Inhaltsverzeichnis

„Während ich dies schreibe, sitze ich in einem Haus mit vielen Einrichtungsgegenständen, die ein Heer von speziell ausgebildeten Menschen für mich geschaffen haben: Es geht ja nicht nur um die Baufirmen, die Elektriker, die Installateure, die Möbelfirmen usw., die unmittelbar sichtbar sind, sondern um deren unzählige Zulieferanten und Subauftragnehmer, die selbst wieder solche haben und die alle wiederum von Infrastruktureinrichtungen abhängig sind (Strom, Straßen, Wasser...)“.

[Maurer01]

1 Einleitung

Hohe Anforderungen an moderne Computersysteme bezüglich der Entwicklungszeit und der Qualität machen eine prinzipiell neue Vorgehensweise bei der Softwareentwicklung erforderlich. Moderne Software:

- beinhaltet zahlreiche unterschiedlich komplexe Funktionen, die in Teilfunktionen zerlegt und jeweils eigenständig bearbeitet werden können;
- hat eine flexible klare leicht anpassbare Architektur und funktioniert in einer heterogenen verteilten Umgebung, z.B. im Internet;
- ist mühelos erweiterbar und kann das vorhandene Wissen in Form von Komponenten aufnehmen, wobei deren Austauschbarkeit und Wiederverwendbarkeit gewährleistet sind.

Diese Erwartungen haben Multiagentensysteme ins Leben gerufen. Die Multiagentenmetapher ist die Grundlage zur Entwicklung von komplexen Systemen auf Basis von unabhängigen autonomen intelligenten Subsystemen, die miteinander handeln und Ergebnisse gemeinsam erzielen.

Multiagentensysteme haben unterschiedliche Ausprägungen und Problemstellungen. Die Grundlage für fast alle Themenbereiche in Multiagentensystemen, beispielsweise für die Planung und Planausführung, ist eine Wissensrepräsentation bzw. Planrepräsentation, die sowohl statische, als auch dynamische Aspekte mit ausreichender Genauigkeit beschreibt.

Im Rahmen dieser Diplomarbeit wurde eine Planrepräsentation entwickelt, die die Planungs- und die Planausführungsvorgänge für kooperative Multiagentensysteme beschreibt. Dabei waren folgende Fragen geordnet nach absteigender Wichtigkeit von besonderer Bedeutung:

- Ist es möglich, eine Planrepräsentation zu finden, die sowohl für die Planung, als auch für die Planausführung in einem Multiagentensystem geeignet ist, das in einer dynamischen Umwelt funktioniert? Ist die ausgewählte Planrepräsentation flexibel und erweiterbar? Wie wird ein solcher Plan ausgeführt?

- Kann das entwickelte Multiagentensystem in einer heterogenen verteilten Umgebung funktionieren? Ist das entwickelte Multiagentensystem erweiterbar? Sind einzelne Komponenten austauschbar? Ist die Wiederverwendbarkeit von Komponenten gewährleistet?

1.1 Aufbau der Arbeit

In Kapitel 2 werden die wichtigsten Begriffe der verteilten KI, insbesondere die der Multiagentensysteme aufgezeigt. Es werden Themenbereiche von Multiagentensystemen besprochen, die das Gerüst dieser Ausarbeitung bilden. In Kapitel 3 wird auf die Wissensrepräsentation u.a. auf die Planrepräsentation genauer eingegangen. In Kapitel 4 werden die Ansätze zur Planung daraufhin untersucht, wie sie zur Planrepräsentation beitragen. Kapitel 5 untersucht die Planausführung, deren Probleme und Ansätze. Einen weiteren Themenbereich in Multiagentensystemen bilden die Kommunikation, die Verteilung und die Kooperation. Diese Gebiete stehen im Mittelpunkt in Kapitel 6.

Nachdem die wichtigsten Themenbereiche und deren Ansätze veranschaulicht sind, wird in Kapitel 7 ausführlich auf die Lösung eingegangen. Es wird zu jedem Themenbereich der ausgewählte Ansatz genannt. Es werden sowie die Planrepräsentation für ein Szenario zur Übergabe eines Gegenstandes aufgezeigt, als auch das Multiagentensystem und dessen Module dargestellt, die dieses Szenario implementieren.

Die Ergebnisse dieser Diplomarbeit und das Fazit folgen in Kapitel 8. Die Ausarbeitung schließt mit einem Anhang in Kapitel 9.

2 Einführung

Im Weiteren werden die wichtigsten Begriffe und Themenbereiche in der verteilten KI aufgezeigt.

2.1 Einführung in die VKI

Die verteilte Künstliche Intelligenz (VKI) ist ein Teilgebiet der KI, das sich mit Problemen von verteilten, interagierenden Systemen beschäftigt. Es wird dabei versucht, eine Lösung für Probleme der Entwicklung von Organisationsstrukturen, der Problemlösungsstrategien, sowie der Koordinations- und Kooperationsmechanismen zu finden.

Die Teilgebiete der verteilten KI sind in Abbildung 1 aufgezeigt (s. auch [Brenner98]).

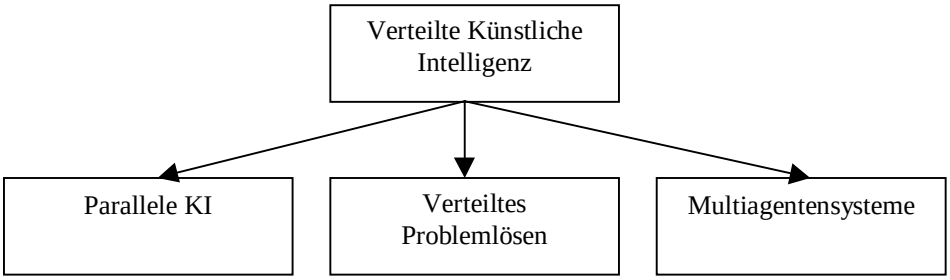


Abbildung 1. Teilgebiete der verteilten KI.

Die parallele KI untersucht, inwieweit das Problemlösen beschleunigt wird, wenn Systeme in mehrere Untersysteme unterteilt werden und parallel funktionieren. Der Nachteil ist, daß auf die Kooperations- und Koordinationsaufgaben keine große Rolle zukommt und das System nicht die aggregierte Kraft der Untersysteme erreichen kann.

Im Rahmen des verteilten Problemlösens wird ein komplexes Problem in mehrere Teile zerlegt, wobei jedes Teilproblem von einem Untersystem bearbeitet wird. Als Nachteile dieses Verfahrens sind folgende zu nennen:

- die an sich komplizierten Algorithmen, die dieses Konzept realisieren, beruhen darauf, daß ein System von einem Entwickler konzipiert und entwickelt wurde. Das ist nicht gegeben, wenn Systemkomponenten von mehreren Entwicklern entworfen sind;
- das verteilte Problemlösen erfordert eine zu Beginn der Entwicklung fest definierte Aufgabenstellung.

Als Lösung der Probleme der parallelen KI und des verteilten Problemlösens sind Multiagentensysteme entstanden[1]. Es ist nicht möglich eine einheitliche Definition zu geben, da die Diskussion um diesen Begriff noch nicht abgeschlossen ist. Darum werden im Weiteren zwei unterschiedliche Definitionen aufgezeigt.

[1] Eine Einführung in Multiagentensysteme ist in [Burkhard00] nachzuschlagen.

[Ferber99] definiert ein Multiagentensystem als ein Tupel:

M = {E, O, A, R, Op}, wo

- E – die Systemumwelt;
- O – eine Menge von Objekten O, die in der Umwelt eine Position einnehmen;
- A – eine Menge von Agenten A, die eine Untermenge von Menge O bilden. Das sind die aktiven Objekte im System;
- R – eine Menge von Relationen R, die Abhängigkeiten zwischen Objekten O definiert.
- Op – eine Menge von Operationen Op, die den Agenten A Handlungen mit Objekten aus Menge O ermöglicht.

[Haddadi96] gibt eine ausführlichere Definition. Dementsprechend ist ein Multiagentensystem ein Tupel:

M = {W, T, U, MT, MSG, R, Act, Agt, Π, B, D, I, C, Φ}, wo

W – eine Menge von Welten ist, wobei eine Welt durch ein Tupel mit Zeitpunkten, deren Relationen, Aktionen und dem Erfolg oder Mißerfolg einer Aktion, repräsentiert wird.
T – eine nicht leere Menge von Zeitpunkten;
U – gibt die Domäne an (Objekte, Pläne etc.).
MT – eine Menge von Nachrichtentypen;
MSG – eine Menge von Nachrichten;
R – eine binäre Relation zwischen benachbarten Zeitpunkten;
Act – eine elementare Aktion, die auch in Relation R abgebildet ist;
Agt – gibt jede Aktion der ausführenden Agenten an.
Π – gibt die Planbibliothek eines jeden Agenten an.
B – eine Relation, die die aktuelle Konstellation auf eine Menge von Agentenüberzeugungen entsprechenden Welten abbildet;
D – eine Relation, die die aktuelle Konstellation auf eine Menge von den Agentenwünschen entsprechenden Welten abbildet;
I – eine Relation, die die aktuelle Konstellation auf eine Menge von den Agentenintentionen entsprechenden Welten abbildet;
C – gibt eine Interpretation von Konstanten an, z.B. ihre Werte;
Φ – gibt eine Interpretation eines Prädikats in einer Situation an.

Aus den beiden Definitionen wird ersichtlich, daß die Stärke eines Multiagentensystems sich primär aus Fähigkeiten von einzelnen Agenten zusammensetzt. Agenten implementieren in einem Multiagentensystem spezifische Dienste, die für andere Systemkomponenten zugänglich sind und können von ihnen benutzt werden. Agenten führen Aufgaben aus, die ihnen von anderen Agenten vergeben werden können. Es wird i.A. kein System konzipiert, das eine komplexe Hauptaufgabe löst, wobei ein Verwaltungsmodul die Teilaufgaben zuweist, sondern Untersysteme (Agenten), die an der Ausführung der Hauptaufgabe dadurch teilnehmen, daß sie Teilaufgaben übernehmen. Die Fähigkeit zur Lösung von Teilaufgaben wird vor Beginn der Ausführung der Aufgabe im Multiagentensystem bekanntgegeben.

Hiermit gestaltet sich ein Multiagentensystem als ein dynamisches System, dessen Fähigkeiten aus den Fähigkeiten der einzelnen Agenten resultieren. Die Agenten in einem Multiagentensystem sind entweder Menschen oder Softwarekomponenten oder technische Geräte, die in einer Arbeitsumgebung zur Ausführung bestimmter Aufgaben eingesetzt werden.

Genauso wie bei der Definition von Multiagentensystemen gibt es keine einheitliche Definition des Agentenbegriffs. [Brenner98] bezeichnet einen intelligenten Agenten als ein System:

- das zur Ausführung seiner Aufgaben einen Wissensgrad besitzt und dieses Wissen aktiv einsetzt;
- das eine Schlußfolgerungs- und Lernfähigkeit besitzt.

[Ferber99] definiert einen intelligenten Agenten als eine physikalische oder virtuelle Entität:

- die in der Lage ist, in seiner Umwelt zu handeln;
- die mit anderen Agenten kommunizieren kann;
- die von seinen Zielen getrieben wird;
- die eigene Ressourcen besitzt;
- die in der Lage ist, Informationen aus seiner Umwelt wahrzunehmen;
- die eine Repräsentation seiner Umwelt hat;
- die Fähigkeiten besitzt und sie anderen Agenten zur Verfügung stellt;
- die sich reproduzieren kann.

Zusammengefaßt ist ein intelligenter Agent ein System, das unter Berücksichtigung sowohl von vorhandenen Ressourcen und Fähigkeiten, als auch Informationen aus der Umwelt und von anderen Agenten seine Ziele verfolgt.

Die Taxonomie intelligenter Agenten ist in Abbildung 2 dargestellt [Brenner98]. Die Unterteilung in Software-Agenten enthält wegen Einfachheit ausschließlich die wichtigsten Agententypen und ist nicht vollständig. Sie kann ggf. um weitere Agententypen ergänzt werden.

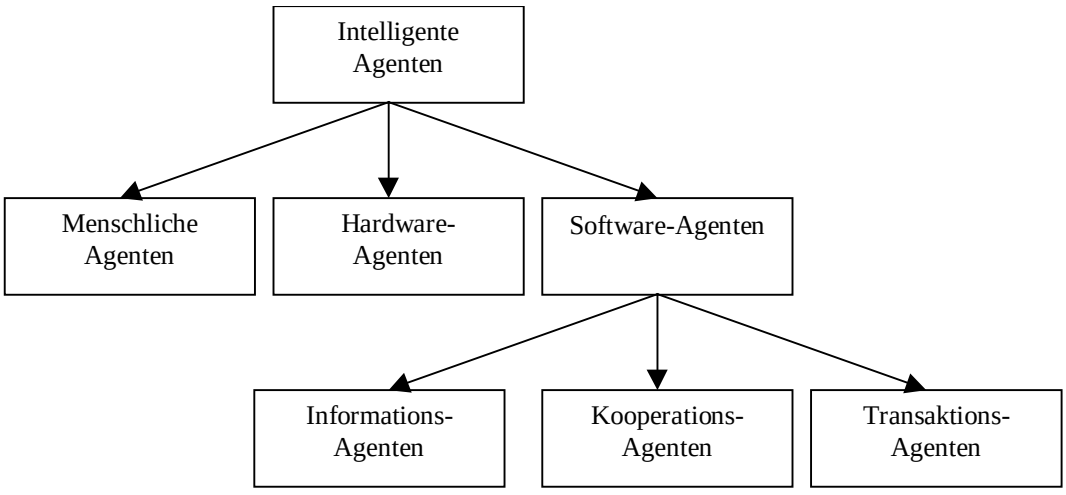


Abbildung 2. Taxonomie intelligenter Agenten.

Der Sinn der Begriffe „Menschliche Agenten“, „Hardware-Agenten“ und „Software-Agenten“ wird am besten durch Beispiele aufgezeigt. Ein Sachbearbeiter könnte als ein menschlicher Agent bezeichnet werden, der im Rahmen eines Betriebes seine Aufgaben erledigt. Ein Hardware-Agent ist z.B. ein Roboter, während ein Software-Agent beispielsweise ein Programm ist.

Auf der nächsten Hierarchieebene stehen Informations-, Kooperations- und Transaktionsagenten.

Ein Informationsagent ist ein Agent zur Unterstützung der Informationsbeschaffung. Ein solcher Agent kann Informationsquellen aufspüren, Informationen extrahieren und sie analysieren.

Die Kooperationsagenten führen ihre Aufgaben aus, indem sie mit den anderen Agenten im System kommunizieren und kooperieren. So können die Kooperationsagenten durch die Zusammenarbeit ihre eigenen Fähigkeiten erweitern. Abhängig davon, wie die Agenten in einem System miteinander agieren, werden kooperative und nicht kooperative Agenten unterschieden. Bei den kooperativen Agenten handelt es sich um Komponenten, die zur Erreichung eines gemeinsamen Ziels in einem System miteinander kooperieren, um für das ganze System bessere Ergebnisse zu erzielen. Die nicht kooperativen Agenten sind dagegen die Komponenten, die in einem Multiagentensystem miteinander handeln, um bessere Ergebnisse für sich selbst zu erzielen. Die Kooperation ermöglicht eine Steigerung der Effizienz durch eine nebenläufige Bearbeitung der Benutzeranfragen.

Die Transaktionsagenten lösen Aufgaben aus dem Datenbankbereich. Ihre Aufgabe ist die Verwaltung, die Beobachtung und die Ausführung von Transaktionen.

Das Multiagentensystem funktioniert in einer Umwelt, deren Eigenschaften die Anforderungen an das System drastisch beeinflussen können. Dementsprechend werden folgende Charakteristika der Umwelt unterschieden ([Burkhard00]):

- Abgeschlossenheit der Umwelt;
- Vollständigkeit/Zugänglichkeit der sensorischen Informationen;
- Zuverlässigkeit der sensorischen Informationen;
- Determiniertheit der Umwelt;
- Episodische Umwelt;
- Zuverlässigkeit bzgl. des Resultats von Aktionen (Präzision, Erfolg);
- Zeitliche Anforderungen an Entscheidungen (Echtzeit);
- Dynamik der Umwelt;
- Granularität, Skalierungen, Wertebereiche;
- Sicherheitsaspekte.

Die Charakteristika intelligenter Agenten können in zwei Kategorien aufgeteilt werden: interne und externe Eigenschaften (Abbildung 3).

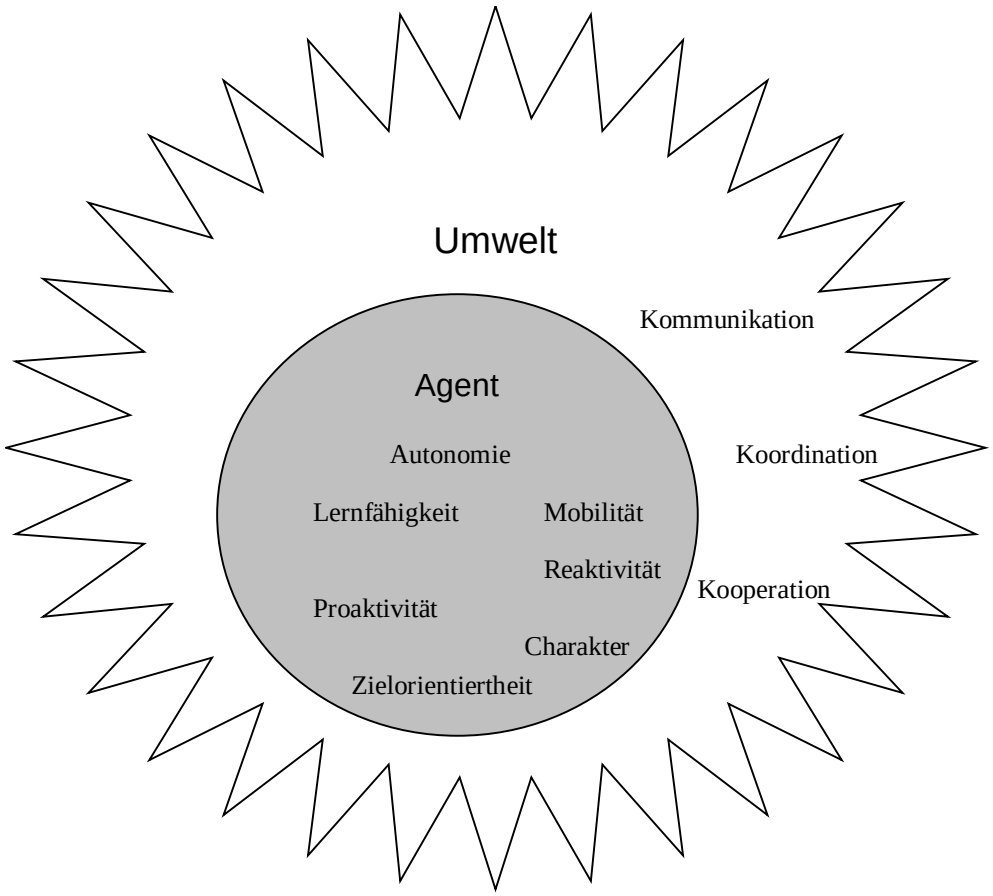


Abbildung 3. Charakteristika intelligenter Agenten.

Im Weiteren wird auf die Charakteristika intelligenter Agenten näher eingegangen.

Autonomes Verhalten ist die Fähigkeit eines Agenten, seine Ziele selbständig zu erreichen. Ein Agent ist in der Lage, seine Aufgaben selbst auszuführen, ohne die Arbeit anderer Agenten im System durch die Abstimmung zu verhindern. Aus der Sicht der Benutzer nimmt dadurch die Intelligenz des Agenten zu.

Als Reaktivität bezeichnet man die Fähigkeit eines Agenten in einer angemessenen Art und Weise auf sich schnell ändernde Konditionen in der Umwelt zu reagieren. Um dieser Anforderung nachgehen zu können, muß ein Agent entweder über Sensoren verfügen, um seine Umwelt wahrnehmen zu können, oder eine interne Repräsentation verwalten, aus der er benötigte Informationen erhalten kann. Von daher unterscheidet man reaktive (ohne Weltmodell), deliberative (mit einem Weltmodell) und hybride Agenten.

Proaktivität/Zielorierntiertheit sind die Fähigkeiten eines Agenten zum aktiven Eingreifen in die Umwelt, um gewisse Ziele zu erreichen. Ein Agent ist nicht immer ein passives Objekt, das in seiner Umwelt „existiert“ und sie abhört. Er kann auch seine Ziele verfolgen und aktiv sein.

Schlußfolgerungsfähigkeit ist die Fähigkeit eines Agenten, aufgrund seines Wissens und der Umweltbeobachtungen logische Schlüsse zu ziehen. Die Schlußfolgerungsfähigkeit ist ausschlaggebend für die Fähigkeit des Agenten, in seiner Umwelt intelligent zu handeln.

Die Lernfähigkeit ermöglicht die Änderung der Wissensbasis eines Agenten aufgrund von Schlußfolgerungen, Umweltbeobachtungen oder Benutzereingaben.

Die Kommunikation ist ein grundlegendes Charakteristikum eines Agenten. Sie ist die Grundlage für sämtlichen Informationsverkehr im Multiagentensystem. Um die Aufgaben des Gesamtsystems ausführen zu können, kommunizieren die Agenten im System miteinander.

Die Kooperation ist eine weitere Fähigkeit eines Agenten, die besagt, wie eine eventuelle Zusammenarbeit im System abläuft. Wie oben bereits erwähnt, kann die Gesamteffizienz des Systems gesteigert werden, indem die Agenten miteinander kooperieren.

Bei einer komplizierten Kooperationsart spricht man nicht nur von kommunikativen bzw. kooperativen Agenten, sondern allgemein über den Charakter des Agenten. Der Charakter ist eine Agenteneigenschaft in Bezug auf sein Verhalten gegenüber den anderen Agenten im System. Der Charakter bewirkt, daß ein Agent nur bedingt an der Durchführung der Ziele im System teilnimmt, wenn seine privaten Teilziele dadurch behindert werden.

Die Bereiche, die in Bezug auf die Multiagentensysteme erforscht werden, sind in Abbildung 4 aufgezeigt (s. auch [Brenner98]).

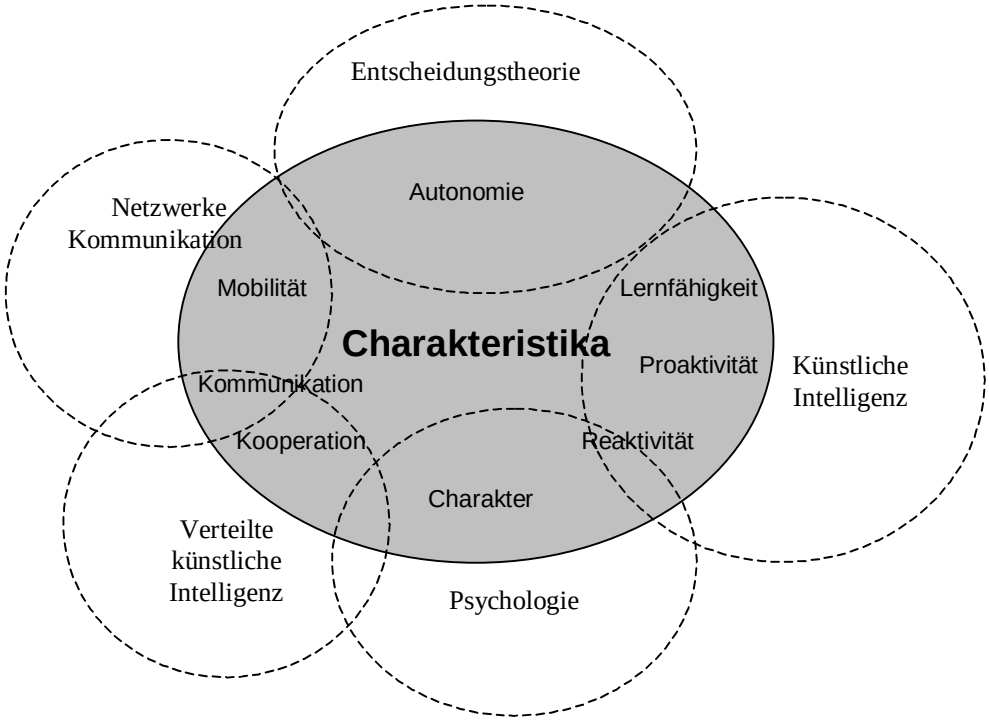


Abbildung 4. Einflußgebiete.

Abbildung 4 verdeutlicht, daß in der verteilten KI zwei Bereiche eine besonders wichtige Rolle spielen – Kommunikation und Kooperation, in der KI – Lernfähigkeit, Reaktivität und Proaktivität.

In den weiteren Abschnitten wird auf die Themenbereiche in Multiagentensystemen näher eingegangen.

2.2 Themenbereiche in Multiagentensystemen

Um Anforderungen an das zu entwickelnde Multiagentensystem, insbesondere an die Planrepräsentation formulieren zu können, werden im Weiteren die wichtigsten Begriffe definiert und erläutert und Themenbereiche in Multiagentensystemen aufgezeigt.

Die bei der Entwicklung eines Multiagentensystems wichtigsten Themenbereiche sind folgende (vgl. [Kühnel01]):

- Wissensrepräsentation (Welt- und Planrepräsentation);
- Planung;
- Planausführung;
- Kommunikation, Verteilung und Kooperation;
- Management;
- Wissensaktualisierung (Lernen);
- Sicherheit.

Zu beachten ist, daß Themenbereiche „Wissensrepräsentation“, „Planung“ und „Wissensaktualisierung“ die Bereiche sind, für die die Aktivität sowohl für einen einzigen Agenten, als auch für das gesamte Multiagentensystem möglich ist. Die Themenbereiche „Kommunikation, Verteilung und Kooperation“ und „Sicherheit“ werden primär für das ganze System dargestellt. Wenn eine Unterscheidung zwischen einem Einzelagenten und dem Gesamtsystem wichtig ist, wird im Weiteren darauf ausdrücklich hingewiesen.

In den folgenden Abschnitten wird auf die angegebenen Themenbereiche kurz eingegangen.

Die wichtigste Aufgabe beim Entwurf eines Multiagentensystems ist die Ausarbeitung einer Wissensrepräsentation. Die Wissensrepräsentation bietet eine Darstellung der Welt, in der das Multiagentensystem funktioniert, und auch der Aktionen, die ausgeführt werden. Die Wissensrepräsentation beeinflußt die Effizienz und die Flexibilität einzelner Agenten und folglich die Effizienz und die Flexibilität des gesamten Multiagentensystems. Näher auf die Wissensrepräsentation in einem Multiagentensystem wird in Kapitel 3 eingegangen.

Damit ein Multiagentensystem eine Aufgabe lösen kann, ist ein Modul erforderlich, das die Planung von notwendigen Aktionen durchführt. Andersherum ist das Ziel der Planung die Generierung einer Reihenfolge der Aktionen in einem Multiagentensystem, die zur Erreichung eines Ziels führt. Ferner auf die Planung wird in Kapitel 4 eingegangen.

Die Planausführung ermöglicht es, die von einem Planungsmodul erzeugten Pläne in wirkliche Taten umzusetzen. Näher auf die Planausführung wird in Kapitel 5 eingegangen.

Die Fähigkeiten eines Multiagentensystems setzen sich zusammen aus den Fähigkeiten seiner einzelnen Agenten. Die Agenten kommunizieren und kooperieren miteinander und verteilen die anstehenden Aufgaben. Ferner auf die Kommunikation, Verteilung und Kooperation wird in Kapitel 6 eingegangen.

Ein Managementmodul verwaltet das Multiagentensystem – es startet und beendet die Agenten und alloziert notwendige Ressourcen.

Damit ein Agent sein neues Wissen, das er im Laufe der Arbeit im Multiagentensystem gesammelt hat, nochmals einsetzen kann, ohne die gleichen Berechnungen durchführen zu

müssen, benötigt er Wissensaktualisierungs- bzw. Lernfähigkeit. Dabei werden Schlußfolgerungen für die weitere Anwendung vermerkt.

Die Sicherheit in einem Multiagentensystem wird in Bezug auf die Ausführung und die Interaktion definiert. Insbesondere sind folgende Aspekte zu beachten:

- ein Agent muß in der Lage sein, die Identität eines anderen Agenten und die Echtheit der Quelle von Nachrichten festzustellen;
- der Datenaustausch sollte verschlüsselt und Datenverfälschungen sollten bemerkbar sein;
- ein Agent sollte seine Fähigkeiten in Abhängigkeit von vergebenen Benutzungsrechten für andere Agenten einsetzen;
- ein Auftrag, der an einen anderen Agenten vergeben wurde, sollte von diesem Agenten gemäß der Vereinbarung rechtzeitig erledigt werden und das erwartete Ergebnis haben, während der andere Agent die ihm gelieferten Daten nicht anderweitig verwenden sollte;
- 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.

Da die Themenbereiche „Wissensaktualisierung“, „Sicherheit“ und „Management“ in dieser Abhandlung von einer eingeschränkten Bedeutung sind, werden sie im Weiteren nicht näher betrachtet. Eine Sammlung von Informationen über das Lernen in Multiagentensystemen findet sich in [Prasad01]. Über die Sicherheit in Multiagentensystemen ist in [Wagner97] nachzulesen. Informationen über das Management in Multiagentensystemen finden sich in [Kühnel01].

Auf die Themenbereiche „Wissensrepräsentation“, „Planung“, „Planausführung“ und „Kommunikation, Verteilung und Kooperation“ wird in den nachstehenden Kapiteln näher eingegangen.

2.3 Funktionelle Architektur der Multiagentensysteme

Da ein Plan oder seine Teile für das ganze Multiagentensystem erstellt werden, ist es erforderlich aufzuzeigen, welche Architektur diesem System zugrunde liegen kann. Im Weiteren wird die hypothetische Architektur eines Multiagentensystems dargestellt, die in Abschnitt 2.2 aufgezeigten Themenbereiche realisiert (s. Abbildung 5 – in Anlehnung an [Brenner98] und [Lüth98]). Obwohl in [Brenner98] die Architektur am Beispiel eines Einzelagenten dargestellt ist, läßt sie sich auf ein Multiagentensystem erweitern.

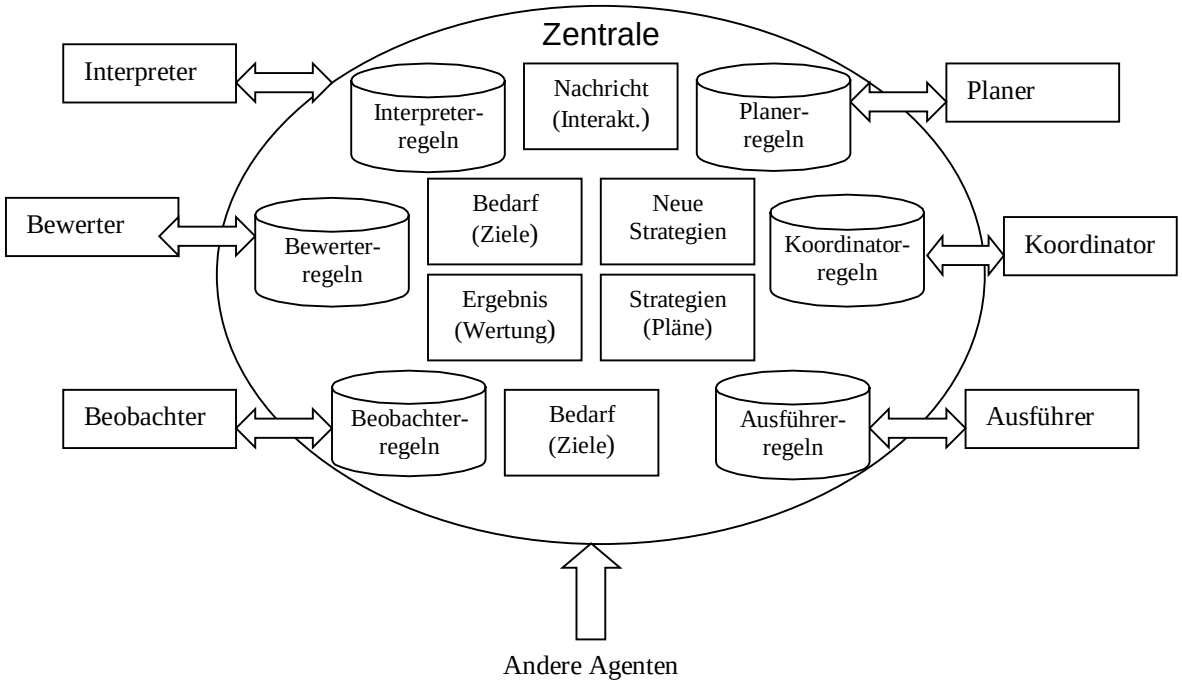


Abbildung 5. Architektur eines Multiagentensystems.

In der in Abbildung 5 dargestellten Architektur sind die wichtigsten Komponenten eines Multiagentensystems definiert – der Planer, der Bewerter, der Koordinator, der Beobachter, der Ausführer und die Zentrale des Multiagentensystems. Im Weiteren wird auf jede Komponente kurz eingegangen.

Der *Beobachter* ist eine Komponente im Multiagentensystem, welche die eingehenden Informationen z.B. Sensorinformationen von einem Hardware-Agenten erfaßt und sie als Prozeßzustände modelliert.

Der *Bewerter* schätzt permanent die Güte der vom Agenten eingesetzten Pläne ab. Er prüft auch die Qualität anderer Strategien, die eventuell zum Einsatz kommen können.

Der *Interpreter* ist eine Komponente im Multiagentensystem, die Prozeßzustände, Nachrichten und Bewertungen analysiert und eine Anpassung der aktiven oder den Einsatz der neuen Strategien ausarbeitet.

Der *Planer* wählt oder generiert für den aktuellen Bedarf eine Strategie für den Ausführer, mit der die aktuellen Ziele erreicht werden sollen.

Der *Koordinator* ermöglicht eine nebenläufige, sequentielle oder alternative Ausführung für die aktiven und neuen Strategien.

Die ausführende Komponente verwaltet die Ausführung der im Plan definierten Handlungen.

Wegen Einfachheit wird in Abbildung 5 eine Architektur mit einer Zentrale aufgezeigt, die im Gesamtsystem die Kommunikationsaufgabe ausführt. Die Kommunikation zwischen den einzelnen Agenten erfolgt daher implizit durch die Zentrale und nicht direkt.

Bezüglich der Verteilung in der hypothetischen Architektur aus Abbildung 5 kann folgendes angemerkt werden: Jede Komponente ist als ein Interface zu verstehen, das spezifische Dienste implementiert und im Multiagentensystem durch einen Agenten oder ein Multiagentensystem repräsentiert wird. Es ist z.B. möglich, daß der Interpreter, der Planer, der Bewerter, der Koordinator, der Beobachter und der Ausführer unabhängige Agenten sind, wobei die Wissensbasis als ein Multiagentensystem mit der Funktionalität des schwarzen Bretts [Hayes-Roth85] implementiert ist. Möglich wäre auch, daß einige der skizzierten Komponenten zusammenschmelzen, z.B. daß physisch der Koordinator und der Ausführer zu einem Subsystem werden.

Ein mögliches Szenario für die Zusammenarbeit der Komponenten in diesem System kann wie folgt definiert sein (s. Abbildung 6): ein GUI-Agent startet eine Anfrage zur Ausführung eines Plans, die Anfrage wird vom Beobachter empfangen und an den Interpreter weitergeleitet. Der Interpreter entscheidet sich für den Einsatz einer neuen Strategie und schickt eine entsprechende Anfrage an den Planer, der eine Strategie, die für das zu erreichende Ziel nötig ist, zur Verfügung stellt. Anschließend übergibt der Planer den neuen Plan an den Koordinator, der diesen Plan als eine neue Strategie unter den laufenden Plänen zur sequentiellen oder nebenläufigen Ausführung empfängt und den Ausführer initialisiert. Der Ausführer führt den Plan aus und das Ergebnis dieser Ausführung geht an den GUI-Agenten über den Koordinator, den Interpreter, den Beobachter zurück und wird ausgegeben.

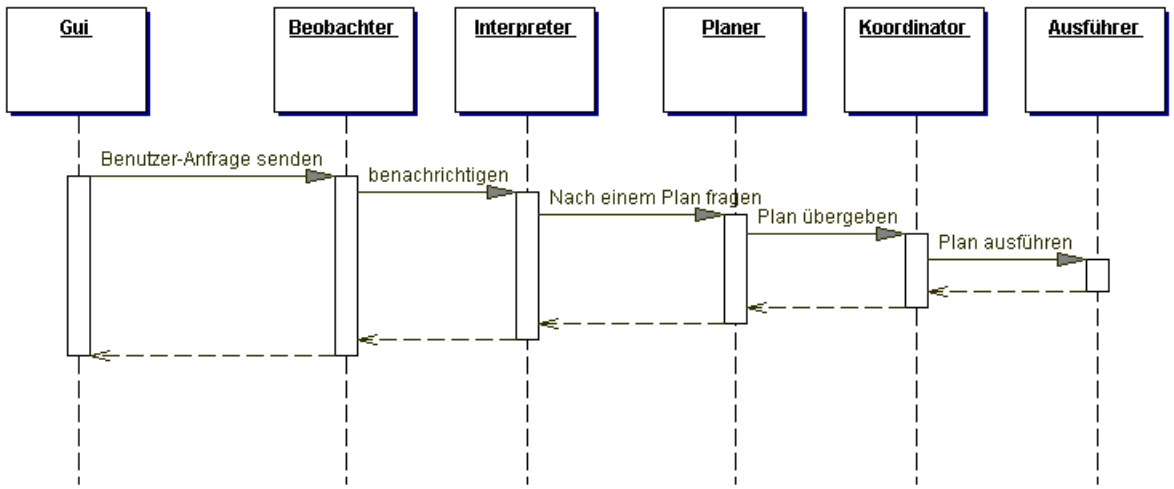


Abbildung 6. Ein mögliches Szenario für das Multiagentensystem.

In Bezug auf die in Abschnitt 2.2 geschilderten Themenbereiche kann folgende Aufgabenverteilung auf Komponenten eines Multiagentensystems vorgenommen werden:

- Die Planung wird vom Planer durchgeführt.
- Die Planausführung wird vom Koordinator und dem Ausführer fertiggemacht;
- Kommunikation wird von einem Modul innerhalb jedes Agenten durchgeführt. Hier wird dieses Modul nicht aufgezeigt;
- Verteilung und Kooperation werden vom Koordinator verwaltet;
- Das Management wird vom Koordinator und dem Ausführer durchgeführt.

Nachdem eine mögliche Architektur eines Multiagentensystems dargestellt wurde, wird im Weiteren auf die Themenbereiche „Wissensrepräsentation“, „Planung“, „Planausführung“ und „Kommunikation, Verteilung und Kooperation“ ausführlich eingegangen.

3 Wissensrepräsentation

Die Auswahl einer Wissensrepräsentation ist ein grundlegender Themenbereich in einem wissensbasierten System bzw. einem Multiagentensystem. Die Wissensrepräsentation beeinflußt die Effizienz und die Flexibilität einzelner Agenten und folglich die des gesamten Multiagentensystems (vgl. [Owsnicki-Klewe00]). Da im Rahmen dieser Ausarbeitung die Wissensrepräsentation von großer Bedeutung ist, wird in diesem Kapitel auf sie ausführlich eingegangen.

Mit Hilfe der Wissensrepräsentation kann das Systemverhalten analysiert werden. Sie bietet eine Darstellung der Welt, in der das Multiagentensystem funktioniert, und repräsentiert die Aktionen, die von ihm ausgeführt werden. Ferner legt die Wissensrepräsentation fest, wie die Gesamtziele im System erreicht werden, welche Fähigkeiten die einzelnen Agenten im System besitzen und wie sie modelliert werden, wie Ereignisse im System dargestellt werden, wie und wann die Agenten auf sie reagieren und wie Nachrichten im System modelliert und ausgetauscht werden.

Prinzipiell können Multiagentensysteme ohne eine Wissensrepräsentation ausschließlich anhand des Programmcodes von Agenten analysiert werden. Eine explizite Wissensrepräsentation ist jedoch aus folgenden Gründen vorteilhafter ([Ferber99]):

1. Es ist extrem aufwendig, nur anhand des Programmcodes die Agentenverhalten nachzuvollziehen und zu kontrollieren, ohne genaue Daten darüber explizit zu halten;
2. Der Programmcode einzelner Agenten enthält viele Details, die nicht zum Verständnis des Agentenverhaltens beitragen;
3. Verschiedene Implementierungen stellen dasselbe Verhalten dar, wobei kleine Codeänderungen große Verhaltensunterschiede bewirken können;
4. Bei der nebenläufigen Ausführung mehrerer Agenten wird ihre Funktionsweise besonders schwer nachzuvollziehbar;
5. Multiagentensysteme sind kompliziert im Entwurf. Ein Weltmodell erlaubt das separate Design einzelner Agenten, ohne den Überblick über das gesamte System zu verlieren.

Um das Wissen zu repräsentieren, wird eine domänen- und aufgabenabhängige Notation gefordert. Von dieser Notation werden folgende Eigenschaften verlangt:

- Kommunikativität;
- Ausdrucksfähigkeit;
- Operationalisierbarkeit.

Die Kommunikativität setzt voraus, daß die dargebotene Notation eindeutig ist. Diese Eigenschaft ist die Grundlage für eine sachliche Diskussion des Weltmodells in Arbeitsgruppen. Das Weltmodell ist unbrauchbar, wenn es Mehrdeutigkeit zuläßt.

Die Ausdrucksfähigkeit ist die Eigenschaft einer Notation, die beobachteten oder angenommenen Phänomene in der Welt genau darstellen zu können. Dazu muß die Notation die Mittel zur Verfügung stellen, die die Weltdomäne beschreiben können.

Die Operationalisierbarkeit besagt, wie praxisnah die ausgewählte Notation ist, d.h. wie das dargestellte Weltmodell in ein lauffähiges System umgewandelt wird, das die vorgegebenen

Aufgaben löst. Das Weltmodell ist unbrauchbar, wenn die ausgesuchte Notation nicht operationalisierbar ist.

Es können grundsätzliche Anforderungen an die Wissensrepräsentation wie folgt formuliert werden ([Morgenstern96]):

1. Ein Gleichgewicht zwischen Generalisierung und Spezialisierung;
2. Selbsterklärbarkeit;
3. Wahrheit und Wichtigkeit;
4. Konkretheit;
5. Möglichkeit der praktischen Umsetzung;
6. Theoriebasiertheit;
7. Einfachheit.

Die Modellierung stellt sich als eine anspruchsvolle Aufgabe dar. Folgendes muß festgelegt werden ([Ferber99]):

- welche Charakteristika im Multiagentensystem wichtig sind;
- welche Parameter und Werte im System modelliert werden müssen;
- was genau beschrieben wird;
- welche Handlungen ausgeführt werden und wie das erfolgt.

Zur Wissensrepräsentation in einem Multiagentensystem gibt es folgende Ansätze:

1. Das Multiagentensystem wird durch Handlungen und deren Auswirkungen modelliert;
2. Das Multiagentensystem wird durch sowohl interne, als auch externe Zustände erfaßt;
3. Das Multiagentensystem wird durch Interaktion und deren Modi modelliert, was auf Kooperation, Koordination von Aktionen und Auflösung von Konflikten hinausläuft;
4. Das Multiagentensystem wird als Ganzes auf der Basis der Experimentierung und Formalisierung modelliert.

Diese Modellierungsmethoden werden an den verschiedenen Planungsansätzen in Abschnitt 4.4 ausführlich aufgezeigt.

Wie oben bereits erwähnt, ist die Auswahl einer Wissensrepräsentation nicht nur für einen einzigen Agenten möglich, sondern für das gesamte Multiagentensystem (s. Abschnitt 2.2). Im Rahmen dieser Diplomarbeit ist zusätzlich eine Wissensrepräsentation erforderlich, die einen Plan für ein Multiagentensystem mit Hardware- bzw. Softwareagenten darstellt.

Die Wissensrepräsentation ist ein Sammelbegriff für die Welt- und Planrepräsentation. Die Weltrepräsentation bietet eine statische Darstellung der Welt, in der das Multiagentensystem funktioniert, und beschreibt diejenigen Fakten über die Welt, welche im System von Bedeutung sind. Die Planrepräsentation konzentriert sich dagegen auf dynamische Aspekte. Sie bietet eine Darstellung von Aktionen in Multiagentensystemen und zeigt wie diese Aktionen die Umwelt verändern.

Im Weiteren wird auf die Welt- und Planrepräsentation näher eingegangen.

3.1 Weltrepräsentation

Die Agenten im Multiagentensystem funktionieren in einer physischen oder virtuellen Umwelt. Oft stellt sich dabei die Frage nach Vorhersage oder Erklärung von bestimmten Phänomenen (s. Abbildung 7).

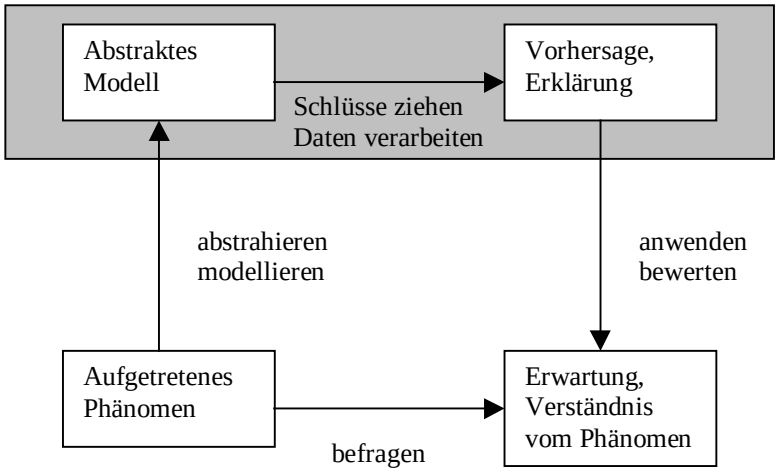


Abbildung 7. Benutzung des Weltmodells.

Ein aufgetretenes Phänomen wird modelliert, indem ausschließlich die wichtigsten Informationen für den Wissensbereich im Weltmodell erfaßt und verarbeitet werden. Danach wird eine Erklärung erzeugt, wie dieses Phänomen zustande kommen konnte. Diese kann im Weiteren für eine Vorhersage benutzt werden.

Eine Welt wird durch Zustände beschrieben. Ein Zustand (eine Situation) in einem Multiagentensystem ist eine Menge von Attributen oder Eigenschaften, die in einer Umgebung zu einem bestimmten Zeitpunkt als meßbar oder feststellbar angenommen werden ([Lüth98]).

Ein Weltmodell kann auf folgende Arten dargestellt werden:

1. Logisch;
2. Symbolisch;
3. Analogisch.

Die logische Darstellung stellt mithilfe von logischen Formeln (Prädikatenlogik) die Welt dar.

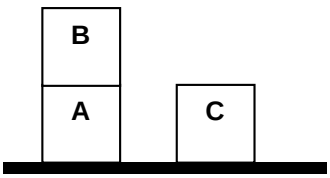


Abbildung 8. Die Klötzchenwelt.

Die in Abbildung 8 dargestellte Situation in der Klötzchenwelt wird beschrieben:

```
on(B,A)
on(A,Tisch)
on(C,Tisch)
```

Die symbolische Weltdarstellung benutzt einige Symbole, um den Weltzustand zu beschreiben (als ein Array). Für das in Abbildung 8 aufgezeigte Beispiel:

```
< <B, A>, <C> >
```

Die analogische Weltdarstellung ist z.B. eine Momentaufnahme der Weltkonstellation als ein Bitmap-File.

3.2 Planrepräsentation

Die Weltrepräsentation liefert in jedem konkreten Moment ein statisches Bild über den Zustand eines Multiagentensystems. Zur Ausführung von Aufgaben und zur vorhergehenden Planung sind jedoch Informationen notwendig, die eine dynamische Modellierung erlauben.

Um veränderliche Informationen über die Systemhandlungen zu modellieren, ist eine Planrepräsentation unabdingbar. Sie legt fest, welche Aktionen, unter welchen Bedingungen gestartet werden, und welche Änderungen diese Aktionen hervorrufen, wobei eine Aktion im Multiagentensystem eine Handlung ist, die von einem oder mehreren Agenten im System ausgeführt wird.

Die Planrepräsentation bietet eine Darstellung von Aktionen im Multiagentensystem und wie diese Aktionen ausgeführt werden. Es gibt mehrere Möglichkeiten, eine Aktion zu modellieren ([Ferber99]):

1. Eine Aktion bewirkt eine Änderung des Zustandes des Gesamtsystems, z.B. die Aktion „verschieben“ in der Klötzchenwelt kann eine Änderung der Position mehrerer Gegenstände und hiermit des Zustandes des gesamten Multiagentensystems bewirken. Das ist die häufigste und am besten untersuchte Repräsentationsart (s. Abschnitt 4.4).
2. Eine Aktion wird als eine Handlung mit möglichen Reaktionen modelliert, so hat z.B. das Verschieben eines Gegenstandes eine mögliche Reaktion, daß der Gegenstand nicht mehr an der ursprünglichen Stelle bleibt. Für den Fall, daß das Ziel von einem Objekt besetzt ist, wird als eine mögliche Reaktion vorgegeben, den Gegenstand möglichst näher zum Ziel abzustellen.
3. Eine Aktion kann mithilfe von Ereignissen repräsentiert werden, z.B. als ein Petri-Netz.
4. Eine Aktion bewirkt eine Änderung des Zustandes eines *einzigen* Agenten, z.B. die Aktion „gehen“ bewirkt eine Änderung der Position eines *einzigen* Agenten.
5. Eine Aktion wird durch physische Veränderungen im geometrischen und physischen Weltmodell beschrieben, z.B. die Aktion „verschieben“ kann Koordinaten mehrerer Gegenstände verändern.
6. Eine Aktion wird als ein reaktives Kommando repräsentiert (s. Abschnitt 4.3).

Eine Zustandsänderung liegt vor, wenn infolge der Ausführung einer Handlung das System in einen anderen Zustand übergeht.

Ein Plan ist eine Aktionssequenz, die das System von einem Startzustand in einen Zielzustand überführt [Lüth98]. Die Aufgabe der Planung besteht darin, diese Aktionssequenz zu finden und sie für eine weitere Ausführung abzuspeichern.

Ein Aktionsnetz ist ein Plan, der als Graph dargestellt wird, in dem ein Knoten einem Zustand des Systems und eine Kante einer Zustandsänderung entspricht.

Abhängig davon, ob bereits während des Planungsvorgangs eine genaue Reihenfolge von Aktionen oder nur das abstrakte Plangerüst festgelegt wurde, werden totale bzw. partiell geordnete Pläne unterschieden. Ein total geordneter Plan enthält eine Folge der Aktionen, die sich nicht während der Ausführung ändern kann (s. Abschnitt 4.4.1). Ein partiell geordneter Plan dagegen definiert ausschließlich Zusammenhänge zwischen einzelnen Aktionen. Eine genaue Reihenfolge von Aktionen wird direkt bei der Planausführung bestimmt (s. Abschnitt 4.4.2).

Pläne können mit Hilfe einer temporalen Logik repräsentiert werden, z.B. der Zeitlogik von Allen (nach [Heinsohn99]). In diesem Fall wird vorgegeben, unter welchen jetzt oder früher geltenden Bedingungen welche jetzigen und zukünftigen Bedingungen gelten werden.

Abhängig von der Planstruktur werden unterschieden (vgl. [Lüth98]):

- sequentielle Pläne – Pläne ohne Alternativen, in denen eine Plansequenz keine Verzweigungen enthält;
- hierarchische Pläne – Pläne mit einer hierarchischen Struktur;
- Alternativpläne mit deterministischer, stochastischer oder chaotischer Verzweigung;
- iterative Pläne – Pläne mit Schleifen;
- rekursive Pläne;
- parallele bzw. nebenläufige Pläne.

Die Planrepräsentation kann in einer netzförmigen Struktur abgespeichert werden, in der die Fähigkeiten und ihre Beziehungen dargestellt sind. In dieser Datenbasis können auch Annahmen und Verpflichtungen einzelner Agenten vermerkt werden.

Es ist naheliegend, daß die Aktionen in einem Multiagentensystem oft als Dienste von Agenten realisiert werden. In diesem Fall wird ein Plan normalerweise als ein Graph dargestellt, in dem die Knoten Systemzustände und die Kanten Agentendienste sind (s. Kapitel 5).

Um einen Einblick darüber zu bekommen, welche Probleme bei der Planrepräsentation eine wichtige Rolle spielen, wird im Folgenden darauf näher eingegangen.

Es werden folgende Probleme unterschieden, die eng mit der Planrepräsentation verbunden sind ([Schneeberger00]):

- Das *Qualification* Problem stellt die Unmöglichkeit dar, korrekte Vorhersagen über die Fakten nach Ausführung einer Aktionssequenz zu machen, ohne alle vorhandenen Informationen über die Situation vor ihrer Ausführung berücksichtigen zu müssen;
- Das *Prediction* Problem ist die Unmöglichkeit vorherzusagen, welche Fakten nach Ausführung einer Aktionssequenz wahr sein werden;
- Das *Persistence* Problem ist die Unmöglichkeit vorherzusagen, welche Fakten durch Ausführung einer Aktionssequenz unverändert bleiben werden;

- Das *Frame* Problem ([McCarthy69]) ist das grundlegende Problem der Planrepräsentation in der KI. Es stellt die Unmöglichkeit dar, vorherzusagen, welche Fakten bei der Ausführung einer Aktion unverändert bleiben und welche Ausschnitte der realen Welt modelliert werden müssen. Das Frame Problem wirft folgende Fragen auf:

  1. Welche Aspekte der Modellierung eines Ausschnitts der realen Welt spielen eine wichtige und welche eine nebensächliche Rolle?
  2. Wie verändert eine Aktion die Umwelt und was muß dabei modelliert werden?

- Das *Ramification* Problem ist die Unmöglichkeit vorherzusagen, welche weiteren Fakten wahr werden, wenn durch Ausführung einer Aktion in einer Situation ein Faktum wahr wird.

Im Allgemeinen sind diese Probleme unentscheidbar ([Schneeberger00]). Bei der Wahl einer Planrepräsentation müssen jedoch diese Probleme berücksichtigt werden.

In diesem Kapitel wurde die Terminologie der Wissensrepräsentation, der Planrepräsentation und deren Probleme ausführlich aufgezeigt. Da sich ein ausgewählter Ansatz zur Planung auf die Wissensrepräsentation maßgebend auswirkt, werden in Kapitel 4 die Ansätze zur Planung aufgezeigt. Es werden u.a. die Möglichkeiten zur Lösung der obengenannten Planungsprobleme dargestellt.

4 Planung

Um seine Aufgaben auszuführen, nimmt das Multiagentensystem eine Planung vor. Im Rahmen dieser Ausarbeitung ist die Planung umso interessanter, weil das ausgewählte Planungsverfahren die Planrepräsentation stark beeinflussen kann.

Im Weiteren werden die wichtigsten Ansätze zur Planung dargestellt, die wichtigsten Begriffe definiert und erläutert. Es wird erläutert, welche Eigenschaften der Pläne im Zusammenhang mit der Planung eine wichtige Rolle spielen. Die Ansätze werden nicht vollständig aufgezeigt und dienen ausschließlich zum Verständnis der Eigenschaften von Plänen.

Die dargestellten Ansätze sind nicht auf Multiagentensysteme bezogen, können jedoch auf sie verallgemeinert werden. Ausführliche Informationen über die Planung finden sich in [Schneeberger00].

4.1 Einführung in die Planung

Die Planung bedeutet die Suche nach Plänen bzw. Aktionen, die gewisse Aufgaben im Multiagentensystem ausführen. In dem Fall, wenn mehrere Planalternativen vorhanden sind, schließt die Planung die Auswahl der besten Alternative ein.

Abhängig davon, auf welchen Informationen die Planung basiert, werden folgende Ansätze zur Planung unterschieden (s. die Reaktivität in Abschnitt 2.1):

- Ansatz zur reaktiven Planung;
- Ansatz zur deliberativen Planung;
- Hybrider Ansatz.

Der Ansatz zur reaktiven Planung benutzt Informationen aus der Systemumwelt, z.B. Sensorinformationen, um Handlungen zu planen, und hat kein Weltmodell. Ausführliche Informationen darüber sind in Abschnitt 4.3 nachzulesen.

Für einen deliberativen Planungsvorgang wird eine Weltdarstellung benötigt. Die für die Planung erforderlichen Informationen werden im Weltmodell gespeichert. Mehr dazu findet sich in Abschnitt 4.4.

Die deliberativen bzw. reaktiven Ansätze haben ihre Vor- und Nachteile (s. Abschnitt 4.3 und Abschnitt 4.4). Als Lösung ist der hybride Ansatz entstanden, der die deliberativen und die reaktiven Ansätze vereinigt (s. Abschnitt 4.5).

4.2 Taxonomie der Planung

In diesem Abschnitt wird auf die Taxonomie der Planung eingegangen. Die Taxonomie der Planung ist insofern wichtig, da sie zusätzliche Anforderungen an die Planrepräsentation und die Planung stellt.

Die Klassifizierung beruht darauf, daß jeder Agent als ein unabhängiges autonomes Modul einen Teil seiner Aufgaben an einen anderen Agenten im Multiagentensystem delegieren kann. In Bezug auf die Planung bedeutet dies, daß ein Planungsagent ein Teil des Planungsvorgangs an einen anderen Agenten übergibt, der über mehr Informationen darüber verfügt (Abbildung 9).

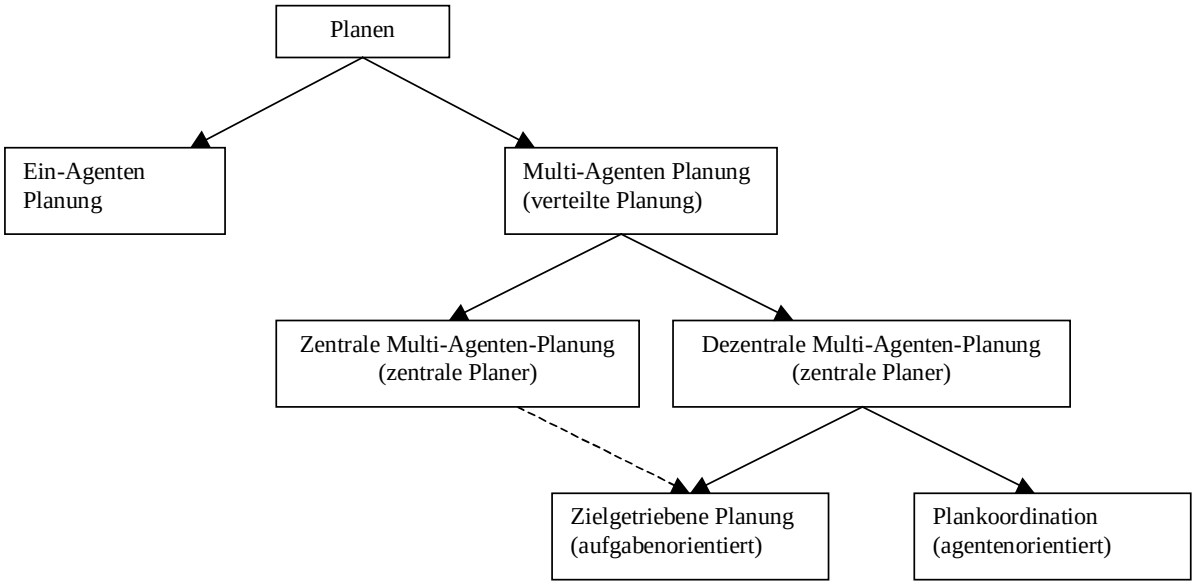


Abbildung 9. Planungstaxonomie.

Die Ein-Agenten-Planung ist der klassische Ansatz mit der Annahme, daß der Planungsagent genaue Informationen oder Annahmen über den Zustand des Systems bereits gesammelt hat und selbständig planen kann. Dieser Ansatz ist gut geeignet für Systeme, die in einer statischen Umgebung funktionieren, d.h. daß die Umwelt genau modellierbar ist und die Konditionen in dieser Umwelt ausschließlich durch Agentenaktionen verändert werden.

Für den Fall, daß ein Agent in einer dynamischen Umwelt funktioniert, ist es sinnvoll, den Ansatz zur Multi-Agenten-Planung in Kauf zu nehmen. Dabei arbeiten mehrere Agenten an der Planung der Gesamtaufgabe zusammen. Jeder Agent generiert für sich einen Teilplan, um eine Teilaufgabe zu realisieren, im einfachsten Fall einen Plan für eine eigene Fähigkeit, die zur Planung der Gesamtaufgabe beiträgt. Dieser Ansatz ist erforderlich, wenn die Planung dynamisch stattfindet und der planende Agent nicht alle Informationen zur Verfügung hat und die Teilaufgaben an besser informierte Agenten delegiert.

Bei der zentralen Multi-Agenten-Planung wird die Planungsaufgabe von einem Agenten ausgeführt, der für die Gesamtaufgabe einen abstrakten globalen Plan erstellt. Der globale Plan hat keine genauen Vorgaben und enthält lediglich die Hierarchie der Teilaufgaben, die für die weitere Planung an die besser informierten Agenten übergeben werden.

Bei der dezentralen Multi-Agenten-Planung wird auf einen globalen Plan verzichtet. Statt dessen wird der Planungsprozeß von mehreren Agenten durchgeführt. Die Planung auf diese Art wird dann sinnvoll, wenn die zentrale Multi-Agenten-Planung wegen zu hoher Kosten nicht anwendbar oder wenn der Aufbau einer globalen Wissensbasis gar nicht möglich ist.

Es werden außerdem zielgetriebene Planung, sowie Plankoordination unterschieden. Bei der zielgetriebenen Planung steht im Mittelpunkt des Planungsprozesses das Gesamtziel im

System, während bei der Plankoordination (oder der agentenorientierten Planung) die Suche und Beseitigung von Konflikten, sowie Optimierung der Ergebnisse ausschlaggebend ist.

Die dargestellten Varianten der Planung (Ein-Agenten- und Multi-Agenten-Planung) lassen sich in einem Multiagentensystem auf eine natürliche Art und Weise implementieren. Es werden im Weiteren entsprechende Ansätze aufgezeigt.

4.3 Ansatz zur reaktiven Planung

Die Ansätze zur reaktiven Planung beruhen ausschließlich auf der Perzeption von Agenten und haben kein Weltmodell. Die einzige Grundlage zur Planung und Planausführung sind die Fähigkeiten der einzelnen Agenten und die aktuellen Meßwerte ihrer Sensoren. Die Planung basiert auf der Hierarchie von Handlungen, d.h. wie einzelne Verhalten aufeinander aufbauen und wie sie strukturiert sind (mehr dazu in Abschnitt 4.3.2).

Im Folgenden werden Ansätze zur reaktiven Planung aufgezeigt.

4.3.1 Ansatz mit Rückkopplung

Ein Ansatz zur reaktiven Planung ist der Ansatz mit Rückkopplung. Er basiert darauf, daß Abweichungen zwischen realen und prognostizierten Zustandsänderungen kontrolliert werden. Wenn eine Abweichung festgestellt wird, wird auf der Basis des tatsächlichen Zustands neu geplant ([Lüth98]). Das System funktioniert nach dem Prinzip der Rückkopplung (Abbildung 10).

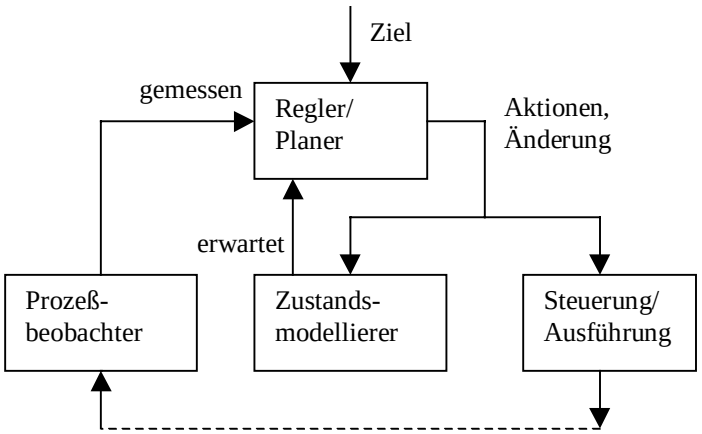


Abbildung 10. Das Prinzip der Rückkopplung.

Dieser Ansatz hat folgende Vorteile:

- Hohe Geschwindigkeit;
- Einfachheit der Umsetzung;
- Die Modellierung der realen Welt und der entsprechende Aufwand entfallen.

Die Nachteile dieses Verfahrens sind folgende:

- die einzelnen Verhaltensmuster sind nicht strukturiert;
- das Hinzufügen weiterer Verhalten ist schwierig;
- die Verhaltensmuster sind im Planungs-/Steuerungsalgorithmus nicht explizit vorhanden;
- typische Nachteile der reaktiven Ansätze wegen des fehlenden Weltmodells (beispielsweise sind die Plan- und Lernfähigkeit begrenzt).

Im Weiteren wird ein anderer Ansatz zur reaktiven Planung aufgezeigt, der einige dieser Probleme bewältigen kann.

4.3.2 Subsumption Architecture

Ein wichtiger Ansatz zur reaktiven Planung und Steuerung ist die Subsumption Architektur (s. Abbildung 11). Im Mittelpunkt dieser Architektur steht der Verhaltensmuster-Begriff, der definiert, welche Aktionen ein Agent ausführt, falls eine Bedingung erfüllt ist ([Brooks86]). Als Beispiele für Verhaltensmuster in der Subsumption Architektur sind Enter-Doorway-Verhalten (In die Tür hineingehen), Wall-Following (der Wand folgen), Avoid Obstacles (Hindernissen ausweichen), Move Forward (Vorwärts fahren) zu erwähnen.

Diese Architektur hat folgende Eigenschaften:

- es gibt keine Unterteilung in die Perzeption, Informationsverarbeitung und Handlungsausführung;
- die Architektur beinhaltet eine Priorisierung der Teilschichten – die oberste Schicht hat gegenüber den unteren Schichten Vorrang in der Durchführung;
- Es gibt nur die aktivitäten-orientierte Unterteilung, in der die einzelnen Schichten des Systems hierarchisch geordnet sind. Eine in der Hierarchie höher liegende Schicht weist ein komplexeres Verhaltensmuster auf, während die untersten Ebenen die primitivsten Verhalten implementieren;
- Jede neue Schicht erweitert die Fähigkeiten des Gesamtsystems. Nachrichten werden zwischen einzelnen Schichten ausgetauscht, wobei die einzelnen Verhalten parallel laufen.

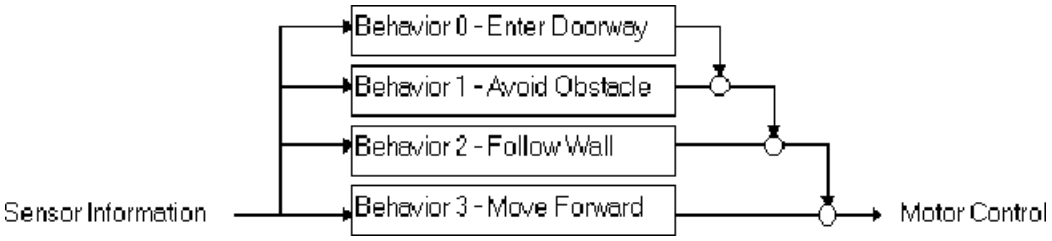


Abbildung 11. Eine typische Subsumption Architektur.

Die Funktionsweise eines Systems mit dieser Architektur kann am folgenden Beispiel aufgezeigt werden: unter normalen Umständen fährt ein Roboter vorwärts (Behaviour 3). Wird währenddessen von seinen Sensoren eine Wand registriert wird, so wird zusätzlich das Verhalten 2 „Der Wand folgen“ ausgeführt. Zusätzlich kann das Verhalten 1 „Hindernissen ausweichen“ oder Verhalten 0 „In die Tür hineinfahren“ dazu geschaltet werden.

Diese Architektur hat folgende Vorteile:

- Die einzelnen Teilschichten können mit einfachen endlichen Automaten realisiert werden;
- Das Hinzufügen einer weiteren Schicht, d.h. eines Verhaltens, ist einfach. Weitere Schichten werden schlicht in das bestehende System hierarchisch eingeordnet.
- Hohe Geschwindigkeit;
- Einfachheit der Umsetzung;
- Die Modellierung der realen Welt und der entsprechende Aufwand entfallen.

Als Nachteile der Subsumption Architektur sind folgende zu erwähnen:

- typische Nachteile der reaktiven Ansätze wegen des fehlenden Weltmodells (beispielsweise sind die Plan- und Lernfähigkeit begrenzt);
- In einem komplexen System mit einer großen Anzahl der Verhalten (Systemschichten) ist der Synchronisations- und Managementaufwand sehr groß.

4.4 Ansatz zur deliberativen Planung

Um eine fortgeschrittene Planungs- und Lernfähigkeit zu implementieren, reichen die reaktiven Ansätze nicht aus. Die Planung und das Lernen setzen voraus, daß es im

Multiagentensystem ein Weltmodell gibt, auf dessen Grundlage die Planung und das Lernen erfolgen. Dabei können in so einem System verschiedene Agenten mit dem BDI-Ansatz (Belief-Desire-Intention) zusammenarbeiten, die jeweils Intentionen, Ziele, Wünsche, Überzeugungen, Wissen und Pläne einzelner Agenten zur Planung benötigen (Abbildung 12).

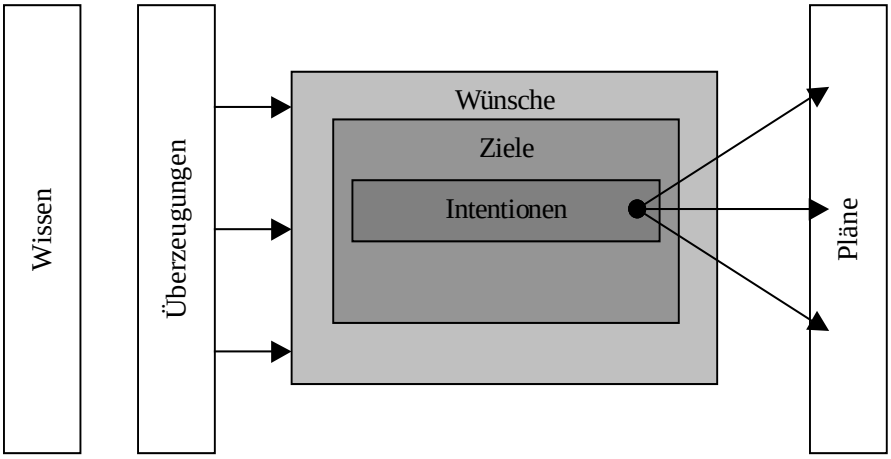


Abbildung 12. Struktur eines deliberativen Planungsmoduls.

Überzeugungen sind Annahmen eines deliberativen Agenten über die Umwelt, in der er funktioniert. In einem Multiagentensystem können sich die Annahmen eines Planungsagenten aus den Teilannahmen einzelner Agenten zusammensetzen.

Die Wünsche basieren auf Überzeugungen und stellen eine Absicht zur Erreichung eines Systemzustandes dar. Ob ein Wunsch realistisch und erfüllbar ist, ist nicht entscheidend.

Die Ziele bilden eine Untermenge der Wünsche. Es handelt sich um die Wünsche, die realistisch und erfüllbar sind, und nicht in Konflikt zueinander stehen.

Aus den Zielen werden Intentionen abgeleitet. Dieser Fall tritt auf, wenn sich ein Agent für ein Ziel entscheidet. In jeder Intention können sowohl Angaben über notwendige Ressourcen, als auch über die Priorität des Ziels gespeichert werden.

Pläne stellen eine Zusammenfassung der Agentenintentionen dar und geben eine Auskunft über Teilpläne des jeweiligen Agenten.

Überzeugungen über die Wünsche, Ziele und Intentionen eines Agenten werden in der Wissensdatenbank gespeichert. Aus diesen Informationen werden Pläne abgeleitet.

Weitere Informationen über den BDI-Ansatz finden sich in [Rao91].

Der Ansatz zur deliberativen Planung hat folgende Vorteile, die sich hauptsächlich aus dem Weltmodell ergeben:

- Die Planung wird wegen eines vorhandenen Weltmodells zuverlässiger – das Weltmodell ermöglicht die Fähigkeit des Systems zu logischen Schlußfolgerungen;
- Die Lernfähigkeit ist größer; bessere Planalternativen und die Gründe für deren Auswahl können im Weltmodell abgespeichert werden;

- Das Weltmodell ermöglicht eine Protokollierung von Änderungen des Weltmodells und einen Vergleich von geplanten und erreichten Zielen, was einer eventuellen Umplanung und der Lernfähigkeit zugrunde liegen kann.

Die Nachteile des deliberativen Ansatzes sind folgende:

- Die Weltmodellierung stellt eine zeit- und arbeitsaufwendige Aufgabe dar. Noch anspruchsvoller ist sie in einem Multiagentensystem mit Hardware-Agenten, da die Informationserfassung nur ungenau erfolgen kann.

Die deliberative Planung benutzt folgende konventionelle Ansätze:

- Logikbasierte Ansätze;
- Planbasierte Ansätze.

Die logikbasierten und planbasierten Ansätze unterscheiden sich hauptsächlich darin, wie Zusammenhänge zwischen einzelnen Operatoren im Plan definiert sind. Die logikbasierten funktionieren nach dem Prinzip der Planung auf der Basis von total geordneten Plänen. Die planbasierten Ansätze machen Gebrauch von der Planung auf Basis von partiell geordneten Plänen. Die Entscheidung darüber, welche Alternative im Plan ausgewählt wird, fällt erst während der Planausführung (s. Abschnitt 3.2).

In den weiteren Abschnitten werden logikbasierte und planbasierte Ansätze ausführlich dargestellt. Es werden dabei die wichtigsten Eigenschaften einer deliberativen Planrepräsentation aufgezeigt.

4.4.1 Logikbasiertes Planen (STRIPS)

Das logikbasierte Planen ist der grundlegende Ansatz zur Planung ([Fikes72]). Es benutzt zur Planung eine Weltrepräsentation, in der die Welt durch logische Formeln (Prädikate, Literale) beschrieben ist.

Am Anfang des Planungsvorgangs werden ein Startzustand und ein Zielzustand angegeben. Das Planungsmodul sucht eine Aktionssequenz, die das System, basierend auf dem Startzustand, in den Zielzustand überführt (Abbildung 13).

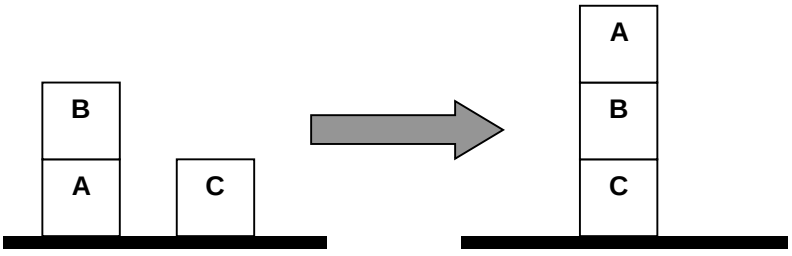


Abbildung 13. Der STRIPS-Planungsvorgang.

Eine geplante Aktion in dieser Welt entspricht einem Operator (STRIPS-Operator). Ein STRIPS-Operator wird definiert als:

1. Operatorvorbedingungen;
2. `Add`-Liste, die Fakten, die nach Ausführung des Operators gültig werden;
3. `Del`-Liste, die Fakten, die nach Ausführung des Operators nicht mehr gültig sind.

Der Planungsalgorithmus wird wie folgt definiert:

```
STRIPS-Algorithmus (Ziel)

   Für jedes Element des Ziels {

      1. Nimm ein beliebiges Fakt aus dem Ziel

      2. Suche einen Operator, der dieses Fakt herstellt.

      //Stelle (rekursiv) seine Vorbedingung her.
      3. STRIPS-Algorithmus (Vorbedingung)

      4. Führe diesen Operator aus
   }
```

Algorithmus 1. STRIPS-Algorithmus.

Im Weiteren wird aufgezeigt, wie die Welt aus Abbildung 13 modelliert wird.

Der Anfangszustand (die linke Seite aus Abbildung 13) wird durch folgende Literale beschrieben:

```
auf(A, Tisch), auf(B, A), auf(C, Tisch), frei(B), frei(C)
```

Der Zielzustand (die rechte Seite aus der Abbildung 13):

```
auf(A, B), auf(B, C), auf(C, Tisch), frei(A)
```

Für die angegebene Problemstellung werden Operatoren der Form `stelle (Was, Worauf, Start)` benutzt, die wie folgt definiert sind:

Vorbedingung: `auf(Was, Start), frei(Was), frei(Worauf)`
Nachbedingung: `auf(Was, Worauf), frei(Start)`

oder

Vorbedingungen: `auf(Was, Start), frei(Was), frei(Worauf)`
Add-Liste: `auf(Was, Worauf), frei(Start)`
Del-Liste: `auf(Was, Start), frei(Worauf)`

Der Operator `stelle` definiert die Bewegung des Klötzchens `Was` vom Klötzchen `Start` auf das Klötzchen `Worauf`.

Das Planungsmodul verwaltet eine sogenannte Agenda – eine Menge von Zielen, die erreicht sind. Die Agenda wird mit den Nachbedingungen des Startzustandes initialisiert.

Für das Beispiel aus Abbildung 13 wendet das Planungsmodul die Operatoren in der folgenden Reihenfolge an:

Agenda vor der Ausführung	Operator	Agenda nach der Ausführung
`auf(A, Tisch), auf(B, A), auf(C, Tisch), frei(B), frei(C)`	`stelle(B,C,A)`	`Auf(A, Tisch), auf(B, C), frei(A), auf(C, Tisch), frei(B)`
`auf(A, Tisch), auf(B, C), frei(A), auf(C, Tisch), frei(B)`	`stelle(A,B,Tisch)`	`Auf(A,B), auf(B,C), auf(C, Tisch), frei(A)`

Tabelle 1. Logikbasiertes Planen.

Der erstellte Plan enthält zwei Operatoren: `stelle(B,C,A)` und `stelle(A,B,Tisch)`.

In vielen Fällen ist es sinnvoll die Rückwärtsverkettung anzuwenden, d.h. vom Zielzustand ausgehend eine Aktionssequenz zu suchen, die in umgekehrter Richtung propagiert den Startzustand ergibt.

Im aufgezeigten Vorgang wurde eine Aktionssequenz für die Klötzchenwelt in Abbildung 13 gesucht und erfolgreich gefunden. Leider funktioniert der STRIPS-Algorithmus nicht für alle Konstellationen in dieser Welt, z.B. liefert er keine Lösung für die in der Abbildung 14 aufgezeigten Situation (Sussman Anomalie).

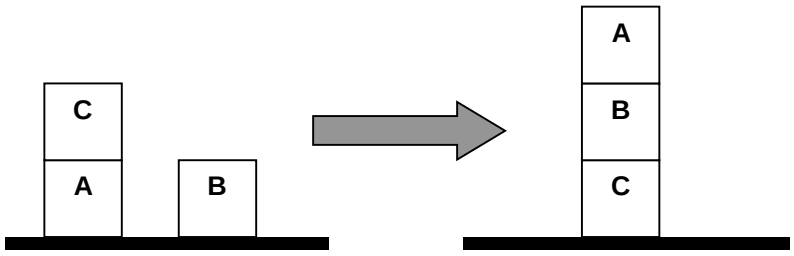


Abbildung 14. Die Klötzchenwelt bei der Sussman Anomalie.

Der Startzustand `auf(A, Tisch), auf(C, A), auf(B, Tisch), frei(B), frei(C)`.

Für diesen Fall wendet das Planungsmodul die Operatoren in der folgenden Reihenfolge an:

Agenda vor der Ausführung	Operator	Agenda nach der Ausführung
`auf(A, Tisch), auf(C, A), auf(B, Tisch), frei(B), frei(C)`	`stelle(C,B,A)`	`auf(A, Tisch), auf(C, B), frei(A), frei(B), auf(C, Tisch)`
`auf(A, Tisch), auf(C, B), frei(A), frei(B), auf(C, Tisch)`	`stelle(C,A,B)`	*`auf(A, Tisch), auf(C, A), auf(B, Tisch), frei(B), frei(C)`*

Tabelle 2. Die Sussman Anomalie.

Es ist dabei zu beachten, daß die zweite Aktion ein Zwischenziel gelöscht hat, das von der ersten Aktion hergestellt wurde. Die Möglichkeit, den Zielzustand zu erreichen, ist deswegen nicht gegeben.

Die Vorteile dieses Ansatzes sind:

- Vollständigkeit – der STRIPS-Planer prüft bei der Suche der Reihenfolge von Aktionen alle möglichen Kombinationen von Operatoren;
- Genauigkeit – die in Abschnitt 3.2 aufgezeigten Probleme der Planrepräsentation (Die Qualification, Prediction, Persistence, Frame, Ramification Probleme) werden durch Verwendung von Operatoren minimiert. Bei einem Operator mit einer gut definierten Logik fällt die Vorhersage der Zustände, die erreicht werden können, relativ leicht, wobei die Genauigkeit der Vorhersage von der Genauigkeit der einzelnen Operatoren abhängt.

Die Nachteile:

- Polynominale Komplexität, weil der STRIPS-Algorithmus im ungünstigsten Fall alle vorhandenen Operatoren untersucht;
- Unfähigkeit zum Lösen bestimmter Planaufgaben wegen fehlenden Zusammenhängen zwischen einzelnen Aktionen.

[Ferber99] weist auf weitere Einschränkungen hin:

- zu jedem Zeitpunkt kann ausschließlich ein einziger Operator angewendet werden;
- alle Auswirkungen einer Aktion müssen explizit angegeben werden, was nicht immer möglich ist;
- ein Operator bietet eine momentane Darstellung einer Aktion dar und repräsentiert nicht den Progreß der Aktionsausführung, was für die dynamische Planung eine wesentliche Einschränkung darstellt.

In diesem Abschnitt wurde das konventionelle logikbasierte Verfahren (STRIPS-Planer) aufgezeigt. Im nächsten Abschnitt geht es um den planbasierten Ansatz.

4.4.2 Planbasiertes Planen

Das logikbasierte Planen liefert einen total geordneten Plan, d.h. es legt eine Kombination von Aktionen fest, die das System von einem Startzustand in einen Zielzustand überführt. In einigen Fällen ist es jedoch sinnvoller, nicht an einer eindeutig definierten Aktionsreihenfolge festzuhalten, sondern sie bei der Planausführung abhängig von den Konditionen in der Umwelt zu bestimmen.

Der Unterschied zwischen logikbasierten und planbasierten Verfahren wird am folgenden Beispiel veranschaulicht. Ein total geordneter Plan im logikbasierten Ansatz enthält eine Reihenfolge ohne Abhängigkeiten, z.B. der Plan „A1, A3, A2“ besagt, daß Aktion A1, dann Aktion A3, anschließend Aktion A2 ausgeführt wird. Ein partiell geordneter Plan definiert Zusammenhänge zwischen möglichen Alternativen. Beispielsweise besagt der Plan „A1<A3<A2“ besagt, daß Aktion A1 zeitlich vor Aktion A3 und A2, Aktion A3 nach Aktion A1 und vor Aktion A2 und abschließend Aktion A2 erfolgen.

Das planbasierte Verfahren erstellt partiell geordnete Pläne, die Abhängigkeiten zwischen Aktionen im Plan festlegen. Formal wird ein partiell geordneter Plan als ein Tripel `<Aktionen A, Ordnungsconstraints O, Beziehungen L>` definiert, wobei A eine Menge von Aktionen ist, O – eine Menge von Ordnungsconstraints über A, und L – eine Menge von Abhängigkeiten.

Die Constraintsmenge über die Menge A gibt vor, in welcher Beziehung die Aktionen zueinander stehen, z.B. die Menge O (A1<A2, A2<A3) gibt vor, daß Aktion A1 vor Aktion A2 und Aktion A3 ausgeführt wird. Die Constraintsmenge heißt konsistent, wenn sie sich zu einer totalen Ordnung vervollständigen läßt – für sie sind eindeutige Vergleichsoperatoren definiert.

Eine Abhängigkeit (A1(Q)→A2) beschreibt die Tatsache, daß eine Aktion A1 das Literal Q erzeugt, das von Aktion A2 verarbeitet wird. Mit Hilfe von Abhängigkeiten läßt sich herausfinden, ob eine neu geplante Aktion mit den anderen Aktionen im Konflikt steht. Die Abhängigkeit (A1(Q)→A2) steht im Konflikt zur Aktion A, wenn die folgenden Bedingungen erfüllt sind:

1. $O \cup \{A1 < A < A2\}$
2. A hat ¬Q als Nachbedingung.

Im Laufe des Planungsvorgangs werden die Abhängigkeiten erzeugt und gelöscht. Es werden aufgetretene Konflikte zwischen Abhängigkeiten aufgelöst.

Der Algorithmus für den planbasierten Ansatz ist wie folgt definiert ([Schneeberger00]):

```
PlanbasiertesPlanen(<A,O,L>, Agenda, Operatoren)
  1. Terminierung:
         Wenn die Agenda leer ist, liefere <A,O,L>.
  2. Wahl eines Ziels:
         Wähle ein Paar <Q, Aneed> aus der Agenda.
  3. Wahl einer Aktion:
     Wähle zufällig eine Aktion Aadd aus A oder einen Operator aus der Menge der
     Operatoren, die Q als Effekt hat.
        wenn keine solche Aktion existiert
        dann Fehlschlag
        anderenfalls
         L´ = L ∪ Aadd(Q)→Aneed
         O´ = O ∪ {Aadd < Aneed}
        wenn Aadd eine neu geplante Aktion ist
        dann
         A´ = A ∪ {Aadd}
         O´ = O´ ∪ {A0 < Aadd < Az}
        andernfalls A´ = A
  4. Anpassung der Ziele:
     Agenda´ = Agenda \ {Q, Aneed}
        wenn Aadd eine neu geplante Aktion ist
        dann
            Für alle Vorbedingungen Qi füge <Qi, Aadd> der Agenda hinzu.
  5. Schutz der Abhängigkeiten:
     Für jede Aktion At, die in Konflikt zu einer Abhängigkeit Ap(R)→Ac steht:
        Demotion:
         O´ = O´ ∪ {At < Ap} oder
        Promotion:
         O´ = O´ ∪ {Ac < At} oder
  6. Rekursion:
        Liefere PlanbasiertesPlanen(<A´,O´,L´>, Agenda´, Operatoren)
```

Algorithmus 2. Planbasiertes Planen

Die Initialisierung des Algorithmus wird wie folgt ausgeführt:

```
RunPlanbasiertesPlanen {

   1. Initialisiere Aktionenmenge A mit {A0,Az}
   2. Initialisiere die Menge der Constraints O mit {A0 < Az}
   3. Initialisiere die Menge der Abhängigkeiten L mit einer leeren Menge {}
   4. Initialisiere die Agenda mit den Vorbedingungen des Zustandes Az.
   5. Initialisiere die Menge der Operatoren
   6. PlanbasiertesPlanen(<A, O, L>, Agenda, Operatoren)

}
```

Die Funktion PlanbasiertesPlanen(<A,O,L>, Agenda, Operatoren) erweitert sukzessiv partiell geordnete Pläne, basierend auf dem Zielzustand. Dabei entspricht der Planungsvorgang der Rückwärtsverkettung. Aktionen A0 und Az sind Sonderaktionen, die keine Vorbedingungen bzw. Nachbedingungen haben.

Um einen partiell geordneten Plan zu erzeugen, wird die Agenda mit den Paaren <Vorbedingung, Ziel> initialisiert. Im nächsten rekursiven Aufruf der Funktion PlanbasiertesPlanen wird die Vorbedingung zum Ziel gewählt, ggf. die Mengen der Abhängigkeiten und der Constraints erweitert und für das neue Zwischenziel der Planungsvorgang gemacht. Falls eine neue Abhängigkeit zu einer in die Menge bereits eingetragenen Abhängigkeit im Konflikt steht, wird dieser Konflikt aufgelöst, indem die entsprechende Abhängigkeit gelöscht wird.

Die Funktionsweise des planbasierten Verfahrens für die in Abbildung 13 aufgezeigte Klötzchenwelt stellt Abbildung 15 dar. In dieser und in den folgenden Abbildungen sind Vor- und Nachbedingungen der einzelnen Aktionen aus Übersichtlichkeitsgründen nicht aufgezeigt.

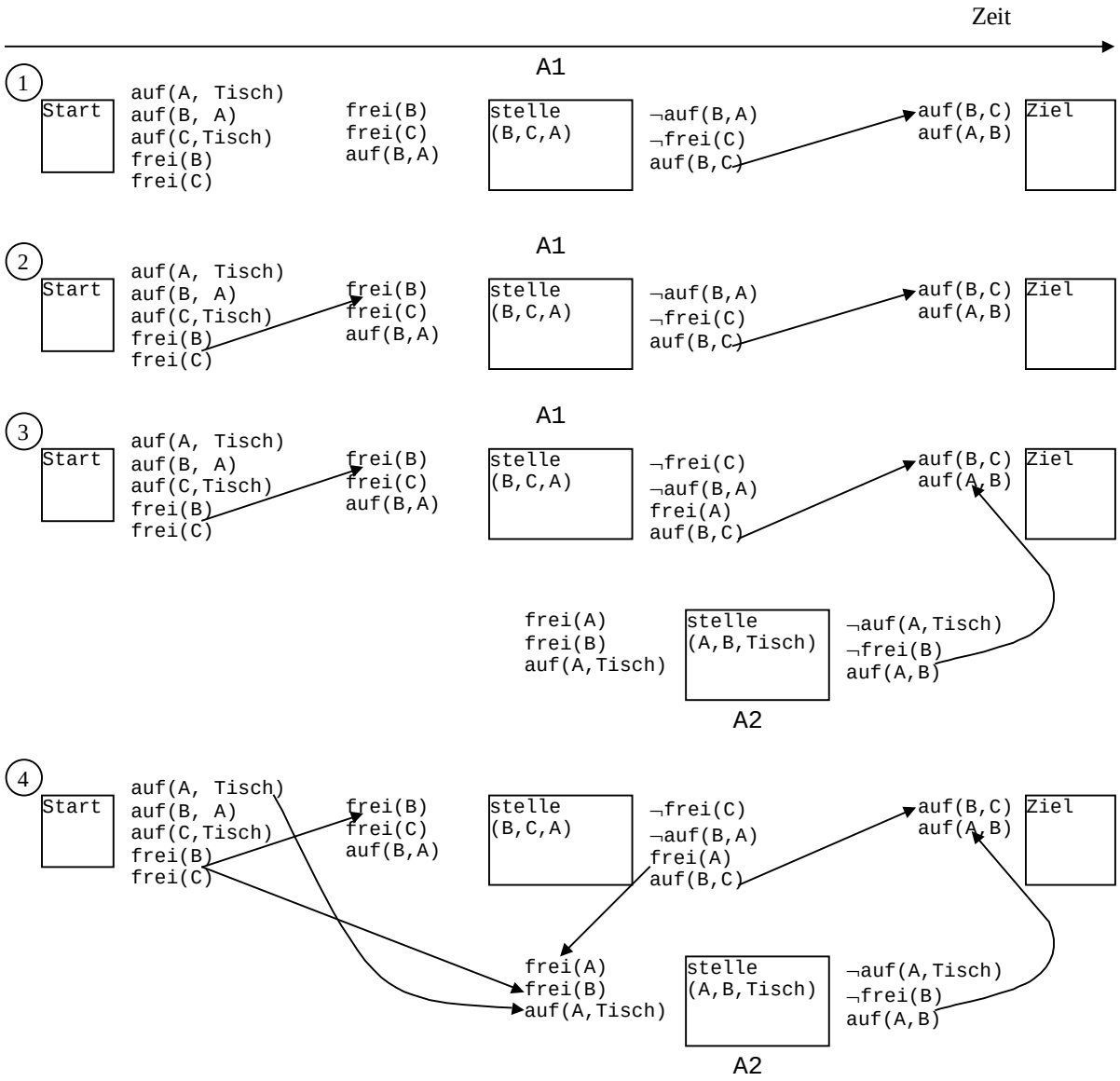


Abbildung 15. Funktionsweise des planbasierten Verfahrens.

Die Pfeile in Abbildung 15 zeigen die Abhängigkeiten, die im Laufe des Planungsvorgangs erzeugt werden. Rechtecke sind neu geplante Aktionen A1, A2. Links neben den Aktionen (Rechtecken) sind die jeweiligen Vorbedingungen aufgeführt und rechts die jeweiligen Nachbedingungen. Die Zeitachse zeigt die zeitliche Anordnung von Aktionen. Beispielsweise steht Aktion A1 links von Aktion A2, d.h. Aktion A1 wird früher ausgeführt als Aktion A2. Das Planungsmodul prüft die im Moment eingetragenen Abhängigkeiten, um eventuelle Konflikte aufzulösen.

Vereinfacht funktioniert das planbasierte Verfahren wie folgt (s. die Schritte in Abbildung 15, sowie Algorithmus 2):

0. Die Agenda wird mit den Paaren {<Vorbedingung1(Az), Az>, <Vorbedingung2(Az), Az>}, also mit den Paaren {<auf(B,C), Ziel>, <auf(B,C), Ziel>} initialisiert. Dieser Schritt kommt in Abbildung 15 wegen dessen Einfachheit nicht vor.
1. Es wird aus der Agenda ein zufälliges Paar ausgewählt <auf(B,C), Ziel>. Da der Startzustand keine Bedingung „auf(B,C)“ enthält, wird eine Aktion gesucht, welche die Bedingung „auf(B,C)“ herstellt. Diese Aktion wird im aufgeführten Beispiel als Aktion

„stelle(B,C,A)“ dargestellt. D.h. um die Vorbedingung des Ziels erfüllen zu können, muß vorher die Aktion „stelle(B,C,A)“ (Schritt 1 in der Abbildung 15) ausgeführt sein. Anschließend wird Abhängigkeit „A1(auf(B,C))→Az“ in die Abhängigkeitsmenge L und das Constraint „A0<A1<Az“ in die Constraintsmenge O eingetragen.

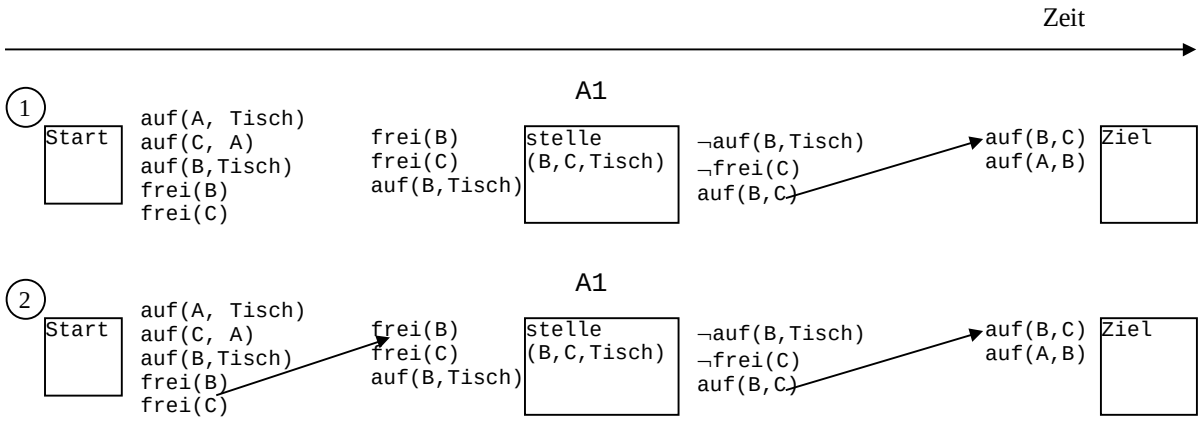
2. Um die Aktion A1 ausführen zu können, muß gewährleistet sein, daß ihre Vorbedingungen (frei(B), frei(C), auf(B,A)) erfüllt sind. Es wird die Bedingung „frei(B)“ ausgewählt. Da sie im Startzustand vorkommt, sind keine weiteren Aktionen mehr nötig, um sie zu erzeugen. Es wird lediglich eine Abhängigkeit „A0(frei(B))→A1“ hinzugefügt. Dasselbe erfolgt jeweils für die Bedingungen „frei(C)“ und „auf(B,A)“, denn sie tritt wie die Bedingung „frei(B)“ im Startzustand auf. Anpassung der Constraintsmenge ist nicht notwendig.

   Die Aktionssequenz, die für die Erfüllung der ersten Vorbedingung des Zielzustandes „auf(B,C)“ notwendig ist, ist geplant.

3. Der in Schritt 1 aufgezeigte Vorgang wird für die zweite Vorbedingung des Zielzustandes „auf(A,B)“ wiederholt. Um sie herzustellen, werden dem Plan eine neue Aktion A2 „stelle(A, B, Tisch)“, anschließend eine neue Abhängigkeit „A2(auf(A,B)) → Az“ und ein neues Constraint „A0<A2<Az“ hinzugefügt.

4. Der in Schritt 2 aufgezeigte Vorgang wird für die Vorbedingungen der Aktion A2 „frei(A)“, „frei(B)“, „auf(A,Tisch)“ wiederholt. Daraus entstehen drei jeweilige Abhängigkeiten „A1(frei(A) → A2)“, „A0(frei(B))→A1“ und „A0(auf(A,Tisch))“.

Abbildung 16 stellt den Planungsvorgang für die in der Abbildung 14 aufgezeigte Klötzchenwelt dar (s. auch Sussman Anomalie in Abschnitt 4.4.1). Aus Übersichtlichkeitsgründen sind einige offensichtliche Abhängigkeiten nicht aufgezeigt.



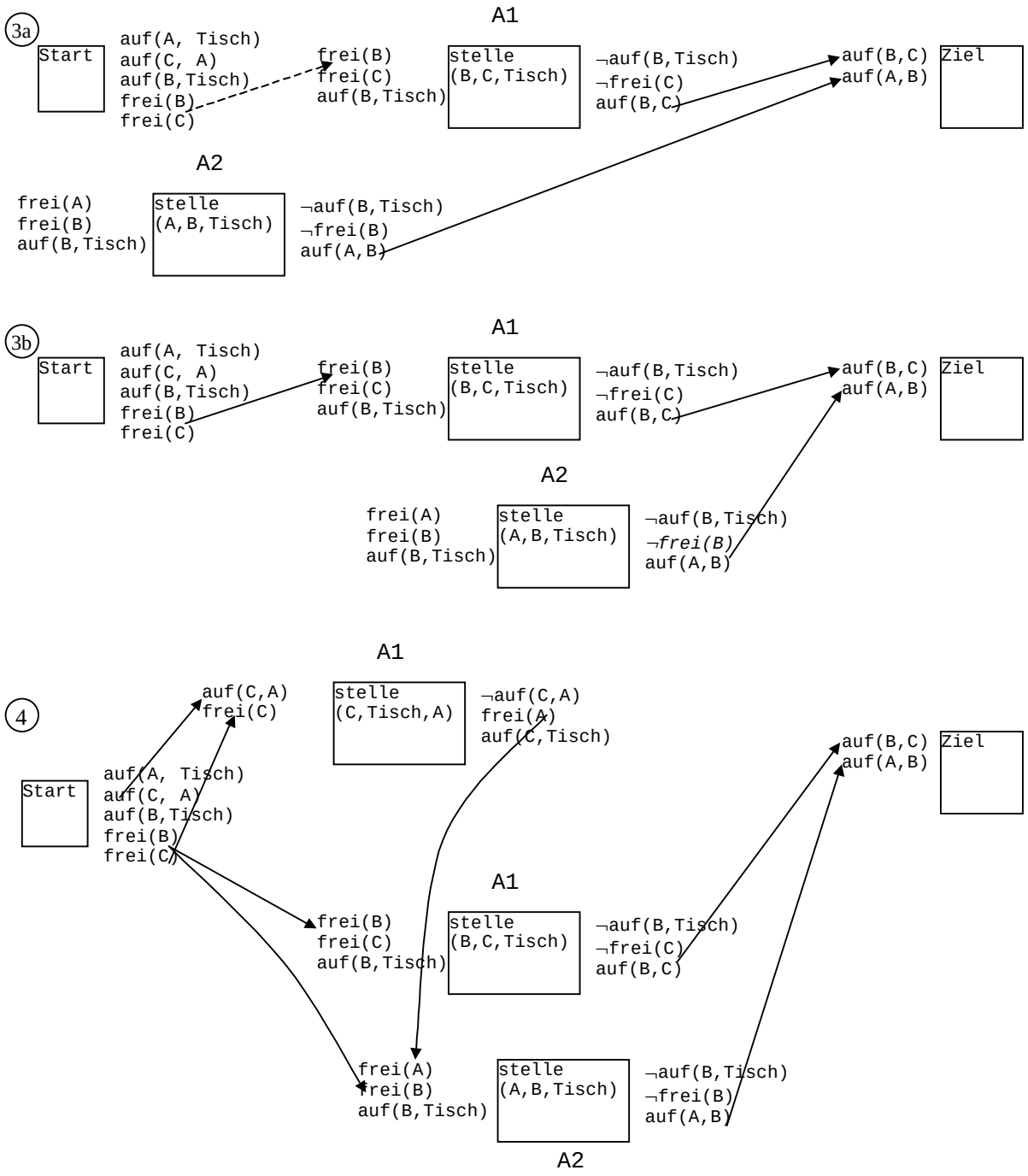


Abbildung 16. Das planbasierte Verfahren für die Sussman Anomalie.

Abbildung 16 zeigt den vollständigen partiell geordneten Plan, den das planbasierte Verfahren für die Sussman Anomalie erzeugt.

Der Vorteil von partiell geordneten Plänen liegt darin, daß Entscheidungen bezüglich der Auswahl möglicher Planalternativen erst während der Planausführung anhand von vorliegenden Informationen getroffen werden, wodurch die Planung genauer wird (s. Least Commitment in [Russell95]). Im Rahmen eines Multiagentensystems ist die Planung auf Basis von partiell geordneten Plänen um so interessanter, da solche Pläne keine unnötigen Vorschriften für den Ausführer eines Plans machen ([Schneeberger00]).

Der aufgezeigte Algorithmus funktioniert ausschließlich mit konstanten Werten, läßt keine Variablen zu und muß um die Variablenbindungen erweitert werden. Ausführliche Informationen darüber finden sich in [Schneeberger00].

Bei der Modellierung der zeitlichen Abhängigkeiten zwischen Aktionen kommt die temporale Planung zum Einsatz (s. auch [Heinsohn99]).

Ein weiterer Ansatz zur Planung auf Basis von partiell geordneten Plänen ist das graphbasierte Verfahren. Nähere Informationen finden sich in [Schneeberger00].

4.5 Hybrider Ansatz

Der Vergleich von reaktiven und deliberativen Ansätzen liefert das folgende Resultat:

- deliberative Ansätze sind besser für mittel-/langfristige Ziele geeignet, wobei die reaktive Planung hauptsächlich benutzt wird, um auf situierte, unvorhersehbare Einflüsse aus der Außenwelt zu reagieren ([Schneeberger00]);
- Die Qualität der deliberativen Planung steigt durch Benutzung eines Weltmodells. Außerdem ermöglicht ein Weltmodell die Lernfähigkeit des Systems. Die Kosten dafür sind ein hoher Zeit- und Arbeitsaufwand (s. Abschnitt 3.2).

Um die Vorteile der reaktiven (eine schnelle Reaktion auf die Änderungen in der Umwelt) und deliberativen (eine bessere Planungs- und Lernfähigkeit) Ansätze nutzen zu können, wurde der hybride Ansatz ins Leben gerufen.

Der hybride Ansatz vereinigt die reaktiven und deliberativen Ansätze. Seine Architektur definiert für einen Agenten folgende Schichten ([Burkhard00]):

1. Die verhaltensbasierte untere Schicht mit parallel arbeitenden Verhaltensmustern, die durch Umweltbedingungen oder Aufrufe der lokalen Planung aktiviert werden.
2. Lokale Planungsebene mit einer Bibliothek von Skelettplänen, die zur Laufzeit instanziiert und expandiert werden.
3. Kooperative (oberste) Planungsebene mit Verhandlungsprotokollen und Verhandlungsstrategien.

Abbildung 17 zeigt die Architektur eines Agenten im hybriden Ansatz.

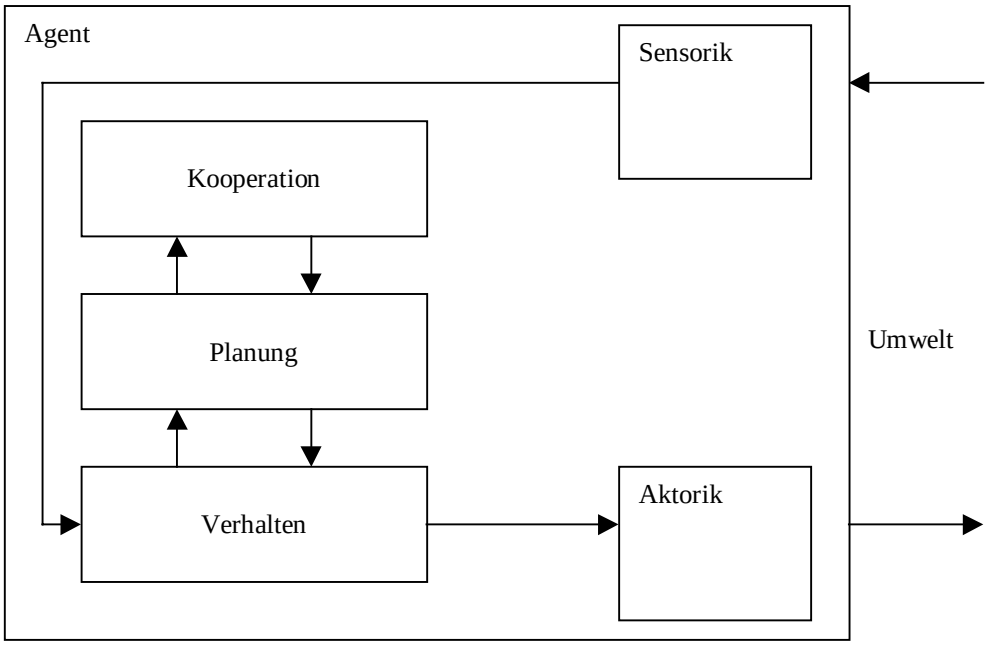


Abbildung 17. Der hybride Ansatz.

Es liegt nahe, daß die Verhaltensebene wegen der Anforderung an eine schnelle Reaktion auf Änderungen von Konditionen in der Umwelt mit dem Ansatz zur reaktiven Planung realisiert wird (s. Abschnitt 4.3). Für die Kooperations- und die Planungsebene ist der deliberative Ansatz besser geeignet, da keine größere Geschwindigkeit, sondern bessere Planungspräzision gefragt ist (s. Abschnitt 4.4).

4.5.1 Der Act-Formalismus

Im Rahmen dieser Diplomarbeit wurde ein Multiagentensystem entwickelt, an dem sowohl Hardware- als auch Software-Agenten beteiligt sind. Aus diesem Grund wurde der hybride Ansatz als Grundlage zur Planung und Planrepräsentation ausgewählt.

Im Weiteren wird der Act-Formalismus ausführlich dargestellt. Eine detaillierte Beschreibung ist insofern wichtig, da die ausgewählte Planrepresentation auf dem Act-Formalismus beruht (s. auch Abschnitt 7.5.1).

Der Act-Formalismus ist ein Formalismus zur Beschreibung von Plänen, in denen Informationen sowohl für die Plangenerierung, als auch für die reaktive Planausführung enthalten sind. Die Planinformation im Act-Formalismus wird in Aufgaben, Pläne und Akten aufgeteilt (Abbildung 18).

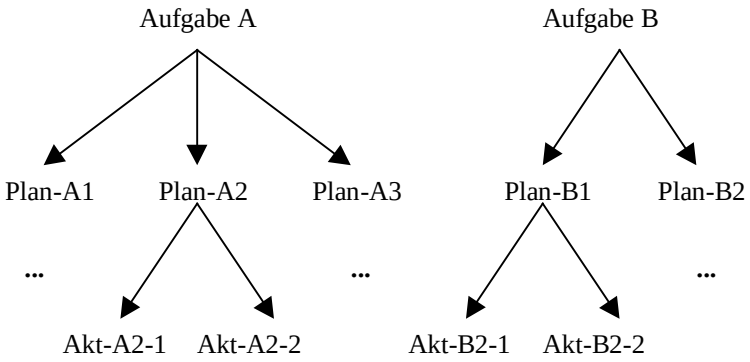


Abbildung 18. Planorganisation im Act-Formalismus.

Eine Aufgabe (task) im Act-Formalismus enthält eine Menge von Zielen, den aktuellen Zustand, sowohl andere Informationen, Annahmen, Voraussetzungen und Constraints, die für die Aufgabe bedeutend sind.

In einem komplexen System kann es erforderlich sein, für eine Aufgabe mehrere Planalternativen zu erzeugen. Eine solche Alternative heißt im Act-Formalismus ein Plan (plan) (um Verwechslungen mit dem konventionellen Plan zu vermeiden, im Weiteren als ein Act-Plan bezeichnet).

Ein Act-Plan wird in Akte (acts) unterteilt. Ein Akt (ein Aktionsnetz) beschreibt eine Menge von Aktionen, die unter gewissen Bedingungen ausgeführt werden (s. Abbildung 19). Ein Akt im Act-Formalismus entspricht einem konventionellen Plan.

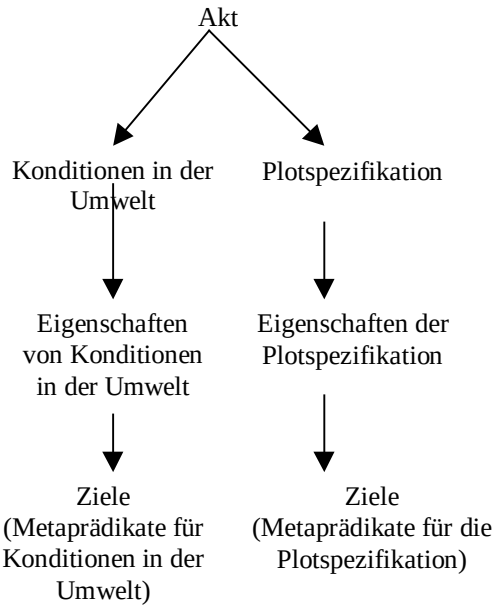


Abbildung 19. Struktur eines Aktes.

Die Konditionen in der Umwelt (environment conditions) geben an, welche Bedingungen erfüllt sein müssen, um den Akt ausführen zu können. Die Plotspezifikation enthält einen hierarchischen Plan mit Aktionen und Unterzielen (s. auch Abschnitt 4.4.2), in dem die

zeitlichen Zusammenhänge zwischen Aktionen mit Hilfe der Zeitlogik von Allen beschrieben werden (nach [Heinsohn99]).

Konditionen in der Umwelt werden durch folgende Eigenschaften beschrieben:

- Die `Name` Eigenschaft gibt den Namen des Aktes an.
- Die `Comment` Eigenschaft gibt einen Kommentar zum Akt an.
- Die `Cue` Eigenschaft gibt einen Verwendungszweck des Aktes an.
- Die `Precondition` Eigenschaft gibt die momentanen situativen Bedingungen in der Umwelt an, unter denen der Akt ausgeführt werden kann.
- Die `Setting` Eigenschaft gibt zusätzliche Bedingungen der Verwendung des Aktes an. Funktionell bedeutet diese Eigenschaft dasselbe, wie die `Precondition` Eigenschaft, wird jedoch üblicherweise benutzt, um Werte von Variablen zu speichern.
- Die `Resources` Eigenschaft gibt die Ressourcen für den Akt an.
- Die `Properties` Eigenschaft gibt Informationen über den Akt an:
  - zu Dokumentationszwecken;
  - zur Speicherung von Werten, die für eine konkrete Anwendung oder ein Planungs-/Planausführungssystem spezifisch sind;
  - zur Repräsentation des Wissens, das entweder vom Planungs-, oder vom Planausführungsmodul benutzt wird. Die Informationen werden paarweise Eigenschaft/Wert definiert.

Die Plotspezifikation ist eine Aktionsbeschreibung, welche die Darstellung eines Aktionsgraphen ist. Jeder Knoten in diesem Graphen bedeutet entweder eine parallele oder eine bedingte Ausführung der Nachfolgeknoten. Dafür wird in der Definition eines Knoten sein Typ angegeben:

- `type conditional` für die bedingte Ausführung der Nachfolgerknoten;
- `type parallel` für die parallele Ausführung der Nachfolgerknoten.

Sowohl in Konditionen in der Umwelt, als auch in der Plotspezifikation werden Ziele mit Hilfe von einem oder mehreren Metaprädikaten aus der folgenden vordefinierten Menge angegeben:

- Das `achieve` Metaprädikat gibt ein Ziel im Plan an, welches das System erreichen soll.
- Das `achieve-by` Metaprädikat gibt ein Ziel im Plan und eine Menge von Akten an, wobei einer davon zur Erreichung des Ziels angewendet werden muß.
- Das `achieve-all` Metaprädikat gibt eine Menge von `achieve` Metaprädikaten an, die ausgeführt werden.
- Das `wait-until` Metaprädikat gibt das Warten des Systems auf Eintreten eines spezifizierten Ereignisses an.
- Das `test` Metaprädikat prüft die Wahrheit einer Bedingung.
- Das `conclude` Metaprädikat gibt ein Fakt an, das einer internen Faktensammlung hinzugefügt wird.
- Das `retract` Metaprädikat löscht das angegebene Fakt aus einer internen Faktensammlung.
- Das `require-until` Metaprädikat gibt Ziele an, die solange erreicht werden, bis eine Bedingung erfüllt wird.
- Das `use-resource` Metaprädikat gibt die Ressourcen für die aktuelle Plotspezifikation an.

Die oben aufgeführte Beschreibung von Metaprädikaten wird am Beispiel der Fakultätsberechnung verdeutlicht ([Myers97]). In Abbildung 20 wird ein Akt dargestellt, der die iterative Fakultätsberechnung ausführt.

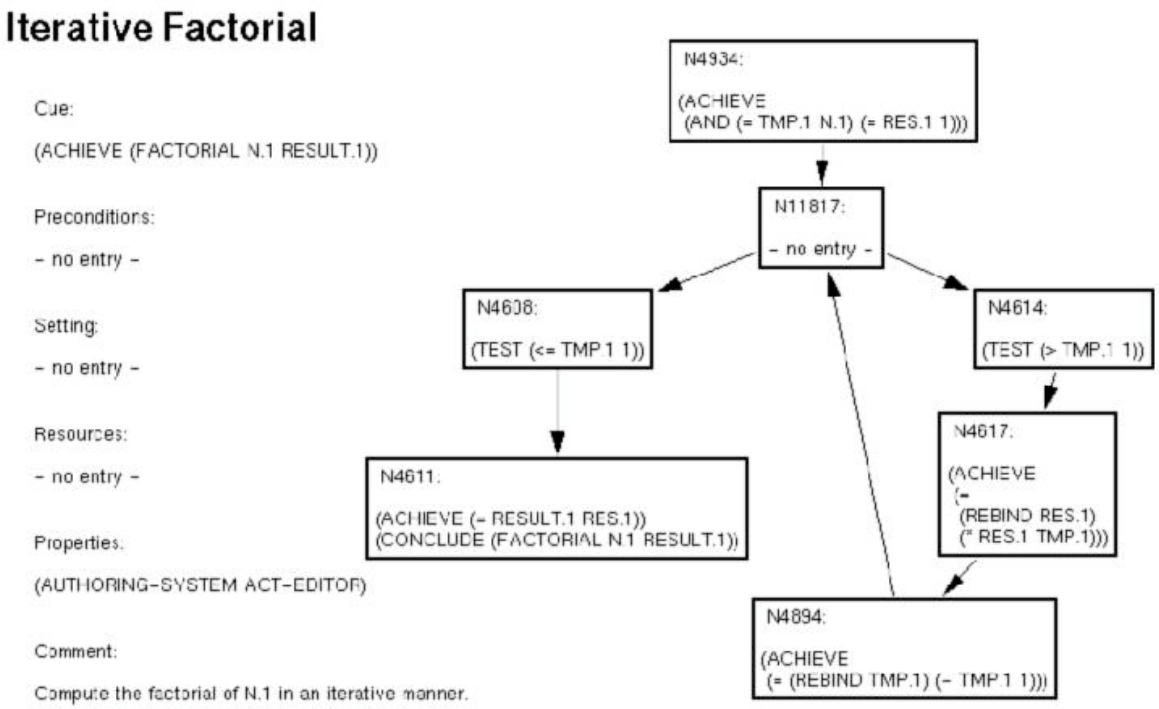


Abbildung 20. Ein Akt zur iterativen Berechnung der Fakultät.

Der folgende Algorithmus beschreibt vereinfacht die iterative Fakultätsberechnung aus der Aktbeschreibung in Abbildung 20:

```
Fakultät (int N, int result)

      //Initialisierung (N4934)
      1.  int tmp = N;
          int res = 1;

      //Paralleler Knoten (N11817)
      2. while (true)

          //prüfe die Abbruchbedingung (N4608)
          2.1.   if (tmp<=1)
                    return res;

          // (N4614)
          2.2.   if (tmp>1) {

              //Multiplikation (N4617).
              2.2.1. res = res * tmp;

              //Dekrementierung des Zählers (N4894)
              2.2.2. tmp = tmp – 1;
      }
   }
```

Algorithmus 3. Iterative Fakultätsberechnung.

Die oben aufgeführte Beschreibung des Act-Formalismus ist nicht vollständig. Dessen Spezifikation ist in [Myers97] nachzuschlagen. Eine BNF Spezifikation des Act-Formalismus findet sich in Abschnitt 9.3.

Eine Architektur zur Planung in einem Multiagentensystem auf Basis des Act-Formalismus ist in [Wilkins98] nachzulesen.

4.5.2 Analyse des Act-Formalismus

Ein Plan im Act-Formalismus macht es in einer Planrepräsentation möglich, die reaktiven und deliberativen Ansätze zu vereinigen. Von einer Seite ist es denkbar, situative Verhalten zu beschreiben, von einer anderen Seite – Planungsvorgänge durchzuführen. Das ist v.a. vorteilhaft für Systeme, die in einer dynamischen Umwelt funktionieren.

Ein reaktiver Plan im Act-Formalismus kann wie folgt repräsentiert werden (vgl. Abschnitt 4.3):

- Die `Precondition` und `Setting` Eigenschaften eines Aktes geben die Bedingungen für dessen Ausführung an.

Ein Beispiel für eine Implementierung des reaktiven Ansatzes mit dem Act-Formalismus ist das PRS-System (Procedural Reasoning System) – ein reaktives Framework zur Robotersteuerung ([PRS01]).

Ein deliberativer Plan wird im Act-Formalismus wie folgt repräsentiert (vgl. Abschnitt 4.4):

- Die `Precondition` und `Setting` Eigenschaften eines Aktes geben dessen Vorbedingungen an, ähnlich wie ein STRIPS-Operator in der logikbasierten Planung (s. Abschnitt 4.4.1);
- In der Plotspezifikation können hierarchische Pläne angegeben werden.

Ein Beispiel für eine Implementierung des deliberativen Ansatzes mit dem Act-Formalismus ist das Sipe-2-System ([Sipe01]).

Der Act-Formalismus beschreibt Pläne auf einem abstrakten Niveau. Um solche Pläne für ein konkretes Szenario in einem Multiagentensystem benutzen zu können, müssen zusätzliche Prädikate in die Planrepräsentation aufgenommen werden, welche mögliche Vorgänge, z.B. die Planausführung, konkret beschreiben. Ein Ansatz dafür ist in Abschnitt 7.5.1 aufgezeigt.

5 Planausführung

Die im Laufe der Planung erstellten Pläne werden an das Planausführungsmodul übergeben und von ihm ausgeführt. Für den Fall, daß das Multiagentensystem in einer dynamischen Umwelt funktioniert und der Planungsvorgang auf unvollständigen oder ungenauen Daten basiert, kann die Interaktion zwischen dem Planungs- und dem Planausführungsmodul tiefgreifender, als eine einfache Planübergabe ausfallen. Es kann eine Umplanung notwendig sein.

5.1 Probleme der Planausführung

In einem System, das in einer sich ändernden Umwelt funktioniert, sind die Planung und Planausführung eng miteinander verzahnt. Dabei treten folgende Probleme auf ([Bastié95]):

1. Eine Planrepräsentation muß in eine für die Ausführung geeignete Form umgewandelt werden. Eine eventuelle Plananpassung muß gewährleistet sein.

2. Die Planung und die Planausführung haben unterschiedliche Eigenschaften. Während die Planung eher statischer Natur ist, ist die Planausführung dynamisch ist und verläuft nicht immer wie geplant.

Im Weiteren werden Ansätze zur Planausführung dargestellt.

5.2 Ansätze zur Planausführung

Um eine erfolgreiche Zusammenarbeit zu ermöglichen, werden bei der Entwicklung des Planausführungsmoduls die reaktiven und deliberativen Ansätze zur Planung berücksichtigt (s. Kapitel 4).

Die Ansätze zur reaktiven Planung verkörpern die Planung und die Planausführung auf eine sehr natürliche Art, so daß eine Trennung nicht möglich ist. Dabei beinhalten die geplante Verhaltensmuster bereits die auszuführenden Aktionen.

Mit den Ansätzen zur deliberativen Planung fällt die Planausführung etwas schwerer, da die Abstimmung zwischen dem Planungs- und Planausführungsmodul permanent erfolgt. Dabei wird der Vorgang zur Ausführung eines Plans wie folgt definiert ([Bastié95]):

1. Das Planungsmodul generiert einen Plan anhand von bestimmten Annahmen und schickt ihn zum Planausführungsmodul.
2. Das Planausführungsmodul startet und kontrolliert die Planausführung. Falls es feststellt, daß eine Annahme bei der Planung falsch war, ergreift es notwendige Maßnahmen (eventuell einen Umplanungsvorgang), um das geplante Ziel zu erreichen.

Die Planausführung beim hybriden Ansatz zur Planung vereinigt einerseits reaktive, andererseits deliberative Verhalten. Eine Wissensrepräsentation liefert der Act-Formalismus (s. Abschnitt 4.5.1). Ausführliche Informationen über die im Rahmen dieser Arbeit entwickelte Implementierung des hybriden Ansatzes finden sich in Kapitel 7.

6 Kommunikation, Verteilung und Kooperation

Wie bereits oben erwähnt, setzen sich die Fähigkeiten eines Multiagentensystems aus den Fähigkeiten der einzelnen Agenten zusammen. Aus diesem Grund kommt auf die Kommunikation, die Verteilung und die Kooperation im Rahmen eines Multiagentensystems eine wichtige Rolle zu. Die Grundlage für die weiteren Überlegungen in diesem Kapitel bildet [Ferber99].

6.1 Einführung in die Kommunikation, Verteilung und Kooperation

In diesem Abschnitt werden sowohl einige Begriffe, als auch Probleme aufgezeigt, die in Bezug auf die Kommunikation, Verteilung und Kooperation auftreten können.

Um miteinander zu kooperieren, müssen die Agenten miteinander kommunizieren und Nachrichten austauschen können. Da die physische Umsetzung des Nachrichtenaustauschs gut erforscht ist ([Kühnel01]), wird im Weiteren unter Kommunikation ausschließlich der Nachrichtenaustausch und das Nachrichtenformat verstanden.

Kooperation ist eine verallgemeinerte Form der Interaktion zwischen Agenten im Multiagentensystem. Sie wirft in Bezug auf den in Abschnitt 4.4 aufgezeigten BDI-Ansatz folgende Fragen auf ([Haddadi96]):

- Wie wirken sich Überzeugungen, Wünsche und Intentionen des gesamten Agententeams auf individuelle Überzeugungen, Wünsche und Intentionen eines einzelnen Agenten aus?
- Wie wirken sich Intentionen des Agententeams zur Ausführung einer komplexen Aufgabe auf Intentionen eines einzelnen Agenten zur Ausführung eines Aufgabenteils aus?
- Was (und wann) müssen Agenten im Multiagentensystem bezüglich der Systemgesamtleistung, deren Anteil und dem Anteil der anderen Agenten wissen?
- Wie wird die kooperative Arbeit protokolliert?

Als Koordination wird eine Menge von ergänzenden Aktivitäten bezeichnet, die ein Multiagentensystem durchführt und die ein Agent mit denselben Zielen alleine nicht durchführen kann.

Die Koordination ist eine allgemeinere Form der Kooperation. Dabei wird nicht gefordert, daß die Agenten für ein gemeinsames Ziel arbeiten, sondern daß lediglich ihre Aktionen nicht im Konflikt stehen.

Die Vorteile der Koordination gegenüber der individuellen Arbeit sind:

- Informationsakquisition – ein Agent braucht Daten, die ausschließlich andere Agenten zur Verfügung stellen können;
- Ressourcenmangel – die Ressourcen im Multiagentensystem sind limitiert;
- Kostenoptimierung – unnötige Aktionen werden vermieden;
- Agentenunabhängigkeit – Agenten können unabhängig voneinander arbeiten und dabei gemeinsam Ergebnisse erzielen.

Diese Vorteile machen die koordinierte Datenverarbeitung sehr attraktiv. Es entstehen jedoch einige Probleme:

- Gesteuerte Objekte – welche Objekte nehmen an der Koordination teil?
- Deadlocks – es können gegenseitige Abhängigkeiten zwischen Aktionen entstehen, die zu Deadlocks führen;
- Konflikte – es können Abhängigkeiten zwischen Aktionen entstehen, die zu Konflikten führen.

Die Ansätze zur Kommunikation, zur Verteilung und zur Kooperation werden im nächsten Abschnitt dargestellt.

6.2 Ansätze zur Kommunikation, Verteilung und Kooperation

Wie bereits im letzten Abschnitt erwähnt, bringen die Kommunikation, die Verteilung und Kooperation einige Probleme mit sich. Das erste Problem bezüglich der gesteuerten Objekte entfällt meistens automatisch, da zumeist bekannt ist, welche Objekte an der Koordination teilnehmen.

Um die Probleme von Abhängigkeiten zwischen Aktionen und von Deadlocks zu lösen, gibt es folgende Ansätze:

1. Koordination mittels Synchronisation;
2. Koordination mittels Planung;
3. Reaktive Synchronisation;
4. Koordination mittels Regulation.

Koordination mittels Synchronisation setzt voraus, daß eine zeitliche Abhängigkeit zwischen Aktionen existiert, welche die Koordination steuert. Das bekannteste Verfahren zur Koordination mittels Synchronisation sind Petri-Netze.

Koordination mittels Planung erfolgt in der Planungsphase. Dabei werden partiell geordnete Pläne erstellt und weiter vervollständigt, bis ein konfliktfreier Plan vorliegt (s. Abschnitt 4.4).

Die reaktive Koordination ähnelt dem reaktiven Ansatz zur Planung, indem sie kein Weltmodell für Handlungsausführung benötigt und alle Aktionen situativ als Reaktion auf Ereignisse in der Umwelt ausführt (s. Abschnitt 4.3). Die Koordination erfolgt auf der Grundlage untergeordneter parallel laufender Verhalten (s. auch Abschnitt 4.3.2).

Koordination mittels Regulation liefert einen prinzipiell anderen Ansatz zur Koordination. Dabei wird nicht eine allgemeingültige Lösung für ein Problem gesucht, sondern das Problem wird in individuelle Zielstellungen für jeden Agenten transformiert, so daß die erforderliche Gesamtaufgabe ausgeführt werden kann.

Nähere Informationen über die Ansätze zur Koordination finden sich in [Ferber99].

Im Weiteren werden Ansätze zur Kommunikation, Verteilung und Kooperation dargestellt.

6.2.1 KQML

KQML – „Knowledge Query and Manipulation Language“ – ist konzipiert worden, um Interaktionen zwischen intelligenten Agenten in einer heterogenen Umgebung zu unterstützen. Sie gibt vor, wie Informationen und Wissen zwischen einzelnen Agenten in einem Multiagentensystem ausgetauscht werden und welche Syntax diese Informationen haben. KQML definiert einerseits eine Sprache für Nachrichten, andererseits eine Architektur, die darunter liegt.

Im Folgenden werden die KQML Architektur und anschließend die Syntax und Möglichkeiten der KQML Sprache kurz dargestellt. Eine ausführliche Beschreibung von KQML findet sich in [KQML93].

6.2.1.1 KQML Architektur

Die Spezifikation von KQML schreibt nicht eine genaue Architektur vor. Es existieren mehrere Ansätze, die miteinander kompatibel sind. In diesem Abschnitt wird auf einen Ansatz näher eingegangen ([Finin97]).

Die KQML Architektur basiert auf zwei Agententypen – Facilitators und Routers (Abbildung 21).

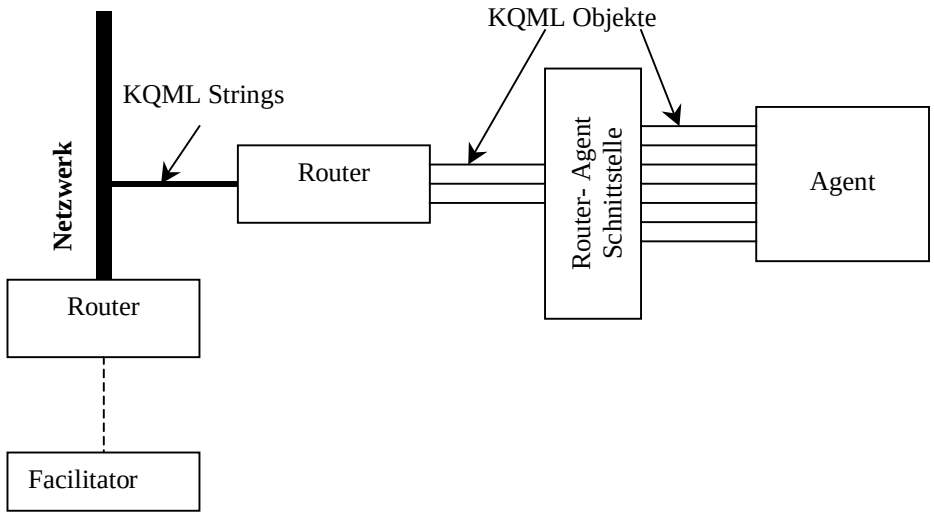


Abbildung 21. Die KQML Architektur.

Ein Router koordiniert die Kommunikation eines Agenten mit einer Zentrale – dem Facilitator und ist für die Adressierung verantwortlich. Jeder Agent verfügt über einen eigenen Router-Prozeß. Wenn ein Router die Adresse in einer Nachricht nicht kennt, so fragt er den Facilitator nach dieser Adresse.

Der Facilitator verwaltet das Multiagentensystem und ist ein spezieller Agent mit einer Verwaltungsfunktion. Er speichert Informationen über Agentenfähigkeiten in einer internen Datenbank und stellt sie den Routers auf Anfrage zur Verfügung.

6.2.1.2 KQML Syntax

Eine Nachricht in KQML ist ein ASCII String in der LISP Syntax. Die KQML Sprache definiert den Begriff „Performativ“, der für eine Aktion im Multiagentensystem steht.

Eine Nachricht enthält einen Performativ und seine Argumente. Beispielsweise startet die Nachricht

```
(ask-one
       :content (PRICE IBM ?price)
       :receiver stock-server
       :language LPROLOG
       :ontology NYSE-TICKS)
```

eine Anfrage der Ontologie `NYSE-TICKS` (`:ontology NYSE-TICKS`)) in der LPROLOG Sprache (`:language LPROLOG`) an den `stock-server` Agenten (`:receiver stock-server`) über den Preis einer IBM-Aktie (`:content (PRICE IBM ?price)`). Das Resultat enthält ein einzelnes Ergebnis (`ask-one`).

KQML definiert folgende Performatives ([Finin97]):

- Basis Performatives – `evaluate, ask-if, ask-in, ask-one, ask-all`, ...
- Mengen Query Performatives – `stream-in, stream-all`, ...
- Response Performatives – `reply, sorry`, ...
- Generische Informationsperformatives - `tell, achieve, cancel, untell, unachieve`, ...
- Generator Performatives – `standby, ready, next, rest, discard, generator`, ...
- Fähigkeiten Definition Performatives – `advertise, subscribe, monitor, import, export`, ...
- Netzwerk Performatives – `register, unregister, forward, broadcast, route`, ...

Argumente eines Performatives:

- `:content` – der Inhalt einer Nachricht;
- `:language` – der Name der zur Darstellung des Inhaltes ausgewählten Sprache (`:content` Parameter);
- `:ontology` – der Name der Ontologie im Inhalt (der `:content` Parameter);
- `:receiver` – der Empfänger des Performatives;
- `:sender` – der Sender des Performatives.

Eine ausführliche Beschreibung der KQML Syntax und der KQML Semantik ist in [KQML93] nachzuschlagen. Ein Demo-Programm für ein Framework zur Erstellung von Multiagentensystemen auf Basis der KQML findet sich in [JATDemo01].

6.2.2 OAA

Die OAA Architektur ist ein Ansatz zur Entwicklung von Multiagentensystemen ([Martin99]). Sie weist die Standardeigenschaften von modernen Computersystemen auf:

- Die Erweiterbarkeit und die Wiederverwendbarkeit sind gewährleistet;
- Die Austauschbarkeit von einzelnen Komponenten.

Zu den Eigenschaften der OAA Architektur gehört:

1. Verteilte Architektur, die weitgehend plattformunabhängig ist;
2. Nebenläufigkeit. Aufgaben im System können von mehreren Agenten gleichzeitig ausgeführt werden;
3. Dynamische Aufnahme neuer Agenten. Jeder Agent kann sich im System jederzeit anmelden, wobei seine Dienste für alle Agenten im System zugänglich werden;
4. Kleiner Ressourcenbedarf. Die Architektur kann u.a. auf Rechnern mit wenig Speicher laufen, z.B. auf Laptops. Die Kommunikation zwischen Agenten ist schnell genug, um die dynamische Robotersteuerung zu gewährleisten.

6.2.2.1 OAA Architektur

Im Mittelpunkt des OAA Konzepts steht der Facilitator – die Dienstzentrale, die das Multiagentensystem verwaltet und überwacht. Er speichert Informationen über die Adressen der Agenten im Netz und deren Dienste. Es können Multiagentensysteme mit mehreren Facilitators aufgebaut werden, die hierarchisch gegliedert sind. Die OAA Architektur besitzt Triggers, die gefeuert werden, wenn angegebene Bedingungen in der Systemumgebung erfüllt sind. Daten im System werden am schwarzen Brett ([Hayes-Roth85]) – einer globalen Datenbank – gespeichert und abgefragt. Der Facilitator verwaltet diese Operationen.

In Abbildung 22 ist ein Beispiel für ein OAA System dargestellt.

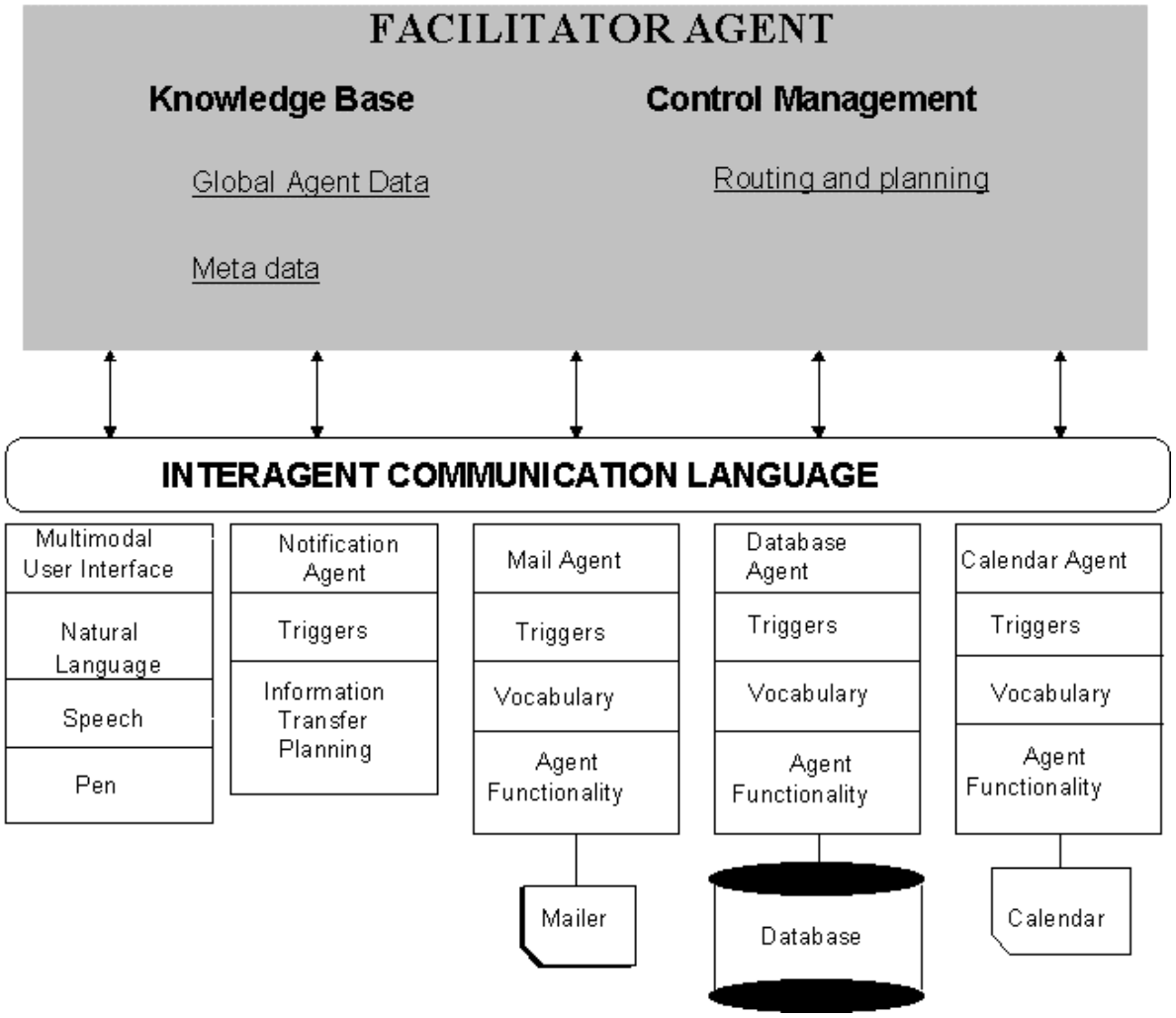


Abbildung 22. Ein Beispiel für ein OAA System.

Die OAA Architektur ist ereignisgesteuert. Ein OAA System wartet auf Anfragen von Agenten zur Ausführung von angegebenen Diensten und führt diese Anfragen aus, wodurch u.a. Dienste der anderen Agenten im System angefragt werden können. Es ist möglich, daß Bedingungen für einen Trigger infolge einer Zustandsänderung im System wahr werden, so daß der entsprechende Trigger vom Facilitator gefeuert wird.

Die Arbeitsweise wird am folgenden Szenario veranschaulicht. Neue Agenten melden sich beim Facilitator an. Anschließend teilen sie ihm mit, welche Dienste sie im Multiagentensystem bereitstellen, und welche Triggers installiert werden müssen. Nun gibt es zwei mögliche Ablaufvarianten. Entweder bearbeitet der Facilitator eine Anfrage zur Ausführung eines Dienstes, indem er den Agenten aussucht, der diesen Dienst implementiert, und leitet die Anfrage an ihn zur Ausführung weiter. Die andere Variante ist es, daß ein Trigger gefeuert wird, wenn der Facilitator feststellt, daß die Triggerbedingung eines Agenten erfüllt ist.

Das OAA Framework definiert folgende Nachrichtentypen:

- Zur Registrierung von Agentenfähigkeiten beim Facilitator;
- Zur Ausführung von Agentendiensten von einem Agenten im System;
- Zur Koordination der Ausführung von dem Facilitator im System.

Die einzelnen Agenten kommunizieren miteinander über den Facilitator. Die Kommunikation erfolgt auf Basis des TCP/IP Netzprotokolls. Alle Messages im System werden in einer speziellen Sprache (ICL) geschrieben (s. nächsten Abschnitt).

6.2.2.2 ICL

Die Kommunikation in OAA erfolgt mit Nachrichten in der ICL Sprache (Interagent Communication Language). Eine Nachricht in OAA ist ein ASCII String in folgender Syntax ([Martin99]):

```
ev_post_solve(Goal, Params)
```

Parameter `ev_post_solve` ist ein Nachrichtentyp, z.B. `oaa_Solve`, Parameter `Goal` – der Inhalt der Nachricht, z.B. `goto(Place)`, und Parameter `Params` sind die Parameter der Nachricht, z. B. `block(false)`.

Ein Beispiel für eine OAA Nachricht:

```
oaa_Solve ( goto(transferplace), [block(false)] ) ,
```

welche eine Nachricht des Typs `oaa_Solve` zur Ausführung eines Dienstes (`goto(transferplace)`) mit einem Parameter (`block(false)`) startet.

OAA definiert folgende Nachrichten ([SRI2]):

- Basisnachrichten – `oaa_Register`, `oaa_RegisterCallback`, `oaa_Ready`, `oaa_MainLoop`, `...`;
- Dienstausführung – `oaa_Solve`, `oaa_DelaySolution`, `oaa_ReturnDelayedSolutions`, `oaa_AddDelayedContextParams`, `oaa_Interpret`;
- Netzwerkfunktionen – `com_Connect`, `com_Connected`, `com_Disconnect`, `oaa_IsConnected`, `oaa_Disconnect`;
- Triggerfunktionen – `oaa_AddTrigger`, `oaa_CheckTriggers`, `oaa_RemoveTrigger`.

Parameter einer Nachricht:

- `cache(T_F)` – gibt an, ob das Ergebnis der Anfrage im Cache abgespeichert wird;
- `block(T_F)` – gibt an, ob die Arbeit des Systems blockiert wird, bis die Antwort auf die Anfrage ankommt;
- `address(Addr)` – gibt an, an welchen Agenten der Facilitator die Anfrage weiterleitet;
- `reply(Mode)` – gibt an, ob das Ergebnis der Anfrage deren Absender zurückgeschickt werden muß.
- `solution_limit(N)` – die Anzahl der Ergebnisse von einer Anfrage wird auf N begrenzt.

Eine ausführliche Beschreibung von OAA Funktionen findet sich in [SRI2].

6.2.2.3 OAA vs. KQML

Die OAA und die KQML Architekturen haben viele Ähnlichkeiten:

- Benutzung von speziellen Agenten mit ähnlichen Funktionen – den Facilitators, die Multiagentensysteme verwalten;
- Plattformunabhängigkeit durch einen Nachrichtenaustausch in einer neutralen Sprache. Die Sprachen (KQML und ICL) definieren ähnliche Möglichkeiten.
- Verteilte Architektur.

[Martin99] weist jedoch auf folgende Einschränkungen von KQML hin:

- Die Menge von KQML Performatives ist nicht universell in Bezug auf die Beschreibungsmöglichkeiten von Arten der Interaktion und von Anfragetypen;
- Der KQML Ansatz hat eine eingeschränkte Flexibilität, da er eine fest definierte Menge von atomaren Performatives festlegt.

Im nächsten Kapitel wird auf die ausgewählten Ansätze und das entwickelte Multiagentensystem eingegangen.

7 Die Lösung

In den vorangegangenen Kapiteln wurde aufgezeigt, wie die Planrepräsentation und die Planausführung in einem Multiagentensystem mit Hardware- und Software-Agenten aussehen können und welche Ansätze für die Themenbereiche im Multiagentensystem möglich sind. Nun werden die Ansätze, die ausgewählt wurden (s. auch Abschnitt 2.2), sowie die Architektur und die Funktionsweise des entwickelten Multiagentensystems dargestellt.

7.1 Szenario

Die Planrepräsentation und die Planausführung werden an einem Szenario untersucht, dessen Aufgabe es ist, einen Gegenstand von einem Raum in einen anderen zu bringen. Dabei werden zwei oder mehr Hardware-Agenten benutzt. Ein Hardware-Agent hat Verantwortungsbereiche, d.h. ihm ist ein Flächenbereich von einem Systemadministrator zugewiesen, den dieser Agent bedient (Abbildung 23).

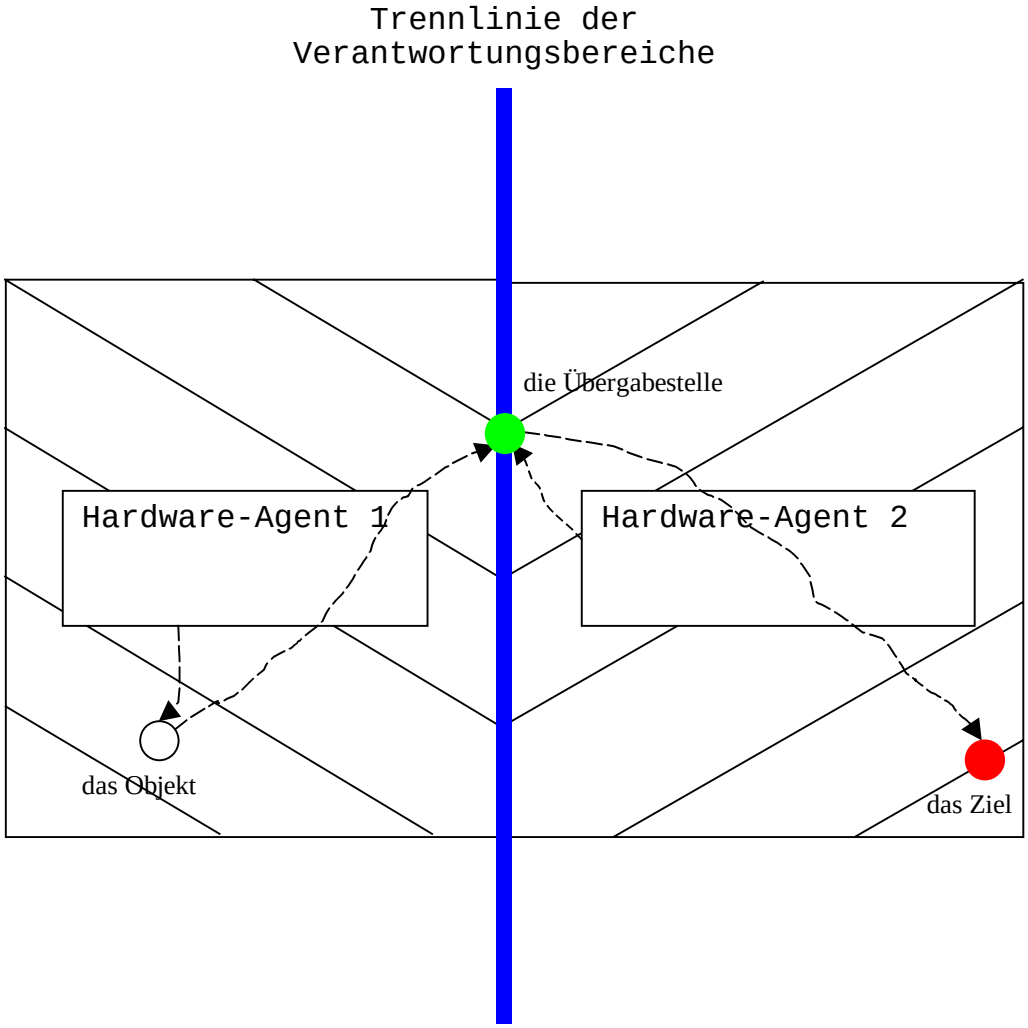


Abbildung 23. Verantwortungsbereiche der Hardware-Agenten.

Eine wesentliche Einschränkung ist es, daß die Hardwareagenten ihre Verantwortungsbereiche nicht verlassen dürfen. Das impliziert, daß die Arbeit von Hardware-Agenten koordiniert werden muß, damit sie die Gesamtaufgabe kooperativ ausführen können, ohne ihre Bereiche zu verlassen.

7.1.1 Allgemeine Bemerkungen

Um in einer realen Welt funktionieren und seine Handlungen intelligent ausführen zu können, beinhaltet das entwickelte System sowohl Hardware-Agenten für die physischen Eingriffe in dessen Umwelt, als auch Software-Agenten zur Koordination von Handlungen. Der Start von Handlungen wird von einem Benutzer initialisiert.

Die allgemeinen Eigenschaften der Agenten im System lassen sich wie folgt ausdrücken (s. Abschnitt 2.1):

- im System sind Hardware-, Software- und menschliche Agenten vorhanden.
- die Agenten sind Kooperationsagenten;
- die Agenten besitzen die Fähigkeit zum autonomen Verhalten, Reaktivität, Zielorientiertheit, Fähigkeit zur Kommunikation und Kooperation.

Im Rahmen des Multiagentensystems wird ein menschlicher Agent durch einen Software-GUI-Agenten repräsentiert, in dem Ein- und Ausgaben erfolgen und der diese Informationen an die anderen Agenten im System überträgt. Ein Hardware-Agent verfügt über eine Softwareschnittstelle, welche die Kommunikations- und Interaktionsaufgaben implementiert, sowie einen Greifer für den Transport von Gegenständen.

7.1.2 Anwendungsfall

Das entwickelte System implementiert den folgenden Anwendungsfall (Abbildung 24).

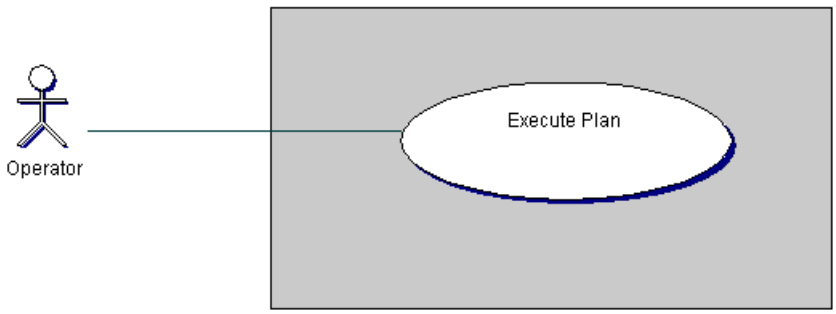


Abbildung 24. Das Anwendungsfalldiagramm für das System.

Für die Übergabeoperation wird ein Plan ausgeführt. Es lassen sich die Anfangsbedingungen wie folgt beschreiben:

- das Objekt befindet sich irgendwo auf dem Feld;
- es gibt zwei Hardware-Agenten (Roboter) im System, welche die Übergabe eines angegebenen Gegenstandes physikalisch ausführen können.

Die erforderlichen Aktionen sind folgende:

1. Es werden zwei Hardware-Agenten zur Ausführung der Übergabe ausgewählt;
2. Der erste Hardware-Agent nimmt das vom Benutzer angegebene Objekt mit, trägt es zur Übergabestelle und setzt es ab;
3. Anschließend holt der zweite Hardware-Agent das Objekt ab und bringt es zum Ziel.

Die Übergabeoperation kann in folgende Schritte aufgeteilt werden:

1. a) Die Suche nach den Hardware-Agenten, welche die Operation durchführen können und im Moment frei sind.
   b) Die Belegung dieser Agenten.
2. Beauftragung des ersten belegten Hardware-Agenten zur Mitnahme des Gegenstandes zur Übergabestelle.
3. Beauftragung des zweiten belegten Roboters zur Mitnahme des Gegenstandes zum Ziel.

7.1.3 Einschränkungen der Aufgabe

In dieser Diplomarbeit werden Pläne auf einer hohen Abstraktionsebene beschrieben. Sie enthalten keine Einzelheiten über reaktive Verhalten von Agenten. Es wird vorausgesetzt, daß die erforderlichen reaktiven Muster, welche die Hardware-Agenten implementieren, in den vorhandenen Colbert-Skripts realisiert sind (s. Abschnitt 7.3.2).

Da die Planung ausschließlich im Zusammenhang mit der Planrepräsentation und der Planausführung betrachtet wurde, enthält die Planrepräsentation nur die Plotspezifikation und nicht die für die Planung notwendigen Konditionen in der Umwelt aus dem Act-Formalismus (s. Abschnitt 4.5.1). Der Plan für die Übergabe wurde im Quellcode des Planungsmoduls fest kodiert. Die Colbert-Skripts zur Ausführung reaktiver Aufgaben, z.B. das Einsammeln und das Finden eines Objektes usw. beruhen auf [Jaghoori00].

Für die Implementierung der benötigten Funktionalität waren nur wenige API-Funktionen nötig, u.a. Funktionen für die Initialisierung der Saphira Software und für den Start der Colbert-Skripts. Für die Beschreibung der Saphira Software siehe Abschnitt 7.3.2.

7.2 Ausgewählte Ansätze

Im Weiteren wird aufgezeigt, welche Ansätze in Bezug auf die Themenbereiche in Multiagentensystemen ausgewählt wurden (s. Abschnitt 2.2).

- Auswahl einer Wissensrepräsentation:
  Eine explizite Weltrepräsentation wurde für die vorgeschlagene Aufgabe nicht erforderlich (s. Gründe für ein explizites Weltmodell in Abschnitt 3.1). Es wurde eine Planrepräsentation (basierend auf dem Act-Formalismus) für den hybriden Ansatz zur Planung ausgewählt, da in einem Multiagentensystem mit Hardware- und Softwareagenten gleichzeitig die deliberativen und reaktiven Ansätze von großer Bedeutung sind (s. Abschnitt 7.5.1).

- Planung:
  Ein Planungsmodul war nicht nötig, da der Schwerpunkt dieser Diplomarbeit auf der

Planrepräsentation und Planausführung lag. Ein Plan für die vorgeschlagene Aufgabe war im Quellcode fest kodiert (s. die `PlanServer` Klasse in Abschnitt 7.5).

- Planausführung:
  Die Planausführungsaktionen wurde in den Ausführungsmethoden der Planprimitivklassen implementiert. Die Aktionenbeschreibung liefert die auf dem Act-Formalismus basierende Wissensrepräsentation (s. Kapitel 5) mit Erweiterungen, die für ein Multiagentensystem notwendig sind (s. Abschnitt 7.5.1).

- Kommunikation, Verteilung und Kooperation:
  Im Rahmen dieser Arbeit wurde der Ansatz zur Koordination und Verteilung mittels Planung ausgewählt, d.h. daß ein konfliktfreier Plan bereits in einer früheren Phase ausgearbeitet worden war. Die Kommunikationsaufgabe realisiert das OAA Framework auf Basis der ICL Sprache. Das OAA Framework wurde wegen seiner an einem Projekt nachgewiesenen Tauglichkeit für ähnliche Aufgaben ausgewählt ([Guzzoni97]). Außerdem ist das Framework eine gute Plattform für praktische Anwendungen, die mit dem Roboter-API kompatibel sind.

- Management:
  Die Managementaufgaben führt das OAA Framework aus.

- Wissensaktualisierung (Lernfähigkeit) und Gewährleistung der Sicherheit:
  Wie oben bereits erwähnt, wurden die Fragen der Sicherheit und Wissensaktualisierung wegen ihrer eingeschränkten Bedeutung nicht näher betrachtet.

Im nächsten Abschnitt wird auf das Systemdesign und die Implementierung der ausgewählten Ansätze ausführlich eingegangen.

7.3 Benutzte Fremdsoftware

7.3.1 OAA Software

Das entwickelte Multiagentensystem wurde auf Basis der OAA Architektur entwickelt, welche die geforderten Standardeigenschaften von modernen Computersystemen bezüglich der Austauschbarkeit, der Erweiterbarkeit, der Wiederverwendbarkeit und der Verteilung aufweist (s. auch Kapitel 1). Diese Architektur wurde ausführlich in Abschnitt 6.2.2.1 beschrieben.

Die Distribution enthält außer dem Facilitator-Programm mehrere Agenten zwecks Demonstration und Unterstützung:

- einen Debug-Agenten, der in einem Multiagentensystem ein System-GUI repräsentieren kann und dem System die Fähigkeiten zum Anzeigen von Informationen zur Verfügung stellt;
- einen Monitor-Agenten, der die im Multiagentensystem angemeldeten Agenten und deren Fähigkeiten anzeigen kann;
- einen Nlg-Agenten zur Unterstützung natürlichsprachlicher Eingaben;
- einen Startit-Agenten zum Starten und Beenden von mehreren Agenten.

Allgemeine Schritte zur Erstellung neuer Agenten in OAA sind ([OAA01], [SRI1], [SRI2]):

1. Definition in der ICL Sprache der Solvables (Dienste), die der neue Agent im System implementiert.
2. Eine eventuelle Anpassung der Defaultverhalten, z.B. Angabe des Timeouts in der Kommunikation zwischen Agenten.
3. Eine eventuelle Deklaration und Definition der Agententriggers.
4. Implementierung der Callback-Funktionen für jeden Dienst des Agenten.
5. Eine eventuelle Deklaration und Definition der Callback-Funktionen, die beim Eintreffen eines Triggerereignisses ausgeführt werden.
6. Erstellung einer Funktion, die eine Verbindung mit dem Facilitator herstellt und anschließend Dienste des Agenten, dessen Callbacks, Triggers und Daten beim Facilitator anmeldet.

Die OAA Version 2.0 wird z.Z. in Prolog, C, Sun's JAVA und Compaq's Web Language ausgeliefert. Bei der Entscheidung über die Programmiersprache für die Implementierung wurde Java ausgewählt. Von einer Seite muß die Realisierungssprache gut C-Funktionen unterstützen, weil das API für die Roboter-Software C-basiert ist. Von der anderen Seite kann Programmiersprache C nur begrenzt benutzt werden, da sie keine objektorientierte Programmierung zuläßt. C++ wird außerdem vom SRI unzureichend unterstützt. Um vom Java-Programmcode aus auf die C-Funktionen des Roboter-APIs (s. Abschnitt 7.3.2) zugreifen zu können, wird das JNI (Java Native Interface) Interface verwendet. Eine ausführliche Beschreibung des JNI Interfaces findet sich in [Stearns01].

Weitere Informationen und Beispielsagenten finden sich in [SRI3].

7.3.2 Roboter-Software (Saphira)

Die Hardware-Agenten im entwickelten Multiagentensystem sind Pioneer Roboter ([ActiveMedia01]). Die allgemeine Robotersteuerung erfolgt über einen externen Rechner, der an einen seriellen Port an dem Roboter angeschlossen ist. Der Roboter benutzt ein integriertes serielles Funkmodem, um mit den anderen Rechnern zu kommunizieren. Zur Standardausstattung wurde für die Zwecke dieser Abhandlung ein Greifer eingebaut, mithilfe dessen unterschiedliche Objekte gegriffen und gefahren werden (Abbildung 25).

Abbildung 25. Pioneer II CE.

Für die Steuerung der Roboter wurde im Rahmen dieser Diplomarbeit die Saphira Software eingesetzt. Sie ist erhältlich für Microsoft Windows NT/95 und UNIX. Die Distribution 6.2d findet sich in [Konolige00].

Die Saphira Software arbeitet in einer Client/Server Umgebung. Der Server ist dabei ein Roboter, der Client – ein Programm mit einem GUI. Die Kommunikation mit dem Client erfolgt direkt über ein Texteingabefeld oder Auswahlmenüs im GUI. Es werden Sensordaten des Roboters und erkannte Artefakte grafisch dargestellt.

Die Programmiersprache Colbert in Saphira dient dazu, einen angeschlossenen Roboter zu steuern. Komplexe Verhalten können in Aktivitäten und Behaviors, die dann als eigenständige Tasks laufen, zerlegt werden. Colbert beinhaltet bereits eine Vielzahl von Basisfunktionen, die dem Benutzer den Einstieg in die Thematik erleichtern. Darüber hinaus gehende Funktionalitäten müssen selbst entwickelt und als dynamische Bibliotheken hinzugeladen werden.

Die Saphira Software stellt Softwareentwicklern ein API zur Verfügung, über das die meisten Funktionen der Software verwendbar sind. Folglich ist es möglich, auf das Standard-Saphira-GUI vollständig zu verzichten und die Steuerung der Roboter vom Benutzer-Programmcode aus zu führen.

Die Saphira Software beinhaltet u.a. mehrere Demo-Programme, einen Pioneer Simulator und die C-ähnliche Programmiersprache Colbert. Weitere Informationen finden sich in [Konolige00].

7.4 Systemdesign

Der Anhaltspunkt für das Systemdesign war die in Abschnitt 2.3 dargestellte hypothetische Architektur eines Multiagentensystems, die OAA Architektur (s. Abschnitt 6.2.2.1) und das an einem Projekt realisierte System mit Saphira-Hardware- und OAA-Software-Agenten ([Guzzoni97]). Dementsprechend hat das entwickelte Multiagentensystem folgende Agenten:

- Ein Facilitator-Agent:
  Der Facilitator-Agent ist ein Standard-OAA-Agent in der OAA Distribution, der die Arbeit des Multiagentensystems steuert (s. Abschnitt 7.3.1).
- Ein Debug-Agent:
  Ein Debug-Agent ist ein Standard-GUI-Agent (s. Abschnitt 7.3.1).
- PlanExecutor-Agent:
  Ein PlanExecutor ist ein Software-Agent, der die Planausführung steuert. Er liest die Planrepräsentation für die angegebene Aufgabe und führt den Plan aus (s. Abschnitt 7.5). In diesem Fall entspricht er dem Ausführer aus dem Abschnitt 2.3.
- ResourceManager-Agent:
  Hier handelt es sich um einen ResourceManager ist ein Agent, der die Ressourcen im System verwaltet (s. Abschnitt 7.6). Er übernimmt einen Teil der Aufgaben des Koordinators aus dem Abschnitt 2.3.
- Carrier-Agenten:
  Ein Carrier-Agent ist ein Hardware-Agent, der die Übergabe physikalisch in der Umwelt ausführt (s. Abschnitt 7.7).

Das Sequenzdiagramm für die Übergabe ist in Abbildung 26 aufgezeigt. Der Vorgang wird von einem Benutzer (`Kai von Luck`) im GUI-Agenten (`Debug`) gestartet. Die Anfrage (`execute(transfer)`) geht an den `PlanExecutor` Agenten, der vom `PlanServer` den folgenden Plan für die Übergabe-Operation erhält (vgl. Abschnitt 7.5.1):

1. Allokation von zwei Hardware-Agenten;
2. Beauftragung des ersten Roboters zum Transfer von Gegenstand `cup` zur Übergabestelle `tp` (Operation `bringto`);
3. Um zu prüfen, ob die Operation `bringto` abgeschlossen ist, wird die Operation `taskfinished` ausgeführt.
4. Beauftragung des zweiten Roboters zum Transfer von Gegenstand `cup` zum Ziel `finish` (Operation `bringto`);
5. Es wird geprüft, ob Operation `bringto` abgeschlossen ist, indem die Operation `taskfinished` gestartet wird.

Das Sequenzdiagramm für die Ausführung der Übergabe ist in Abbildung 26 aufgezeigt.

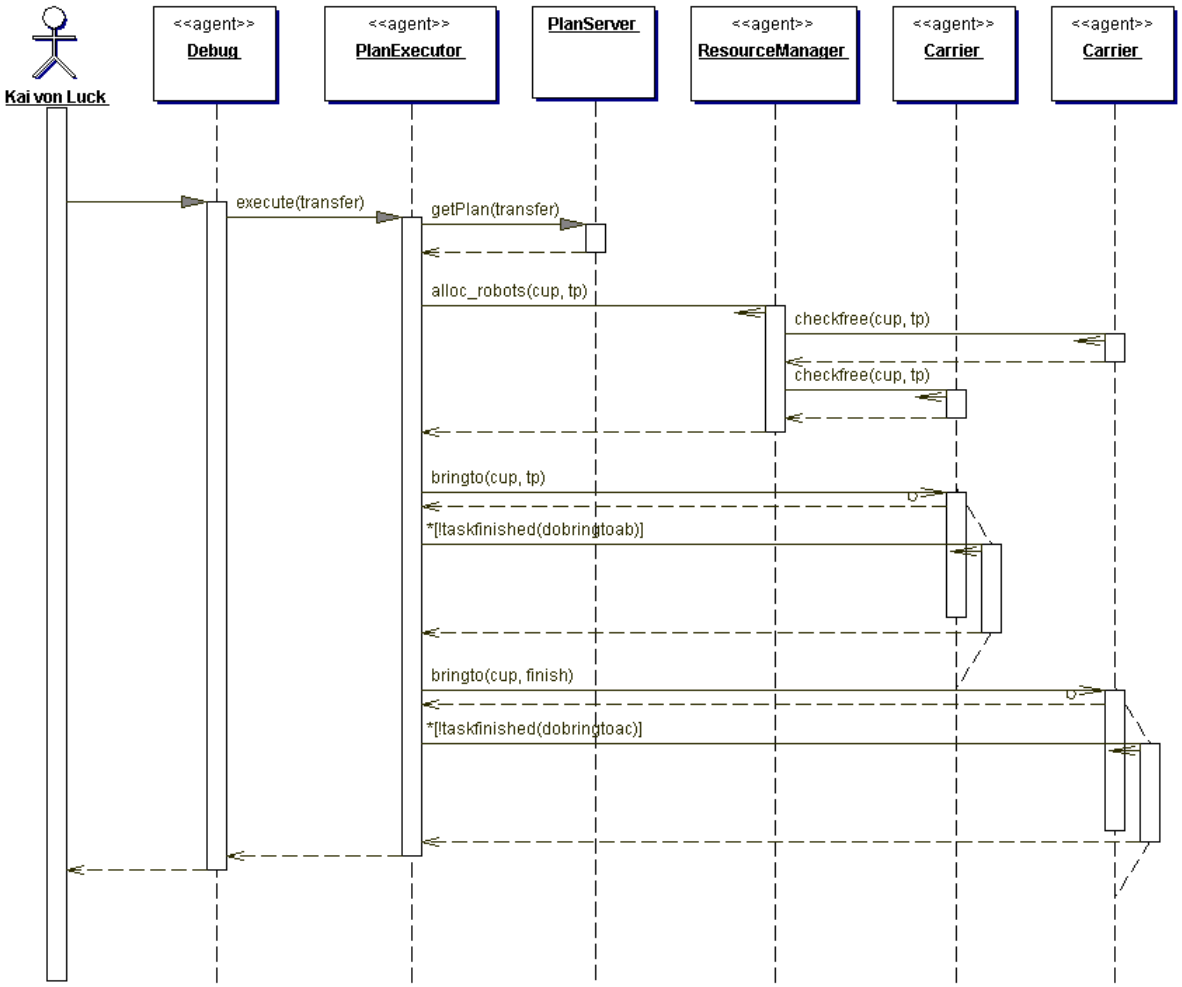


Abbildung 26. Sequenzdiagramm "Übergabe".

Abbildung 27 zeigt das Komponentendiagramm des Multiagentensystems.

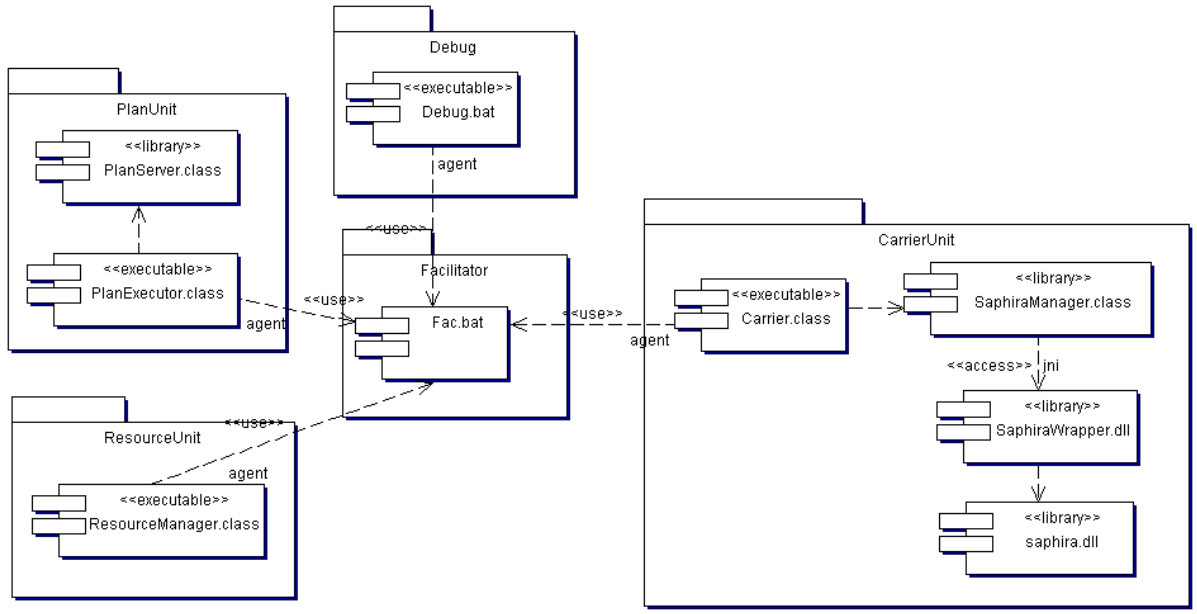


Abbildung 27. Das Komponentendiagramm des Multiagentensystems.

In Abbildung 27 werden folgende Komponenten aufgezeigt (vgl. Abschnitt 6.2.2.1):

- Subsystem „PlanUnit“ stellt den `planexecutor` Agenten dar, der Pläne ausführt (s. Abschnitt 7.5.2).
- Subsystem „ResourceUnit“ stellt den `resourcemanager` Agenten dar, der die Ressourcen im System verwaltet (s. Abschnitt 7.6).
- Zwei Subsysteme „CarrierUnit“ zeigen jeweils einen `carrier` Agenten auf, der im System einen Roboter darstellt (s. Abschnitt 7.7).
- Subsystem „Facilitator“ stellt den OAA-Facilitator (die Systemzentrale) dar (s. Abschnitt 7.3.1).
- Subsystem „Debug“ stellt den OAA-Debug-Agenten dar, der sämtliche Informationen im System ausgibt (s. Abschnitt 7.3.1).

Eine typische Verteilung von Agenten ist in Abbildung 28 aufgezeigt. Es werden zwei Laptops benutzt, die jeweils auf einem Roboter installiert sind und ihn steuern (s. Abschnitt 7.3.2). Auf dem Server laufen der OAA-Facilitator sowie der OAA-Debug-, der PlanExecutor- und der ResourceManager-Agent. Die Kommunikation mit Hardware-Agenten erfolgt auf Basis einer Funk-Ethernet-Verbindung (Abbildung 28).

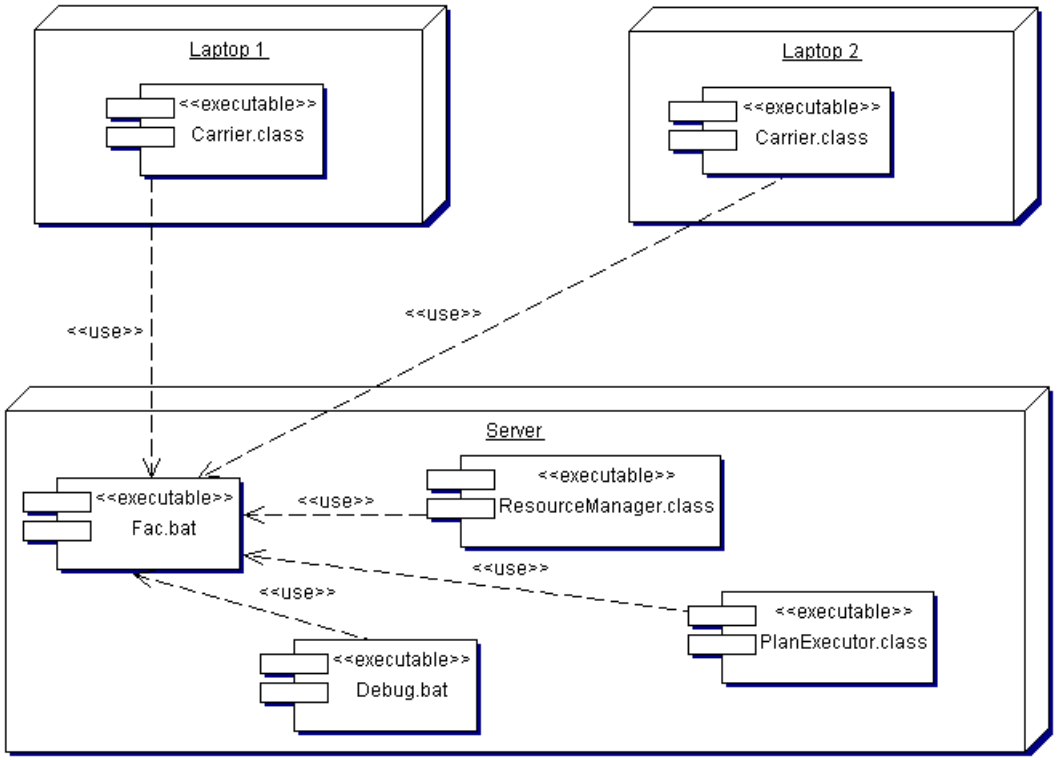


Abbildung 28. Eine typische Verteilung von Agenten.

Der Grund für die in Abbildung 28 dargestellte Verteilung liegt darin, daß jeder Roboter einen eigenen Laptop mit sich trägt, auf dem jeweils ein Agenten-Programm läuft. Die anderen Agenten können auf einem Server-Rechner ausgeführt werden.

Im Weiteren werden die aufgezeigten Komponenten ausführlich dargestellt.

7.5 PlanUnit

Das `PlanUnit` Paket enthält die Planrepräsentation und implementiert die Planausführung im Multiagentensystem. Das Paket beinhaltet einen Planserver, der die Pläne zur Verfügung stellt, und Klassen, welche die jeweiligen Primitive des Plans ausführen. Das Klassendiagramm für das `PlanUnit` Paket ist in Abbildung 29 aufgezeigt.

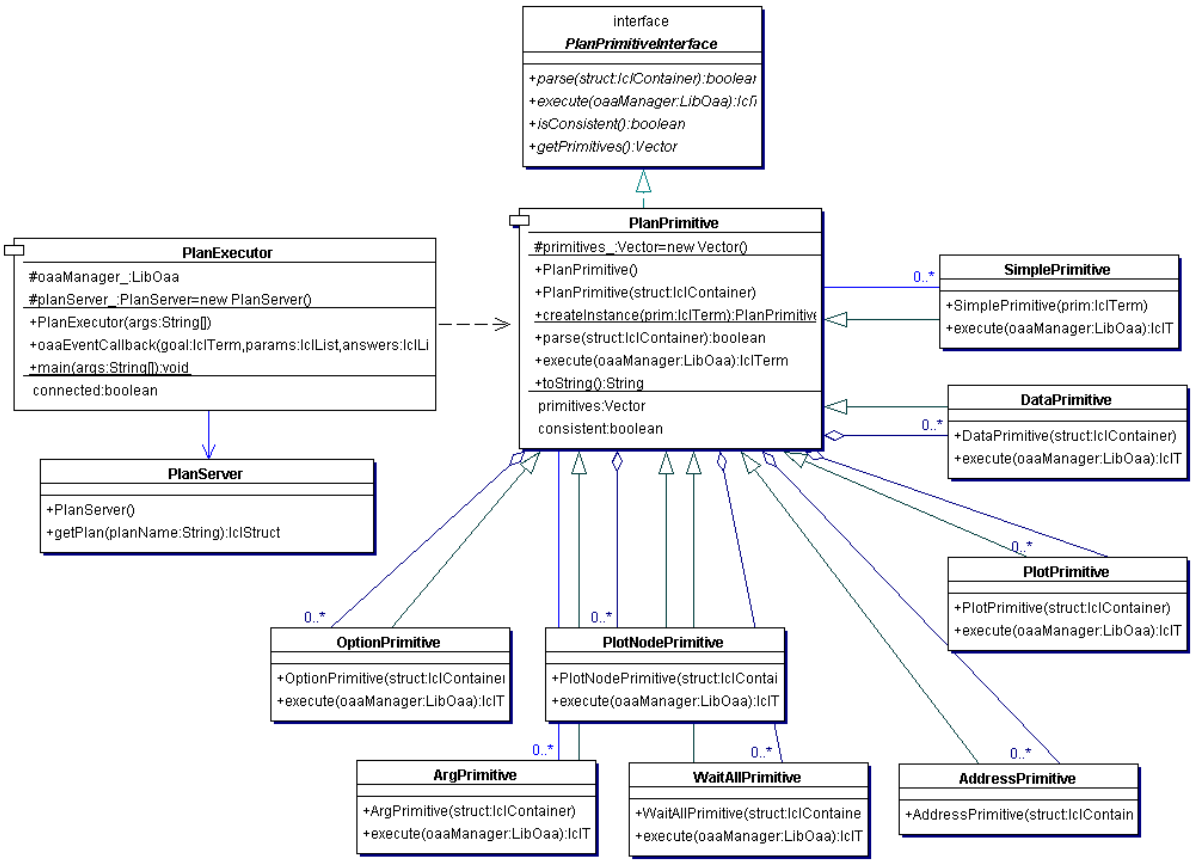


Abbildung 29. Klassendiagramm für das PlanUnit Paket.

Zur Planausführung wird im Paket die `PlanExecutor` Klasse benutzt. Eine textuelle Planrepräsentation wird von der `getPlan` Methode der `PlanServer` Klasse erstellt und in der `createInstance` Methode der `PlanPrimitive` Klasse analysiert. Für jedes Element des Plans (Planprimitiv) wird eine Instanz einer der von `PlanPrimitive` abgeleiteten Klassen erzeugt, die dieses Element im Programm bearbeiten.

Da der Schwerpunkt dieser Arbeit nicht auf der Planung lag, enthält z. Z. die `PlanServer` Klasse einen einzigen Plan, der im Quellcode fest kodiert ist (s. Abschnitt 8.2).

Aus Flexibilitätsgründen implementiert jedes Primitiv ein Composite Design Muster ([Gamma95]). Zur Ausführung eines Plans werden rekursiv die Ausführungsfunktion der Nachfolger aufgerufen, bis alle Knoten im Plangraphen abgearbeitet sind.

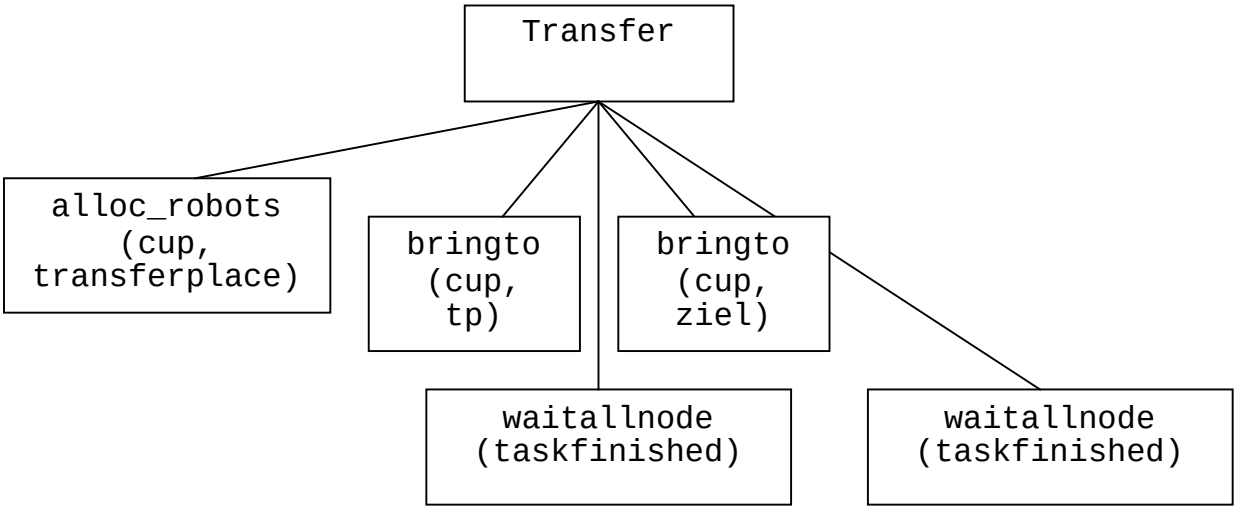
7.5.1 Planrepräsentation

Im Weiteren wird die Implementierung des ausgewählten Ansatzes zur Planrepräsentation dargestellt. Die Weltrepräsentation ist nicht notwendig, da für die Übergabeoperation mit den angegebenen Bedingungen kein explizites Weltmodell nötig ist.

Die Planrepräsentation basiert auf dem oben aufgezeigten Act-Formalismus und erforderlichen Änderungen in Bezug auf die OAA Architektur. Folglich vereinigt die ausgewählte Planrepräsentation die Act- mit der ICL-Syntax. Da der Schwerpunkt dieser Arbeit primär auf der Planrepräsentation, sowie der Planausführung und nicht auf der Planung

(Planerzeugung) lag, wurden die vom Act-Formalismus vorgegebenen Angaben von Konditionen in der Umwelt nicht implementiert.

Abbildung 30 zeigt einen Plangraphen für die Übergabeoperation und eine dazu notwendige Planrepräsentation.



tp – transferplace (Übergabestelle)

Abbildung 30. Plangraph für die Übergabeoperation.

Eine Textrepräsentation dieses Planes sieht wie folgt aus:

```
plot
   ((plotnode
        (alloc_robots(args(Object, Transferplace), options(block(true)))),

   (plotnode
        (bringto
            (args(cup, transferplace))),
            options(block(true)),
               solution_limit(1)), address(data(free_robots1(Robot1)))))))))))),
   (waitallnode
        (plotnode
            (taskfinished
                  (args(dobringtoab),
                  options(block(true),
                  solution_limit(1),address(data(free_robots1(Robot1)))))))),
   (plotnode
        (bringto(args(cup, finish)),
        options(block(true)),solution_limit(1),address(data(free_robots2(Robot2)))),

   (waitallnode
        (plotnode
            (taskfinished
               (args(dobringtoac),
            options(block(true),solution_limit(1),address(data(free_robots2(Robot2)))))))))
```

In der aufgezeigten Planrepräsentation sind folgende Primitive definiert (vgl. Abschnitt 4.5.1):

- Das `plot` Primitiv.
  Das `plot` Primitiv stellt die Plotspezifikation aus dem Act-Formalismus dar.

- Das `plotnode` Primitiv.
  Das `plotnode` Primitiv stellt im entwickelten Multiagentensystem ein `achieve` Metaprädikat aus dem Act-Formalismus dar.
- Das `waitallnode` Primitiv.
  Das `waitallnode` Primitiv faßt die gemeinsame Funktionalität der `achieve-all` und `wait-until` Act-Prädikate zusammen. Es wartet solange, bis alle enthaltenen Primitive ausgeführt sind und eine Bedingung erfüllt ist.
- Primitive für Agentendienste, die `alloc_robots` und `bringto` Primitive;
- ICL Primitive, z.B. das `address` Primitiv (s. Abschnitt 6.2.2.2).

Einige Aktionen werden als Agentendienste implementiert. Für die Übergabeoperation sind folgende Dienste notwendig:

- `alloc_robots(Object, Transferplace)` im `resourcemanager` Agenten – das Suchen zweier freier Roboter, welche die Übergabeoperation für den Gegenstand `Objekt` („cup“) und Übergabestelle `Transferplace` ausführen können; das Vermerken ihrer Adressen im Multiagentensystem für die weiteren Aktionen.
- `bringto(Objekt, Stelle)` – der Transport eines Gegenstandes `Objekt` („cup“) zur Stelle `Transferplace`.
- `taskfinished(Operation)` – das Warten, bis eine Operation abgeschlossen ist.

Im Weiteren wird die Vorgehensweise bei der Implementierung der Klassen für neue Primitive in der Planrepräsentation aufgezeigt:

- Neue Klassen müssen von der `PlanPrimitive` Klasse oder deren Unterklassen abgeleitet sein.
- Aufnahme der Erzeugungsoperatoren von Instanzen neuer Primitive durch Erweiterung der `createInstance` Methode der `PlanPrimitive` Klasse;
- Umdefinition der `execute` Methode neuer Primitivklassen, falls sie anders ausgeführt werden, als das die `execute` Methode der Basis-Klasse vorgibt (für die Beschreibung der `execute` Methode siehe den nächsten Abschnitt). Default wird eine textuelle Darstellung des Primitivs ohne Änderungen, so wie sie in der Planrepräsentation stand, ausgegeben.

Die genaue Beschreibung davon, wie die Primitive ausgeführt werden, ist im nächsten Abschnitt aufgezeigt.

7.5.2 Planausführung

Nachdem die Planrepräsentation analysiert wurde und entsprechende Instanzen von Planprimitiv-Klassen erzeugt sind, wird der Plan ausgeführt. Jede Planprimitiv-Klasse enthält für diesen Zweck eine Ausführungsmethode, die `execute` Methode, die eine spezifische Implementierung für das jeweilige Planprimitiv realisiert.

Die Ausführung bzw. die Interpretation eines Primitivs ist sehr unterschiedlich. In einigen Fällen gibt die `execute` Methode eine textuelle Darstellung des Primitivs zurück, z.B. wird für das ICL-Primitiv `block(true)` im Ausdruck `option(block(true))` als Resultat der String `block(true)` zurückgegeben. In diesem Fall unterschiedet sich die textuelle Darstellung des Primitivs und der String nicht wesentlich voneinander.

Aus Flexibilitätsgründen kann jedoch jede Primitiv-Klasse eine Sammlung von weiteren Primitiven enthalten. In diesem Fall werden zusätzlich enthaltene Primitive rekursiv ausgeführt. Beispielsweise wird der Ausdruck `option(block(checkFree(Robot1, Transferplace)))` ausgeführt, indem zuerst der Agentendienst `checkFree` mit Argumenten `Robot1` und `Transferplace` und danach das `block` Planprimitiv mit dem Ergebnis der Ausführung des `checkFree` Primitivs als Argument ausgeführt werden.

Die `execute` Methode der `PlanPrimitive` Klasse und deren abgeleiteter Klassen hat folgende Bedeutung:

- Die `execute` Methode der `PlanPrimitive` Klasse ist eine allgemeingültige Methode, die eine textuelle Darstellung des Primitivs zurückgibt, ohne eine weitere Bearbeitung durchzuführen;
- Die `execute` Methode der `PlotPrimitive` Klasse gibt eine Darstellung der Plotspezifikation zurück (s. auch Abschnitt 4.5.1);
- Die `execute` Methode der `PlotNodePrimitive` Klasse implementiert die Funktionalität des `achieve` Act-Prädikates (s. Abschnitt 4.5.1) mit allen ICL-Prädikaten (s. Abschnitt 6.2.2.2), z.B. `plotnode(bringto(...), options(...))`. In vielen Fällen wird dabei ein Dienst eines Agenten in Anspruch genommen;
- Die `execute` Methode der `WaitForAllPrimitive` Klasse führt die angegebenen Primitive solange aus, z.B. ein `waitallnode(plotnode(taskfinished))` Primitiv, bis jedes davon ein Ergebnis zurückgibt. Somit hat das `waitallnode` Primitiv die Bedeutung der `achieve-all` und `wait-until` Act-Prädikate;
- Die `execute` Methode der `DataPrimitive` Klasse gibt den Wert einer Variablen zurück, die i.A. am schwarzen Brett gespeichert ist ([Hayes-Roth85]), z.B. `data(free_robots1(Robot1))`;
- Die `execute` Methoden weiterer Klassen geben folgende Werte zurück:
  - die `execute` Methode der `SimplePrimitive` Klasse – den Wert einer ICL Konstante;
  - die `execute` Methode der `AddressPrimitive` Klasse –eine textuelle Darstellung einer ICL Adresse;
  - die `execute` Methode der `OptionPrimitive` Klasse – eine textuelle Darstellung der Optionen eines `plotnode` Primitivs;
  - die `execute` Methode der `ArgPrimitive` Klasse – eine textuelle Darstellung der Argumente eines `plotnode` Primitivs.

7.6 ResourceUnit

Das `ResourceUnit` Paket beinhaltet Funktionen für die Verwaltung der Ressourcen von Akten. Das Klassendiagramm ist in Abbildung 31 dargestellt.

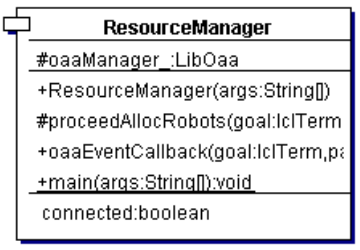


Abbildung 31. Das Klassendiagramm für das ResourceUnit Paket.

Das `ResourceUnit` Paket implementiert einen `resourcemanager` Agenten mit der `alloc_robots` Fähigkeit (s. Abschnitt 7.5.1).

7.7 CarrierUnit

Das `CarrierUnit` Paket implementiert einen Hardware-Agenten mit seinen Fähigkeiten (Abbildung 32).

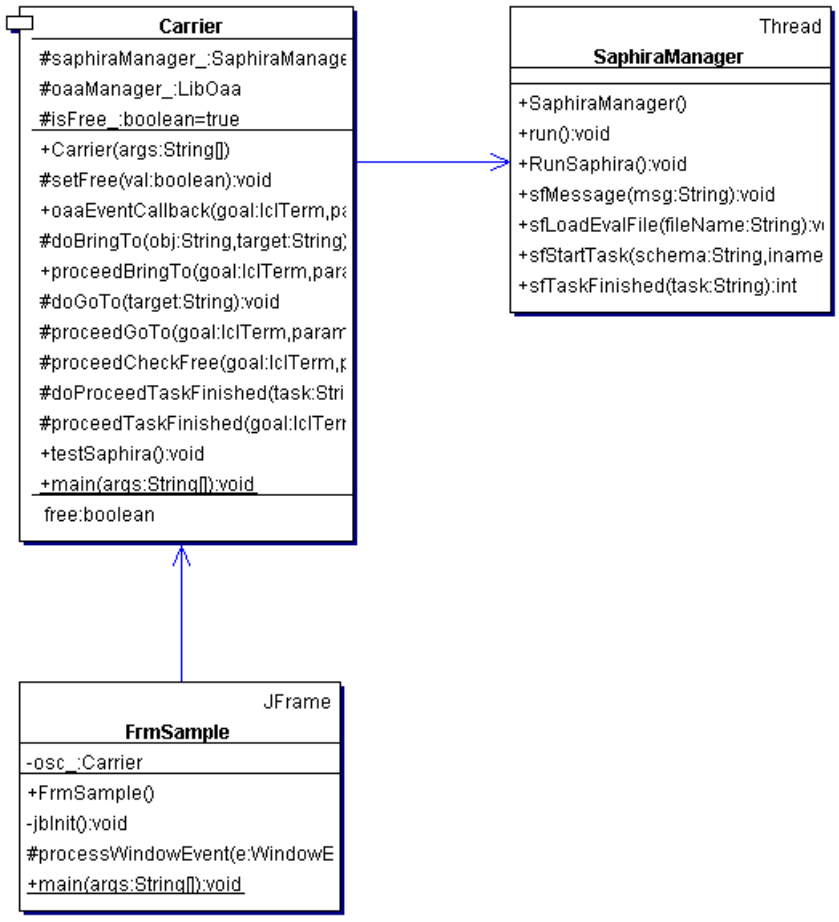


Abbildung 32. Das Klassendiagramm für das CarrierUnit Paket.

Die `Carrier` Klasse ist eine Schnittstelle zur OAA, während die `SaphiraManager` Klasse eine Schnittstelle zum Saphira API (s. Abschnitt 7.3.2) auf Basis vom JNI Interface ist ([Stearns01]). Die Namensgebung der Methoden dieser Klasse entspricht der Namensgebung aus dem Saphira Developer Guide [Konolige00]. Die `FrmSample` Klasse beinhaltet eine Funktion zum Testen des JNI Interfaces.

Der `Carrier` Agent führt folgende Dienste aus (vgl. Abschnitt 7.5.1):

- der `bringto(cup, tp)` Dienst ermöglicht den Transfer von Gegenstand `cup` zur Übergabestelle `tp`;
- Dienst `taskfinished(Operation)` prüft, ob eine Operation in Saphira abgeschlossen ist;
- Dienst `checkFree(cup,tp)` prüft, ob ein `Carrier` Agent frei ist.

8 Fazit

8.1 Resümee

Im Rahmen dieser Diplomarbeit wurde ein Multiagentensystem sowohl mit Hardware- als auch mit Software-Agenten auf Basis der OAA Architektur entwickelt. Die Funktionsweise wurde am Beispiel eines Szenarios zur Übergabe eines Gegenstandes veranschaulicht. In Bezug auf die Fragen, die im Mittelpunkt standen (s. auch Kapitel 1), sind folgende Ergebnisse festzustellen:

- Die ausgewählte Planrepräsentation auf Basis des Act-Formalismus ist sowohl für die Planung, als auch für die Planausführung in einem in einer dynamischen Umwelt funktionierenden Multiagentensystem geeignet. Sie modelliert dynamische und statische Aspekte von Plänen. Die ausgewählte Planrepräsentation ist hierarchisch und läßt sich leicht erweitern und verändern (s. Abschnitt 7.5.1).

  Der Umfang der implementierten Planrepräsentation richtete sich nach der Planrepräsentation und der Planausführung des angegebenen Szenarios. Aus diesem Grunde sind die Planprimitive, die auf die Planung (Planerzeugung) hinauslaufen, wie Konditionen in der Umwelt, nicht realisiert. Die Vorgehensweise bei der Programmierung neuer Planprimitive wurde in Abschnitt 7.5.1 dargestellt. Planausführung im entwickelten Multiagentensystem wurde in Abschnitt 7.5.2 aufgezeigt.

- Das entwickelte Multiagentensystem kann in einer heterogenen verteilten Umgebung funktionieren und ist erweiterbar, wobei die einzelnen Komponenten austauschbar und wiederverwendbar sind. Die Grundlage dafür sind die Erweiterbarkeit, die Wiederverwendbarkeit, die Austauschbarkeit und die Plattformunabhängigkeit der OAA Architektur und deren Implementierung (s. Abschnitt 6.2.2).

8.2 Systemerweiterungen

Folgende Erweiterungen des aufgezeigten Multiagentensystems sind vorzuschlagen:

1. Unterstützung der natürlichsprachlichen Eingabe auf Basis des OAA Nlg-Agenten (s. Abschnitt 7.3.1). Die OAA Distribution beinhaltet einen Agenten zur Unterstützung der natürlichsprachlichen Eingabe. Der Nlg-Agent erzeugt aus eingegebenen Sätzen Prolog-ähnliche Anfragen auf Grundlage einer Datenbank mit Wörtern und deren grammatikalischen Bedeutungen.
2. Realisierung weiterer Act-Primitive, z.B. zur Unterstützung von Konditionen in der Umwelt aus dem Act-Formalismus (s. auch Abschnitt 4.5.1). Eine Implementierung von Konditionen in der Umwelt schafft eine gute Grundlage zur Realisierung eines Planungsmoduls (s. auch Abschnitt 4.5.2).
3. Planrepräsentation und Darstellung in einem neutralen Format, z.B. im XML Format. Ein Ansatz zur XML Repräsentation von Verhalten ist in [Jaghoori01] nachzuschlagen. Das VGJ Tool zur visuellen Darstellung und Bearbeitung von Plänen in XML Format, das zur Erstellung von Planbibliotheken benutzt werden kann, findet sich in [VGJ01].
4. Realisierung eines Moduls zur Planung bzw. Umplanung (s. auch Punkt 2). Wenn die Planung auf unvollständigen oder vagen Daten basiert, z.B. in einem System mit

Robotern, kann die Ausführung des geplanten Vorgangs dazu führen, daß die Abweichungen zwischen dem geplanten und dem tatsächlichen Vorgang so groß sind, daß eine einfache Korrektur des Planes unzureichend ist und es erforderlich ist, neu zu planen. Dementsprechend kann die `PlanServer` Klasse in einen Planungsagenten mit der Fähigkeit `getPlan` umgewandelt werden. Die Klassenschnittstelle ist so konzipiert, daß dies keinen größeren Aufwand zur Folge haben sollte (s. Abschnitt 7.5).

8.3 Ausblick

Um eine bessere Qualität des Multiagentensystems in Bezug auf die Planung und die Planausführung in einem hybriden System, sowie auf die Methodik der Entwicklung von Multiagentensystemen zu gewährleisten, können in den nächsten Versionen des Systems folgende Aspekte weiter erforscht werden:

1. Planung vs. Planausführung:

   - Das Frame Problem (s. Abschnitt 3.2)

     Ist das Frame Problem allgemeingültig nicht lösbar ([vanBrakel92]), so gibt es Ansätze zu seiner praktischen Lösung ([Morgenstern96]):

     1. *Monotone Ansätze*. Monotone Ansätze nehmen axiomatisch an, daß die Umwelt unverändert bleibt, wenn eine Aktion ausgeführt wird (s. auch [Reiter91]);
     2. *Prozedurale Ansätze*. Prozedurale Ansätze basieren darauf, daß eine Reihenfolge von Aktionen bestimmt wird, um das System von einem Zustand in einen anderen zu überführen. Ein Beispiel für einen solchen Ansatz ist der STRIPS-Planer (s. Abschnitt 4.4.1);
     3. *Nichtmonotone Ansätze*. Nichtmonotone Ansätze machen Gebrauch von der nichtmonotonen Logik, die in die konventionelle Logik Annahmen und Ausnahmen einführt. Grundlegende Informationen über nichtmonotone Ansätze sind in [McCarthy80] nachzuschlagen;
     4. *Statistische und probabalistische Ansätze*. Diese Ansätze setzen die Planung unter Unsicherheit und basieren auf der Wahrscheinlichkeitstheorie ([Pearl91]).

     In weiteren Versionen des aufgezeigten Multiagentensystems kann die Wissensrepräsentation daraufhin untersucht werden, wie das Frame Problem und die Ansätze zu seiner Lösung die Wissensdarstellung beeinflussen können.

- Umplanung

  Wie bereits erwähnt, kann es erforderlich sein, einen Umplanungsvorgang durchzuführen, wenn die Planung auf unsicheren und vagen Daten basiert. Dadurch wird eine genauere Planausführung möglich und die Qualität des Systems gesteigert.

  In den nächsten Versionen des Multiagentensystems kann außer einem Planungsmodul ein Umplanungskonzept entwickelt und umgesetzt werden. Beispiele für einige Ansätze zur Umplanung sind in [Beetz94] und [Atkins01] nachzulesen.

- Erweiterte Planrepräsentation

  Die Ergebnisse der Erforschung des Frame Problems und der Umplanung können eine entsprechende Erweiterung der Planrepräsentation notwendig machen. In den nächsten Versionen des Multiagentensystems wäre es erforderlich, die Planrepräsentation auf die Lösung des Frame Problems und auf das Konzept der Umplanung anzupassen.

2. Methodik:

   - Agent-Oriented Software Engineering

     Das Bereitstellen der notwendigen Mittel, mit denen der Entwicklungsprozeß von Multiagentensystemen standardisiert werden kann, hat zur Entstehung des Agent-Oriented Software Engineerings geführt. Das Agent-Oriented Software Engineering beschreibt Methodiken, um die Analyse, das Design und die Implementierung von Multiagentensystemen zu verbessern ([Wood00]). Weitere Informationen sind in [UniTrento01] nachzulesen.

     In den nächsten Versionen des Multiagentensystems können die Ansätze aus diesem Bereich berücksichtigt und vertieft werden.

   - Ein IDE zur Entwicklung und zum Debugging von Multiagentensystemen

     Zur praktischen Umsetzung von Methodiken des Agent-Oriented Software Engineerings, sowie zur weiteren Forschung und zur Experimentierung in diesem Bereich kann in den nächsten Versionen ein IDE zur Entwicklung und zum Debugging von Multiagentensystemen implementiert werden. Eine Übersicht über existierende Systeme in diesem Umfeld gibt [AgentBuilder01].

9 Anhang

9.1 Programmcode

Der Quellcode des entwickelten Multiagentensystems ist über [PioneerProject01] verfügbar. Die Verzeichnisstruktur ist selbsterklärend. Zum Quellcode gibt es eine Dokumentation im Javadoc-Format.

9.2 Installation und Start des Multiagentensystems

Notwendige Schritte zur Installation und zum Start des Multiagentensystems:

1. Prüfen, daß der Port 3378 freigegeben ist.
2. Die Rechnernamen in der „setup.pl“ Datei sowohl im OAA Verzeichnis, als auch im Arbeitsverzeichnis des Multiagentensystems anpassen. Die Namen werden klein geschrieben.
3. OAA installieren. Die OAA Distribution findet sich in [SRI4].
4. Verbindung mit einer Saphira-Installation herstellen (z.B. „`Start, ConnectTo, Roboter, Saphira for Pioneer`“) oder Saphira lokal installieren. Die Saphira Distribution findet sich in [Konolige00]. Die `dobringtoab.act` und `dobringtoac.act` Skripts in das Colbert-Verzeichnis der Saphira Installation kopieren.
5. Prüfen, ob das Verzeichnis mit SaphiraWrapper.dll in der `PATH` Variablen vorhanden ist.
6. Den OAA Facilitator starten und anschließend das StartOaaSaphira.bat File ausführen.
7. Im Debug-Agenten `oaa_Solve(execute(transfer), [])` eingeben.

9.3 Act Syntax in der BNF Form

```
wff          ::=   (pred-name ) | (UNKNOWN (pred-name {term})) |
                   (NOT wff) | (AND {wff}+) | (OR {wff}+)
term         ::=   simple-term | function | (REBIND variable )
simple-term  ::=   individual | variable
variable     ::=   {class} . {integer}
function     ::=   (fn-name {term}*)
pred-name    ::=   the name of a predicate
fn-name      ::=   the name of a function
individual   ::=   a domain ob|ect
class        ::=   the name of a class
integer      ::=   a positive integer

meta-pred    ::=   test | conclude | achieve | achieve-by
                   | use-resource | wait-until | require-until
test   ::=         (TEST {wff | wff-list} )
achieve      ::=   (ACHIEVE {wff | wff-list } )
achieve-by   ::=   (ACHIEVE-BY {wff+acts | (fwff+acts}+) )
conclude     ::=   (CONCLUDE {wff | wff-list}
```

9.4 Danksagung

An dieser Stelle möchte ich mich bei den Menschen bedanken, die diese Arbeit in vielfältiger Art und Weise unterstützt haben. Besonders möchte ich Herrn Prof. von Luck für sein persönliches Engagement und seine Anregungen danken. Bei Herrn Prof. Klauck bedanke ich mich für die Zweitbegutachtung dieser Arbeit und seine konstruktive Kritik. Weiterhin ein Dankeschön an Herrn Prof. Kahlbrandt für seine Unterstützung bei der Erstellung der UML-Diagramme. Für eine ausgesprochen plausible Beratung in Bezug auf die Zeit- und Rahmenbedingungen danke ich Herrn Prof. Fähnders. Bei Herrn Dipl.-Inform. Wichern und Herrn Dipl.-Math. Arnold bedanke ich mich für ihre Korrekturen, die dieser Ausarbeitung eine bessere sprachliche Qualität gegeben haben. Für die technische Unterstützung danke ich Herrn Abrams. Zum Schluss möchte ich mich bei meiner Frau Olga, die mit ihrem Verständnis, ihrer Rücksichtnahme und nicht zuletzt ihrer Motivationsgabe mir den benötigten Freiraum während der letzten Zeit einräumte.

9.5 Versicherung

Hiermit versichere ich, daß ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §24(4) bzw. §25(4) ohne fremde Hilfe selbständig verfaßt und nur die angegebenen Hilfsmittel benutzt habe.

Ort, Datum Unterschrift

Abbildungsverzeichnis

Literaturverzeichnis

1. [ActiveMedia01]. Pioneer Documentation. URL: http://robots.activmedia.com/docs. ActiveMedia. 2001.

2. [AgentBuilder01]. Agent Construction Tools. Reticular Systems, Inc. URL: http://www.agentbuilder.com/AgentTools. 2001.

3. [Atkins01]. Atkins E. M., Abdelzaher T. F., Shin K.G., Durfee E. H. Planning and Resource Allocation for Hard Real-time, Fault-Tolerant Plan Execution. Autonomous Agents and Multi-Agent Systems, 4, S. 57-78, Kluwer Academic Publishers. 2001.

4. [Bastié95]. Bastié C., Régnier P. Planning and execution in a dynamic environment: the supervision of execution in SPEEDY. Department of Computer Science, University of Essex. URL: http://cswww.essex.ac.uk/conferences/ukpssig/essex-14/regnier-uue-2.ps.gz. 1995.

5. [Beetz94]. Beetz M., McDermott D. Improving Robot Plans During Their Execution. Yale University, Department of Computer Science. mailto: beetz@cs.yale.edu, mcdermott@cs.yale.edu. URL: http://www.informatik.uni-bonn.de/~rhino/publications /improving_robot_plans_during_execution.ps.gz. 1994.

6. [Brenner98]. Brenner W., Zarnekow R., Wittig H. Intelligente Softwareagenten. Grundlagen und Anwendungen. Springer Verlag. 1998.

7. [Brooks86]. Brooks, R. A. "A Robust Layered Control System for a Mobile Robot". IEEE Journal of Robotics and Automation. URL: http://www.ai.mit.edu/people/brooks /papers/AIM-864.pdf. Vol. 2, No. 1, March 1986, pp. 14-23; also MIT AI Memo 864, September 1985.

8. [Burkhard00]. Burkhard H.-D. Software-Agenten. In: Görz G. (Hrsg.), Rollinger C.-R., Schneeberger J. Handbuch der künstlichen Intelligenz. 3 Auflage. S. 941-1018. Oldenbourg Wissenschaftsverlag GmbH. 2000.

9. [Ferber99]. Ferber J. Multi-Agent Systems. An Introduction to distributed artificial intelligence. Addison-Wesley. 1999.

10. [Fikes72]. Fikes R. E., Hart P. E., Nilsson N. J. Learning and executing generalized robot. Artificial Intelligence, 3 : 251-288. 1972.

11. [Finin97]. Finin T., Labrou Y., Mayfield J., Bradshaw J. (Ed.). KQML as an agent communication language. In: „Software Agents“, MIT Press, Cambridge, to appear, 1997.

12. [Gamma95]. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns. Elements of Reusable Object-Oriented Software. Addison Wesley. 1995.

13. [Guzzoni97]. Guzzoni D., Cheyer A., Julia L., Konolige K. Many Robots Make Short Work. AAAI Journal. 1997.

14. [Haddadi96]. Haddadi, A. Communication and cooperation in agent systems: a pragmatic theory. Springer Verlag. 1996.

15. [Hayes-Roth85]. Hayes-Roth B. A Blackboard Architecture for Control. Artificial Intelligence 26(3). P. 251-321. 1985.

16. [Heinsohn99]. Heinsohn J., Socher-Ambrosius R. Wissensverarbeitung. Eine Einführung. Spektrum Akademischer Verlag. 1999.

17. [Jaghoori00]. Jaghoori B., Hinrichs M. FH Hamburg. URL: http://www.informatik.fh-hamburg.de/~pioneer/Projekte/projekte.html. 2000.

18. [Jaghoori01]. Jaghoori B. Ein Baukasten Verhaltensbasierter Module für Agentenprogramme. FH-Hamburg. URL: http://www.informatik.fh-hamburg.de/~pioneer/Projekte/baschir/html/index.html. 2001.

19. [JATDemo01]. JATLite Online Demo. Standford University. URL: http://java.stanford.edu/JATLiteDemo.html. 2001.

20. [Konolige00]. Konolige K. SRI International Artificial Intelligence Center. SAPHIRA Robot Control System. URL: http://www.ai.sri.com/~konolige/saphira. mailto: konolige@ai.sri.com. 2000.

21. [KQML93]. Finin T., Weber J., Wiederhold G., Genesereth M., Fritzson R., McKay D., McGuire J., Pelavin R., Shapiro S., Beck C. KQML Specification Document. URL: http://www.cs.umbc.edu/kqml/kqmlspec.ps. 1993.

22. [Kühnel01]. Kühnel R. Agentenbasierte Softwareentwicklung. Aufgaben und Anwendungen. Addison-Wesley Verlag. 2001.

23. [Lüth98]. Lüth T. Technische Multiagenten Systeme: verteilte autonome Roboter- und Fertigungssysteme. Carl Hanser Verlag. 1998.

24. [Martin99]. Martin D. L., Cheyer A. J., Moran D. B. The Open Agent Architecture: A Framework for Building Distributed Software Systems. SRI International Artificial Intelligence Center. URL: http://www.ai.sri.com/~cheyer/papers/oaa.pdf. mailto: {fmartin,cheyer,morang}@ai.sri.com.

25. [Maurer01]. Maurer H. Die (Informatik-)Welt in 100 Jahren. Informatik Spektrum. Band 24. Heft 2. Springer Verlag. April 2001.

26. [McCarthy69]. McCarthy J., Hayes P. J. Some Philosophical Problems From The Standpoint Of Artificial Intelligence. Computer Science Department. Stanford University. URL: http://www-formal.stanford.edu/jmc/mcchay69.pdf. 1969.

27. [McCarthy80]. McCarthy J. Circumscription - a form of nonmonotonic reasoning. Artificial Intelligence, 13. S. 27-40. URL: http://www-formal.stanford.edu/jmc/circumscription/circumscription.html. 1980.

28. [Morgenstern96]. Morgenstern L. The Problem with Solutions to the Frame Problem. Third Symposium on Logical Formalization of Commonsense Reasoning. URL: http://www-formal.stanford.edu/leora/fp.ps. 1996.

29. [Myers00]. Myers K. L. Artificial Intelligence Center. SRI International. A Procedural Knowledge Approach to Task-level Control. mailto: myers@ai.sri.com. URL: http://www.ai.sri.com/~prs/prslite.ps.

30. [Myers97]. Myers K. L., Wilkins D. E. The Act Formalism. SRI International Artificial Intelligence Center. URL: http://www.ai.sri.com/~act/act-spec.ps. 1997.

31. [OAA01]. Open Agent Architecture (OAA) Developer's Guide. SRI International Artificial Intelligence Center. URL: http://www.ai.sri.com/~oaa/sri-private/doc/proguid2.ps. 2001.

32. [Owsnicki-Klewe00]. Owsnicki-Klewe B., v. Luck K., Nebel B. Wissensrepräsentation und Logik – eine Einführung. In: Görz G. (Hrsg.), Rollinger C.-R., Schneeberger J. Handbuch der künstlichen Intelligenz. 3 Auflage. S. 153-197. Oldenbourg Wissenschaftsverlag GmbH. 2000.

33. [Pearl91]. Pearl J. Probabilistic reasoning in intelligent systems. San Mateo, CA: Morgan Kaufmann. 1991.

34. [Pioneer00]. Pioneer Documentation. ActiveMedia. URL: http://robots.activmedia.com. 2000.

35. [PioneerProject01]. Das Pioneer Project. FH Hamburg. URL: http://www.informatik.fh-hamburg.de/~pioneer. 2001.

36. [Prasad01]. Prasad N., Haynes A., Haynes T. Learning in Multi-Agent Systems Webliography. University of Massachusetts. URL: http://dis.cs.umass.edu/research/agents-learn.html.

37. [PRS01]. PRS-CL: A Procedural Reasoning System. SRI International Artificial Intelligence Center. URL: http://www.ai.sri.com/~prs. 2001.

38. [Rao91]. Rao A. S. and Georgeff M. P. Modeling rational agents within a BDI-architecture. In J. Allen, R. Fikes, and E. Sandewall, editors, Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning (KR'91), S. 473-484. Morgan Kaufmann. URL: http://citeseer.nj.nec.com/rao91modeling.html. 1991.

39. [Reiter91]. Reiter R. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy, Vladimir Lifschitz (ed.), Academic Press, San Diego, CA, S.359-380. URL: http://www.cs.toronto.edu/cogrobo/simple.ps.Z. 1991.

40. [Russell95]. Russell S., Norvig P. Artificial Intelligence: A Modern Approach. Prentice Hall, Englewood Cliffs, NJ. 1995.

41. [Schneeberger00]. Schneeberger J. Planen. In: Görz G. (Hrsg.), Rollinger C.-R., Schneeberger J. Handbuch der künstlichen Intelligenz. 3 Auflage. S. 491-515. Oldenbourg Wissenschaftsverlag GmbH. 2000.

42. [Sipe01]. SIPE-2: System for Interactive Planning and Execution. SRI International Artificial Intelligence Center. URL: http://www.ai.sri.com/~sipe. 2001.

43. [SRI1]. OAA® 2.0 Documentation. SRI International Artificial Intelligence Center. URL: http://www.ai.sri.com/~oaa/sri-private/distribv2/. 2001.

44. [SRI2]. Open Agent Architecture (OAA). Agent Library Reference Manual. Version 2.0. SRI International Artificial Intelligence Center. URL: http://www.ai.sri.com/~oaa/sri-private/doc/refmanual.html. 2001.

45. [SRI3]. OAA® Contact and Community. SRI International Artificial Intelligence Center. URL: http://www.ai.sri.com/~oaa/contributions. 2001.

46. [SRI4]. OAA® Documentation and Distribution. SRI International Artificial Intelligence Center. URL: http://www.ai.sri.com/~oaa/distrib.html. 2001.

47. [Stearns01]. Stearns B. Trail: Java Native Interface. Sun Microsystems. URL: http://www.javasoft.com/docs/books/tutorial/native1.1/index.html. 2001.

48. [UniTrento01]. University of Trento, University of Toronto, IRST. Agent Oriented Software Engineering Project. URL: http://www.science.unitn.it/~pgiorgio/aose. 2001.

49. [vanBrakel92]. Van Brakel J. The Complete Description Of The Frame Problem. Book Review of Ford & Hayes on the Frame-Problem. URL: http://www.monash.edu.au/journals/psycoloquy/volume_3/psyc.92.3.60.frame-problem.2.vanbrakel. 1992.

50. [VGJ01]. Drawing Graphs with VGJ. Auburn University Department of Computer Science and Software Engineering. URL: http://www.eng.auburn.edu/department/cse /research/graph_drawing/graph_drawing.html. 2001.

51. [Wagner97]. Wagner, G. Multi-Level Security in MultiAgent Systems. In: Proc. of 1st Int. Workshop on Cooperative Information Agents (CIA-97), Springer LNAI. URL: http://www.inf.fu-berlin.de/~wagnerg/cia97.ps.gz. 1997.

52. [Wilkins94]. Wilkins D. E., Myers K. L. A Common Knowledge Representation for Plan Generation and Reactive Execution. SRI International Artificial Intelligence Center. URL: http://www.ai.sri.com/~wilkins/papers/jlc-www.ps. 2000.

53. [Wilkins98]. Wilkins D. E., Myers K. E., Marie desJardins, Berry P. M. Multiagent Planning Architecture. SRI International Artificial Intelligence Center URL: http://www.ai.sri.com/~wilkins/mpa/mpa.ps. 1998.

54. [Wood00]. Wood M. F., DeLoach S. A. An Overview of the Multiagent Systems Engineering Methodology. The First International Workshop on Agent-Oriented Software Engineering (AOSE-2000), June 10, 2000 - Limerick, Ireland.