswortes aufgebauten Syntaxbaum ist ein Auftrag zusammen mit den für ihn notwendigen Teilaufträgen repräsentiert. Allgemeiner gesagt, ist hier ein (Teil-)Auftrag also ein (Teil-)Baum eines Wortes der Auftragsbeschreibungssprache. Die Blätter des Syntaxbaumes repräsentieren direkt ausführbare Aktionen.

Da in reinen kontextfreien Sprachen Nichtterminale durch Regeln unabhängig von ihrem Kontext (d.h. anderen Aufträgen) abgeleitet werden, in einer Folge von Aufträgen Beziehungen zwischen Aufträgen jedoch nichtsdestotrotz bestehen, führt Kirn eine Menge von Attributen für jedes Grammatiksymbol (Terminal, Nichtterminal) ein, die während der Ableitung des Syntaxbaumes Informationen zwischen den Aufträgen vermitteln. Die Berechnungsvorschriften für die Attributwerte werden den Produktionsregeln der Grammatik zugeordnet. Das entspricht dem Begriff der syntaxgesteuerten Definition. Solche Attribute können einerseits von Attributen des Vorgängerknotens oder linker Nachbarknoten im Syntaxbaum abhängen. Diese Attribute können also während des Aufbaus des Syntaxbaumes, vor der Ausführung des Auftrages bestimmt werden (top-down; ererbte Attribute). Andere Attribute berechnen sich aus Attributen rechter Nachbarknoten oder der Nachfolgerknoten im Syntaxbaum. Diese können erst nach dem vollständigen Aufbau des Baumes in

einem Berechnungsvorgang von den Blättern zur Wurzel synthetisiert werden (bottom-up; synthetisierte Attribute), repräsentieren also Ergebnisse der Auftragsausführung.

Neben dieser operationalen Beschreibung der Aufträge geht es in der allgemeinen Auftragsbeschreibung darum, allgemein zur Auftragsbearbeitung notwendige Kriterien zu nennen. Es sind im Allgemeinen die operationale Beschreibung ergänzende Merkmale notwendig. Zuerst sind das offensichtlich die aktuellen Parameter. Kirn nennt entsprechend der Anwendung in der Verknüpfung wissensbasierter Systeme als charakterisierende Merkmale außerdem den Inferenztyp (vorwärts oder rückwärts verkettend), den Problemlösungstyp (z. B. Diagnose, Konstruktion oder Simulation), den Kooperationstyp (Abgabe, Konsultation, Überprüfung, Beauftragung, Zusammenarbeit), den Wissenstyp (nichtmonoton, fehlerbehaftet, heuristisch, temporal, qualitativ), die Problemlösestrategie (Form der Wissensrepräsentation, Struktur des Lösungsraumes), benötigte Ressourcen (Hard-, Software, Peripherie), die verwendete Sprache zur operationalen Beschreibung des Auftrags (TDL, wie oben) und die Priorität des Auftrages. Diese Merkmale kennzeichnen also einen Auftrag näher und können der Kompetenzeinschätzung eines potentiellen Auftragnehmers dienen. Die Auflistung dieser Merkmale ist offensichtlich abhängig von der Anwendung und bietet im Wesentlichen die Grundlage einer Heuristik für die Kompetenzeinschätzung.

Ein in der Kooperation beteiligtes Expertensystem muss dazu neben der eigentlichen Wissensbasis eine Wissensbasis ihrer Fähigkeiten aufbauen. In diese zweite Wissensbasis fließen dieselben Kategorien ein, wie in eine allgemeine Auftragsbeschreibung. Die hier verwendeten Mechanismen werden später im Abschnitt zur Kooperation zwischen Agenten genauer betrachtet.

Zusammengefasst verfolgt dieser Ansatz mit seinem Schwerpunkt auf der Kooperation also ein ganz anderes Ziel als etwa die Planung in dynamischen Systemen. Mit der Repräsentation von Aufträgen (letztlich Plänen und Aktionen) als Worte einer Sprache und der Angabe zusätzlicher charakterisierender Merkmale wird eine effektive Auftragsvergabe möglich. Die operationale Beschreibung entspricht dabei dem Konzept des hierarchischen Planens. Allerdings ist der Ansatz entsprechend der vorgesehenen Anwendung eingeschränkt. Für die Bearbeitung eines Auftrages gibt es hier für einen Agenten immer genau eine Variante. Diese ist durch die Auftragsbeschreibungssprache vorgegeben. Besitzen zwei Agenten unterschiedliche Lösungswege (unterschiedliche Ableitungsbäume), so operieren sie über unterschiedlichen Grammatiken. Die benötigte Ausführungszeit für Aufträge spielt keine Rolle.

Andere Ansätze

Neben den direkt im Bereich der KI betrachteten Varianten der Aktions- und Plandarstellung gibt es weitere Ansätze aus anderen Bereichen der Informatik, die in Agentensystemen angewendet wurden. So verwendet Concurrent

MetateM ([Fisher 94]) eine temporale Logik zur Verhaltensbeschreibung und MAGSY ([Fischer 93]) Petrinetze als Ausgangspunkt für Aktionen. Beide Ansätze eignen sich gut, um Eigenschaften im Agenten- und Systemverhalten zu überprüfen. Sie gehen jedoch jeweils von fest vorgegebenen und vollständigen Beschreibungen des Diskursbereichs aus. Das ist nicht in allen Anwendungsgebieten möglich. Allerdings sollte man diese Verfahren beachten, wenn sie anwendbar sind. So könnte ein Teil des Agentenverhaltens sehr wohl fest vorgegeben sein und auch in Hinblick auf Verklemmungen, Fairness- und Lebendigkeitseigenschaften untersucht werden.

Zusammenfassung

Die Repräsentation von Aktionen und Plänen muss hinsichtlich sehr unterschiedlicher Kriterien betrachtet werden. Im Allgemeinen gibt das konkrete Anwendungsgebiet, die Domäne, die primitiven Operationen vor, die ein Agent ausführen kann. Von ihr hängen auch weitere Entscheidungen ab, so ob eine häufige Neuplanung, die Zusammenarbeit mit anderen Agenten, die zeitliche Ausdehnung und Überlappung oder die gleichzeitige Ausführung von Aktionen eine Rolle spielen. Die vorgestellten Arbeiten zeigen jeweils, wie diese unterschiedlichen Kriterien in einem Aktionsformalismus berücksichtigt werden können. Bei der Aktionsdarstellung kommt es außerdem nicht nur auf die Nutzbarkeit in Planungsalgorithmen, sondern auch auf die Nutzbarkeit für die Kooperation zwischen Agenten an. Dabei spielen dann neben der operationalen Beschreibung und den Vor- und Nachbedingungen auch andere charakterisierende Eigenschaften eine Rolle.

3.2.2 Ausprägungsvarianten

Für ein Framework zur Programmierung von Agenten wird es nicht ausreichen, eine Art der Aktionsrepräsentation und der Planung anzubieten, da kein Mechanismus auf alle Domänencharakteristiken gleichzeitig gut passt. Es müssen stattdessen Kriterien für die Verwendung unterschiedlicher Varianten aufgestellt werden. Auch die Verwendung eines neuen Mechanismus muss möglich sein. Zwischen unterschiedlichen Aktionsdarstellungen soll es, wenn möglich, Übersetzungsalgorithmen geben oder Mechanismen zum Lesen und Schreiben eines gemeinsamen Formats.

Die hier vorgestellte Architektur benutzt für die Repräsentation von Operationen, die ein Agent ausführen kann, eine Trennung der operationalen Beschreibung (das »Wie«, als Aktion) von der charakterisierenden Beschreibung (das »Was«). Durch diese Trennung wird die Kombination unterschiedlicher Darstellungsformen beider Beschreibungsaspekte möglich.

Operationale Beschreibung

Wie im letzten Abschnitt herausgearbeitet, lässt sich die Art der operationalen Beschreibung hinsichtlich ihrer Transparenz für den Agenten charakterisieren. Vorkompilierte (nicht transparente) Aktionen sind in einer Programmiersprache programmiert und dem Agenten nur zur Ausführung bzw. Unterbrechung zugänglich. Bei Plänen kann der Agent dagegen die Einzelschritte und deren zeitliche Beziehungen feststellen. Für vorkompilierte Aktionen muss das Framework keine weitere Hilfe anbieten, der Programmierer muss für jede vorgesehene derartige Aktion eine spezielle Aktionsklasse programmieren, die die vorgegebene Schnittstelle einhält. Durch den Klassennamen erhält die Aktion einen eindeutigen Namen. Zur Ausführung wird die Klasse instantiiert (hier ist ggf. eine Wiederverwendung von Objekten möglich), das Objekt mit Parametern versehen und dessen `perform()`-Methode aufgerufen.

Pläne können dagegen ganz unterschiedliche Gestalt haben und sind in einer Plansprache (im einfachsten Fall lässt diese nur sequentielle Verknüpfungen zu) formuliert. Das einzige gemeinsame Charakteristikum ist ihr Bezug auf Teilaktionen. Teilaktionen eines Plans sind offensichtlich nicht mit ihrer operationalen Beschreibung im Plan enthalten, sondern durch ihren Aufruf: einen eindeutigen Bezeichner und aktuelle Parameter. Dieser Bezeichner kann nun sowohl vorkompilierte Aktionen als auch Pläne identifizieren, Teilaktionen können dadurch auch wieder Pläne sein. Diese Darstellung ist also offen für hierarchisches Planen. Der Aufruf einer vorkompilierten Aktion wird zur Erzeugung einer entsprechenden Verpflichtung oder Aktivität führen, der Aufruf eines Plans im Allgemeinen dazu, dass er erzeugt wird.

Als Quelle von Plänen kommen verschiedene Varianten in Frage: Sie können sowohl durch zustandsbasierte oder hierarchische Planung entstehen als auch innerhalb einer Fähigkeit mit speziellen vorher festgelegten Mitteln berechnet werden (wie z. B. in der Fußballanwendung). Wie auch immer sie zustande kommen, das Ergebnis ist ein Plan-Objekt. Soll es für diese Objekte nun aber spezielle Plan-Klassen geben (analog zu den Aktionen), oder genügt eine allgemeine Plan-Klasse? Das Hauptmerkmal von Plänen ist die symbolische Repräsentation der Teilaktionen und ihrer Verknüpfungen. Diese müssen also auf jeden Fall zugreifbar sein. Andererseits unterscheiden sich Pläne einer Repräsentationsform nur durch die enthaltenen Teilaktionen, der Mechanismus ihrer Bearbeitung (ihre Interpretation) ist gleich. Es muss deshalb Planklassen für unterschiedliche Repräsentationsformen geben. Konkrete Pläne existieren jedoch innerhalb des laufenden Agenten nur als Plan-Objekte oder in Fähigkeitsklassen, außerhalb von ihm in Form einer Datensammlung (z. B. textuelle Planbibliothek). Falls allerdings Flexibilität zugunsten einer besseren Effizienz aufgegeben werden soll, so ist es möglich, aus Plänen vorkompilierte Aktionen (also Klassen) herzustellen.

Bei einem Plan muss man strikt zwischen einem zur Ausführung konkret instantiierten Plan und einem Plan als Beschreibung einer Aktionsabfolge (z. B. in

einer Planbibliothek) unterscheiden. Die hier betrachtete Datenstruktur betrifft hauptsächlich den ersten Gesichtspunkt. Die Ausführung eines Plans unterscheidet sich jedoch von der Ausführung einer vorkompilierten Aktion. Bei einem Plan müssen Teilpläne eventuell erst erzeugt werden, erst enthaltene vorkompilierte Aktionen können zur direkten Ausführung gelangen. Neben der Transparenz der Beschreibung gibt es demnach auch einen konzeptionellen Unterschied zwischen Aktion und Plan. Die vorgeschlagene Verwendung des Entwurfsmusters Composite verwischt diesen Konzeptionsunterschied. Trotzdem kann es in bestimmten Anwendungen notwendig sein, einen Plan auszuführen. Die Methode `perform()` eines Plans muss deshalb die Aufgaben eines Planinterpreters übernehmen. Diese Herangehensweise ist durch die oben begründete Benutzung von verschiedenen Planklassen für verschiedene Formen der Planrepräsentation unterstützt.

Von den Planvarianten, die bei der Assistenz-Anwendung vorgestellt wurden, spielt hier nur die Struktur eine Rolle. Die Zuordnung der Planteile zu ausführenden Agenten wird erst bei der Ausführung interessant. Die Neuplanung ist konzeptionell auf einer höheren Stufe angesiedelt. Die Struktur eines Planes betrifft die Reihenfolge der Planteile, ihre zeitliche Beziehung. Sie kann sowohl eine lineare oder partielle Ordnung als auch ein Allensches Constraintnetz sein.

Aus diesen Betrachtungen ergibt sich das Klassendiagramm in Abbildung 3.7.

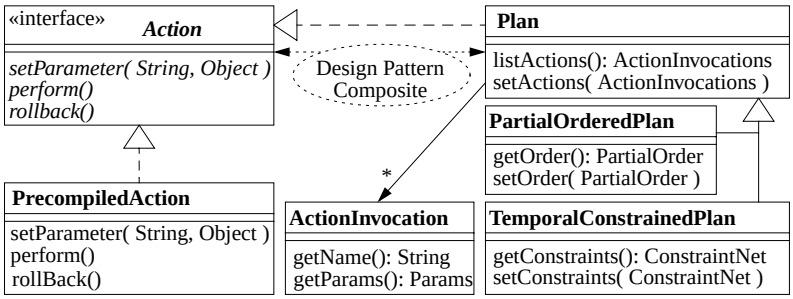

Abbildung 3.7: Diagramm für die operationale Beschreibung

Charakterisierende Beschreibung

Eine Fähigkeit wird nun als ein Tripel aus Name, operationaler Beschreibung und charakterisierender Beschreibung definiert. Im Gegensatz zu der bisherigen Begriffsbildung ist die charakterisierende Beschreibung hier nochmal extra gekapselt. Diese Wahl unterstützt die Kombinationsmöglichkeit beider Aspekte besser.

Die charakterisierenden Beschreibungen von Aktionen und Plänen in den eingangs vorgestellten Systemen besitzen als gemeinsame Information die Vor- und Nachbedingungen. Die Bedeutung dieser Vor- und Nachbedingung variiert allerdings. Beide dienen jedoch für den Agenten dazu einzuschätzen, ob die

Aktion ausführbar ist, und dazu, Konsequenzen der Aktion in die aktuelle Situation einzuarbeiten, falls ihre Ausführung aus Sicht des Agenten erfolgreich war. Neben den Vorbedingungen kann man die möglichen Parameter und ihre Wertebereiche sowie die benötigten Ressourcen beschreiben. Ressourcen definieren Synchronisationspunkte zwischen Aktionen: im Allgemeinen wird nur eine Aktion die Ressource zur selben Zeit benutzen können (siehe Abbildung 3.8).

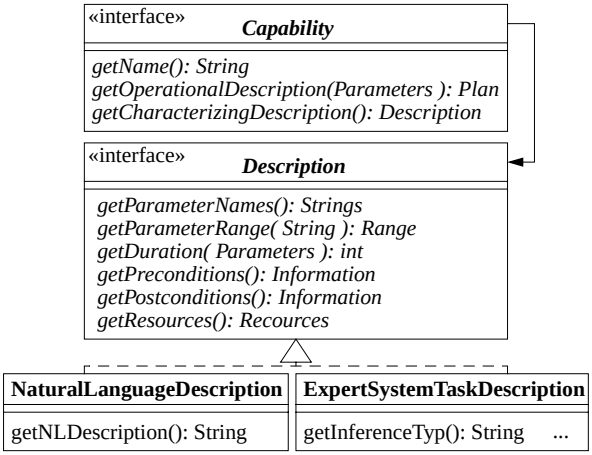

Abbildung 3.8: Diagramm für die charakterisierende Beschreibung

Weitere Charakterisierungen sind wieder von der Anwendung abhängig. So würde eine Beschreibung für die Anwendung bei der Kooperation mehrerer Expertensysteme wie in [Kirn 91] Angaben zum Inferenztyp, Problemlösungstyp etc. enthalten. Für das Assistenz-Programm wäre eine zusätzliche natürlichsprachliche Kurzbeschreibung wichtig.

Planen als Fähigkeit

Für die Formulierung der operationalen Beschreibung einer Fähigkeit gibt es nun verschiedene Varianten, die sich für unterschiedliche Anwendungen eignen. Für die Planung in dynamischen Umgebungen, wie in [Wood 93] oder beim Fußballer, wo es darum geht, in der Umgebung durch eine geringe Anzahl von berechneten Aktionen gezielt zu agieren, bieten sich speziell programmierte Fähigkeitsklassen an. Aus den übergebenen Parametern wird die entsprechend parametrisierte Aktionsfolge mit mathematischen Methoden oder Methoden der Simulation berechnet. Hat man jedoch die Pläne und Aktionen für einen hierarchischen Planungsprozess etwa durch die Konstruktion einer Planbibliothek vorbereitet, so können die zugehörigen Fähigkeiten automatisch als Objekte einer speziellen Fähigkeitsklasse erzeugt werden. Alle diese Objekte besitzen eine Referenz auf den hierarchischen Planer. Wird eine solche Fähigkeit mit aktuellen Parametern aktiviert, so leiten sie die Anfrage an

den hierarchischen Planer weiter, der die notwendigen Teilpläne anhand des Namens der Fähigkeit und der Parameter zusammenstellt.

Auch das klassische zustandsbasierte Planen kann man nun als spezielle Fähigkeit formulieren: Die Parameter dieser Fähigkeit enthalten das Ziel der Planung, und der Planungsalgorithmus sucht aus der Menge der Fähigkeiten (zugreifbar über die Expertise-Wissensbasis) eine passende Kombination aus (siehe Abbildung 3.9). Ein so gefundener Plan könnte Grundlage einer neuen Fähigkeit werden.

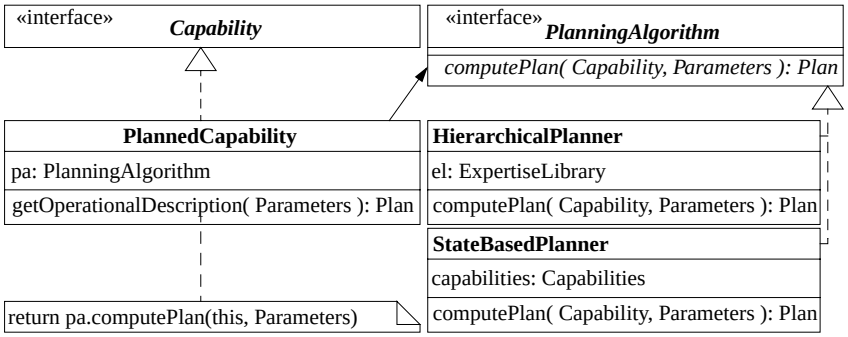

Abbildung 3.9: Diagramm für geplante Fähigkeiten

Auswirkungen von Aktionen

Was das oben besprochene Problem der Auswirkungen von Aktionen auf die intern gespeicherte Umgebungssituation angeht, so sind diese zunächst in der charakterisierenden Beschreibung mit enthalten. Wurde die Aktion aus Sicht des Agenten ohne Fehler ausgeführt, so muss man die durch die Aktion bewirkten Veränderungen in die Situation mit aufnehmen und speziell kennzeichnen. Aus diesen Veränderungen können sich aufgrund der Umgebungsgesetzmäßigkeiten weitere Änderungen ergeben. Auch diese sind zu kennzeichnen. Der Agent kann weitere Entscheidungen auf den so aktualisierten Daten treffen. Es kann jedoch sein, dass die angenommene Entwicklung der Umgebung von der tatsächlichen abweicht. Dieser Fehler kann erst durch die nächsten aufgenommenen Umgebungsinformationen korrigiert werden. In sehr dynamischen Umgebungen ist es dann nur sinnvoll, sehr kurze Aktionsfolgen zu erzeugen und dann oft neu zu planen.

Fähigkeiten als Angebotsbeschreibung

Sind nun Fähigkeiten durch ihren Namen, ihre operationale und charakterisierende Beschreibung festgelegt, so eignet sich der charakterisierende Anteil sehr gut für die Formulierung von Angeboten. Ein Angebot ist nichts anderes als der Name und die charakterisierende Beschreibung einer Fähigkeit, die ein Agent in der Lage ist anzuwenden.

Fähigkeiten und Aufträge

Ein Auftrag besteht dagegen aus dem Namen einer anzuwendenden Fähigkeit und den aktuellen Parametern oder aus einem auszuführenden Plan. Unter der Voraussetzung, dass der Agent die einzelnen Planschritte ausführen kann, sind also beide Auftragsformen formulierbar: der detaillierte operational beschriebene Auftrag und der Auftrag ohne Angabe der dazu notwendigen Schritte.

Neue Fähigkeiten

Der letzte Abschnitt erwähnte die Möglichkeit der Neuaufnahme neuer Fähigkeiten. Diese Neuaufnahme hat durch die Trennung von operationaler Beschreibung und charakterisierender Beschreibung nun mehrere Aspekte. Zunächst können neue Fähigkeiten durch Planung (siehe nächster Punkt) oder durch Kommunikation entstehen. Gegenstand der Kommunikation kann nun sowohl die gesamte Fähigkeit sein (der Agent weiß danach, wie sie auszuführen ist und was es mit ihr auf sich hat) als auch nur der charakterisierende Anteil. Der Agent wüsste damit, dass ein anderer Agent in der Lage ist, die Fähigkeit anzuwenden. Dann könnte der Agent sie analog zu eigenen Fähigkeiten benutzen, Aufrufe aber an den anderen Agenten weiterleiten. Das setzt natürlich voraus, dass dieser den Auftrag auch ausführt.

Eine andere Quelle neuer Fähigkeiten ist das Experiment. Dazu ist es notwendig, dass der Agent eine Reihe schon vorhandener Fähigkeiten zu einem vorher nicht benutzten Plan zusammenstellt und das Ergebnis seiner Ausführung in der Umgebung überprüft. Hier gibt es jedoch von praktischer Seite Einschränkungen; nicht jedes machbare Experiment ist erlaubt. Man müsste eine Testumgebung schaffen, in der der Agent sich die Fähigkeiten erzeugt.

Effizienz vs. Flexibilität

Im letzten Abschnitt wurde schon darauf hingewiesen, dass das Verhältnis zwischen vorkompilierten Aktionen und Plänen das Verhältnis zwischen Effizienz und Flexibilität widerspiegelt. Hier sei kurz als Idee skizziert, wie aus einem Plan eine Aktion kompiliert werden kann.

Pläne entstehen zur Laufzeit als Ergebnis der Anwendung einer Fähigkeit. Pläne sind in Objekten gespeichert, in Objekten, die die Aktion-Schnittstelle erfüllen. Ein solches Objekt kann man abspeichern. Konstruiert man nun zu dieser abgespeicherten operationalen Beschreibung noch einen Namen und die zugehörige charakterisierende Beschreibung (aus der Ausgangsfähigkeit und den Anforderungen ihrer Anwendung), so kann bei nochmaligem Gebrauch der Fähigkeit die operationale Beschreibung (der Plan) geladen und nicht berechnet werden. Dieses Verfahren ist natürlich nur sinnvoll, wenn das Laden schneller geht als das Berechnen und wenn als Folge dieses Verfahrens die Menge an Fähigkeiten nicht unverhältnismäßig wächst. Als zusätzliche Bedingung darf

der Plan nur noch vorkompilierte Aktionen und keine anderen Pläne mehr enthalten.

Dieses Verfahren ist jedoch nur ein erster möglicher Schritt, denn noch muss der Planinterpreter diesen Plan ausführen, noch ist er für den Agenten nicht ein Ganzes. Der zweite Schritt besteht darin, eine Aktionsklasse zu erzeugen und diese zu kompilieren. Namen und charakterisierende Beschreibung kann man aus dem letzten Schritt übernehmen. Die Konstruktion der Klasse ist nun von der verwendeten Planungssprache abhängig. Im allgemeinen wird sie aber weniger mächtig als die benutzte Programmiersprache sein. Die Konstruktion besteht dann darin, die Schritte, die der Planinterpreter für den konkreten Plan ausführen würde, eins zu eins in die Methode `perform()` der konstruierten Aktionsklasse zu übernehmen und die Klasse dann zu übersetzen. Die so konstruierte vorkompilierte Aktion wäre für den Agenten nicht mehr transparent, könnte also z. B. nicht mehr genau mit anderen Plänen koordiniert werden (statt einer verzahnten Ausführung könnte z. B. eine Hintereinanderausführung nötig werden), aber die Ausführung der Aktion bedürfte weder der Planung noch der Interpretation.

3.2.3 Zusammenfassung

Dieser Abschnitt führte unterschiedliche Varianten und Aspekte der Aktions- und Plandarstellung sowie der Planung zusammen und machte sie für das in dieser Arbeit entwickelte Framework zugänglich. Für den Programmierer von Agenten ist die Formulierung der durch den Agenten ausführbaren Aktionen ein wesentlicher Gesichtspunkt. Trotz der Programmierung in einer objektorientierten Sprache ist hier durch die Anbindung hierarchischer und zustandsbasierter Planung eine Formulierung gemäß der dort verwendeten Vorgehensweise möglich. Lade- und Speichermethoden sorgen für die Umsetzung in Objekte. Besondere Merkmale der hier verwendeten Architektur sind die Einbindung der Planungsverfahren als Fähigkeiten, die Möglichkeit der Plankompilierung und die Möglichkeit der Neuaufnahme von Fähigkeiten.

3.3 Aktionsauswahl

Ein wichtiger Aspekt nach der Repräsentation der Fähigkeiten ist die Auswahl der zur momentanen Umgebungssituation und zu den aktuellen Zielen passenden Aktionen.

3.3.1 Aufgaben und Quellen

Dieser Auswahlmechanismus muss verschiedenen Anforderungen genügen: Er muss erstens zur Umgebungssituation und den Zielen passende Aktionsfolgen

berechnen, und er muss das zweitens rechtzeitig tun (Prinzip der »Rationalität«). So rechtzeitig, dass die ausgeführten Aktionen auch noch auf die berücksichtigten Bedingungen treffen. Zusätzlich soll der Algorithmus drittens für eine Stabilität des Verhaltens sorgen. Die Stabilität bezieht sich auf die Häufigkeit des Umentscheidens, auf das Abwägen zwischen Beibehalten und Verwerfen von Absichten.

Die Problematik des automatischen Auswählens einer von mehreren Alternativen ist thematisiert im Gebiet der Spieltheorie, dort vor allem im Bezug auf eine Gewinnstrategie (z. B. für Zwei-Personen- oder Matrixspiele). Dort spielen allerdings die Rechtzeitigkeit und die Stabilität keine Rolle. Eine reichhaltige Quelle ist auch die theoretische Forschung zu Agenten, insbesondere zur BDI-Architektur (Belief, Desire, Intention), die auf Arbeiten von Bratman ([Bratman 87, Bratman et al. 88]) zurückgeht. In manchen Arbeiten (z. B. [Wood 93]) gilt bereits das Planen als Aktionsauswahl. Schließlich müssen alle umgesetzten Agentensysteme eine Lösung für dieses Problem anbieten.

BDI

Folgt man den Arbeiten zur BDI-Architektur ([Cohen/Levesque 90, Rao/Georgeff 91, Bratman 87, Bratman et al. 88] und [d'Inverno et al. 98, Jung 98] für aktuelle Arbeiten), dann geht es bei der Auswahl um die Bestimmung von Zielen und Absichten aus Annahmen und Wünschen (Desires). Hinter den Absichten stehen (partielle) Pläne. Die Pläne geben die auszuführenden Aktionen vor. Absichten sind untereinander und in Bezug auf die Annahmen konsistent. Sie sorgen für die Stabilität des Verhaltens. Insofern liefern diese Arbeiten Hinweise auf die dritte Anforderung an den Auswahlalgorithmus, geben aber keine Anhaltspunkte für die zweite Anforderung. Wann sollte eine Absicht beibehalten oder verworfen werden? Cohen und Levesque formulieren in ihrer Arbeit mit dem Slogan »Eine Absicht ist Auswahl (eines Planes) und Verpflichtung (zu dessen Ausführung)« Kriterien für Absichten:

- Der Agent muss einen Weg (Plan) kennen, die Absicht zu erfüllen.
- Die Absichten des Agenten dürfen sich nicht widersprechen. Eine einmal gefasste Absicht kann zum Verwerfen widersprechender Absichtskandidaten führen.
- Der Agent soll erkennen, wann er eine Absicht erfolgreich umgesetzt hat oder wenn ihre Umsetzung fehlgeschlagen ist. Er soll Absichten bei Fehlschlag erneut verfolgen.
- Der Agent nimmt an, dass die Ausführung einer Absicht möglich ist.
- Der Agent nimmt nicht an, dass er die Absicht nicht erfüllen wird (Verpflichtung).
- Es gibt einen Zeitpunkt, zu dem der Agent annimmt, dass er den hinter der Absicht stehenden Plan ausführen wird.

- Der Agent beabsichtigt nicht alle logischen Folgerungen seiner Absichten.

Daraus geht hervor, dass der Agent die Absicht so lange verfolgen soll, bis er sie entweder erfolgreich umgesetzt hat, oder er annimmt, dass er sie nicht umsetzen kann. Diese beiden Alternativen sind in Implementierungen im Allgemeinen als erfolgreiche Endbedingung bzw. Fehler- (oder Ausnahme-)Bedingung für Fähigkeiten formuliert. Für die tatsächliche Auswahl der Absichten in Bezug auf Wünsche und Annahmen geben die theoretischen Arbeiten wenig Hinweise. Aus den obigen Kriterien geht lediglich hervor, dass sich die gewählten Absichten nicht widersprechen sollen. Implementierungen müssen die Auswahl deshalb konkretisieren; sie bewerten, berücksichtigen benötigte Ressourcen oder serialisieren die Ausführung (nur eine Absicht zu jedem Zeitpunkt).

Planen versus Aktionsauswahl

Einige Arbeiten (z. B. [Wood 93]) fassen, wie gesagt, das Planen als Aktionsauswahl auf. Dabei vermischen sie aber Aspekte, die zu einer klareren Strukturierung und zu einer besseren Effizienz führen können. Das eigentliche Planen bezieht sich nämlich nur auf die Wahl der Aktion nach Vorgabe eines Planziels, nicht jedoch auf die Wahl des Planziels. Absichten verkörpern Planziele. Sind die Planziele permanent und vorgegeben, dann reicht dieser Ansatz aus. Anderenfalls ist vor der Planung die Wahl der Absicht notwendig. Für die Wahl der Absicht sollte aus Effizienzgründen keine Planung, sondern eine Nutzenberechnung stattfinden.

Aktionsauswahl in Agent0

In Agent0 ([Shoham 93]) erfolgt die Aktionsauswahl nach einem zweistufigen fest vorgegebenen Algorithmus. Zunächst führen nur neu eintreffende Nachrichten potentiell zu neuen Aktionen. Der erste Schritt der Aktionsauswahl besteht in der Auswahl einer auf die Nachricht passenden Reaktionsregel. »Passen« bedeutet hier, dass die Nachricht einem bestimmten Muster genügt und dass weitere Bedingungen erfüllt sind. Agent0 wendet die erste passende Regel an. Sie führt im Allgemeinen zur Speicherung von Verpflichtungen, zu einer gewissen Zeit eine Aktion auszuführen. Der zweite Auswahlschritt betrifft die Verpflichtungen: Agent0 prüft die Bedingungen aller zum aktuellen Zeitpunkt auszuführenden Verpflichtungen (das sind auch solche, deren Ausführungszeitpunkt schon überschritten ist, die aber bisher nicht bearbeitet wurden) und führt sie davon abhängig aus. Die eigentliche Auswahl ist somit der erste Schritt, der zweite Schritt führt eine nochmalige Prüfung von Bedingungen aus.

Die Auswahl in Agent0 gleicht also im wesentlichen der Auswahl einer Regel durch einen Regelinterpreter. Regelinterpreter sind jedoch flexibler bei der Auswahl einer aus mehreren anwendbaren Regeln. Agent0 (zumindest seine Prolog-Variante) wendet, wie gesagt, die erste passende an.

Für die erste Anforderung an den Algorithmus, das Passen, ist hier der Programmierer zuständig. Er muss die Reaktionsregeln formulieren. Die zweite Anforderung, die Rechtzeitigkeit, wird in Agent0 nicht besonders beachtet. Zwar sind Verpflichtungen mit einem Zeitstempel versehen, die Einhaltung des Termins ist jedoch nicht garantiert. Auch die durch den Auswahlalgorithmus benötigte Zeit wird nicht überwacht. Diese ist aber von der Kompliziertheit des Bedingungstests und linear von der Anzahl der Regeln abhängig. Auch für die Abwägung der Stabilität sind in Agent0 keine speziellen Mittel vorgesehen. Der Algorithmus kann aber zumindest die eingegangenen und noch nicht abgearbeiteten Verpflichtungen in die Aktionsauswahl einbeziehen. Allerdings muss mit einer solchen Verpflichtung nicht die dahinterliegende Absicht bekannt sein.

Aktionsauswahl in GPGP

Für die Aktionsauswahl in GPGP ([Lesser 98]) spielt die Repräsentation der Aktionen und Aktivitäten in einer Aufgabenstruktur in der Repräsentationssprache TAEMS eine wesentliche Rolle. Eine Aufgabenstruktur umfasst, wie bereits früher erwähnt, Aufträge, mögliche Bearbeitungswege und die dazu notwendigen Fähigkeiten, versehen mit Bewertungen zu Qualität, Kosten und Dauer. Trifft ein Auftrag ein, so bewertet der Auswahlalgorithmus zunächst die Handlungsvarianten und bestimmt dann die am besten bewertete. Ein Scheduler ordnet sie entsprechend der von ihr benötigten Ressourcen in einen Abarbeitungszeitplan ein.

Interessant ist bei diesem Ansatz die Einbeziehung quantitativer Größen in die Definition der Anforderung »Passen«. Nicht nur das reine Ergebnis, sondern auch Dauer, Kosten und Qualität sind wichtig. Was die Rechtzeitigkeit der Entscheidung betrifft, so hängt sie hier von der für die Bewertung und den Vergleich benötigten Zeit ab, wenn die Aufgabenstruktur einmal aufgebaut ist. Da für die Ausführung von Aktionen Ressourcen nötig sein können, hängt der tatsächliche Ausführungszeitpunkt auch von deren Verfügbarkeit und so von der Einordnung in den Abarbeitungszeitplan durch den Scheduler ab. Die Stabilität des Verhaltens ergibt sich aus der Auftragsverfolgung.

Aktionsauswahl in INTERRAP

Wie bereits in vorigen Abschnitten erwähnt, repräsentiert INTERRAP ([Müller 96]) Umgebungsinformationen, Motivation und Expertise in drei Schichten, eine Reaktionsschicht, eine lokale Planungsschicht und eine Kooperationsschicht. An diese Einteilung in Schichten hält sich auch die Aktionsauswahl in INTERRAP.

Das Auswahlverfahren der Reaktionsschicht untersucht alle nicht aktiven Verhaltensmuster. Es aktiviert Reaktionen (erste Sorte von Verhaltensmustern),

deren Aktivierungsbedingung erfüllt ist, und Prozeduren (zweite Sorte von Verhaltensmustern), deren Ausführung durch die lokale Planungsschicht vorgesehen ist. Von allen aktivierten Verhaltensmustern kommt eine sich nicht widersprechende Teilmenge zur Ausführung. Auch hier besteht die Auswahl also im Test von Bedingungen aller möglichen Fähigkeiten.

In der lokalen Planungsschicht entstehen Aktionsfolgen durch hierarchisches Planen zur Erfüllung eines Planziels. Pläne und Planziele sind gemeinsam in einer Planbibliothek abgespeichert. Die Auswahl besteht in der Wahl der Planziele (durch Aufträge oder die Kooperationsschicht vorgegeben) und der Wahl der zu den Zielen gehörigen Pläne. Gibt es zu einem Ziel verschiedene Pläne, so entscheidet eine Bewertungsfunktion über die Wahl. Pläne sind in der Planbibliothek so abgespeichert, dass ein effizienter Zugriff mit Angabe des gewünschten Planziels möglich ist.

Die Auswahl in der Kooperationsschicht erfolgt auf Grundlage von Verhandlungen, die in dem entsprechenden Abschnitt näher betrachtet werden. Hier ist nur wichtig, dass der Verhandlungsprozess schließlich zur Bestimmung eines Kooperationsplanes und damit zu Planzielen für die lokale Planungsschicht führt.

Insgesamt betrachtet, kann jede der drei Schichten eine Auswahl treffen. Die Auswahl in der Reaktionsschicht führt sofort zu Aktionen (Reaktionen), die Auswahl von Plänen in der Planungsschicht führt zum Aufruf von Prozeduren in der Reaktionsschicht, und die Auswahl von Kooperationsplänen in der Kooperationsschicht führt zur Abarbeitung von Plänen der lokalen Planungsschicht.

Das Passen betrifft wieder die Abbildung von Umgebungssituationen und Zielen auf Aktionen und Pläne. Der Programmierer muss die Abbildung in den Aktivierungsbedingungen, den benötigten Ressourcen und der Zuordnung von Planzielen zu Plänen und schließlich zu Prozedurfolgen kodieren. Die Rechtzeitigkeit der Auswahl kann der Programmierer in der INTERRAP-Architektur durch die Einordnung von Entscheidungen in die geeignete Schicht beeinflussen. Auswahlentscheidungen für das sofortige Handeln in dynamischen Umgebungen gehören zum Beispiel in die Reaktionsschicht, während die Auswahl zielgerichteter Handlungen in der Planungsschicht mehr Zeit beanspruchen können. Aus den Arbeiten geht allerdings nicht hervor, ob die Laufzeit von Planungs- und Kooperationsprozessen überwacht werden kann. Durch die Einschränkung der tatsächlich ausgeführten Verhaltensmuster auf eine sich nicht widersprechende Teilmenge (Verhaltensmuster konkurrieren dann z. B. nicht mehr um die Verwendung einer Ressource) kann es außerdem passieren, dass eine gewählte Aktion erst verzögert ausgeführt wird. Die Stabilität im Verhalten ist durch verschiedene Mechanismen reguliert. Zwar sind in INTERRAP Absichten nicht explizit repräsentiert (sie sind vielmehr aus dem Abarbeitungsstand der momentan aktiven operationalen Primitive abzulesen). Trotzdem sind die ak-

tuell verfolgten Planziele und Pläne bekannt. Sie werden erst verworfen, wenn eine enthaltene Prozedur in einen Fehler- oder Ausnahmezustand gerät.

Zusammenfassung

Die Aktionsauswahl besteht im Allgemeinen aus mehreren Schritten: der Wahl einer Absicht, der Wahl einer Aktionsfolge, die diese Absicht umsetzt, und schließlich die Auswahl der tatsächlich zu diesem Zeitpunkt auszuführenden Aktionen. Absichten hängen von Zielen und der Umgebungssituation ab. Bei der Wahl der Absicht spielt eher die Bewertung eine Rolle. Ein effizienter Vergleich zwischen Umgebungssituation, Zielen und Fähigkeiten sollte realisiert sein. Der zweite Schritt ist die Planung. Der dritte Schritt muss die Parallelisierbarkeit von Aktionen berücksichtigen.

Für die Verbindung der Umgebungssituation zu Fähigkeiten werden Regelsysteme oder Bewertungsfunktionen benutzt. Sie gewährleisten die erste Anforderung: das Passen. Die Rechtzeitigkeit ist zuerst eine Frage der Laufzeit des Auswahlalgorithmus. Der Agent müsste diese abschätzen können und eventuell verschiedene Algorithmen unterschiedlicher Laufzeit besitzen (z. B. Schichtenarchitektur). Die Effizienz eines Regelauswahlmechanismus hängt insbesondere davon ab, wie schnell die linke Regelseite auf ihre Gültigkeit hin getestet werden kann. Hier ist der Algorithmus auf die Effizienz der Wissensbasis angewiesen. Die Rechtzeitigkeit hängt darüber hinaus von der Umsetzungszeit ab. Wann kann also die gewählte Aktion aufgrund benötigter Ressourcen (z. B. des Prozessors) tatsächlich ausgeführt werden? Ohne die Einführung von Absichten ist die Stabilität im Verhalten nicht zu realisieren. Die Auswahl soll nicht nur von der aktuellen Situation, sondern auch von den gerade verfolgten Absichten abhängen. Diese können dann beibehalten oder verworfen werden. Wooldridge und Parsons ([Wooldridge/Parsons 98]) schlagen hier ein Meta-Level-Control-Modul vor, das über die Beibehaltung von Absichten entscheidet.

3.3.2 Ausprägungsvarianten

In dem in dieser Arbeit hergeleiteten Framework ist die Verbindung zwischen Zielen, Umgebungssituation und Fähigkeiten durch den Begriff »Handlungsvariante« geprägt. Während eine Fähigkeit aus der Sicht der enthaltenen operationalen Beschreibung in der charakterisierenden Beschreibung Anforderungen an die Umgebungssituation formuliert, bei deren Gültigkeit sie anwendbar ist, ist der Ausgangspunkt der Handlungsvariante die Umgebungssituation. Die Handlungsvariante soll den effizienten Zugriff auf Fähigkeiten unterstützen. Fähigkeiten und Handlungsvarianten sind allerdings eng gekoppelt. Handlungsvarianten könnten Produkt einer automatischen Berechnung aus Fähigkeiten (Entscheidungsbaum) oder Ergebnis der Bewertung der Anwendung von Fähigkeiten (Fallbasis) sein. Ein anderer Ansatzpunkt für die Formulierung von Handlungsvarianten ist die Berechnung einer Nützlichkeit der zugehörige Fähigkeit

in Abhängigkeit von der aktuellen Situation durch eine Domänen-abhängige Berechnungsvorschrift (Aufgabe des Programmierers) (siehe Abbildung 3.10).

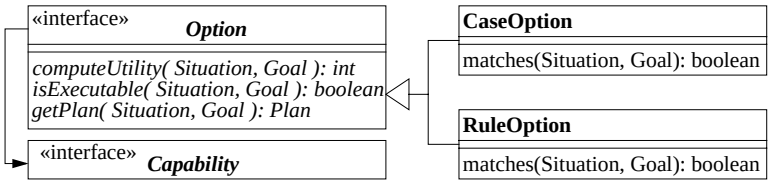

Abbildung 3.10: Diagramm für eine Handlungsvariante

Wie im vorletzten Abschnitt besprochen, sind Fähigkeiten und Handlungsvarianten in der Expertise-Wissensbasis enthalten. Hier geht es zunächst um die Auswahl und Repräsentation der passenden Handlungsvariante: der Absicht.

Auswahl von Absichten

Ausgangspunkte für die Auswahl einer Absicht sind die Umgebungssituation, Ziele und momentan verfolgte Absichten. Zunächst ist zu testen, ob es sinnvoll ist, die momentanen Absichten weiterzuverfolgen. Sie könnten erstens bereits vollständig umgesetzt sein, zugehörige Aktivitäten könnten fehlerhaft unterbrochen worden sein, oder sie könnten nicht mehr zur Umgebungssituation oder zu den Zielen passen.

Eine Absicht entsteht durch die Auswahl einer passenden Handlungsvariante und steht dann für den zur Handlungsvariante bzw. zur Fähigkeit gehörigen und abzuarbeitenden Plan. Der abzuarbeitende Plan ist manifestiert in bereits bearbeiteten Aktionen, momentan ausgeführten Aktionen (Aktivitäten) und noch auszuführenden Aktionen (Verpflichtungen, wenn diese bereits in den Bearbeitungszeitplan aufgenommen sind). Eine Absicht ist demnach vollständig umgesetzt, wenn die Aktionen des abzuarbeitenden Planes vollständig ausgeführt sind. Eine Absicht passt nicht mehr zur Umgebungssituation, wenn die Voraussetzungen der benutzten Fähigkeit nicht mehr erfüllt sind. Eine Absicht passt nicht mehr zu den Zielen, wenn das zu ihr gehörende Ziel zurückgezogen wurde (z. B. durch Rücknahme eines Auftrags). In allen Fällen des Zurückziehens einer Absicht sind die entsprechenden Aktivitäten abzubrechen und die schon zeitlich eingeplanten Verpflichtungen zu streichen.

Nach der Entfernung nicht mehr umsetzbarer Absichten bleibt eine Teilmenge von bisherigen Absichten übrig, gegen deren Weiterverfolgung zunächst nichts spricht. Sie sind allerdings mit neu entstehenden Absichtskandidaten zu vergleichen. Die Kandidaten lassen sich berechnen, indem für jedes Ziel die der Umgebungssituation entsprechenden (am besten passenden) Handlungsvarianten ausgesucht werden. Das ist ein Suchprozess in der Expertise-Wissensbasis: das Durchlaufen eines Entscheidungsbaums, das »Matchen« einer Regel, das

Retrieval in einer Fallbasis, die Berechnung von Nützlichkeiten oder eine Aktivierung in einem neuronalen Netz. Für diesen Suchprozess soll es nicht nötig sein, die tatsächlichen Pläne von Fähigkeiten auszurechnen, er soll sich vielmehr auf direkt zugreifbare oder schnell berechenbare Attribute stützen (wie die Nützlichkeit).

Es folgt ein Abwägen zwischen neu bestimmten Absichtskandidaten und bisherigen Absichten, die zu ein und demselben Ziel gehören. Dieses Abwägen kann nur durch eine Bewertungsfunktion geschehen. Einfache Strategien sind Funktionen, die immer die bisherige Absicht oder immer die neuen Absichtskandidaten bevorzugen. Im letzten Schritt gilt es, die Menge der tatsächlichen Absichten zu bestimmen. Auch hierfür gibt es verschiedene Strategien: zum Beispiel die Bestimmung einer konsistenten Teilmenge mit höchster summarischer Bewertung oder die Wahl nur einer Absicht mit höchster Bewertung.

Was heißt es, dass Absichten untereinander und in Bezug auf die Umgebungssituation konsistent sind? Für den zweiten Punkt sorgt bereits der Auswahlprozess, der nur solche Absichtskandidaten bestimmt, deren charakterisierende Beschreibung (durch die zugehörige Fähigkeit festgelegt) zur Umgebungssituation passt, deren Voraussetzungen also erfüllt sind. Auch die Konsistenz untereinander kann nur in Bezug auf die charakterisierende Beschreibung der benutzten Fähigkeiten bestimmt sein: die Nachbedingungen der Fähigkeiten müssen konsistent sein. Die Konsistenz von Bedingungen ist durch die benutzte Umgebungswissensbasis festgelegt. Zwei Fähigkeiten können aber konsistent sein, auch wenn sie dieselben Ressourcen benötigen. Dieser Konflikt wird weiter unten besprochen.

Eine Absicht ist ein Paar aus einer zur Ausführung ausgewählten Handlungsvariante und des durch die zugehörigen Fähigkeiten gemäß der aktuellen Umgebungssituation berechneten Plans. Handlungsvarianten sind dabei in dieser Architektur nur als Bindeglied zwischen Situationen und Fähigkeiten gedacht, es ist genauso möglich, auf sie zu verzichten und Fähigkeiten direkt mit Situationen zu vergleichen.

Auswahl von Plänen

Sind nach dem ersten Auswahlschritt die Absichten festgelegt, so müssen für neue Absichten die zugehörigen Pläne bestimmt werden. Das geschieht durch Abfrage bei den entsprechenden Fähigkeiten. Wie im letzten Abschnitt beschrieben, kann sich hinter dieser Abfrage eine hierarchische oder zustandsbasierte Planung befinden.

In gewissen Fällen ist es sinnvoll, die Absicht beizubehalten, jedoch den realisierenden Plan zu ändern: wenn eine Aktivität fehlerhaft abgebrochen wurde oder wenn der bisherige Plan wegen der Änderung der Umgebungssituation die Absicht nicht mehr umsetzen wird. Ein neuer Plan muss auch hier durch die

entsprechende Fähigkeit berechnet werden. In sehr dynamischen Umgebungen (wie bei der Fußballanwendung) ist es zweckmäßig, jeweils nur ein Stück eines Planes zu berechnen; die Absicht bleibt bestehen, der Plan wird jeweils aktuell festgelegt.

Auswahl von Aktionen

Nach der Auswahl der Pläne müssen die entsprechenden Aktionen zeitlich angeordnet werden. Der Plan gliedert sich in einen Zeitplan ein. Die Anordnung der Aktionen ist einerseits durch den Plan selbst bestimmt (Reihenfolge, evtl. zeitliche Beziehungen), andererseits von der Verfügbarkeit benötigter Ressourcen abhängig. Ein Schedulingalgorithmus muss diese beiden Vorgaben berücksichtigen. Der Algorithmus hat die Möglichkeit, bereits mit Terminen versehene Aktionen neu anzuordnen.

Die tatsächliche Ausführung obliegt dann dem Ausführungsmodul. Das Steuerungsmodul startet dazu für jede mit dem aktuellen Termin versehene Verpflichtung eine Aktivität. Dieser Vorgang ist bereits ein Thema des nächsten Abschnitts: der allgemeinen Ablaufsteuerung im Agenten.

Insgesamt ergibt sich folgendes Bild für die Aktionsauswahl (siehe Abbildung 3.11).

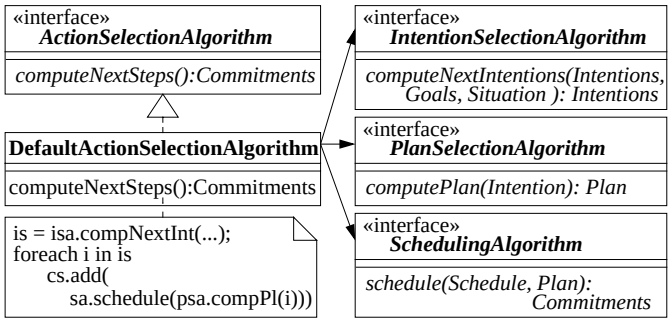

Abbildung 3.11: Diagramm für die Aktionsauswahl

Rechtzeitigkeit vs. Reaktion

Viele Arbeiten auf dem Gebiet der Agenten unterscheiden reaktive und deliberative Systeme. Der Unterschied besteht hauptsächlich in der Frage, ob die Aktionsauswahl auf Grundlage interner Repräsentation von Umgebung und Aktionen oder auf Grundlage fest eingestellter Situations-Aktions-Regeln geschieht. Was jedoch die Reaktionszeit betrifft, so ist diese nicht unmittelbar von diesem Unterschied abhängig. Eine durch einen Inferenzprozess ausgewählte Aktion kann durchaus wie eine Reaktion wirken, wenn sie schnell genug ausgeführt wird. Es kommt in dem Fall also eher darauf an, die Rechtzeitigkeit von

Aktionen zu betrachten. Ist die Rechtzeitigkeit wichtig, so müssen Auswahlmechanismen verwendet werden, die dieser Anforderung genügen. Reaktionsregeln sind ein Mittel dafür. Liegen diese aber in einem Regelsystem vor, so kann die Auswahl der richtigen Regel ebenfalls zu lange dauern.

Entsprechend der vorgenommenen Dreiteilung muss die Rechtzeitigkeit bei der Wahl der Absicht, der Wahl des Planes und bei der Zeitplanung der Aktionen berücksichtigt werden. Hat der Agent Informationen zum Laufzeitverhalten der Algorithmen, so kann er selbst auf die Rechtzeitigkeit Einfluss nehmen. Für die genaue Ausarbeitung dieser Anforderung müssten Verfahren aus Echtzeitsystemen studiert und zur Verfügung gestellt werden. Es bietet sich hier auch ein Any-Time-Algorithmus an, ein Algorithmus, der zu jedem Zeitpunkt ein Ergebnis liefert, dessen Qualität sich jedoch mit steigender Rechenzeit verbessern kann. Für den speziellen Fall der Aktionsauswahl könnte ein solcher Algorithmus mit einer Reaktionsregel starten und dann weitere länger dauernde Inferenzmechanismen starten. Es ist dann aber die zusätzliche Entscheidung zu treffen, wann der Auswahlprozess abzubrechen ist. In manchen Systemen (wie bei der Fußball-Anwendung) ist die Entscheidung einfach: der Algorithmus ist abzubrechen, wenn die nächsten Aktionen auszuführen sind. Das ist in diesem Fall durch eine Frequenz vorgegeben.

Effizienz vs. Flexibilität

Ein Thema dieses Kapitels ist das Abwägen zwischen Effizienz und Flexibilität. Bei der Aktionsauswahl ist das Konzept der Handlungsvarianten bereits ein Mittel zur Effizienzsteigerung, indem es den Suchraum für mögliche Aktionen einschränkt bzw. die hier nötige Richtung von der Umgebungssituation zu den Fähigkeiten explizit macht. Stehen dem Agenten mehrere Auswahlmechanismen oder ein Any-Time-Algorithmus zur Verfügung, so kann er die Güte der und den Zeitbedarf der Auswahl entsprechend den obigen Bemerkungen steuern.

3.3.3 Zusammenfassung

Am Anfang dieses Abschnitts wurden drei Anforderungen an einen Aktions-Auswahlmechanismus gestellt: das Passen, die Rechtzeitigkeit der Entscheidung und die Stabilität des Verhaltens. Für alle drei Anforderungen wurden Lösungsansätze in veröffentlichten Arbeiten identifiziert und für das Framework aufbereitet: Handlungsvarianten gewährleisten das Passen und sind ebenfalls ein Mittel für Effizienzsteigerungen. Absichten sind ein Mittel, das Verhalten zu stabilisieren. Für die Rechtzeitigkeit von Entscheidungen sind Angaben über die Laufzeit von Auswahlalgorithmen nötig. Ein Schritt in diese Richtung besteht bereits darin, dass ein Teil der Aktionsauswahl, die Planung, im letzten Abschnitt als Fähigkeit formuliert wurde. Zur charakterisierenden Beschreibung einer Fähigkeit gehört die Angabe ihres Zeitbedarfs.

Bezüglich der Begriffe »Ziel« (mit den Attributen »potentiell«, »erfüllbar«, »effektiv«), »Wunsch« (Desire), »Option«, »Absicht« und »Verpflichtung« findet man in der Literatur einen sehr uneinheitlichen Gebrauch. Teilweise werden die einen mit den anderen erklärt. Alle betreffen jedoch letztlich die Aktionsauswahl und sollen hier bezüglich des in dieser Arbeit abgeleiteten Frameworks kurz zusammenfassend erläutert werden.

Nach Rao und Georgeff ([Rao/Georgeff 98]) ergeben Annahmen eine Struktur möglicher Welten (im Sinne der Kripke-Semantik). Die *Ziele*$_{\text{BDI}}$ eines Agenten verweisen auf eine Teilmenge dieser Welten. *Ziele*$_{\text{BDI}}$ sollen untereinander konsistent sein und bilden nach den Worten der Autoren eine Auswahl aus den möglicherweise untereinander inkonsistenten *Wünschen*$_{\text{BDI}}$. *Absichten*$_{\text{BDI}}$ repräsentieren wiederum eine Auswahl der Zielwelten, die zu erreichen sich der Agent *verpflichtet* hat. In der operationalen Umsetzung (dMars) steht eine *Absicht*$_{\text{BDI}}$ für einen partiellen Plan, zu dessen Ausführung sich der Agent *verpflichtet* hat (den er also gerade ausführt, oder dessen Schritte bereits im Zeitplan eingereiht sind) und von dem er annimmt, dass er zu der Zielwelt führen wird. Zusätzlich sind hier also *Ziele*$_{\text{BDI}}$ mit Plänen ihrer Umsetzung verknüpft und die Wahl von *Absichten*$_{\text{BDI}}$ bedeutet die Festlegung auf konkrete Pläne. In diesem Sinn kann man die in INTERRAP benutzten Begriffe ([Müller 96]) wie folgt zuordnen: *Potentielle Ziele* entsprechen den *Wünschen*$_{\text{BDI}}$ oder den in der Planbibliothek vorgesehenen Zielen, *erfüllbare Ziele* den *Zielen*$_{\text{BDI}}$ und *effektive Ziele* den *Absichten*$_{\text{BDI}}$. Die in Agent0 benutzten *Verpflichtungen* korrespondieren mit den in einen Zeitplan eingereihten Schritten und gehören so zu einer dort allerdings nicht explizit repräsentierten *Absicht*$_{\text{BDI}}$.

Die in dieser Arbeit benutzten Begriffe passen in dieses Schema wie folgt: Die *Handlungsvarianten* (Optionen) beschreiben die aus der momentanen Situation erreichbaren Welten mit einer Bewertung. Diese Bewertung ist abhängig von den *Zielen* des Agenten. Definiert man eine Bewertungsschwelle, dann ist also auch hier mit den über der Schwelle liegenden Welten eine Teilmenge von Zielwelten definiert. Die Optionen sind mit einer Fähigkeit und so mit einem Plan zum Erreichen der durch sie repräsentierten Welt verbunden. Eine *Absicht* entsteht durch Auswahl einer Option und steht wie die in dMars operationalisierte *Absicht*$_{\text{BDI}}$ für den letztlich gewählten Plan mit seinen momentan bearbeiteten Schritten (Aktivität) und den zukünftigen Schritten (*Verpflichtung*).

3.4 Ablaufsteuerung und Parallelisierung

Die Verbindung der bisher betrachteten einzelnen Strukturen und Abläufe ist durch die Ablaufsteuerung innerhalb des Agenten zu leisten.

3.4.1 Aufgaben und Quellen

Es geht bei der Ablaufsteuerung wieder um die Kriterien des Passens und der Rechtzeitigkeit. Alle Mechanismen, die im Agenten wirken, müssen koordiniert und aufeinander abgestimmt werden.

Wie sollen zum Beispiel die zur Ausführung bestimmten Aktionen tatsächlich bearbeitet werden: parallel oder sequentiell? Soll die Aktionsauswahl die Aktionsausführung unterbrechen? Wann soll der Agent über seine Interaktionsmodule mit der Umgebung kommunizieren?

Anhaltspunkte für diese Fragen finden sich vor allem in den praktischen Umsetzungen von Agentensystemen. Dort muss der verwendete Mechanismus genau angegeben sein, sonst kann man das Verhalten eines Agenten nicht vernünftig steuern.

Ablaufsteuerung bei ACTORS

Eine der ersten Arbeiten zur Ablaufsteuerung in aktiven Objekten ist Aghas ACTOR-Modell ([Agha 90]). Ein Actor wartet an einer Nachrichtenschlange, bis eine Nachricht eintrifft. Er kann auf die Nachricht reagieren, indem er neue Actor-Objekte erzeugt, Nachrichten an andere Actor-Objekte sendet oder sein Verhalten (Menge der Reaktionsregeln) ändert. Die Ablaufsteuerung entspricht einer Sequenz aus Nachrichtenempfang und Nachrichtenbearbeitung. Sobald ein Actor eine Nachricht empfängt, wird ein »Folgeactor« mit einem eventuell geänderten Verhalten erzeugt, der für die nächste Nachricht in der Nachrichtenschlange verantwortlich ist. Damit ist die Bearbeitung der Nachrichtenschlange parallelisiert. Die Struktur eines Actors ist jedoch nicht weiter gegliedert. Rechtzeitigkeit und Passen spielen in dem Modell keine Rolle.

Ablaufsteuerung in Agent0

Ein Agent0-Agent ([Shoham 93]) arbeitet in einem Zyklus, der aus folgenden Einzelschritten besteht: dem Lesen aller in diesem Takt eingetroffenen Nachrichten, der Anwendung einer Reaktionsregel (Commitment-Regel) je Nachricht (dabei kann der Agent z. B. Verpflichtungen eingehen) und der sequentiellen Abarbeitung aller Verpflichtungen, die zum aktuellen Zeitpunkt auszuführen sind, oder die vor dem aktuellen Zeitpunkt auszuführen waren, es aber noch nicht sind. Das ist ein sehr einfaches Schema, das aber in keiner Weise dem Kriterium der Rechtzeitigkeit gerecht werden kann. Das Schema ist nicht geeignet, wenn die Nachrichtenzahl sehr hoch ist: dann ist die Zeit zum Lesen der Nachrichten und zum Anwenden der Regeln sehr hoch, in dieser Zeit kann der Agent aber keine Verpflichtungen bearbeiten. Deren Ausführungszeit verstreicht. Auch der Zeitbedarf einer Verpflichtung kann sehr hoch sein, dann ist der Agent nicht in der Lage, Nachrichten zu empfangen. Nachrichten und Verpflichtungen sind nicht priorisiert, die Reihenfolge ihrer Bearbeitung ist aber essentiell für das Verhalten des Agenten.

Ablaufsteuerung in BDI-Systemen

BDI-Systeme (wie z. B. dMARS: www.aaii.oz.au/proj/dmars_tech_overview/dMARS-1.html oder [Rao/Georgeff 91, Rao/Georgeff 98, d'Inverno et al. 98]) laufen ebenfalls in einem Interpreterzyklus ab. Dieser besteht aus der Aktualisierung der Annahmen, der Bestimmung der momentan anwendbaren Pläne (abhängig von den Annahmen), der Wahl eines dieser Pläne als zu verfolgende Absicht, der Instantiierung dieses Plans (Ausformulierung) und seiner Abarbeitung. Dieser Zyklus ist sequentiell, eine Parallelisierung wird im Allgemeinen nicht betrachtet. Für diesen Zyklus gelten ähnliche Anmerkungen wie für Agent0: Die Rechtzeitigkeit spielt hier keine Rolle, wichtig ist das Passen.

Ablaufsteuerung in INTERRAP

Einen feiner strukturierten Ablaufmechanismus besitzt INTERRAP ([Müller 96]). Jede der drei Schichten (Reaktionsschicht, lokale Planungsschicht, Kooperationsschicht) besitzt nämlich eine eigene Ablaufsteuerung, einen eigenen Ablauffaden (Thread), der von einem gemeinsamen Grundmuster abgeleitet ist. Damit ist zwar ein Synchronisationsaufwand notwendig, die Steuerung lässt aber eine bessere Anpassung an verschiedene Umgebungen zu.

Das Grundmuster ist in einem gemeinsamen Steuerzyklus beschrieben. Er besteht aus der Aufnahme neuer Umgebungsdaten (Aktualisierung der Annahmen); dem Erkennen bestimmter Situationen (Nachrichten der darunterliegenden Schicht werden einbezogen); der Zielauswahl; der Entscheidung, ausgewählte Ziele selbst zu bearbeiten oder an die übergeordnete Schicht weiterzuleiten; der Planauswahl für das Ziel; der Zeitplanung und schließlich der Ausführung.

In der Reaktionsschicht läuft dementsprechend ein Zyklus, der nach der Aktualisierung der Annahmen zunächst die positiven und negativen Abbruchkriterien aktiver Verhaltensmuster testet und gegebenenfalls deaktiviert. Im zweiten Schritt erfolgt die Aktionsauswahl entsprechend der Beschreibung im letzten Abschnitt. Diese Auswahl führt zur Aktivierung einer Menge von Verhaltensmustern, aus denen die Ablaufsteuerung im dritten Schritt eine Teilmenge kompatibler Verhaltensmuster tatsächlich ausführt. Ausführen bedeutet hier, dass alle gewählten Verhaltensmuster einen Abarbeitungsschritt vollziehen können. Mit dieser schrittweisen Abarbeitung führt J. P. Müller ein Mittel ein, die Rechtzeitigkeit von Aktionen anzugehen. Ein Verhaltensmuster mit hohem Zeitbedarf soll den weiteren Ablauf im Agenten nicht zu lange unterbrechen. Stattdessen führt die Ablaufsteuerung die Verhaltensmuster in Portionen mit geringerem Zeitbedarf aus. Das ist zumindest das Ziel dieses Mechanismus, der Programmierer wird jedoch nicht gezwungen, die Verhaltensmuster tatsächlich in kleine Portionen einzuteilen. Es liegt also in der Hand des Programmierers, Verhaltensmuster passend zu strukturieren. Es könnte ja andererseits notwendig sein, dass ein Verhaltensmuster nicht geteilt wird.

Die lokale Planungsschicht nimmt neue Ziele von der Kooperationsschicht und Informationen zu beendeten oder fehlgeschlagenen Verhaltensmustern von der Reaktionsschicht entgegen, plant und stößt Verhaltensmuster in der Reaktionsschicht an oder meldet Kooperationsbedarf an die Kooperationsschicht. Die Kooperationsschicht nimmt Kooperationsanforderungen von der eigenen lokalen Planungsschicht oder über ein Kommunikationsmodul von anderen Agenten entgegen, steuert den Verhandlungsprozess und übergibt schließlich Kooperationsziele an die lokale Planungsschicht.

In diesem dreigeteilten Ablaufschema sind zwei Mittel umgesetzt, die Rechtzeitigkeit von Aktionen zu unterstützen: zum einen die Aufteilung der Aktionsausführung in Teilschritte und zum anderen die drei eingesetzten unterschiedlich schnellen Auswahlverfahren. Die Synchronisation zwischen den drei Ablaufsteuerungen erfolgt durch das Senden von Nachrichten. Man könnte demnach jede der drei Schichten als eine BDI-Schicht auffassen.

Ausführen oder Auswählen

Wooldridge und Parsons ([Wooldridge/Parsons 98]) schlagen ein Meta-Level-Control-Modul vor. Dieses entscheidet, wann der Agent Aktivitäten ausführen und wann er Absichten überprüfen soll. Mit diesem Modul abstrahieren sie von den starren Recognize-Act-Zyklen, wie sie z. B. in Agent0 oder dMARS umgesetzt sind.

Parallelität in der Ablaufsteuerung

In INTERRAP ist die Ablaufsteuerung bereits in drei parallele Stränge unterteilt. In [Bussmann 98] schlägt S. Bussmann vor, die Ablaufsteuerung überhaupt zu parallelisieren. Er stellt zunächst fest, dass im Bereich seiner Anwendungsdomäne, der Steuerung von Produktionsanlagen, häufig die parallele Ausführung von Aktionen notwendig ist. Berücksichtigt man diese Anforderung in der Ablaufsteuerung, so müssen die gleichzeitig ausgeführten Aktionen kompatibel bzw. konfliktfrei sein (wie auch schon in INTERRAP angemerkt). Konfliktfrei bedeutet einerseits, dass sie nicht exklusive Ressourcen gleichzeitig benutzen dürfen (Mutual Exclusion), und andererseits, dass sie nicht zeitlich voneinander abhängen (die zweite Aktion ist erst nach der ersten möglich).

Führt man aber einmal ein Schema zur Parallelisierung von Aktionen ein, so kann man dieses auch auf die internen Abläufe im Agenten ausdehnen. Zu einem Zeitpunkt ist dann eine Menge von Aktivitäten (Aktionsausführungen, Planung neuer Aktionen, Kommunikation etc.: Tasks) parallel zu bearbeiten. Der Ablauf innerhalb einer solchen Aktivität ist durch ein Skript (von S. Bussmann so genannt, in dieser Arbeit würde der Begriff »Plan« passen) bestimmt. Da auf einer Einprozessormaschine sich trotzdem immer nur eine Aktivität in der Abarbeitung befinden kann, muss es einen Mechanismus geben, der die

Ressource Prozessor den Aktivitäten zuweist. Das ist ebenfalls eine Schedulingaufgabe. Ein im Bereich dieser Schedulingaufgabe übliches Verfahren (z. B. in Betriebssystemen) ist die Priorisierung von Aktivitäten und die Betrachtung von Zuständen der Aktivitäten (bereit, blockiert). Diese Idee führt hier dazu, dass allen Skripten Prioritäten zugewiesen werden müssten. Das wäre ein sehr starres Konzept, das vielleicht besagen könnte, dass eine Aktionsauswahl immer Vorrang vor einer Aktionsausführung hat.

Stattdessen schlägt S. Bussmann vor, die Ereignisse (dort so genannt, hier: eintreffende Umgebungsinformationen und interne Auslösebedingungen) zu priorisieren. Dabei besitzen äußere Ereignisse eine feste Priorität, die Priorität interner Ereignisse wird durch das auslösende Skript berechnet. Die Priorität des Ereignisses überträgt sich dann auf das ausgelöste Skript. Zeitkritische äußere Ereignisse (z. B. ein Auftrag oder die eintreffende Umgebungssituation bei der Fußballanwendung) erhalten die höchste Priorität. Die notwendigen Skripts werden dementsprechend bevorzugt ausgeführt. Eine zweite Klasse von Ereignissen sind die normal priorisierten (die z. B. Steuerungsaktivitäten auslösen). Optionale Ereignisse (z. B. für die Beobachtung und Diagnose) besitzen die geringste Priorität.

Die Prozessorzuweisung erfolgt nun gestaffelt nach den Prioritäten. Zwischen den Modulen (z. B. Kommunikation, Sensorik, Aktionsauswahl, Aktionsausführung) wird die Prozessorzeit nach einem Zeitscheibenverfahren aufgeteilt. Damit ist ein der Anforderung der Rechtzeitigkeit entgegenkommendes Schema entworfen. Allerdings reduziert S. Bussmann die Aktionsauswahl auf die Wahl eines Skripts zu einem Ereignis. Er argumentiert, dass man komplexere Mechanismen (wie BDI) für die Produktionssteuerung nicht benötigt. Ein Agent soll hier nicht über eigene Absichten, Fähigkeiten etc. räsonieren, sondern eine Maschine steuern und deren Verhalten überwachen. Gerade die Rechtzeitigkeit von Aktionen steht hier im Vordergrund.

Aber wie schon weiter oben bemerkt, schließen sich die Benutzung von komplexeren Auswahlmechanismen und die Rechtzeitigkeit nicht aus. Das Konzept der Absicht könnte durchaus in einer Maschinensteuerung eine Rolle spielen, um etwa ein stabiles Verhalten aufrechtzuerhalten.

Zusammenfassung

Bei der Ablaufsteuerung geht es, wie gesagt, um die Integration aller im Agenten notwendigen Abläufe. Dazu gehören die Aufnahme von Umgebungsinformationen über die Interaktionsmodule, die Aktualisierung der Annahmen über die Umgebung, die Aktionsauswahl und schließlich die Ausführung. In den untersuchten Systemen lassen sich zwei allgemeine Lösungsmuster für dieses Problem finden:

- ein sich während der Laufzeit des Agenten permanent wiederholender Zyklus, der alle oben genannten Teilabläufe nacheinander bearbeitet und

- ein Scheduler, der einer Menge paralleler Aktivitäten (aus allen Teilabläufen) anhand von Prioritäten und anderen Mechanismen den Prozessor zuweist.

Die erste Variante ist einfach umzusetzen, hat jedoch den Nachteil, dass sich Teilabläufe gegenseitig blockieren können; dauert eine Aktionsausführung lange, wird der Agent in dieser Zeit nicht auf neue Nachrichten reagieren. Mittel zur Abschwächung dieses Problems unter Beibehaltung des Grundprinzips sind die Einführung von Schichten und die Unterteilung der Aktionsausführung in kurze Einzelschritte wie in INTERRAP oder die Einschaltung eines Meta-Level-Control-Moduls wie in [Wooldridge/Parsons 98].

Mit der zweiten Variante kann man bei entsprechender Wahl des Schedulingalgorithmus die erste nachbilden. Der Sinn ist es aber, ein flexibleres Ablaufschema zu realisieren. Dringende Nachrichten sollen unwichtigere Aktionen unterbrechen können. Solche Anforderungen lassen sich mit der zweiten Variante umsetzen. Allerdings ist ein zusätzlicher Synchronisationsaufwand erforderlich und auch dieses Scheduling kostet Zeit. Es ist außerdem zu untersuchen, ob das Prinzip umsetzbar ist, wenn die Aktionsauswahl nicht nur aus einer Skriptaktivierung durch Ereignisse besteht, sondern aus mehreren Schritten, wie im letzten Abschnitt beschrieben. Ein weiterer Nachteil könnte darin bestehen, dass der Agent keinen Einblick in die Interna des Schedulingalgorithmus hat. Insbesondere ist der momentane Abarbeitungszustand der Aktivitäten nicht explizit repräsentiert.

3.4.2 Ausprägungsvarianten

Für das in dieser Arbeit hergeleitete Framework geht es bei der Ablaufsteuerung vor allem um das Steuerungsmodul mit seiner Parallelisierungsstrategie und die Wechselwirkung der verschiedenen Module. Der Begriff »Aktivität« wurde bereits herausgearbeitet. Er wird jetzt im Sinne von [Bussmann 98] breiter gefasst und soll auch Abläufe in anderen Modulen umfassen, nicht nur die Ausführung von Aktionen. Die Ablaufsteuerung soll transparent für Auswahlprozesse innerhalb des Agenten und auf die Anforderungen der Anwendung einstellbar sein.

Aktivitäten

Als potentielle Aktivitäten kommen in Frage: die Interaktion, die Aktualisierung der Annahmen, die Aktionsauswahl und die Aktionsausführung. Interaktionsaktivitäten können durch eintreffende Nachrichten hervorgerufen werden. Ein Agent kann mehrere Interaktionsmodule besitzen. Annahmen sind zu aktualisieren, wenn Nachrichten eingetroffen sind oder wenn Aktionen ausgeführt worden sind. Eine Aktionsauswahl ist nötig, wenn neue Annahmen dafür sprechen. Ausgewählte Aktionen sind schließlich auszuführen. Einige

Aktivitäten sind offensichtlich nicht unabhängig voneinander. Eine Interaktionsaktivität kann eine Aktualisierung, diese eine Auswahl und diese wiederum eine Ausführung auslösen.

Andererseits könnten über verschiedene Interaktionsmodule gleichzeitig Nachrichten eintreffen. Die dadurch ausgelösten Aktivitäten wären zunächst unabhängig voneinander. Die Aktualisierung der Annahmen muss nun aber synchronisiert sein, um einen konkurrierenden Zugriff zu verhindern.

Auslöser von Aktivitäten sind einerseits äußere Ereignisse (eingetroffene Nachrichten oder ein Zeitimpuls) und andererseits vorher ausgeführte Aktivitäten (im Prinzip innere Ereignisse). Es ist also ein Mechanismus nötig, äußere Ereignisse in Aktivitäten umzusetzen. Das lässt sich realisieren, indem jede Ereignisquelle einen Auslöse-Thread kennt (ein eigener oder ein gemeinsamer). Das ist ein Thread, der in einer Endlosschleife arbeitet und immer bis zum nächsten Ereignis blockiert, um dann für das Ereignis eine Aktivität zu starten. Für das Auslösen interner Aktivitäten gibt es zwei Varianten, sie könnten direkt durch andere Aktivitäten oder als Folge interner Nachrichten ebenfalls durch einen Auslöse-Thread gestartet werden.

Im Hinblick auf die Transparenz für Auswahlprozesse ist hier die zweite Variante vorzuziehen. Dadurch gibt es fest definierte Punkte, an denen Aktivitäten entstehen können. Es bietet sich an, für jedes Modul des Agenten einen Auslöse-Thread vorzusehen. Diese müssen die entsprechenden Aktivitäten auslösen. Sie sind die meiste Zeit blockiert und verbrauchen keine lange Rechenzeit.

Es stellt sich die Frage, wie Aktivitäten im Agenten repräsentiert sind. Ein wichtiges Kriterium ist, wie gesagt, die Transparenz für Auswahlprozesse. Die Merkmale, anhand derer die Auswahl stattfinden kann, sind die Priorität, Zeitbedingungen und der Initiator der Aktivität. Für die Ausführung sind Methoden zur Steuerung des Ablaufs notwendig (siehe Abbildung 3.12).

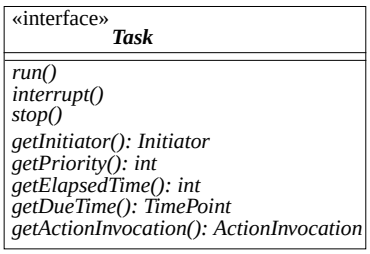

Abbildung 3.12: Diagramm für eine Aktivität

Ausführung von Aktionen

Die Ausführung einer Aktion hat jetzt keinen besonderen Stellenwert mehr gegenüber anderen internen Abläufen. Nur muss das Ende einer Aktionsausführung wieder zu anderen Aktivitäten, wie der Aktualisierung der Wissensbasis

führen. Im Initiator der Aktivität muss für eine Aktionsausführung die dahinter stehende Absicht kodiert sein.

Nun gibt es allerdings schon zwei Schedulingprozesse zur Auswahl der tatsächlich ausgeführten Aktionen: Im letzten Abschnitt wurde als dritter Schritt der Aktionsauswahl die zeitliche Anordnung der Aktionen genannt, hier nun soll der Scheduler aus den zur Ausführung bestimmten Aktionen die aussuchen, die auf verfügbare Threads aufgeteilt werden. Sind das wirklich zwei Schedulingprozesse? Es gibt zwar Anforderungen von zwei Seiten: die Anzahl der verfügbaren Threads auf der einen, die Abhängigkeit zwischen Aktionen (z. B. Benutzung gleicher Ressourcen) auf der anderen, die voneinander unabhängige Betrachtung ist jedoch nicht sinnvoll. Eine Aufteilung für die eine Seite muss nicht kompatibel zu einer Aufteilung für die andere Seite sein. Hätte zum Beispiel der erste Schedulingprozess eine Menge von Aktionen für den aktuellen Zeittakt eingeplant und eine weitere Menge von Aktionen zu einem späteren (die erste Menge muss dann schon beendet sein), um Konflikte zwischen Aktionen beider Mengen aufzulösen, dann könnte nun die geringe Anzahl von verfügbaren Threads dazu führen, dass die erste Menge zum späteren Zeitpunkt nicht beendet ist. Wird die zweite Menge nun trotzdem aktiviert, so muss auch der zweite Schedulingprozess Konflikte zwischen Aktionen berücksichtigen, oder er müsste den ersten Prozess dazu veranlassen, die Ausführungszeiten neu aufzuteilen.

Diese Argumentation führt zu der Anforderung, beide Aspekte des Scheduling in einem Algorithmus zu beachten:

- Konflikte zwischen Aktionen (Benutzung exklusiver Ressourcen oder zeitliche Abhängigkeit) und
- die Anzahl der verfügbaren Threads.

Diese Anforderung kann man auf alle Aktivitäten ausdehnen. Auch zwischen anderen Aktivitäten bestehen zeitliche Abhängigkeiten oder Ressourcenkonflikte.

Auswahl der Aktivitäten

Der Schedulingalgorithmus muss die Aktivitäten den Ressourcen zuordnen und die Ausführung der Aktivitäten überwachen, sie eventuell unterbrechen oder beenden.

Die momentan gestarteten Aktivitäten und die im Agenten laufenden Threads (dem Betriebssystem-Mittel, um nebenläufige Abläufe zu realisieren), welche die Aktivitäten wirklich ausführen (beherbergen), korrespondieren nicht eins zu eins. Es kann Aktivitäten geben, die gerade keinem Thread zugeordnet sind, und Threads, die gerade keine Aktivität tragen. Diese zusätzliche Ebene erlaubt es, unabhängig von der darüberliegenden Ablaufstruktur eine auf das benutzte

System zugeschnittene Parallelisierungsstrategie anzuwenden. Es könnte zum Beispiel sinnvoll sein, die Anzahl der gleichzeitig aktiven Threads zu begrenzen. Entsprechend den Erläuterungen im Abschnitt zum Assistenz-Programm kommen für den Einsatz von Threads verschiedene Varianten in Frage:

- die Benutzung nur eines Threads,
- die Benutzung eines Threads pro Aktivität,
- die Benutzung eines Thread-Pools oder
- die Benutzung eines Threads (Thread-Pools) pro Schlüssel.

Die zweite Strategie ist im Allgemeinen nicht praktikabel. Sie berücksichtigt eventuelle Ressourcenbeschränkungen nicht. Der Auswahlmechanismus hätte allerdings nur die Aufgabe, Konflikte aufzulösen, Aktivitäten zu überwachen und eventuell zu beenden. Mit der ersten Strategie lässt sich bei geeigneter Zuordnung ein BDI-Zyklus realisieren. Die dritte Strategie ist ein Kompromiss aus den ersten beiden: Solange es weniger Aktivitäten als Threads im Threadpool gibt, ist keine Zuordnung von mehreren Aktivitäten auf einen Thread nötig; alle Aktivitäten können quasi gleichzeitig ablaufen (analog zur zweiten Strategie). Anderenfalls müssen einige Aktivitäten warten, bis wieder ein Thread frei ist (analog zur ersten Strategie). Der Haupteffekt dieser Strategie ist die Möglichkeit nebenläufiger Arbeit in beschränktem Ausmaß. Die vierte Strategie kann Aktivitäten Kategorien (durch Schlüssel identifiziert) zuordnen und so Threads gleichmäßig aufteilen.

Ist die Strategie zum Einsatz von Threads gewählt, so muss der Schedulingalgorithmus die gestarteten Aktivitäten den verfügbaren Threads zuordnen bzw. wieder entziehen. Dafür gibt es wiederum verschiedene Strategien, die auf aktivitätsspezifischen Informationen wie der Priorität, der Abhängigkeit von anderen Aktivitäten oder dem Endtermin aufbauen können. Zu klären ist, wann eine Aktivität einen Ausführungsthread erhält, und wann er ihr wieder entzogen wird.

Zunächst konkurrieren nur Aktivitäten derselben Kategorie (es gibt eventuell nur eine) um Threads. Aus der Menge der laufenden und wartenden Aktivitäten ist eine kompatible Menge mit dem größten Nutzen für den Agenten zu bestimmen, deren Kardinalität der Anzahl der verfügbaren Threads entspricht. Der Nutzen ist eine Funktion über der Priorität, der bereits für die Aktivität verwendeten Zeit (evtl. Timeouts) und dem gewünschten Endtermin. Die Kompatibilität ist eine Frage der Abhängigkeit. Zwei Aktivitäten sind nicht parallelisierbar, wenn das aus den zugehörigen Beschreibungen (zum Beispiel aus dem durch eine dahinter liegende Absicht verkörperten Plan) hervorgeht. Ein dem Ablaufschema von Agent0 entsprechender Algorithmus würde zum Beispiel alle gerade gestarteten Interaktionsaktivitäten, dann alle Regelauswertungsaktivitäten und zum Schluss alle Ausführungsaktivitäten nacheinander dem einzigen

Thread zuordnen und wieder von vorn beginnen. Das Ablaufschema von INTERRAP könnte man mit drei Threads für jeweils eine Kategorie nachbilden. In jedem dieser Threads erfolgte die Zuordnung in einem Zyklus.

Die nächste zu klärende Frage ist, wann und wie oft Neuzuteilungen (also der erneute Aufruf des Schedulingalgorithmus) stattfinden sollen. Auslöser von Neuzuteilungen könnte ein Zeitsignal, der positive oder negative Abschluss einer Aktivität (also das Freiwerden eines Threads), das Blockieren oder Blockierungsende einer Aktivität, der Neustart von Aktivitäten durch innere oder äußere Signale oder der explizite Aufruf durch eine Aktivität sein. In Agent0 würde zum Beispiel eine Neuzuteilung nur nach dem Ende einer Aktivität stattfinden.

Insgesamt sind in der Aktivitätsauswahl drei Variationspunkte vorhanden: die Einsatzstrategie für Threads, die Zuordnungsstrategie und die Auslösestrategie. Das gesamte Bild der Aktivitätsauswahl stellt die Abbildung 3.13 zusammenfassend dar.

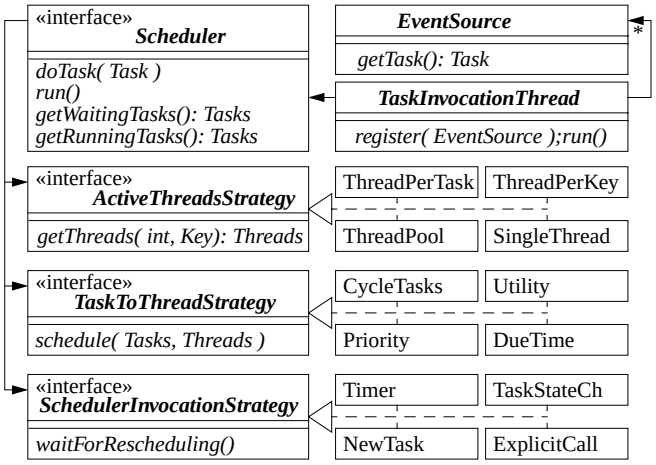

Abbildung 3.13: Diagramm für die Aktivitätsauswahl

Wurde eine Aktivität fehlerhaft abgebrochen, so kann sie zu einer Ersatzaktivität führen. Diese ist vom über den Abbruch informierten Initiator der Aktivität zu erzeugen.

Effizienz vs. Flexibilität

Was das Abwägen zwischen Effizienz und Flexibilität betrifft, so ist hier zunächst der Programmierer des Agenten gefragt. Er muss die zu seiner Anwendung passenden Strategien aussuchen. Die Strategien selbst zur Laufzeit zu ändern, bedeutet einen tiefen Eingriff in das Verhalten des Agenten. Prinzipiell steht dem aber nichts entgegen, so könnte zum Beispiel die Anzahl der verfügbaren Threads in einem Threadpool einstellbar sein.

3.4.3 Zusammenfassung

Dieser Abschnitt führte ein Klassengerüst für verschiedene Ablaufmechanismen ein. Mit ihm kann man Zyklen wie in Agent0 oder raffiniertere Algorithmen wie in INTERRAP umsetzen. Insbesondere ist es möglich, den Aspekt der Rechtzeitigkeit von Aktionen zu berücksichtigen, indem der Scheduler Endtermine von Aktivitäten bei der Auswahl beachtet. Die Basis für ein solches Klassengerüst ist die Ausweitung der bisherigen Aktivitätsdefinition (Aktionen auszuführen) auf interne Abläufe. Eine Aktivität ist mit Informationen ausgestattet, die der Schedulingalgorithmus benutzen kann, sie sind in dieser Hinsicht transparent für Auswahlprozesse im Agenten.

3.5 Kommunikation, Verteilung und Kooperation

Bisher war in diesem Kapitel hauptsächlich von den Strukturen und Abläufen innerhalb eines Agenten die Rede. Hier soll es nun um das gemeinsame oder kooperative Handeln von Agenten gehen.

3.5.1 Aufgaben und Quellen

Die erste Voraussetzung für die Kooperation zwischen Agenten ist die Möglichkeit, Nachrichten zwischen ihnen auszutauschen. Zwar werden in der Literatur auch Modelle vorgestellt, Kooperation ohne Kommunikation zu realisieren (und die Fußballanwendung zeigte eine Variante), dazu muss jedoch ein Agent das Verhalten des anderen kennen oder beobachten können. Hier soll es um explizite Kooperation gehen.

Vor einem Nachrichtenaustausch steht die Suche nach Kooperationspartnern, welche anderen Agenten sind vorhanden, welche Fähigkeiten besitzen sie. Für diese Frage der Verteilung, der Adressierung und des Entdeckens von Agenten sollten Agentenframeworks auf weit entwickelte und gut etablierte Architekturen aus dem Bereich der verteilten Systeme zurückgreifen. Diese Einsicht führte in einigen Systemen (z. B. in Agent0) zur Ausklammerung des Kommunikations-Aspekts. Die Kommunikation wird dort innerhalb eines Prozesses simuliert. Auch die meisten theoretischen Konzepte zur Kooperation legen keinen Wert auf die Beschreibung der tatsächlichen Umsetzung des Nachrichtenaustauschs. Diese Grundlagen sollen hier trotzdem benannt werden, weil sie zum Teil auch Lösungsansätze für Probleme liefern, die enger mit der Kooperation zu tun haben.

Ist die Grundlage für die Kommunikation gelegt, ist zu klären, welches Format die ausgetauschten Nachrichten haben und nach welchen Protokollen der Nachrichtenaustausch stattfindet. Erst dann ist es möglich, Konzepte der nachrichtenbasierten Kooperation zu betrachten. Hier liegen in der Literatur vor

allem theoretische Arbeiten vor, die Anforderungen und Lösungsansätze für kooperatives Handeln aufzeigen.

Verteilung

Mit CORBA, Java-RMI, EnterpriseJavaBeans, Distributed COM und vielen weiteren Verteilungsplattformen ist eine breite Basis vorhanden, die Verteilung von Agenten aufzusetzen. Sie bieten im Allgemeinen einen Namensdienst zur Adressierung und einen Broker zur Bekanntgabe von Diensten und Nachfrage nach Diensten an. Auch die Mobilität von Agenten ist eine Frage der zugrundeliegenden Plattform. Lässt diese die Migration von Code zu, dann können auch Agenten dieses Mittel nutzen. Es ist allerdings zu fragen, wann die Migration des gesamten Agenten sinnvoll ist, wenn dieser sehr umfangreich ist.

Ein in umgesetzten Agentensystemen vorzufindendes Verteilungsprinzip ist die Blackboard-Architektur. In ihr geht jede Nachricht an eine allen bekannte Verteilungsstelle. Ein Agent muss selbst wählen, welche der dort angekommenen Nachrichten für ihn wichtig ist.

Ein weiterer Verteilungsgesichtspunkt für Agenten ist die Art der Nachrichtenzustellung. Hier spielt eher die asynchrone Kommunikation eine Rolle. Insbesondere müssen Nachrichten an einen gerade nicht aktiven Agenten aufbewahrt und zugestellt werden, wenn der Agent wieder aktiv ist.

Nachrichten

Für Nachrichten sind drei Aspekte zu betrachten: ihr Zweck, die Syntax und die Semantik des Inhalts. Die meisten für Agenten vorgeschlagenen Kommunikationssprachen basieren auf der Speech-Act-Theorie (z. B. FIPA ACL: [FIPA 97] oder KQML: [Finin et al. 92, Labrou/Finin 98]), die Nachrichten nach ihrem Zweck kategorisiert. So dienen manche Nachrichten der Bekanntgabe von Informationen (inform, provide-information etc.), manche der Aufforderung, eine bestimmte Aktion auszuführen (request, request-action, command etc.), und manche der Abfrage einer bestimmten Information (query, request-information etc.). Der Zweck der Nachricht wird in diesen Kommunikationssprachen explizit repräsentiert. Er bestimmt das Verhalten des Empfängers. Damit der Empfänger den Inhalt der Nachricht entschlüsseln und in seine interne Repräsentation überführen kann, muss er die benutzte Syntax kennen. Das Nachrichtenformat der Kommunikationssprachen sieht deshalb einen Eintrag für die Kennzeichnung der Syntax vor. Agenten, die dieselbe interne Repräsentation von Informationen besitzen, können diese auch als Austauschsyntax benutzen, wenn dies nicht der Effizienz oder Datensicherheit widerspricht. Agenten mit unterschiedlichen eigenen Formaten müssen sich auf ein Format einigen, das jeweils in das lokale übersetzbar ist. Ein Vorschlag für ein solches Format ist KIF (Knowledge Interchange Format: [Genesereth/Fikes 92]).

Auch nach dieser Einigung ist nicht gesichert, dass die Kommunikation zwischen Agenten das erwartete Verhalten nach sich zieht. Die Agenten müssen zusätzlich mit den verwendeten Primitiven im Inhalt der Nachricht (Begriffe) dieselbe Bedeutung assoziieren. Als ein Lösungsansatz hat sich hier die Kennzeichnung der benutzten Ontologie (Begriffswelt) in einem Eintrag des Nachrichtenformats etabliert. Die Ontologie enthält Konzepte und ihre gegenseitigen Beziehungen, sie repräsentiert ihre Bedeutung. Beide Kennzeichnungen, die der benutzten Syntax und die der benutzten Ontologie für den Inhalt der Nachricht, müssen offensichtlich im Agentensystem eindeutig sein.

Diese beiden Voraussetzungen, gemeinsame Syntax und Ontologie, sind erfüllt, wenn das System von Agenten als Ganzes konzipiert und umgesetzt wird. Dann spielen Agenten anderer Hersteller mit anderen Formaten und anderen Repräsentationsformen keine Rolle.

Auch für andere kommunizierende Systeme stellt sich die Frage der Syntax und Semantik der Nachrichten. Hier hat sich in jüngster Zeit die eigentlich als Kennzeichnungssprache eingeführte Sprache XML ([Behme/Mintert 99, Orchard 99]) als ein sinnvoller Ansatz erwiesen. In einer allgemeinen Syntax kann man hier die konkrete Syntax der Nachricht beschreiben. Eine konkrete Nachricht enthält einen Vermerk auf diese Definition, die sich der Empfänger besorgen kann.

Protokolle

Nachdem geklärt ist, wie die ausgetauschten Nachrichten aussehen, können die Agenten nun Dialoge aufbauen. Für die Kooperation zwischen Agenten reicht es im Allgemeinen nicht aus, Nachrichten zu schicken, sondern es ist eine Folge von Anfragen und Antworten (z. B. für eine Verhandlung) nötig.

Kleine spezielle Protokolle sind bereits in den oben erwähnten Kommunikationssprachen vorhanden. So soll auf eine Anfrage-Nachricht eine Informations-Nachricht mit der Antwort folgen. Die einzusetzenden Dialoge können allerdings komplizierter werden und abhängig von der Anwendungsdomäne sein. Agenten müssen sich demnach auch über das verwendete Protokoll einigen bzw. dieselben Protokolle benutzen.

Als Domänen-unabhängiges Protokoll zum Austausch von Aufträgen und deren Lösungen findet man in der Literatur und implementierten Systemen insbesondere ein Grundmuster: das Contract-Net-Protokoll ([Davis/Smith 83]). Ein Agent sammelt in diesem Protokoll zunächst Angebote für einen Auftrag ein, wählt nach einer gewissen Zeit aus den gesammelten Angeboten eins aus und schließt mit dem Anbieter einen Kontrakt zur Ausführung des Auftrages.

In [Burmeister et al. 93] stellen die Autoren einen Mechanismus zur Entwicklung von Kooperationsprotokollen vor, die auf Sprechakten aufbauen. Ein Protokoll ist hier durch einen Namen, Parameter und einen beschrifteten Baum

möglicher Dialogschritte repräsentiert. Der Name kennzeichnet die das Protokoll auslösende Nachricht (die erste Nachricht des Dialogs). Die Parameter enthalten unter anderem Adressen vom Sender und Empfänger. Die Knoten des Baumes stehen für die Dialogzustände, Kanten für Zustandsübergänge. Gehen von einem Knoten mehrere Kanten aus, so sind das Alternativen in der Dialogführung. Knoten sind mit dem Aufruf von Subprotokollen und dem jeweiligen Sender beschriftet. Der Wurzelknoten repräsentiert die erste Nachricht im Dialog. Sie ist zusätzlich mit der zu benutzenden Empfangs- und Sendeprozedur versehen. Protokolle bauen also ausgehend von primitiven Protokollen für die Sprechakte hierarchisch aufeinander auf. Zwar ist die Protokollbeschreibung ein Baum, die tatsächliche Dialogführung kann aber zyklische Verhandlungen einschließen. So kann ein Protokoll zur Anfrage nach einem Dienst als Antwort einen Vorschlag enthalten und das Protokoll für einen Vorschlag als Reaktion eine erneute Anfrage. Mit dem vorgeschlagenen Mechanismus kann man auch das Contract-Net-Protokoll formulieren. Für die Ausführung von Dialogen sehen die Autoren den Protokollbaum als Planbaum entsprechend dem hierarchischen Planen an.

Nach all diesen Vorbereitungen seien jetzt Arbeiten analysiert, die sich mit den internen Abläufen für die Kooperation zwischen Agenten beschäftigen.

Kooperation und Kompetenzeinschätzung

Für die Kooperation spielt die Kompetenzeinschätzung eine wichtige Rolle. In [Kirn 91] arbeitet S. Kirn Prinzipien für die Kompetenzeinschätzung von Expertensystemen heraus. Sie bauen auf der benutzten Task Description Language (TDL) auf. Die Kompetenz eines Expertensystems ist hier bestimmt durch den Inhalt seiner Wissensbasen, die zur Verfügung stehenden Problemlösestrategien und die vorhandenen Hard- und Softwareressourcen. Die Kompetenzeinschätzung beruht auf einem Vergleich der Anforderungen eines Auftrags (charakterisierende Merkmale wie z. B. den geforderten Inferenztyp) mit den vorhandenen Mitteln nach folgenden Kriterien:

- Der Auftrag enthält eine Anfrage, die einer explizit repräsentierten Fähigkeit entspricht.
- Der Auftrag enthält Eingabedaten, die sich auf lokale Variablen abbilden lassen (Daten-getriebene Inferenz).
- Der Auftrag enthält Konzepte und Instanzen, die auch im Expertensystem definiert sind (unter Beachtung von Synonymen).
- Der Auftragstyp (Problemlösungs-, Inferenz-, Wissenstyp) entspricht dem Typ der Wissensbasis.
- Das Expertensystem besitzt die geforderte Problemlösestrategie (Wissensrepräsentation, Lösungsraum).

- Das Expertensystem verfügt über die notwendigen Ressourcen.
- Das Expertensystem kann die in der Auftragsbeschreibung verwendete Task Description Language interpretieren.

Nachdem klar ist, wie die Kompetenz eines Expertensystems zur Erfüllung eines in TDL beschriebenen Auftrags eingeschätzt werden kann, besteht die Kooperation hier darin, dass ein zunächst zu suchender Auftragsmanager einen vom Benutzer oder einem anderen Agenten übergebenen Auftrag durch Auftragsanalyse und Zerlegungsplanung in Teilaufträge zerlegt, diese entsprechend der Kompetenz zuordnet und verteilt. Der Auftragsmanager muss die Teilergebnisse auch wieder zusammenfügen und in Fehlerfällen Teilaufträge wieder zurücknehmen und eine neue Zerlegung berechnen. Die für die Kooperation notwendigen Aktivitäten innerhalb eines Agenten steuert ein Kooperationsmanager. Er übernimmt Aufträge oder lehnt sie ab.

Über die Kompetenzeinschätzung hinaus gehen Arbeiten, die den Begriff des Vertrauens in Multiagentensystemen thematisieren (wie [Castelfranchi/ Falcone 98]). Ein Agent schätzt dabei neben der Kompetenz auch das tatsächliche Verhalten anderer Agenten ein.

Kooperation als Koordination von Plänen

In [v. Martial 92] führt F. v. Martial die Kooperation als Koordination von individuellen Plänen ein. Grundlage ist das hierarchische Planen. Jeder einzelne Agent besitzt einen Plangraphen, der seine nächsten Aktivitäten vorgibt. Diese sollen koordiniert ausgeführt werden. Im Gegensatz zu einem gemeinsamen Plan, der verteilt wird, gibt es hier also Einzelpläne, deren Ausführung aufeinander abgestimmt werden muss.

F. v. Martial identifiziert mehrere negative und positive Beziehungen zwischen Plänen, die dabei berücksichtigt werden müssen. So passen Pläne nicht zusammen, wenn in ihnen Aktionen enthalten sind, die die gleichen Ressourcen benötigen. Der Zugriff auf diese Ressourcen muss sequenzialisiert werden. Pläne sind inkompatibel, wenn sie Aktionen mit entgegengesetzten Effekten enthalten. Positive Beziehungen zwischen Plänen bestehen, wenn sie identische Aktionen enthalten, ein Agent also auf die Ausführung verzichten könnte; oder wenn eine Aktion eine andere subsumiert. Eine weitere Beziehung zwischen zwei Plänen besteht, wenn eine Aktion von einem Agenten geplant, von einem anderen aber ausgeführt werden soll.

Vor der Ausführung der Pläne übernimmt ein spezieller Agent deshalb deren Koordination. Er schränkt die zeitliche Anordnung der Einzelaktionen weiter ein oder unternimmt andere Aktionen, um Konflikte aufzulösen oder positive Beziehungen zu nutzen. Er teilt die koordinierten Pläne den beteiligten Agenten mit. Die Agenten benutzen dabei ein fest definiertes Kommunikationsprotokoll. Die Agenten entscheiden selbst, ob sie den Vorschlag des Koordinators annehmen oder ob andere Lösungen gesucht werden müssen.

Das System GPGP ([Lesser 98, Decker/Lesser 98]) baut auf diesen Ideen auf. K. S. Decker und V. R. Lesser führen 5 erweiterbare Koordinationsmechanismen ein:

- die Aktualisierung nicht lokaler Informationen,
- die Benachrichtigung über Ergebnisse,
- die Behandlung einfacher Redundanzen in den Aktionen,
- die Behandlung starker Abhängigkeiten zwischen Aktionen und
- die Behandlung schwacher Abhängigkeiten zwischen Aktionen.

Voraussetzung für ihre Mechanismen ist ein lokaler Aktions-Scheduler. Sie gehen außerdem davon aus, dass Agenten anderen Agenten glauben und nicht absichtlich lügen. Der lokale Scheduler berechnet aus der aktuellen Aufgabenstruktur (repräsentiert in TAEMS), einer Menge von Verpflichtungen des Agenten und einer Menge von ihm bekannten Verpflichtungen anderer Agenten (z. B. diesen Agenten zu unterstützen) eine Zuordnung von Methodenaufrufen zu Zeitpunkten, die möglichst alle vorhandenen Einschränkungen erfüllt und die höchste Bewertung aller Varianten hat. In der Aufgabenstruktur sind quantitative Angaben für die Bewertung enthalten. Ziel der Koordinationsmechanismen ist es, den Schedulingalgorithmus mit Informationen zu versorgen, so dass die Aktionsausführung koordiniert mit anderen Agenten stattfindet.

Entsprechend den Eingabedaten für den Scheduler betreffen die Informationen die Aufgabenstruktur, die eigenen Verpflichtungen oder die Verpflichtungen anderer Agenten. Der erste Mechanismus hat als Ziel, Abhängigkeiten zwischen privaten und gemeinsamen Aktionen auszutauschen (also Ausschnitte aus der Aufgabenstruktur). GPGP bietet Agenten eine spezielle Aktion, die solche Abhängigkeiten berechnen kann. Die Abhängigkeiten sind nach der Berechnung in der privaten Aufgabenstruktur enthalten. Sie sind allerdings auch für andere Agenten wichtig, nämlich die, die an den abhängigen Aktionen beteiligt sind. Der Mechanismus erlaubt es deshalb, genau die lokal berechneten Abhängigkeiten an diese Agenten zu übermitteln.

Der zweite Mechanismus kann in folgenden Varianten auftreten: der Ergebnisübermittlung nur zur Erfüllung von Verpflichtungen; der zusätzlichen Übermittlung der für das Ergebnis benötigten Aufgabenstruktur (Task group) oder der Übermittlung aller Ergebnisse an alle anderen Agenten.

Der dritte Mechanismus sorgt dafür, Redundanzen in Aufgabenstrukturen zu erkennen und aufzulösen, indem nur ein nach dem Zufallsprinzip ausgewählter Agent verpflichtet wird, die redundante Aufgabe auszuführen.

Zwischen zwei Aktionen besteht eine starke Abhängigkeit, wenn die eine die andere erst ermöglicht, die eine also vor der anderen stattfinden muss. Der hier eingesetzte Mechanismus versucht, die erste Aktion zum frühestmöglichen Zeitpunkt anzuordnen. Ein Agent, der die vorausgesetzte Aktion ausführt,

verpflichtet sich, das frühestmöglich zu tun. Dieser Zeitpunkt ist der erste Zeitpunkt, für den der Scheduler die Aktion einordnen kann, so dass die Bewertung des Zeitplans für den Agent positiv ist. Schwache Abhängigkeiten werden ähnlich behandelt wie starke Abhängigkeiten, sind aber Gegenstand einer Verhandlung. Zwei Aktionen sind schwach voneinander abhängig, wenn die Ausführung der ersten die Ausführung der zweiten verbessert (höhere Qualität, kürzere Ausführungszeit) oder behindert.

Koordination und Normen

In [Ossowski 99] stellt S. Ossowski die ProsA$_2$-Architektur vor, die speziell die Koordination zwischen Agenten in dynamischen Umgebungen berücksichtigt. Ähnlich zu den Arbeiten von F. v. Martial und in GPGP sind positive und negative Beziehungen zwischen Plänen definiert (unabhängig, unterstützend oder behindernd, ermöglichend und verhindernd). Aufbauend auf den Beziehungen der auszuführenden Pläne sind Beziehungen zwischen Agenten definiert. Ein Agent besitzt eine Menge von Plänen, die er unabhängig von anderen Agenten ausführen kann, und eine Menge von Plänen, die außerdem nicht durch andere Agenten behindert werden. Diese letzte Menge von Plänen kann er ohne Koordination mit anderen ausführen. Für andere Pläne ist die Koordination sinnvoll oder notwendig.

Die Ausführung eines Planes verbindet ein Agent mit einem bestimmten Nutzen (Utility). Agenten koordinieren ihre Pläne mit anderen Agenten, wenn sie sich davon eine Erhöhung des Nutzens versprechen. S. Ossowski benutzt hier Ergebnisse der Utilitäts-Theorie. Der Nutzen eines Planes hängt von den gleichzeitig durch andere Agenten ausgeführten Plänen ab. Für jede Kombination von Plänen unterschiedlicher Agenten ist der für jeden beteiligten Agenten resultierende Nutzen bekannt. Koordination bedeutet dann Suche in einer Nutzenmatrix. Der für einen Agenten mindestens erreichbare Nutzen ohne Koordination ergibt sich als das Maximum der jeweils schlechtesten Bewertungen jedes durch den Agenten ausführbaren Plans. Ein Agent würde also einer anderen Plankombination nur dann zustimmen, wenn er dadurch einen höheren als den mindestens erreichbaren Nutzen erzielt. Gibt es Plankombinationen, die für jeden Agenten einen höheren Nutzen bedeuten, so ist eine dieser Kombinationen Ziel der Koordination. Die letztendlich gewählte Kombination wird pareto-optimal sein (kein Agent kann seinen Nutzen erhöhen, ohne den Nutzen für einen anderen Agenten zu verringern).

Unter den zusätzlichen Voraussetzungen, dass die Nutzenberechnung nur von der aktuellen Situation abhängig ist (zwei Agenten erzielen denselben Nutzen, wenn sie dieselben Fähigkeiten besitzen und in derselben Situation handeln müssen), dass die zu findende Lösung unabhängig von der Skalierung der Nutzenfunktion und von irrelevanten Alternativen ist, gibt es eine eindeutige Lösung der Suche.

Die Nutzenberechnung geschieht nun nicht nur in Hinblick auf den eigenen Vorteil, sondern auch in Abhängigkeit von globalen Normen (für alle Agenten gültige Vorgaben). ProsA_2 sieht dafür eine weitere Schicht über der Kooperationsschicht (wie sie in INTERRAP zu finden ist) vor: eine Normschicht. In dieser Schicht sind Verbote (welche Aktionen ein Agent nicht ausführen darf) und Warnungen (welche Aktionen ein Agent nicht ausführen sollte) als Regeln formuliert. Ein Agent könnte also seinen Plan ausführen, wenn störende Aktionen anderer Agenten durch die Normschicht ausgeschlossen wären. Die Kooperationsschicht dient hier der Sammlung und Aktualisierung von Informationen über andere Agenten, der Identifizierung von Plankonflikten und der Auflösung der Konflikte.

Die Koordination zwischen Agenten verläuft dann in drei Schritten: Zuerst tauschen alle Agenten ausführbare, pareto-optimale Plankombinationen aus. Im zweiten Schritt führt ein Agent die oben beschriebene Suche nach einer optimalen Kombinationsmöglichkeit aus (Näherungsverfahren) und teilt sie im dritten Schritt den beteiligten Agenten mit.

Ähnliche Ideen zur Benutzung von Normen zur Steuerung des Verhaltens eines Systems von Agenten werden auch in anderen Arbeiten betrachtet (z. B. [Rosenschein/Zlotkin 94] oder [Conte/Castelfranchi 95]).

Kooperation als gemeinsames Handeln für ein gemeinsames Ziel

Andere Autoren sehen in der Kooperation nicht nur die Koordination individueller Pläne, sondern das gemeinsame Handeln hin zu einem gemeinsamen Ziel. Beispiele dafür sind die Arbeiten von A. Haddadi ([Haddadi 95]) am System COSY, die Kooperation in Grate ([Jennings 94]), in MECCA ([Lux/Steiner 98]) und in INTERRAP ([Müller 96, Müller 97]). Die Fragestellungen sind dann, wie gemeinsame Annahmen, Ziele und Absichten in einer Gruppe von Agenten zu Annahmen, Zielen und Absichten eines einzelnen Agenten in Beziehung stehen; wie die Absicht, eine gemeinsame komplexe Handlung auszuführen, zu individuellen Absichten zu Teilhandlungen führt; welches Wissen ein Agent über die gemeinsame Handlung haben muss.

INTERRAP ([Müller 96, Müller 97]) sieht für die Kooperation zwischen Agenten in jedem Agenten eine so genannte Kooperationsschicht vor. Sie dient dazu, die Notwendigkeit zur Kooperation zu erkennen und Kooperationsprozesse zu steuern. Die Kooperationsschicht liegt über der lokalen Planungsschicht und kann Ziele für diese Schicht vorgeben. Sie besteht aus einem Protokollauswahlmodul, einem Protokollinterpreter, einem Protokollscheduler, einem Protokollausführungsmodul und einem Steuerungsmodul. Das Auswahlmodul wählt aus einer Protokollbibliothek das passende Protokoll aus. Der Interpreter hat Zugriff auf verschiedene Verhandlungsstrategien. Das Steuerungsmodul kann Kooperationspläne erzeugen und deren Ausführung überwachen.

Stellt eine Agent Kooperationsbedarf fest, so kommt es zu Verhandlungsprozessen zwischen Agenten. Ziel der Verhandlung ist ein gegenseitig akzeptierter gemeinsam auszuführender Plan. Grundlage für die Verhandlung ist eine Menge möglicher Lösungen, ein Protokoll und eine Verhandlungsstrategie. Allerdings ist es schon vorher ein Problem, Kooperationspartner festzustellen und den Lösungsraum zu bestimmen. Deshalb erfolgt in INTERRAP vor der Verhandlung eine Phase der Informationsbeschaffung darüber, ob eine Verhandlung überhaupt sinnvoll oder notwendig ist, und über den Gegenstand der Verhandlung. Ein Protokoll besteht aus einer Menge von Kommunikationsprimitiven und einer Abbildung, die einer Rolle und einer eingetroffenen Nachricht eine Menge von möglichen Antworten zuordnet. Eine Verhandlungsstrategie bildet eine eingetroffene Nachricht in Abhängigkeit vom benutzten Protokoll, der Rolle des Agenten in diesem Protokoll, des momentan erreichten Verhandlungsstandes und einer agentenspezifischen Bewertungsfunktion für die Einschätzung des Verhandlungsstandes auf eine Antwort und einen modifizierten Verhandlungsstand ab.

Verhandlungsgegenstand in INTERRAP sind Kooperationspläne aus einer Planbibliothek. Für einen solchen Plan ist angegeben, unter welchen Bedingungen der Plan ausführbar ist, welche Konflikte er löst und welche Planziele einzelne Agenten durch ihn erreichen. Die operationale Beschreibung eines Kooperationsplanes enthält eine Menge von Paaren aus lokalem Plan und ausführendem Agent sowie eine partielle Ordnung zwischen Aktionen der lokalen Pläne. Um aus einem Kooperationsplan einen lokalen Plan herzuleiten, muss der lokale Plan entsprechend der partiellen Ordnung um Synchronisationsanweisungen ergänzt werden, um zu gewährleisten, dass eine bestimmte Aktion erst nach einer anderen stattfindet, wenn dies durch die partielle Ordnung vorgegeben ist.

Der Verhandlungsprozess besteht dann insgesamt aus folgenden Schritten: dem Austausch von Informationen über das momentane Ziel der einzelnen Agenten; der Bestimmung des Verhandlungsgegenstandes; der Festlegung des Protokolls und der Zuweisung der dort enthaltenen Rollen an die beteiligten Agenten; der Wahl eines ausgezeichneten Agenten, der die Menge möglicher Lösungen berechnet; der Benachrichtigung aller Beteiligten über diesen Lösungsvorschlag und der eigentlichen Verhandlung unter Nutzung von Verhandlungsstrategien. Ergebnis der Verhandlung ist eine von allen Beteiligten akzeptierte Lösung für den Verhandlungsgegenstand, in diesem Fall ein Kooperationsplan, der alle bestehenden Konflikte zwischen den Zielen der Agenten auflöst.

Nach der Übereinkunft über einen Kooperationsplan muss das kooperative Handeln auch stattfinden. Als Ergebnis des Verhandlungsprozesses müssen die beteiligten Agenten deshalb die Verpflichtung eingegangen sein, die notwendigen Aktionen auszuführen. N. Jennings nennt in [Jennings 94] folgende gemeinsam benötigte und explizit zu speichernde Informationen, die das Verhalten eines Agenten bezüglich gemeinsamer Ziele beeinflussen:

- die vereinbarten Kooperationsziele,
- die Kooperationspläne, mit denen die Ziele zu erreichen sind,
- den eigenen Anteil an diesen Plänen (Verpflichtungen),
- Konventionen (von ihm so genannte Regeln zur Verhaltensbeschreibung), die festlegen, unter welchen Umständen Verpflichtungen nicht erfüllt werden können,
- Kooperationskonventionen, die beschreiben, welche Informationen an Kooperationspartner gesendet werden sollen (sowohl bei normaler Abarbeitung als auch im Fehlerfall).

Kooperation und Alltagswissen

R. V. Lenat und D. B. Guha, die Leiter des Projekts CYC, nennen in [Guha/Lenat 94] als notwendige Voraussetzung für die Kooperation zwischen Agenten eine gemeinsam verfügbare Wissensbasis, die Alltagswissen repräsentiert. Betrachtet man allerdings Agenten als ein Mittel, komplexe verteilte Softwaresysteme zu implementieren, so tritt diese Forderung in den Hintergrund. Zwar sind allgemeine Mechanismen für die Verhandlung und das gemeinsame Handeln notwendig, aber man kann sie auf spezielle Probleme einschränken. Bei dieser Anwendung muss nicht menschliche Kommunikation und Kooperation nachgebildet werden. Allerdings müssen die Agenten den kommunizierten Nachrichten dieselbe Bedeutung zuordnen. Das wäre eine sinnvolle Anwendung einer solchen gemeinsamen Wissensbasis, einer Ontologie. Doch diese Forderung ist auch erfüllt, wenn die Agenten von vornherein auf einer einzigen Begriffswelt, nämlich der vom Programmierer eingeführten, beruhen.

Kooperation und Lernen

Die Anpassung einer Menge von Agenten an die momentane Situation (Lernen) trägt zur Verbesserung des Systemverhaltens bei und ist deshalb ebenfalls ein Gesichtspunkt der Kooperation. In [Brauer/Weiss 98] beschreiben W. Brauer und G. Weiss die Zusammenarbeit von Agenten in der Produktionssteuerung. Eine Maschine leitet nach dem eigenen Bearbeitungsschritt das Werkstück an die nächste, in einem Berechnungsprozess bestimmte Maschine weiter. Hier setzt die Anpassung an. Die Maschine bewertet die möglichen Nachfolgemaschinen und kann ihr eigenes Verhalten ändern, wenn die tatsächliche Bewertung von der bisherigen Schätzung abweicht. Durch die Berücksichtigung der Diskrepanz zwischen Annahme und tatsächlichem Verhalten gibt es hier wieder einen Ansatzpunkt für das Abwägen zwischen Effizienz und Flexibilität. Gelingt eine Kooperation immer und immer wieder, könnte sie fest eingestellt werden und ein vorheriger Verhandlungsprozess entfallen. Stimmt dann aber das erwartete Verhalten nicht mehr mit dem tatsächlichen überein, ist die feste

Einstellung zu lösen und eine neue Verhandlung zu führen. Einen ähnlichen Ansatz beschreibt V. R. Lesser in [Lesser 98] (siehe Abschnitt 3.7).

In [Sen/Weiss 99] nennen die Autoren weitere Varianten für das Lernen in Bezug auf Kooperation, wie das Lernen der Rollenzuteilung, das Lernen in Marktumgebungen und das Lernen zur Reduktion der Kommunikation.

Zusammenfassung

Ein Framework für die Programmierung agentenbasierter Software muss eine auswechselbare Schicht zwischen die vom Agenten benötigten Kommunikationsprimitive und die benutzte Verteilungsplattform legen. Die Verteilungsplattform bietet schon Möglichkeiten, andere Agenten zu finden und ihre Fähigkeiten abzufragen. Darauf aufbauend können Protokolle definiert werden.

Für die eigentliche Kooperation sind mehrere Punkte wichtig: Ein Agent muss zunächst erkennen, ob Kooperation mit anderen Agenten notwendig ist. Ist das der Fall, muss er die involvierten Agenten und den Verhandlungsgegenstand bestimmen. Unter Nutzung gemeinsam bekannter Protokolle ist eine Verhandlung zu führen, deren Ziel Verpflichtungen der beteiligten Agenten sind, bestimmte Aktionen auszuführen. Für diese Verhandlung gibt es in der Literatur zwei verschiedene Ansatzpunkte: einerseits die Bestimmung eines gemeinsamen Ziels, eines gemeinsamen Plans und dann die Zerlegung des Plans in individuelle Pläne und Absichten (Verpflichtungen); andererseits die Koordination vorher vorhandener individueller Pläne und Absichten. Beide Ansätze können allerdings Nutzen- oder Kostenfunktionen benutzen, um Pläne aus Sicht der Agenten einzuschätzen und darauf die Verhandlungsstrategie aufzubauen. Für die Zerlegung eines Kooperationsplanes in individuelle Pläne muss man die Kompetenz von Agenten berücksichtigen.

Nach der Einigung über die Kooperation müssen Agenten den eingegangenen Verpflichtungen nachgehen und die beteiligten Agenten über den Fortgang der Ausführung informieren. Im Fehlerfall müssen eventuell neue Verhandlungen stattfinden.

Das Konzept der Kooperation steht einer festen Verdrahtung der Kommunikationsbeziehungen entgegen. Ein Agent kann sich mit diesem Konzept flexibel an die vorhandene Umgebung, insbesondere an die vorhandenen anderen Agenten anpassen. Allerdings kostet ein Verhandlungsprozess Zeit. Durch Lernprozesse ist es möglich, beide Varianten auszubalancieren. Ein Agent kann das Verhalten anderer Agenten bewerten und bei der nächsten Entscheidung berücksichtigen. Die Bewertung ist auch ein Mittel gegen Falschinformation. Eine zusätzliche Normschicht von allen Agenten zu beachtender Regeln ist eine weitere Variante der Effizienzsteigerung. Sie vermeidet Verhandlungen zugunsten lokaler Entscheidungen.

3.5.2 Ausprägungsvarianten

Für das hier hergeleitete Framework geht es nicht um eine Modellierung aller denkbaren Kooperationsbeziehungen, sondern darum, die Kooperation zwischen Agenten als ein flexibles Mittel der Zusammenarbeit verteilter Softwareeinheiten der festen Verdrahtung von Kommunikationsbeziehungen entgegenzusetzen. Das entspricht derselben Idee bei der schon besprochenen Auftragsbearbeitung: Ein Auftrag führt nicht zu einem fest verdrahteten Methodenaufruf, sondern löst Auswahlprozesse im Agenten aus. Analog soll ein Agent nicht von vornherein zur Bearbeitung eines Auftrags mit einem vorgegebenen anderen Agenten kommunizieren, einen Teilauftrag weiterleiten und das Teilergebnis benutzen. Er soll stattdessen erst dann, wenn er einen Kooperationsbedarf feststellt, Schritte unternehmen, die zu gemeinsamen (unterstützenden oder koordinierten) Handlungen führen. Dazu gehört zunächst die Identifizierung relevanter anderer Agenten.

Einen Teil der hier geforderten Flexibilität liefern bereits Verteilungsplattformen wie CORBA oder JavaRMI mit Namensdiensten, Brokern und weiteren Konzepten. Diese Konzepte kann man über eine Zwischenschicht im Agenten nutzen. Sie dient dem Auffinden von potentiellen Kooperations- oder Koordinationspartnern und dem Nachrichtenaustausch mit ihnen.

Ziel der Kooperation

Die Kooperation zwischen Softwareagenten verfolgt ähnliche Ziele wie die Kommunikation in verteilten Systemen:

- ein Agent soll einen Auftrag eines anderen übernehmen,
  - weil jener nicht die Fähigkeit dazu besitzt,
  - weil jener sonst überlastet wäre (Loadbalancing),
  - weil dieser eine bessere Qualität liefern kann;
- ein Agent soll seine Aktivität mit der eines anderen koordinieren (Koordination),
  - damit Konflikte (z. B. Ressourcenzugriff) vermieden werden,
  - um die Aktion des anderen möglich zu machen oder zu unterstützen,
  - weil gewisse Handlungen die gleichzeitige Aktivität mehrerer Agenten erfordern;
- Agenten sollen einen gemeinsamen Plan ausführen (Kohärenz),
  - um das gewünschte Systemverhalten zu realisieren,
  - um ein gemeinsames Ziel zu erreichen,
  - um komplexe Aufgaben zu lösen.

Im Unterschied zu fest verdrahteter Kommunikation soll es allerdings Gegenstand von Auswahlprozessen innerhalb des Agenten und nicht des abzuarbeitenden Programmcodes sein, wann und wie er mit anderen kooperiert, ob er Hilfe anfordert oder gewährt.

Die Fähigkeit zu kommunizieren

Die erste Voraussetzung für diese Forderung ist die Möglichkeit zur Kommunikation. Entsprechend der bisherigen Modellierung ist die Formulierung der Kommunikation als Fähigkeit des Agenten angemessen. Fähigkeiten, die Kommunikationsvorgänge beschreiben, bilden gerade die Zwischenschicht zwischen benutzter Verteilungsplattform und dem Agenten. Zur Modellierung als Fähigkeit gehören die operationale und die charakterisierende Beschreibung. Für den Nachrichtenaustausch mit anderen Agenten benötigt der Agent die Fähigkeit, dessen Adresse herauszufinden, Nachrichten zu senden und Nachrichten zu empfangen (siehe Abbildung 3.14).

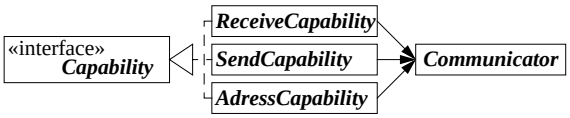

Abbildung 3.14: Diagramm für die Fähigkeiten zur Kommunikation

Fähigkeiten können zwischen Agenten ausgetauscht werden, so auch die Fähigkeit, über bestimmte Verteilungsplattformen zu kommunizieren. Im Framework muss eine Auswahl (eine Ausgangsbasis) dieser Fähigkeiten bereits vorhanden sein. Der Agenten-Programmierer sollte nur in speziellen Fällen eigene Varianten umsetzen müssen.

Die Fähigkeit, Protokolle auszuführen

Auf den Fähigkeiten zu kommunizieren bauen Fähigkeiten auf, Protokolle einzuhalten. Die operationale Beschreibung dieser Fähigkeit kann durch einen hierarchischen Planer geliefert werden. Dieser muss allerdings keinen Plangraph, sondern einen Protokollgraph traversieren und die Knoten entsprechend der Rolle in Sende- oder Empfangsaktionen umsetzen. Insofern kann man ein Protokoll bereits als einen Kooperationsplan mehrerer Agenten auffassen (siehe Abbildung 3.15). Analog zu [Burmeister et al. 93] bauen komplexere Protokolle auf einfachen auf.

Die Analogie zwischen Protokollen und Kooperationsplänen geht weiter: an einem Protokoll sind verschiedene Rollen beteiligt. Nicht jeder Agent kann jede Rolle übernehmen, es sind gewisse Fähigkeiten dafür nötig (siehe Abbildung 3.16). Rollen haben zu Agenten dasselbe Verhältnis wie Schnittstellen zu Klassen. Nur bestimmte Klassen können für eine Schnittstelle eingesetzt werden.

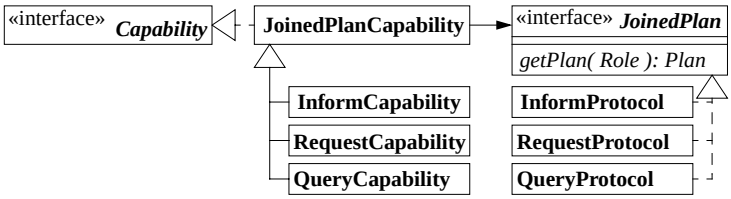

Abbildung 3.15: Diagramm für die Fähigkeit, ein Protokoll auszuführen

Demnach kann man die Rollenverteilung für ein Protokoll auch als Zerlegungsaufgabe für einen Kooperationsplan auffassen.

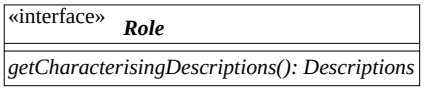

Abbildung 3.16: Diagramm für eine Rolle

Die in einem Protokoll auszuführenden Aktionen sind zumeist Kommunikationsaktionen. Schlagen sie fehl oder überschreiten sie ein Zeitlimit, so sind analog zu Kooperationsplänen Ersatzaktionen notwendig.

Kooperieren als Handlungsvariante

Die Kooperation stellt sich aus der Sicht eines Agenten als Handlungsvariante dar. Ist sie in der aktuellen Situation anwendbar und passt sie zu den Zielen (insbesondere zu den Kooperationszielen), so deutet das auf einen Kooperationsbedarf hin (im weitesten Sinn: entweder durch eintreffende Nachricht ausgelöst, durch einen nicht allein ausführbaren Auftrag oder durch die Erkennung eines Konflikts). Der Agent entscheidet in seinem Aktionsauswahlalgorithmus, ob er diese Variante wählt und Kooperationsabsichten fasst, Kooperationspläne verfolgt, Kooperationsverpflichtungen eingeht, Kooperationsaktivitäten startet (siehe Abbildung 3.17).

Auftragsvergabe

Aufträge und Auftragsbeschreibungen waren schon Thema des Abschnitts über die Repräsentation von Aktionen. Für sie ändert sich nichts, wenn nun auch die Auftragsvergabe von einem Agenten an einen anderen zugelassen ist. Der beauftragende Agent kann die Kompetenz des potentiellen Auftragnehmers mit Hilfe der charakterisierenden Beschreibungen von dessen Fähigkeiten einschätzen. Dafür ist ein Dialog (Protokoll) zum Austausch dieser Beschreibungen notwendig. Komplexere Auftragsvergabemechanismen können mit dem Contract-Net-Protokoll realisiert werden. Ein Auftrag kann die operationale Beschreibung der auszuführenden Handlung ebenfalls enthalten. Hat ein Agent einen Auftrag angenommen, so führen Auftragnehmer und Auftraggeber einen

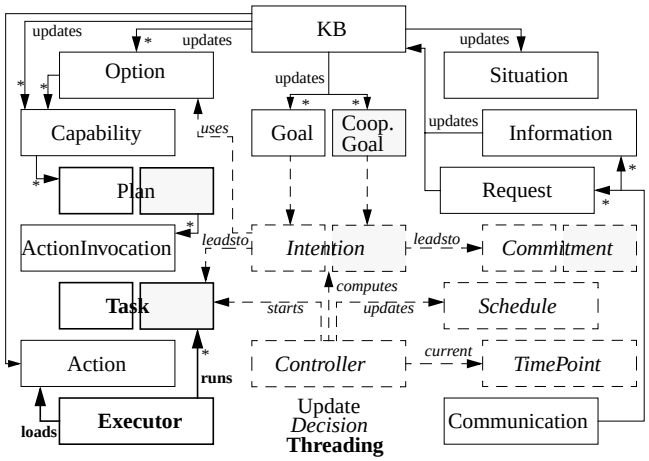

Abbildung 3.17: Kooperation als Handlungsvariante

gemeinsamen einfachen Kooperationsplan aus: Der Auftragnehmer wartet auf die Ausführung durch den Auftraggeber.

Koordination

Koordination (Bottom-up-Kooperation) ist notwendig, wenn individuelle Pläne voneinander abhängen (entsprechend der Planbeziehungen in [v. Martial 92] oder [Ossowski 99]). Die Abhängigkeit muss aus der charakterisierenden Beschreibung hervorgehen. Ein Koordinationsprotokoll soll für eine Ausführungsreihenfolge sorgen, die die Beziehungen berücksichtigt. In der ersten Phase (Verhandlung) müssen sich die Agenten auf die Reihenfolge einigen. In den vorgestellten Arbeiten realisiert das Verhandlungsprotokoll (mit einer Koordinator-Rolle) die Sammlung der individuellen Pläne beim Koordinator, die Berechnung und Verteilung eines Kooperationsplanes (der den individuellen Plänen entspricht) an die Agenten, deren Zustimmung (auf Grundlage einer Bewertung) oder einen weiteren Zyklus. Endet die erste Phase mit einer Zustimmung, führen die Agenten den Kooperationsplan in der zweiten Phase synchronisiert aus. Die Zerlegung des Kooperationsplanes zurück in individuelle Pläne geschieht hier kanonisch, da der Kooperationsplan ja von vornherein aus individuellen Plänen zusammengesetzt ist.

Der Agent, der im Verhandlungsprotokoll die Rolle des Koordinators übernimmt, muss die Fähigkeit besitzen, einen Kooperationsplan aus den individuellen Plänen zu berechnen. Die Übernahme einer Rolle ist wie die Übernahme eines Auftrags an Fähigkeiten gebunden.

Handeln für ein gemeinsames Ziel

Ein gemeinsames Ziel entsteht im Allgemeinen durch einen Auftrag, der nicht durch einen Agenten allein zu bearbeiten ist. In der ersten Phase des hier be-

nötigten Protokolls (Top-down-Kooperation) muss ein Agent in der Rolle des Planers einen Kooperationsplan berechnen, der zum Ziel hinführt. Der Planer muss dafür die zur Zusammenarbeit bereiten Auftragnehmer und deren zur Verfügung gestellte Fähigkeiten (deren charakterisierende Beschreibung: Kompetenz) berücksichtigen. Der Planer kann den Kooperationsplan durch hierarchisches Planen auf Basis einer Planbibliothek oder durch zustandsbasiertes Planen auf der Basis der charakterisierenden Beschreibungen erzeugen. Im zweiten Schritt erfolgt die Planzerlegung (unter Berücksichtigung der Fähigkeiten beteiligter Agenten) und die Verteilung auf die Auftragnehmer. Die Verteilung entspricht der Auftragsvergabe. Sie kann zur Ablehnung und Neuplanung führen. In der letzten Phase ist die synchronisierte Ausführung des Kooperationsplanes nötig.

Die Fähigkeit, Kooperationspläne synchronisiert auszuführen

Es wurde bereits die Analogie zwischen Protokollen und Kooperationsplänen herausgestellt. Hat ein Agent die Absicht, Kooperationspläne auszuführen, so führt das zu zeitlich geordneten Aktivitäten und Verpflichtungen. Ist eine Aktivität von Aktionen anderer Agenten abhängig, so synchronisieren sich beide durch entsprechende Nachrichten (blockierendes Empfangen).

Der Aktionsauswahlmechanismus muss dafür sorgen, dass die zum Kooperationsplan gehörenden Aktivitäten rechtzeitig ausgeführt werden, oder im Fehlerfall Ersatzhandlungen stattfinden. Die Ersatzhandlungen können durch allgemeine Regeln beschrieben (wie in GRATE) oder selbst Bestandteil (als alternativer Abarbeitungszweig) des Kooperationsplans sein.

Effizienz vs. Flexibilität

Kooperation (mit Feststellen des Bedarfs, Auswahl der Partner, Verhandlung und Planerzeugung, synchronisiertem Ausführen) und fest verdrahtete Kommunikation bilden die beiden Eckpunkte bei der Balance zwischen Flexibilität und Effizienz. Für Agenten liegt zunächst mehr Gewicht auf der Flexibilität. Stellt sich allerdings heraus, dass diese nicht in diesem Umfang nötig ist, sollten Möglichkeiten eingeschränkt, die Kommunikationsbeziehungen effizienter gestaltet werden. Wieder bietet es sich hier an, erfolgreich durchgeführte Kooperationen, in diesem Fall den Kooperationsplan und die beteiligten Partner in einer Fallbasis abzuspeichern. Bei der nächsten gleichartigen Situation könnte der abgespeicherte Plan bereits Ausgangspunkt der Verhandlung sein oder sofort zur Beauftragung führen. Erst wenn das fehlschlägt (weil zum Beispiel nicht mehr alle vorigen Kommunikationspartner aktiv sind), ist die volle Bandbreite der Kooperation nötig.

Ein weiterer Ansatzpunkt ist die Bewertung anderer Agenten durch die Sammlung von Informationen (z. B. zu dessen Fähigkeiten). Dann können Teile der oben beschriebenen Protokolle entfallen. Stehen Pläne zweier Agenten immer

wieder im Konflikt zueinander, dann könnte eine Neuaufteilung der notwendigen Fähigkeiten zur Beseitigung des Konflikts führen und damit zur Beseitigung einer Ineffizienz (der wiederholten Koordination).

Ein Schwachpunkt der Kooperation zwischen Agenten kann die Komplexität der verwendeten Protokolle und der notwendigen Kommunikationsstruktur sein ([Wooldridge/Jennings 98]). Für die Vereinfachung der Kommunikationsstruktur sind Mittel der Verteilungsplattform wie die Gruppierung und der Einsatz von Verteilern zu nutzen. Letztlich ist es aber eine Aufgabe des Programmierers, das Agentensystem sinnvoll zu strukturieren.

3.5.3 Zusammenfassung

Dieser Abschnitt zeigte Ansätze, wie die Kooperation zwischen Agenten in das in dieser Arbeit entwickelte Framework einbezogen werden kann. Die Grundideen dafür sind die Formulierung der Kommunikation und Protokollausführung als Fähigkeiten (und so als Schicht zwischen Verteilungsplattform und Agent), die Auffassung eines Protokolls als Kooperationsplan, der Zusammenhang zwischen Rolle und Kompetenz, die Kompetenzeinschätzung für die Auftragsvergabe durch charakterisierende Beschreibungen von Fähigkeiten, die Erkennung des Kooperationsbedarfs durch anwendbare Handlungsvarianten und die Ausführung des Anteils an Kooperationsplänen durch speziell gekennzeichnete Aktivitäten. Das Framework muss eine erweiterbare Auswahl an Kommunikationsfähigkeiten und Protokollen zur Verfügung stellen.

Betrachtet man zusätzlich die Kooperation von Agenten unterschiedlicher Hersteller, unterschiedlicher Bereiche, muss man für die Bedeutungsgleichheit von Konzepten bei allen Agenten sorgen. Auch erst dann ist eine Normschicht (wie in [Ossowski 99] vorgeschlagen) sinnvoll, sie hätte ja hauptsächlich die Aufgabe, unkooperatives Verhalten zu verhindern. In einem als Ganzes entworfenen Agentensystem ist aber im Allgemeinen Kooperation und nicht Konkurrenz gewünscht.

Eine wichtige Frage bei der Kommunikation ist die Datensicherheit, die aber zum Teil durch die benutzte Verteilungsplattform gewährleistet ist.

3.6 Sicherheit: Ausführung und Interaktion

Es bestehen aber auch Anforderungen, die über die Verschlüsselung des Nachrichtenaustauschs hinausgehen. In den Veröffentlichungen zu Agenten spielt das Thema der Sicherheit aber bisher eine untergeordnete Rolle.

3.6.1 Aufgaben und Quellen

Für einen Agenten sind verschiedene Sicherheitsaspekte relevant:

- Ein Agent sollte die Identität eines anderen Agenten und die Echtheit der Quelle von Nachrichten feststellen können.
- Der Datenaustausch sollte verschlüsselt sein und Datenverfälschungen während des Transports sollten bemerkt werden.
- Ein Agent sollte seine Fähigkeiten in Abhängigkeit von vergebenen Benutzungsrechten für andere Agenten einsetzen.
- Ein an einen anderen Agenten vergebener Auftrag sollte von diesem gemäß der Vereinbarung rechtzeitig erledigt werden und das erwartete Ergebnis haben.
- Der andere Agent sollte die ihm gelieferten Daten nicht anderweitig verwenden.
- Die Auftragsbearbeitung durch einen anderen Agenten sollte bei diesem nicht zu Störungen führen, ihn nicht überlasten und durch den Auftragnehmer nicht für andere Zwecke benutzt werden.
- Von anderen Agenten übernommene Fähigkeiten sollten keine unerwünschten Nebeneffekte aufweisen.

Der verschlüsselte Nachrichtenaustausch, die Verhinderung unbemerkter Datenverfälschungen, die Identifizierung von Agenten und die Echtheit der Quelle (Unterzeichnung von Nachrichten) können bereits durch die benutzte Verteilungs- und Kommunikationsplattform gewährleistet werden (siehe z. B. [McGraw/Felton 99]). Ein Agent erhält dafür einen eindeutigen Namen (mit privatem und öffentlichem Schlüssel) und eventuell Angaben über seine Zugehörigkeit (Autor, Organisation etc.).

Selektive Weitergabe und Aufnahme von Informationen

G. Wagner führt in [Wagner 97b] das Konzept der Multi-Level-Security für vivide Agenten ([Wagner 97]) ein. Dieses Konzept hat zum Ziel, Information in Abhängigkeit eines Vertrauensgrades (Clearance Level: natürliche Zahl) weiterzureichen. Die in einer Datenbank enthaltenen Informationen sind deshalb ebenfalls nach ihrer Schutzbedürftigkeit (natürliche Zahl) klassifiziert. Ein Agent bekommt nur korrekte Antworten zu Informationen, die mit einer Zahl gekennzeichnet sind, die kleiner oder gleich dem Vertrauensgrad des Agenten sind. Anderenfalls kann der Agent eine gefälschte (also nicht den Daten der Datenbank entsprechende) Antwort bekommen. Die Möglichkeit, die Frage in diesem Fall nicht zu beantworten, schließt G. Wagner explizit aus. Er geht davon aus, dass der Frager aus der Nichtbeantwortung ebenfalls die gewünschte Antwort ablesen könnte. Der anfragende Agent erfährt dadurch nicht, ob er für die Antwort autorisiert war und ob diese korrekt ist.

Der in diesem System benutzte Inferenzmechanismus zur Beantwortung von Fragen ist also vom Vertrauensgrad des Anfragers abhängig. Je nach dessen

Höhe berechnet er unterschiedliche Antworten. Auch bei der Aufnahme von Informationen eines anderen Agenten spielt dessen Vertrauensgrad eine Rolle: Nur wenn dieser größer oder gleich der Sicherheitsstufe der zu aktualisierenden Information ist, findet die Aufnahme statt.

Das Konzept der selektiven Weitergabe kann man ausdehnen auf die selektive Anwendung von Fähigkeiten für einen anderen Agenten. Das ist dann wieder ein Kriterium für den Auswahlmechanismus: Soll er die Handlungsvariante für die Kooperation wählen oder nicht? Es können explizite Ersatzvarianten angegeben sein, die der Agent anwenden soll, wenn die Anfrage/der Auftrag von einem nicht autorisierten Agenten kommt.

Vertrauen

In vielen Arbeiten (z. B. [Castelfranchi/Falcone 98]) diskutiert C. Castelfranchi das Prinzip des Vertrauens in Multiagentensystemen. Multiagentensysteme beruhen auf der Delegation von Aufträgen von einem Agenten an einen anderen. Die Delegation setzt das Vertrauen voraus, dass dieser den Auftrag erfüllt und die dazu notwendigen Informationen nicht anderweitig nutzt. C. Castelfranchi nennt verschiedene Kriterien für Vertrauen: die Kompetenz eines Agenten, die Vorhersagbarkeit dessen Verhaltens (wird er den Auftrag auch ausführen?), die Wichtigkeit des ausgeführten Auftrags für die eigenen weiteren Aktionen, den Grad der Abhängigkeit vom Agenten (Kann er den Auftrag wirklich besser ausführen?). Die Vorhersagbarkeit ist eine Annahme, die sich auf bisher gezeigtes Verhalten stützen kann.

Normen

Auch die von S. Ossowski ([Ossowski 99]) und anderen vorgeschlagene Normschicht über der Kooperationsschicht kann Sicherheitsaspekte umsetzen, wenn es dem Programmierer des Agenten nicht möglich ist, diese zu umgehen. Durch Normen sind allgemeine Regeln aufgestellt, an die sich jeder Agent halten muss. In einem heterogenen System von Agenten ist es allerdings fraglich, ob eine Normschicht umsetzbar ist.

Zusammenfassung

Insgesamt gibt es im Bereich der Agentensysteme wenige veröffentlichte Arbeiten zur Sicherheit. Wenn man allerdings die Zusammenarbeit von Agenten verschiedener Quellen betrachtet, die nicht von vornherein als ein System konzipiert sind, und die Notwendigkeit gemeinsamer Kommunikationssprachen und Ontologien diskutiert, dann ist auch die Sicherheit mit den oben genannten Kriterien ein Thema. Letztlich kann man aber die Sicherheit nur auf Mechanismen aufbauen, die in der darunterliegenden Verteilungs- und Kommunikationsplattform bereits etabliert sind. agentenspezifische Kriterien

sind durch die Vergabe von Rechten und die Beobachtung und Bewertung des Verhaltens und der Kompetenz anderer Agenten umsetzbar. Dazu benötigt man allerdings nicht nur die Möglichkeit der Formulierung von Aufträgen, sondern auch ein Mittel, die Ergebnisse zu kontrollieren. In die Entscheidung, ob der Agent einen Auftrag delegieren sollte, muss er die Zeit einrechnen, die er hinterher für die Ergebniskontrolle benötigt. In diesem Fall müsste die Ergebniskontrolle demnach bei weitem effektiver oder mit einfacheren Mitteln umsetzbar sein als die Ergebnisberechnung.

In homogenen Agentensystemen treten mit dem Problem der gemeinsamen Kommunikationssprache und der gemeinsamen Ontologie auch die Sicherheitsaspekte zwischen den Agenten (die letzten vier Punkte der oben stehenden Aufzählung) in den Hintergrund. In diesem Fall wäre nur das Gesamtsystem gegen äußere Beeinflussungen und falsche Benutzung zu schützen.

3.6.2 Ausprägungsvarianten

Ein Framework für die Entwicklung von Agenten muss, wie geschildert, auf den Sicherheitsmechanismen der darunterliegenden Verteilungsplattform aufbauen. Damit sind die ersten beiden Anforderungen erfüllbar.

Kommunikationsmodul

Diese ersten beiden Punkte betreffen insbesondere das Kommunikationsmodul. Die über dieses Modul umgesetzten Fähigkeiten zur Kommunikation müssen um Verschlüsselung und Quellenidentifizierung erweitert werden. Es ist auch möglich, Fähigkeiten mit unterschiedlichen Sicherheitsanforderungen (wie z. B. im Assistenz-Programm) zur Verfügung zu stellen. Die Umsetzung der Fähigkeit ist für den Agenten nicht transparent, er könnte nur im Einzelfall entscheiden, welche Fähigkeit er benutzt.

Aktionsauswahl und Ablaufsteuerung

Die Ablaufsteuerung (insbesondere die Wahl der tatsächlich ausgeführten Aktivitäten) muss so gestaltet sein, dass der Agent eingegangene Verpflichtungen einhält. Er soll beteiligte Agenten informieren, wenn das nicht möglich ist oder wenn Fehler aufgetreten sind. Dieses Verhalten kann durch einen Mechanismus des Frameworks fest verankert oder über die explizite Repräsentation von Konventionen einstellbar sein. Die Aktionsauswahl kann bei der Annahme von Aufträgen die Rechte des Auftraggebers berücksichtigen.

Ausführungsmodul

Das Ausführungsmodul muss die Ausführung von Aktionen von den anderen Abläufen im Agenten isolieren. Fehler in den Aktionen sollen nicht zu Fehlern

im Gesamtverhalten führen. Das Ausführungsmodul könnte die erlaubte Funktionalität für von anderen Agenten übernommene Fähigkeiten einschränken.

Kompetenz und Verhalten

Wie bereits erwähnt, kann ein Agent Informationen über andere Agenten sammeln. Die Kompetenz eines Agenten kann er aus den veröffentlichten charakterisierenden Beschreibungen von dessen Fähigkeiten entnehmen. Für die Einschätzung des Verhaltens sind Kriterien wie die Qualität der Lösung und die benötigte Zeit aufzustellen. Diese Informationen könnten Inhalt einer Fallbasis sein.

Fehlertoleranz

Wenn ein Agent einen Auftrag delegiert hat, so soll er nicht für immer auf dessen Erledigung warten. Die Überprüfung von Zeitbedingungen ist hier wichtig. Bei Überschreiten eines Zeitlimits ist die entsprechende Aktivität fehlerhaft abzubrechen. Der Initiator der Aktivität kann Ersatzaktivitäten auslösen.

3.6.3 Zusammenfassung

Die Beachtung von Sicherheitsaspekten in einem Framework für Agenten ist wichtig, wenn die Agenten nicht in einem isolierten homogenen System wirken. Dieser Abschnitt zeigte knapp die Ansatzpunkte für die genannten Forderungen: Fähigkeiten zur verschlüsselten Kommunikation mit Echtheit der Quelle, Isolierung der Aktionsausführung, Rechtzeitigkeit der Aktionsausführung und Benachrichtigung als Ersatzaktivität und Zeitlimits zur Verhinderung von Verklemmungen. Einige dieser Aspekte spielen auch im nächsten Abschnitt eine Rolle.

3.7 Management

3.7.1 Aufgaben und Quellen

Beim Management geht es darum, Agenten zu erzeugen und zu starten; das Verhalten der Agenten zu beobachten und gegebenenfalls zu verändern; Agenten zu beenden.

Als Quelle für die Behandlung dieser Anforderung bietet sich das Gebiet des Managements verteilter Anwendungen ([Schade 99]) und des Managements offener Systeme ([ITU 92]) an. Ein Agent entspricht in der Terminologie verteilter Anwendungen einer Anwendungskomponente. Die Komponenten der verteilten Anwendung und alle Systemressourcen, die dem Management unterliegen, werden hier als gesteuertes System bezeichnet. Ihm gegenüber steht das

steuernde System. Das Management besteht aus der Observierung des gesteuerten Systems durch das steuernde (Beobachtungsinformationen müssen vom gesteuerten zum steuernden System gelangen); der Bewertung der Informationen (Entscheidung, ob das Verhalten verändert werden soll) und der Steuerung durch Managementoperationen (Operationen müssen vom steuernden zum gesteuerten System gelangen). Das OSI-Managementmodell nennt fünf Bereiche des Managements:

- die Konfiguration (Sammeln von Informationen und Ausführen von Operationen zum automatischen Start, Stopp und zur Reinitialisierung der Anwendung, Persistenz);
- die Fehlerbehandlung (Fehleraufzeichnung, -identifikation, -lokalisierung; Ausnahmebehandlung; Testläufe zur Fehlerdiagnose);
- die Buchführung (Ressourcenvergabe);
- die Steuerung der Systemleistung (statistische Informationen zum Ressourcenverbrauch);
- die Gewährleistung der Systemsicherheit (Authentifizierung, Rechtevergabe, Schlüsselverwaltung).

Auch in der Literatur zur Programmierung von Agenten findet man Anmerkungen zu deren Management ([Hannebauer 98, Lesser 98, FIPA 97]). Hier sind die beiden Eigenschaften Steuerbarkeit und Autonomie gegeneinander abzuwägen. Jörg P. Müller ([Müller 96]) setzt Agenten in Bezug mit der Theorie der Steuerung dynamischer Systeme. Er baut eine Analogie zwischen Agenten und steuerndem System sowie der Umgebung des Agenten und gesteuertem System auf. Es soll aber unter Umständen auch möglich sein, Agenten zu steuern.

Managed Objects

In [Schade 99] geht es darum, verteilte CORBA-Anwendungsobjekte im obigen Sinne zu steuern. CORBA-Objekte sind durch eine Schnittstelle in IDL beschrieben. Die Schnittstelle legt die Syntax der Objekt-Benutzung fest, sagt jedoch nichts über deren Wirkungsweise, deren Semantik aus. Die Benutzung von IDL allein bietet so keine ausreichende Grundlage für ein allgemeines Steuerungsprinzip. A. Schade schlägt deshalb vor, die Objektbeschreibung um drei Kategorien zu erweitern, die eine Steuerung und Beobachtung erst möglich machen.

Die ergänzte Beschreibung enthält zum ersten Zustände, die das Objekt annehmen kann; zum zweiten Attribute, die man setzen und lesen oder nur lesen kann; und zum dritten Beschreibungen von Ereignissen, mit denen das Objekt interessierte andere Objekte benachrichtigen kann. Diese drei Ergänzungen unterstützen die Beobachtung durch Abfrage des Zustandes und von

Attributwerten sowie durch die Benachrichtigung über Ereignisse und die Steuerung durch die Veränderung von Attributwerten. Die eigentliche Steuerung übernehmen jedoch Steuerungsmethoden, die wie bisher auch in der Objektbeschreibung mit ihrer Signatur angegeben sind. Wichtig ist hier, dass diese Methoden mit einem Geltungsbereich versehen sind. Sie dürfen nur ausgeführt werden, wenn sich das Objekt in einem der festgelegten Zustände befindet.

Die Managementkomponente eines Objekts ist als eigener Thread realisiert. Sie nimmt Aufrufe von Managementoperationen entgegen und leitet sie an das Objekt weiter, sobald dies sich in einem für die Operation erlaubten Zustand befindet. Der Programmierer eines Managed Objects muss in den eigentlichen Code Management-Anweisungen einfügen: Er muss angeben, an welcher Stelle sich der Zustand des Objekts ändert, und wann Ereignisnachrichten gesendet werden sollen.

Das steuernde System ist ebenfalls mit CORBA-Objekten realisiert. Diese Manager genannten Objekte sind indirekt mit den zu steuernden Anwendungsobjekten verbunden: Sie melden bei sogenannten Benachrichtigungsobjekten Bedingungen an, bei deren Eintreten sie informiert werden wollen. Die Bedingungen betreffen bestimmte Ereignis-Konstellationen, wie z. B. das gleichzeitige oder sequentielle Eintreten gewisser Ereignisse. Die Benachrichtigungsobjekte beziehen die Ereignisse ihrerseits direkt von den Anwendungsobjekten. Ausprägungen von Managern sind passive Manager, bei denen letztendlich ein Mensch die Anwendungsobjekte über eine grafische Benutzungsoberfläche steuert; und aktive Manager, die entweder mit Hilfe von Regeln eine Situation diagnostizieren und geeignete Managementoperationen ableiten (das sind Agenten), oder deren Reaktion auf Ereignisse über Skripts programmierbar und zur Laufzeit einstellbar ist.

Zwei zusätzliche Objekte unterstützen das Management: eines gestattet die Aufzeichnung des Objektverhaltens über einen bestimmten Zeitraum; ein anderes das Anstoßen von Managementoperationen zu bestimmten Zeitpunkten. Insgesamt ist hier also ein Weg gegeben, Anwendungsobjekte (in unserem Fall eine Analogie für Agenten) mit zusätzlichen Managementinformationen auszustatten und steuerbar zu machen.

Diagnose in GPGP

Victor R. Lesser benennt in einer auf GPGP (Generic Partial Global Planning) aufbauenden Agenten-Architektur ([Lesser 98]) ein Modul zur Erkennung und Diagnose fehlerhaften Verhaltens des Agenten als wesentlichen Bestandteil. Das ist eine Managementaufgabe. Sie dient hier der Aufgabenaufteilung unter den Agenten. Sucht man immer erst zur Laufzeit einen passenden Agenten, so ist das zwar flexibel, aber unter Umständen nicht effektiv. Bei gleichartigen Aufgaben ist es daher sinnvoll, wenn sich eine stabile Aufgabenverteilung herausbildet. Erst wenn ein Agent seinen Anteil nicht mehr erfüllen kann, muss

die Aufteilung angepasst werden. Statt nun nach einem anderen Agenten zu suchen, der den nicht erfüllten Anteil übernimmt, soll das Diagnosemodul den Grund des Ausfalls herausfinden. Dieser Grund könnte zu einer Veränderung der Arbeitsorganisation führen, z. B. zu längeren Bearbeitungsfristen, ohne neue Agenten einzubeziehen.

Administration und Management in FIPA

FIPA ([FIPA 97, O'Brien/Nicol 98]) widmet dem Management von Agenten den ersten Teil des Standards. Dieser Teil des Standards versteht unter Management allerdings eher das Bereitstellen der Kommunikationsinfrastruktur; er definiert zuverlässige Nachrichtenverbindungen unabhängig vom zugrundeliegenden Protokoll; Adressen für Agenten; das Anbieten und Auffinden von Diensten.

FIPA beschreibt ein Referenzmodell für das Management von Agenten, das auf einer Agentenplattform beruht. Die Plattform enthält einen speziellen Agenten, der in ihr Dienste vermittelt (Directory Facilitator); einen speziellen Agenten, der in der Plattform andere Agenten erzeugen und beenden, an- und abmelden kann, und der ein Adressverzeichnis aller momentan angemeldeten Agenten führt (Agent Management System, Agent Name Service); und einen Agenten, der Nachrichten zu anderen Agentenplattformen weiterleitet (Agent Communication Channel). Die drei Hauptbestandteile des Managements fasst FIPA demnach auch als Agenten auf, das Management von Agenten als eine Anwendung von Agenten. Die speziellen Managementagenten benutzen dieselbe Kommunikationssprache wie andere Agenten und eine speziell für das Management zugeschnittene Ontologie (z. B.: register, unregister). Während man sich vorstellen kann, dass Dienstvermittler und Nachrichtenvermittler durch den Systemagenten erzeugt oder ausgewechselt werden können, so ist der Systemagent selbst verschieden von allen anderen Agenten, indem er schon existieren muss, bevor andere Agenten arbeiten können.

Mit dem Referenzmodell beschreibt FIPA wichtige Aspekte des Managements, sagt jedoch nichts zur Steuerung und Kontrolle von Agenten selbst. Das überlässt sie der jeweiligen Anwendung.

Beobachtung der Ausführung von Verhaltensmustern in INTERRAP

In INTERRAP ([Müller 96]) wird die Abarbeitung eines Verhaltensmusters in Einzelschritte zerlegt. Zwischen diesen Schritten kann das Ausführungsmodul die Abarbeitung des Musters unterbrechen und bestimmte Bedingungen überprüfen.

Diese Art der Abarbeitung ist ein Beispiel für die interne Beobachtung des Verhaltens. Durch spezielle ergänzende Verhaltensmuster kann man die Beobachtungen auch an der Oberfläche sichtbar machen.

INTERRAP besitzt einen Tracing-Mechanismus, der aktive Verhaltensmuster, lokale und Kooperationspläne anzeigt. Mit einem speziellen Tool kann man Agenten erzeugen, indem man ihre Konfiguration (Name, Adresse, Fähigkeiten, Planungs- und Konfliktlösestrategie) angibt.

Zusammenfassung

Aus der Aufzählung lässt sich erkennen, dass viele einzelne Management-Funktionen in Agentensystemen umgesetzt sind. Diese sind aber meist nicht zusammengefasst und als eigentliches Management gekapselt.

3.7.2 Ausprägungsvarianten

Es geht nun um die Frage, wie die verschiedenen Management-Funktionen in das hier besprochene Framework integriert werden können.

Fehlerbehandlung: Beobachtung und Steuerung

Agenten haben nach dem in dieser Arbeit herausgearbeiteten Vorgehen eine identische Grundstruktur. Aus dieser lassen sich bereits Informationen zum Management herleiten. Alle Bestandteile des Agenten sollen die Beobachtung und Steuerung zulassen.

Die Ablaufsteuerung arbeitet gemäß einem Zustandsdiagramm. Diese Zustände sind auch im Sinne von [Schade 99] verwendbar. Es muss möglich sein, einzelne Aktivitäten zu unterbrechen, fortzusetzen oder abzubrechen. Die Managementschnittstelle kann Zustandsübergänge anzeigen. Es muss immer möglich sein, die Arbeit des Agenten bedingungslos sofort abzubrechen. Neben dem bedingungslosen Abbruch soll der Agent geordnet beendet, unterbrochen und wieder angestoßen werden können. Diese Managementfunktionen sind als Methoden im Steuerungsmodul angesiedelt und über die Managementschnittstelle ansprechbar.

Interaktionsmodule können empfangene und gesendete Nachrichten als Ereignis (im Sinne von [Schade 99]) anzeigen, die Managementschnittstelle sollte selbst Nachrichten platzieren können.

Die Wissensbasis enthält Angaben zum aktuellen Abarbeitungszustand, zu den Zielen und Absichten und zu den eingegangenen Verpflichtungen. Entsprechend [Schade 99] sind dies Attribute, die über die Managementschnittstelle angezeigt und verändert werden können.

Die Managementschnittstelle bietet einen zentralisierten Zugriff auf die Beobachtungs- und Steuermöglichkeiten (Entwurfsmuster Facade, [Gamma et al. 95]). Eine Managementoberfläche entspricht in der Terminologie aus [Schade 99] einem passiven Manager. Sie bietet einem Benutzer oder einem

Programmierer den Zugriff auf die Managementschnittstelle eines Agenten. Es muss demnach möglich sein, vom Agenten eine Referenz auf dessen Managementschnittstelle zu erhalten.

Durch die beschriebenen Punkte ist es möglich, die Arbeit des Agenten auch intern zu beobachten und zu steuern, und das Verhalten nicht nur an ein- und ausgehenden Nachrichten zu erkennen. Es ist nicht möglich, auf diese Weise einzelne Aktivitäten zu steuern. Sie sind hier die Atome des Managed Agent, sie sind im Allgemeinen auch Anwendungs-abhängig (zumindest die Ausführung von Aktionen). Für speziell zu überwachende Aktivitäten kann eine Datenstruktur »Managed Task« (im Sinne eines Managed Object in [Schade 99]) eingeführt werden. Diese definiert selbst wieder Methoden, Ereignisse, Zustände und Attribute, die nach außen sichtbar gemacht werden können. Die Ausführung des Entscheidungsalgorithmus könnte derart realisiert sein.

Steuerung und Autonomie

Widerspricht aber diese Steuerung nicht dem allgemein Agenten zugesprochenen Attribut der Autonomie? Sollten Steuerungs- und Kontrollnachrichten über eine nicht im Einfluss des Agenten liegende Managementschnittstelle oder als normale Nachrichten über ein Interaktionsmodul gesendet werden? Aus Sicht der Programmierung ist ersteres vorzuziehen; es geht ja gerade darum, Verhalten zu überprüfen, Fehler herauszufinden. Man muss an die entsprechenden Informationen unabhängig vom eigentlichen Ablauf des Agenten herankommen. Der in dieser Arbeit vertretenen Auffassung von Autonomie widerspricht die Steuerung ebenfalls nicht: Sie ist ein Hilfsmittel, die eigentliche Aktionsauswahl unterliegt weiterhin dem Agenten selbst.

Konfiguration: Beenden, Starten, Persistenz

Laut dem OSI-Managementmodell ist die Sicherung der Persistenz in einem verteilten System ebenfalls eine Aufgabe des Managements. In einem MAS spielt die Persistenz eine sehr wichtige Rolle ([Kirn 99]). In einem MAS teilen im Allgemeinen mehrere Agenten die Lösung eines Problems untereinander auf, so dass jeder einzelne Agent den Anteil seines Spezialgebietes bearbeitet. Es muss dabei offensichtlich gewährleistet werden, dass kein Teil des notwendigen Wissens zur Problemlösung verloren geht. Ist ein Agent nicht mehr verfügbar, so müssen andere Agenten dessen Anteil übernehmen.

Persistenz eines Agenten

Betrachtet man zunächst die Problematik für einen Agenten, so stellt sich die Frage, welche seiner Komponenten Daten zum internen Zustand speichern, die der Agent beim Neustart wieder benötigt. In der hier vorgestellten Architektur sind derartige Daten einzig in der Wissensbasis gespeichert. Die Persistenz eines

einzelnen Agenten lässt sich also auf das persistente Abspeichern der in seiner Wissensbasis enthaltenen Daten zurückführen. Damit wäre gesichert, dass der Agent beim Neustart über dieselben Informationen verfügt, wie beim Beenden der Arbeit. Die Wissensbasis enthält, wie besprochen, verschiedene Arten von Informationen. Die Fähigkeiten und Handlungsvarianten eines Agenten können sich während seiner Laufzeit verändern. Diese geänderten Informationen können problemlos abgespeichert und beim Neustart wiederverwendet werden.

Die Umgebungsinformationen, über die der Agent verfügt, unterliegen ebenfalls ständigen Änderungen während dessen Laufzeit. Die Wiederbenutzung von persistent abgespeicherten Umgebungsinformationen ist jedoch problematisch. Der Agent kann in der Zeit zwischen dem Beenden der Arbeit und dem Neustart keine Umgebungsänderungen wahrnehmen. Er kann also beim Neustart nicht voraussetzen, dass die gespeicherten Informationen mit der Umgebung übereinstimmen. Diese Diskrepanz kann auf verschiedene Weise berücksichtigt werden. Ändert sich die Umgebung sehr häufig und ist der Agent durch Interaktion sehr schnell in der Lage, die aktuelle Situation zu erfassen, kann man auf die Wiederbenutzung und damit auch auf die Speicherung der Umgebungsinformationen verzichten. Wenn die Umgebung sich nicht so oft ändert und der Agent die Möglichkeit hat, Informationen über die Umgebung zu überprüfen, dann kann der Agent beim Neustart die persistent gespeicherten Umgebungsinformationen laden und testen. Wenn sich die Umgebung selten ändert, reicht es aus, die nach dem Neustart geladenen Umgebungsinformationen entsprechend mit einer Wahrscheinlichkeit ihrer weiteren Gültigkeit zu markieren.

Der dritte Teil der Wissensbasis betrifft die Motivation des Agenten, seine Ziele, Absichten, bearbeiteten Aktivitäten und seine eingegangenen Verpflichtungen. Die Abspeicherung dieser Informationen ist ebenfalls problematisch und im Allgemeinen nicht sinnvoll. Ein abgeschalteter Agent kann seine Verpflichtungen nicht einhalten. Man muss aber bedenken, welche Folgen der Abbruch einer gerade bearbeiteten Aktivität, oder das Verwerfen einer Verpflichtung und damit der zugehörenden Absicht haben, wenn man den Agenten beendet. Die hier getroffene Entscheidung hat Einfluss auf das Verhalten des Gesamtsystems. Bearbeitet der Agent gerade einen Anteil an der Lösung eines umfassenden Problems, so macht der Abbruch dieser Aktivität eine Umverteilung der Problemlösung unter den anderen Agenten nötig. Der Abbruch der Aktivität kann in diesem Fall auch zur Unlösbarkeit des Problems führen, wenn zur Lösung eine Fähigkeit notwendig ist, die einzig der abgebrochene Agent besitzt. Es muss also mindestens dafür gesorgt werden, dass alle Agenten, in deren Auftrag der Agent gearbeitet hat, das Beenden des Agenten bemerken. Das kann explizit durch eine entsprechende Benachrichtigung oder implizit durch das Ausbleiben termingerechter Ergebnisse erfolgen. Im Falle eingegangener Verpflichtungen muss der Auftraggeber benachrichtigt werden, um das zu lösende Problem gegebenenfalls anders verteilen zu können. Vor der tatsächlichen Be-

endigung des Agenten kann die Übertragung einer Problemlösefähigkeit an andere Agenten oder ihre persistente Speicherung in einem anderen Agenten zugänglichen Reservoir notwendig sein.

In speziellen Fällen lässt sich auch das persistente Abspeichern von Zielen, Absichten und Verpflichtungen denken. Wenn diese für den Auftraggeber nicht an konkrete Termine gebunden sind und nicht mit der Arbeit anderer Agenten synchronisiert werden müssen, dann kann der Agent die notwendigen Schritte auch zu einem späteren Zeitpunkt, also nach dem Neustart, ausführen. Beim Neustart lädt der Agent in diesem Fall eine Liste von Zielen, Absichten und Verpflichtungen, denen er neue Termine zuweisen und die er dann abarbeiten muss. Der Grad der Wiederbenutzbarkeit dieser Daten hängt allerdings wiederum von der Umgebung ab. In dynamischen Umgebungen kann der Agent das Ziel (einen Auftrag zu erfüllen) beim Neustart laden, muss jedoch die dafür notwendigen Schritte abhängig von der neuen Umgebung neu auswählen. In diese Neuberechnung fließen dadurch Auswirkungen bereits durchgeführter Schritte ein. Sie sind ja in der Umgebung manifestiert. In statischen Umgebungen könnte der Agent die vorher berechneten Schritte weiter ausführen, also auch Absichten und Verpflichtungen wiederbenutzen, wenn die zur Absicht gehörenden Aktivitäten vor dem Abbruch definiert beendet (d.h. komplett ausgeführt oder zurückgesetzt) wurden. Auf diese Art können halb gelöste Aufgaben nach dem Neustart beendet werden.

Persistenz im System

Will man den Bestand der im System vorhandenen Problemlösefähigkeiten sichern, müssen Agenten vor ihrem Abbruch gegebenenfalls diese Fähigkeiten an andere Agenten übertragen. Diese Übertragung besitzt Grenzen: Die Problemlösefähigkeit kann an Ressourcen gebunden sein, die nur dem einen Agenten zugänglich sind. Die Übertragung würde dann zwar das Wissen konservieren, die Anwendung der Fähigkeit wäre aber nicht mehr möglich. Andererseits kann ein Agent auch unkontrolliert beendet werden (z. B. durch einen Softwarefehler oder einen unbedingten Abbruch). Auch in diesem Fall kann dem System zumindest temporär Wissen über Problemlösefähigkeiten verloren gehen. Diesem Problem kann man durch Redundanz in den Problemlösefähigkeiten begegnen, jede dieser Fähigkeiten ist dann bei mindestens zwei Agenten vorhanden. Für bestimmte Agenten muss es eine identische Kopie im System geben.

Steuerung der Systemleistung

Das Übertragen von Problemlösefähigkeiten an andere Agenten ist auch ein Mittel zur Steuerung der Systemleistung. Ein überlasteter Agent kann dadurch ein Problem und die Fähigkeit, das Problem zu lösen, an einen anderen Agenten delegieren. Der Vorteil dieses Mittels liegt darin, dass es auch ohne zentrale Steuerung eine gleichmäßige Lastverteilung möglich machen kann.

Das Problem der Systemsicherheit in Agentensystemen wurde in einem eigenen Kapitel behandelt (Abschnitt 3.6).

3.7.3 Zusammenfassung

Dieser Abschnitt beschrieb die Einbeziehung von Managementfunktionen in das Framework. Für die Steuerung und Beobachtung stellen die Bestandteile des Agenten Attribute, Ereignisse und Managementmethoden zur Verfügung, die über die Managementschnittstelle des Agenten abgefragt, benutzt oder verändert werden können. Für die Persistenz eines Agenten und eines Agentensystems ist die Konservierung von Informationen der Wissensbasis, insbesondere der Problemlösefähigkeit nötig.