Architektur und Typisierung intelligenter Agenten

Dirk Burkamp

Universität Dortmund

Fachbereich Informatik

burkamp@poboxes.com

Erstellt im Rahmen des Seminares „Lernende Agenten im WWW“, Sommersemester 1997

Lehrstuhl für künstliche Intelligenz, Prof. Dr. Katharina Morik

Inhalt:

1 Einführung

Aufgrund des Siegeszuges der weltweiten Vernetzung durch das Internet und hier insbesondere der Informationsflut des World Wide Web gewinnt das Konzept der Agenten im Bereich der Informatik zunehmend an Bedeutung. Während Agentensoftware früher häufig nur zur Unterstützung der Informationsbeschaffung konzipiert wurde, läßt sich heute eine eindeutige Weiterentwicklung zum persönlichen digitalen Assistenten (PDA) feststellen.

Der erste Teil meiner Ausarbeitung soll einen Überblick über die verschiedenen Typen von Agenten sowie deren Architektur geben und den Leser in die Lage versetzen, diese anhand einer Klassifikation einordnen zu können.
Im zweiten Teil meiner Ausarbeitung diskutiere ich verschiedene Ansätze von Agenten-Architekturen. Diese Ausarbeitung basiert im wesentlichen auf Arbeiten von Michael Woolridge [WOOL*], der unter der Architektur eines Agenten „Modelle des Softwareengeneering“ versteht.

Die Technologie der Agenten erfordert insbesondere im Bereich des Information Retrieval sowie der künstlichen Intelligenz erhebliche Kenntnisse und Fähigkeiten; im folgenden werde ich in erster Linie auf die Aspekte im Zusammenhang mit der KI eingehen.

Wenn ich nach dem Konzept der Agenten produzierter Software im folgenden Eigenschaften wie Motivation, zielgerichtetes Handeln sowie Bewußtsein zuschreibe, geschieht das zur Erleichterung der Kommunikation und nicht weil ich Agenten für lebende Wesen halte.

2 Agententypisierung

Es ist eine Klassifizierung anhand der Mobilität der Agenten möglich. Anhand dieser können die Agenten in zwei Klassen eingeteilt werden:

- Mobile Agenten
- Fixierte Agenten

Die Agenten-Typen dieser Klassen werden in den folgenden Abschnitten noch näher charakterisiert. Diese von Woolridge vorgenommene Klassifizierung könnte um die Klasse der hybrid mobilen Agenten erweitert werden, da fixierte Agenten vorstellbar sind, die autonom mobile Unteragenten beauftragen. Jedoch halte ich diese zusätzliche Klasse für überflüssig, da an diesem Konzept wiederum nur zwei Klassen von Agenten beteiligt sind.

2.1 Begriffseinführung

Zur Klassifizierung von Agenten möchte ich im Vorfeld diverse Begriffe und Fragen einführen. Zur Typisierung der Agenten benötigen wir Begriffe und Methoden aus den Themenbereichen

- Hostsicherheit
- Agentensicherheit
- Heterogenität des Hosts

2.1.1 Hostsicherheit

Im Rahmen dieser Ausarbeitung definiere ich abweichend vom typischen Client/Server-Modell den Begriff „Host“ als den aktuellen Gastgeber eines Agenten. Wenn wir also über die Hostsicherheit sprechen, stellen wir uns primär folgende Fragen:

Welche Risiken bedrohen den Host im Zusammenhang mit Agenten?

- **Sabotageakte**

  Sabotageakte durch Agenten mit „böser“ Absicht stellen insbesondere im Hinblick auf die Agentensicherheit einen Risikofaktor dar. Eine Kontrolle des Hosts und der damit verbundenen Manipulationsmöglichkeiten der anderen Agenten ist denkbar.

  Auf feindliche Agenten lassen sich in diesem Fall die Begriffe der Computer-Sicherheit übertragen. Ein Agenten-Virus wäre demnach ein Agent, der bedrohlichen Code im System des Hosts etablieren kann. [PFLEEGER97]

- **Ressourcen-Überanspruchung**

  Durch übermäßige Beanspruchung von Rechenzeit bzw. Speicherraum wird die Integrität des Host bedroht. Ein „gieriger“ Agent könnte die gesamte Rechenzeit verbrauchen und dadurch die Ausführung der anderen Agenten verhindern. Falls es der Auftrag eines Agenten ist, die Ausführung anderer Agenten zu behindern, kann dieser als eine neuartige Klasse der Rabbits [PFLEEGER97] eingeordnet werden.

- **Seiteneffekte**

  Ein defekter Agent kann unvorhersehbare Auswirkungen auf die Systemsoftware sowie andere Agenten haben. Der Absturz eines fehlerhaft programmierten Agenten kann in einem unsicheren System sogar den Absturz des gesamten Host nach sich ziehen. Dieser Effekt tritt insbesondere bei Systemen mit ungeschützten Prozess-Speicherräumen auf.

  Ein weltbekanntes Beispiel ist, daß eine sinnvolle Arbeit mit MS-Windows 3.x nach dem Absturz eines Tasks kaum noch möglich ist. Auch diese Seiteneffekte könnten als Angriffspunkt feindlicher Agenten genutzt werden und diese als logische Bomben eingesetzt werden.

- **Korrumpierung**

  Zeigertypen unterstützende Programmiersprachen bergen die Gefahr der Korrumpierung des Speicherraumes. Ein direktes Schreiben in gemeinsam genutzte Speicherbereiche bzw. System-Speicherbereiche kann durch feindliche Agenten z.B. zur Ausspähung fremder Agenten genutzt werden. Auch durch die Korrumpierung des Speicherraumes kann der Host zum Absturz gebracht werden.

- **Fehlnutzung**

  Ein wesentlich komplexerer Bereich ist die Fehlnutzung von legalen Aktionen. Manche Aktionen sind unter normalen Umständen harmlos, können jedoch bei Fehlnutzung schnell gefährlich werden.

  Zum Beispiel ist das Absenden einer eMail in der Regel harmlos und kann durch den Agenten zur Kommunikation mit seinem Auftraggeber genutzt werden. Jedoch sind Szenarien denkbar, in denen der eMail-Versand zum Problem wird.

  Ein aktuelles Problem des Internets ist der Versand von Massen eMail. Was geschieht bei einem SPAM-Agenten, der darauf programmiert wurde, eMail-Adressen potentieller Kunden zu sammeln und Werbung an diese zu versenden? Wer übernimmt die Kosten für einen eMail-Versand eines Agenten, der durch den Host ausgeführt wurde?

  Entscheidend für die Antwort auf diese Fragen ist jeweils die Intention der durch den Agenten ausgeführten Aktionen.

Welche Schutzmöglichkeiten gibt es?

- Viele Agenten-Sprachen unterstützen **keine Zeigertypen** (z.B. JAVA); dadurch wird das direkte Schreiben in geschützte Speicherbereiche des Systems verhindert. Diese Sprachen müssen natürlich geeignete Konstrukte zur Intersystem-Kommunikation zur Verfügung stellen. Diese sollte eine Interagenten-Kommunikation beinhalten, da diese eine der grundlegenden Agententechniken darstellt.

- UNIX-ähnliche **Zugriffsrechte** auf alle Ressourcen ermöglichen eine abgestufte Limitierung der Arbeitsmöglichkeiten der Agenten.

- Es sind **sichere Bibliotheken** notwendig, die den Dateizugang, die Speicherung, den Prozeßraum und vieles mehr zur Verfügung stellen müssen. Diese normierten Bibliotheken sind an das jeweilige Hostsystem angepaßt und bieten den Agenten nur „legale“ Zugriffsmöglichkeiten an. Das zur Zeit populärste Beispiel für solche Agenten-Bibliotheken ist die Programmiersprache JAVA. Wie Rainer Stoll in seinem Vortrag referierte, stellt diese über die o.a. Standard-Bibliotheken hinaus Klassen zur Netzwerksteuerung und GUI-Gestaltung zur Verfügung [FLANAGAN96].

- Es müssen Möglichkeiten geschaffen werden, die **Ressourcennutzung zu limitieren**. Ein Ansatzpunkt hierzu wären die Agenten-Sprachen (z.B. TELESCRIPT) aber auch betriebssystemseitig wäre eine Limitierung möglich.

- Einige Agenten-Konzepte bieten die Möglichkeit der **Überprüfung der Agentensicherheit** bei deren Erhalt. Dieses Konzept sollte jeweils durch eine Laufzeitüberwachung ergänzt werden.

- Das Konzept der „**Gummizelle**“ ermöglicht eine physikalische Trennung des Hosts von den Agenten. Die Agenten arbeiten in diesem Konzept auf einem physikalisch abgetrennten Co-Prozessor in einer Art Quarantäne. Die daraus resultierende Trennung des Betriebsystemes von den Agenten stellt sicher, daß im Falle der Korrumpierung des Speicherraumes sowie bei dem Absturz eines Agenten die Systemintegrität nicht tangiert ist.
  Die Agenten „nur“ in einem gesichertem Speicherbereich auf einer virtuellen Maschine zu starten halte ich für keine geeignete Alternative, da die Abschätzung der Seiteneffekte eines Softwareabsturzes zu komplex ist.
  In diesem Zusammenhang wäre es z.B. gut vorstellbar, den von Sun angekündigten JAVA-Chip auf einer Erweiterungs-Karte zu integrieren und so den Prozessor gänzlich von der eigentlichen Agentenausführung zu entbinden.

- Der Host muß „abgestürzte“ Programme **kontrolliert behandeln** können. Speicherschutzkonzepte, getrennte Adressräume und andere Techniken der Betriebssystem-Praxis [SILBER94] stellen geeignete Möglichkeiten zum kontrollierten Abort einzelner Agenten dar.

- „**befreundete**“ Agenten.

  Beim Konzept der befreundeten Agenten werden fremden Agenten(d.h. nicht vom Benutzer beauftragte) Vertrauens-Attribute zugeordnet. Anhand dieser Attribute werden den Agenten verschiedene Rechte erteilt. (z.B. Daten lesen, Server besuchen etc.)

  Diese Lösung diverser Sicherheitsprobleme birgt völlig neue Sicherheitsaspekte in sich. Welche Daten dürfen mit befreundeten Agenten ausgetauscht werden? Wie werden befreundete Agenten zertifiziert und identifiziert?

Der Absturz eines Programmes im Zusammenhang mit Agenten wirft die Frage des weiteren Vorgehens auf. Wie soll sich der Host in diesem Fall verhalten. Eine optimistische Variante wäre ein Neustart des Agenten oder sollte der Host ohne Reaktion fortfahren. Ein sinnvolles Vorgehen wäre es sicherlich, nach dem Absturz eine eMail an den Auftraggeber abzusetzen.

2.1.2 Agentensicherheit

Welche Sicherheitsaspekte gibt es im Zusammenhang mit Agenten?

Agenten haben ein Recht auf **Privatsphäre** (engl. privacy). In diesem Zusammenhang besteht die Möglichkeit, daß die Absicht des Agenten **ausgespäht** und gegen den Auftraggeber verwandt wird. Sogar die Ausspähung der durch den Agenten gesammelten Daten läßt Rückschlüsse auf die Absicht des Agenten bzw. die Interessen des Auftraggebers zu. Insbesondere Lernende Agenten greifen häufig auf Benutzerprofile ihrer Auftraggeber zurück, die für Werbe- und Marktforschungszwecke sicherlich hoch interessant wären und damit potentielle Zielobjekte eines logischen Angriffs darstellen.

Es ist denkbar, daß Agenten ohne Wissen des Benutzers **modifiziert**, ja sogar **sabotiert** werden. Feindliche Agenten könnten die unter Umständen kostenpflichtige Sammlung von Daten an einen anderen Auftraggeber weiterleiten oder die Sammlung bzw. den Auftrag modifizieren.
Diese Art des „Entern“ eines Agenten könnte parasitär oder symbiotisch erfolgen, zum Beispiel wären folgende Szenarien denkbar:

- Ein Agent A erhält den Auftrag, eine Auswahl der Lieferanten eines Produktes festzustellen. Ein anderer Agent B modifiziert die Datensammlung von Agent A, indem er alle Lieferanten, die preisgünstiger als sein Auftraggeber sind, aus der Datensammlung löscht.
- Ein feindlicher Agent A erhält den Auftrag, den Terminkalender-Agenten B aufzusuchen und dessen Terminkalender, also wirtschaftlich äußerst sensible Daten, auszuspähen.

Der **Daten-** und **Programmtransfer** kann über konventionelle kryptographische Verfahren sichergestellt werden. Um eine **Verfälschung** zu verhindern, können digitale Wasserzeichen (Signaturen) und/oder Checkcodes verwandt werden. In diesem Zusammenhang ist sicherlich das RSA Verfahren von Interesse [SCHNEIER96].

2.1.3 Heterogenität des Hosts

Gerade im Internet sind nahezu alle Maschinentypen vertreten. Daher ergeben sich insbesondere im Hinblick auf mobile Agenten spezielle Anforderungen an die Agenten und das Host-System. Da es viele **verschiedene Hardwareplattformen** gibt (PC, MAC, SPARC etc.) müssen Umgebungen geschaffen werden, in denen es möglich ist Agenten unabhängig von der Hardware-Plattform auszuführen.

- Diese Heterogenität legt die Benutzung von **Interpreter-Sprachen** oder die Nutzung eines **virtuellen Maschine-Konzeptes** nahe, da Compilersprachen eine Reduktion auf Maschinen-Code durchführen. Den zur Zeit sicherlich bekanntesten Vertreter des virtuellen Maschine Konzeptes stellte Rainer Stoll in seinem Vortrag vor: die JAVA Virtual Machine (JVM). Diese virtuelle Maschine wurde von SUN dazu konzipiert, möglichst effektive und kleine JAVA-Anwendungen in einer sicheren Laufzeitumgebungen auszuführen. Zur Zeit arbeitet die Hardwareindustrie an einer singlechip Implementation der JVM, die dann gänzlich ohne Betriebssystem lauffähig wäre.

- **Dynamische Link-Bibliotheken** müssen in verschiedenen Umgebungen gemeinsame Schnittstellen anbieten. Die dynamischen Link-Bibliotheken werden zur Laufzeit des Agenten eingebunden und somit wird die Größe des Agenten stark reduziert. Ansonsten würden die Agenten bei ohnehin schon geringer Leitungsbandbreite zu groß. Auch zur Verdeutlichung dieses Aspektes kann die Programmiersprache JAVA herangezogen werden. Alle in Java erstellten Klassen werden in eigene Dateien in einen Bytecode kompiliert und die Klassen werden erst zum Instanziierungszeitpunkt geladen. Das ermöglicht auch umfangreiche Applikationen auf einem Server zu plazieren, da diese sozusagen häppchenweise nachgeladen werden. Ich stimme jedoch nicht mit Rainer Stolls Meinung überein, daß dieses Verhalten optimal sei und sich für große Anwendungen eigne. Meiner Einschätzung nach wäre ein Netzlast abhängiges Nachladen der Klassen sinnvoller und würde Netzspitzen besser ausgleichen. Die derzeitig niedrige Bandbreite der meisten Netze macht sich bei Ausführung von JAVA-Programmen deutlich bemerkbar und der Geschwindigkeitsvorteil des Dynamic Linking wird durch den Geschwindigkeitsverlust des Interpreter wieder ausgeglichen. [FLANAGAN96]

2.2 Mobile Agenten

Nach dieser kurzen Diskussion der verschiedenen Sicherheitsaspekte und des Heterogenitäts-Problemes möchten ich mich der ersten Klasse von Agenten zuwenden.

Mobile Agenten, das heißt der den Agenten repräsentierenden Programmcode, können den Host wechseln. Daher sind diese Agenten in der Lage, unabhängig vom Computer des Auftraggebers Aktionen abzuarbeiten und dem Auftrag entsprechend den passenden Informationsanbieter bei Bedarf aufzusuchen.

Die grobe Klassifizierung in mobile und fixierte Agenten läßt sich durch eine Typisierung aufgrund des Auslösers und der Art der Mobilität verfeinern. Die mobilen Agenten werden durch Woolridge in drei Arten eingeteilt:

- **Selbstständig** mobile Agenten
- auf **Anforderung** mobile Agenten
- **Aktive**-Post Agenten
- mobile Agenten in **Push**-Technologie

Die Agenten-Typen dieser Klassen werden in den folgenden Abschnitten noch näher charakterisiert. Die Aktive-Post Agenten lassen sich keiner der beiden anderen Typen eindeutig zuordnen und werden daher gesondert aufgeführt.

Agenten in Push-Technologie lassen sich meiner Einschätzung nach keiner der drei anderen Arten zuordnen und daher erweitere ich die verfeinerte Klassifizierung von Woolridge um diese Kategorie.

2.2.1 Autonome mobile Agenten

Unter selbständig mobilen Agenten versteht man Agenten, die innerhalb vom Anwender gesetzter Grenzen **selbst** in der Lage sind, über ihren Zielort und die dort zu erledigende Arbeit zu **entscheiden** (z.B. Wieviel eCash diese ausgeben können, die Auswahl des nächsten

Zielortes etc.). Solche Agenten werden im allgemeinen mit einer speziellen Sprache programmiert, die eine ‘Go’-Anweisung anbietet (z.B. TELESCRIPT).
Schlüssel-Begriffe wie **Orte** und **Agenten** kennzeichnen diese Technologie. Agenten können sich innerhalb dieser virtuellen Welt bewegen und Plätze einnehmen. Zu diesem Zweck führt jeder Agent ein Ticket mit **Reiseinformationen** mit sich. Dieses Ticket kann Informationen wie z.B. Zielort, Ursprung, Auftraggeber oder auch benötigte Ressourcen enthalten. Agenten können miteinander **kommunizieren** und sich **treffen**.

Zum Beispiel wäre ein Agenten-Programm nach folgendem Schema denkbar:

```
gehe zu Dirks Rechner
spreche einen Termin mit diesem ab
komme zurück
bestätige diesen Termin bei meinem Terminkalender-Agenten
```

Der Agent könnte bei Dirks Abwesenheit selbständig Dirks Terminkalender-Agenten treffen und mit diesem einen Termin absprechen.

2.2.2 Mobilität auf Anforderung

Die Idee hierbei ist, daß der Host nur ausdrücklich **angeforderte** Agenten ausführen muß.

Das meist bekannte Bsp. sind die in HTML[1] eingebetteten JAVA-Applets. JAVA-Applets sind kleine Programme die in JAVA implementiert wurden und innerhalb eines WWW-Browsers lauffähig sind. Der Benutzer eines JAVA kompatiblen WWW-Browsers lädt diese Applets mit der Seite neben den Grafiken, dem Text und Formularen auf seinen Computer und diese werden dann auf der Maschine des Benutzers innerhalb eines browserinternen Interpreters (einer virtuellen Maschine) ausgeführt (vgl. JAVA-Vortrag in diesem Seminar und [FLANAGAN96]).

[1] **H**yper**t**ext **M**arkup **L**anguage

2.2.3 Aktive-Post Agenten

Die Idee hierbei ist es, den Agenten sozusagen **Huckepack mit einer eMail** zu versenden. Bei Erhalt der eMail wird die Dateianlage **entpackt** und der entpackte Agent wird auf der Zielmaschine durch einen **restriktiven Interpreter** ausgeführt (Aktive eMail). Der restriktive Interpreter ermöglicht einen großen Anteil der Host-Sicherheit.

Das weitverbreitetste Beispiel für diese Art von Agenten ist die **MIME**[2] Erweiterung der eMail in Verbindung mit **sicheren TCL Skripten**. Diese Kombination wird im folgenden Abschnitt näher dargestellt.

2.2.3.1 Fallbeispiel: MIME mit TCL/TK

TCL/TK ist eine Verbindung der Kommandosprache TCL[3] mit TK[4] einem X-Windows basierten Widget-Toolkit, daß die Erzeugung von graphischen Oberflächen sowie Interprozeßkommunikation ermöglicht. Die Kombination TCL/TK ist ein relativ einfach zu benutzendes GUI Entwicklungswerkzeug mit interessanten Eigenschaften im Bezug auf Agenten.

- TCL ist eine Interpretersprache. Dies impliziert die für einen Agenten benötigte Plattformunabhängigkeit und ermöglicht ein virtuelles Maschine Konzept.
- TCL ist leicht erweiterbar und daher sind in TCL implementierte Agenten leicht weiterentwickelbar
- TCL/TK kann als Code in andere Umgebungen eingebettet werden und ermöglicht somit eine große Austauschbarkeit.

Also können TCL-Skripten in Verbindung mit eMail-Systemen und Interpretern als eine Art Agenten aufgefaßt werden. Das zentrale Problem der TCL/TK Agenten waren die Sicherheitsanforderungen die über TCL nur unzureichend abgedeckt werden konnten. Diese erweiterten Anforderungen an die Kommandosprache führten zu Safe-TCL, das sich jedoch nicht durchsetzen konnte [STCL].

---

[2] **M**ultipurpose **I**nternet **M**ail **E**xtension

[3] **T**ool **C**ontrol **L**anguage

[4] **T**ool**k**it

Eine aktuelle Entwicklung in Zusammenhang mit TCL ist das Erscheinen eines TCL-Plugins für den meistverbreiteten WWW Browser Netscape [PLUG]. Dieses Plugin könnte zu einer Belebung der TCL/TK Kombination führen, jedoch vermute ich, daß JAVA seinen aktuellen Vorsprung gegenüber seinen Mitbewerbern noch weiter ausbauen wird.

2.2.4 Mobile Agenten in Push-Technologie

Es wäre ein mobiler Agent in Push-Technologie denkbar.
Z.B. beauftrage ich einen Ticketdienst, mir Informationen zu bestimmten Veranstaltungen zukommen zu lassen und die Veranstaltungstermine über einen Terminkalender Agenten in meinen Filofax eintragen zu lassen.

Der beschriebene Agent wäre weder autonom mobil, da der Agent durch den Ticketservice direkt gepusht wurde, noch wäre seine Aktivität durch Anforderung des Benutzers bestimmt, da der Agent durch den Ticketdienst beauftragt wurde. Diese Begründung basiert natürlich darauf, daß der Benutzer des Agenten in diesem Fall nicht der Informations-Anbieter sondern der Konsument der Information ist.

2.3 *Fixierte Agenten*

Nicht zuletzt aufgrund der immensen Sicherheitsprobleme scheinen mobile Agenten nicht der Weisheit letzter Schluß zu sein. Eine andere Möglichkeit sind die fixierten Agenten, die Programme im traditionellen Sinne sind.
Fixierte Agenten bleiben ortsfest an den Rechner des Auftraggebers gebunden und arbeiten ihren Auftrag ohne Rechnerwechsel ab. Ich möchte ausdrücklich darauf hinweisen, daß auch fixierte Agenten dazu in der Lage sein können über Kommunikationskanäle mit anderen Funktionsträgern (z.B. fremde Agenten, WWW-Server) Daten auszutauschen.

Auch hier läßt sich eine verfeinerte Typisierung vornehmen. Aufgrund der Lokalisation der Agenten teilt Woolridge die fixierten Agenten in zwei Gruppen ein:

- Client-seitige Agenten
- Server-seitige Agenten

Ich möchte diese Einteilung um eine Gruppe erweitern

- verteilte Agenten

In diesem Zusammenhang wird der Begriff des Client als lokale Maschine des Benutzers aufgefaßt. Der Server ist eine mit dem Client verbundene nicht näher lokalisierte Maschine.

2.3.1 Client-seitige Agenten

Bei client-seitigen Agenten bleibt der Agent an die **lokale Maschine** des Benutzers gebunden. Der Agent fungiert als **Vermittler** zwischen dem Benutzer und dem Internet. Die **Bewertung** der Informationen aus dem Netz erfolgt **durch das eigene** intelligente **Programm**. Beispiele für diesen Agententyp sind MAXIMS, NEWT und Personal-WebWatcher[5]. Newt ist ein Newsreader-Assistent und WebWatcher ein Browser-Assistent. In dem einen Fall wird die eingehende Information gefiltert, im anderen wird die erreichbare Information bewertet. In

[5] Das Ursprungsprogramm WebWatcher ist ein server-seitig fixierter Agent.

beiden Fällen verläßt der Agent die lokale Maschine des Benutzers nicht, sondern kommuniziert ortsgebunden mit den jeweiligen Informationsanbietern.

Vorteile:

- keine Sicherheitsprobleme
- keine Verletzung der Privatsphäre (abgesehen von Profilbildungen)

Nachteile:

- hoher Netzwerk-Verkehr durch die ständige Kommunikation mit externen Informationsanbietern
- Limitierung des Agenten in seinen Fähigkeiten

Der von Jochen Buchholz in seinem Vortrag im Rahmen dieses Seminars vorgestellte NEWT ist ein typischer Vertreter des client-seitig fixierten Agenten. Die USENET-Postings werden vom Server über das Netzwerk auf den Clientrechner herunter geladen und erst dort durch den NEWT Agenten ausgewertet. Da dabei das Verfahren TF/IDF und ein genetischer Algorithmus der kein Lernverfahren darstellte angewandt werden, kann dieser Agent jedoch nicht in die Klasse der intelligenten Agenten eingeordnet werden.

2.3.2 Server-seitige Agenten

Bei server-seitigen Agenten ist die „Intelligenz“ an den Server gebunden. Die Bewertung der Informationen aus dem Netz erfolgt in der Regel **durch den Anbieter** (Bsp. LYCOS, INFO-SEEK, jedes PERL-Skript). Insbesondere der im Seminar vorgestellte Shop-Bot Agent stellt einen server-seitigen Agenten da. Nur die gesammelte Information wird durch den Kollektor/Agenten zum Clientrechner des Benutzers übertragen, wobei der Agent beim Server verbleibt.

<u>Vorteile:</u>

- Informationen werden der Informationsquelle zugeordnet. Über die im Information Retrieval üblichen Indexierungsverfahren wird die Information anbieterseitig aufbereitet und dadurch sind präzise Auswertungen möglich.
- Der Aufbau von Benutzermodellen wird möglich und kann zu einem verbessertem Service für den Benutzer führen.
- Da die Informationen auf dem Server gesammelt und ausgewertet werden, ist eine gezielte Informationsübertragung möglich, die nur eine geringe Netzlast verursacht.

<u>Nachteile:</u>

- Eine Verletzung der Privatsphäre wird durch das Erstellen von Benutzerprofilen möglich.

2.3.3 Verteilte Agenten

Ein verteilter Agent kann die zur Ausführung seiner gesamten Funktionalität notwendigen Programmteile sowohl auf den Client als auch auf den Server und auf Zwischenstationen verteilen. Im Seminar wurde ein Agent vorgestellt, der die Aufgabe des Newsfiltering in zwei Arbeitsschritte aufteilte. Zuerst wurden die gesamten News-Postings auf einem Server gefiltert und dann durch den an einen Benutzer angepaßten Teil des Filteragenten bewertet.
In diesem Konzept erfüllt keiner der Programmteile alle Anforderungen an die Eigenschaften eines Agenten. Nur in der Gesamtheit der Funktion werden diese erreicht.

3 Agentenarchitektur

Wir möchten Agenten konstruieren, die alle vorher beschriebenen Eigenschaften aufweisen können. Daher geht es bei Agentenarchitektur um folgende Fragen:

- Wie macht man das?
- Welche Soft- und Hardware-Strukturen sind dafür geeignet?
- Was ist ein angemessener Einsatz bei der Aufteilung bzw. der Vermischung der Konzepte?

3.1 Schließende Agenten

Der klassische Ansatz zur Konstruktion von Agenten ist es, diese als einen bestimmten Teil eines wissenbasierten Systemes zu sehen und alle schon bekannten Konzepte und Methoden dieser Systeme zu übernehmen.

Dieses Paradigma ist als **symbolische KI** bekannt. Die Architektur eines **„beratenden“ Agenten** setzt sich dabei aus zwei Komponenten zusammen:

- Der Agent enthält ein explizit repräsentiertes symbolisches **Modell der realen Welt**.
- Er trifft Entscheidungen auf Grundlage **symbolischer Schlüsse**.

Soll ein Agent in dieser Struktur konstruiert werden, gilt es zwei Probleme zu lösen:

Translation: Wie übersetzt man die reale Welt in akkurate, adäquate symbolische Beschreibungen und schafft dies innerhalb akzeptabler Zeit.

Repräsentation: Wie können Informationen symbolisch repräsentiert werden und wie konstruiert man einen Agenten, der in Lage ist, aus diesen Informationen ‘just in time’ zu schließen. Keines der Probleme ist bisher auch nur annähernd zufriedenstellend gelöst. Einige Probleme modaler Logik scheinen sogar **unentscheidbar** zu sein.

3.1.1 Planende Agenten

Ein Spezialfall der schließenden Agenten sind die planenden Agenten. Planung ist eigentlich eine Form der automatischen Programmierung: „Das Design eines Verfahrens, das angewandt ein bestimmtes Ziel erreicht.“

Ein **planender Agent** besteht aus:

- einer symbolischen **Beschreibung der Welt**
- einer symbolischen **Bescheibung eines Ziel-Status**
- einer Menge von **Aktionsbeschreibungen,** die aus Beschreibungen der mit einer Aktion verknüpften **vorherigen und nachfolgenden Bedingungen** bestehen.

Zum Beispiel enthält die Beschreibung der realen Welt eine Beschreibung einer Tür. Die Beschreibung des Ziel-Status enthält die Beschreibung einer offenen Tür. Die passende Aktionsbeschreibung „öffnen einer Tür“ ist mit der vorhergehenden Bedingung „Tür geschlossen“ und der nachfolgenden Bedingung „Tür geöffnet“ verknüpft.

Die Algorithmen zur Lösung eines Problems basieren in erster Linie auf Vergleichen des aktuellen Status mit dem Zielstatus nach Ausführung diverser Aktionen. Der Algorithmus versucht eine Sequenz von Aktionen zu finden, durch deren Ausführung der Agent sein Ziel erreicht. Einige theoretische Ergebnisse scheinen allerdings zu belegen, daß diese Art von Problemen ab einer gewissen Komplexität nicht effizient lösbar ist.

3.2 Reaktive Architekturen

Aufgrund der wahrscheinlich nicht effizient lösbaren Probleme der schließenden Agenten wurden andere Konzepte verfolgt. Ein reaktiver Agent ist ein Agent, dessen Architektur keinerlei zentrale symbolische Logik-Modelle enthält und daher auch nicht auf komplexe logische Schlüsse zurückgreifen muß.

Zentrale Thesen der reaktiven Architekturen:

Intelligentes Verhalten ist ohne die explizite Repräsentation der Welt und komplexe Schlüsse im Sinne der symbolischen KI möglich. (Brooks)

- Intelligenz ist eine **entstehende** Eigenschaft bestimmter komplexer Systeme
- Intelligentes Verhalten entsteht durch **Interaktion** des Agenten mit seiner Umwelt

Grundlegende Beobachtungen sind:

- Die meisten Aktivitäten sind **Routine ohne neue Schlüsse**. (Agre & Chapmann)
- Die meisten Aufgaben lassen sich durch eine **variierte Routine-Lösung** bewältigen.

Brooks realisierte diverse Roboter-Steuerungen auf Grundlage einer mehrschichtigen Subsumption Architektur. Die Schichten bestehen aus einer Hierarchie von aufgabenorientierten Verhaltensweisen. Jede Verhaltensweise kann mit anderen zur Steuerung des Roboters kombiniert werden. Tiefere Schichten repräsentieren einfachere Verhaltensweisen und haben Vorrang vor komplexeren Verhaltensweisen höherer Schichten. Roboter die mit einer großen Anzahl von Subsumptions-Agenten arbeiten, erfüllen einfache Aufgaben (z.B. das Überwinden von Hindernissen) nahezu optimal.

3.3 Situative Automaten

Die Strategie der situativen Automaten ist es, einen Agenten in einer **regel-ähnlichen Sprache** zu spezifizieren und diese Beschreibung in eine **digitale Maschine** zu kompilieren, die die Aufgabenspezifikation erfüllt. Diese Maschine arbeitet in **beweisbaren Zeitgrenzen.**

Jedoch: Je ausdrucksvoller die Beschreibungssprache wird, desto schwieriger wird die Kompilierung. Einige Forschungsergebnisse sagen aus, daß die Kompilierung ab einer gewissen Komplexitätstiefe der Sprache nicht durchzuführen ist.

3.4 Hybrid-Architektur

Ein hybrider Architekturansatz konstruiert einen Agenten aus zwei oder mehr Teilen.

Ein einfacher Hybrider Ansatz ist es, den Agenten genau aus zwei Komponenten zu erzeugen:

- Einem beratenden Teil
- Einem reaktiven Teil

Der beratende Teil enthält ein symbolisches Welt-Modell, mit dem Pläne entwickelt und Entscheidungen getroffen werden.
Der reaktive Teil ist in der Lage, ohne komplexe Schlüsse auf Ereignisse/Zustände zu reagieren (Häufig besitzt die reaktive Komponente Vorrang vor der Beratenden).
Die hybriden Agenten werden hierarchisch in einem Schichten-Modell aufgebaut. Höheren Schichten werden Informationen höherer Komplexität zugeordnet. Das größte Problem dieser Architektur ist die Interaktion zwischen den Schichten und Teilen.

3.5 Touring-Maschinen

Agenten in der Architektur der Touring-Maschinen bestehen aus drei Subsystemen:

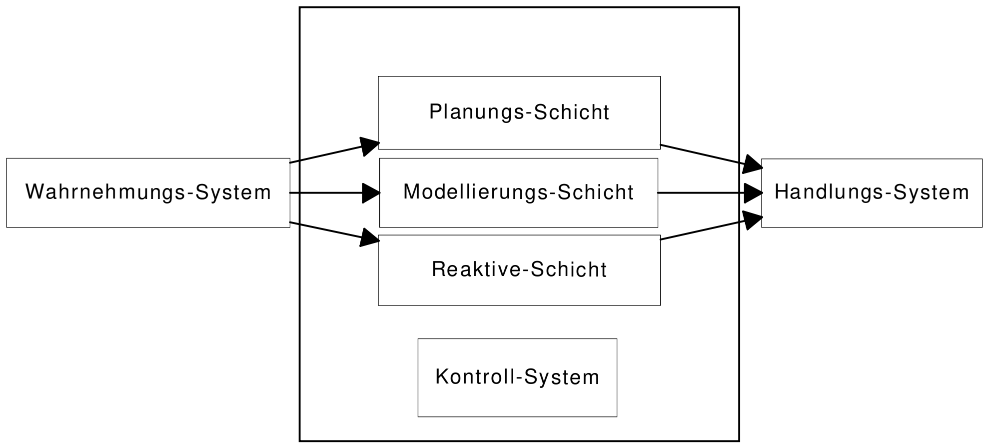
- Einem Wahrnehmungssystem
- Einem Handlungssystem
- Einem Kontrollsystem

Das Wahrnehmungssystem stellt die Sensorik des Agenten dar. Im Wahrnehmungssystem müssen alle zur Erledigung des Agenten-Auftrages notwendigen „Sinne“ des Agenten repräsentiert werden. Die so erworbenen Informationen des Agenten werden im Kontrollsystem verarbeitet.

Das Kontrollsystem setzt sich aus drei Kontroll-Schichten zusammen:

- einer reaktiven Schicht
- einer planenden Schicht
- einer modellierenden Schicht

Die im Kontrollsystem erzeugten Handlungsmuster des Agenten werden durch das Handlungssystem ausgeführt. Das Handlungssystem umfaßt alle Aktoren eines Agenten (Bewegung, Sprache etc.).



Die **reaktive Schicht** wird als eine Menge situativer-Regeln implementiert.

Die **planende Schicht** erzeugt Pläne und wählt Aktionen zur Erreichung des Ziels aus.

Die **modellierende Schicht** enthält symbolische Repräsentationen des „Bewußtseins“ des Agenten im Bezug zu seiner Umgebung.

Alle drei Schichten kommunizieren miteinander und sind in einem Kontrollrahmen eingebettet.

4 Schlußbemerkungen

Die im Verlauf des Seminars vorgestellten Agenten gehörten häufig nicht zur Klasse der intelligenten Agenten da kaum komplexe Lernmethoden, sondern rein statistische-IR-Methoden implementiert wurden (z.B. Relevance Feedback). Diese Tatsache resultiert meiner Einschätzung nach nicht nur aus der Komplexität der einzelnen Algorithmen, sondern auch häufig aus den Problemen, die sich bei einem Einsatz dieser Algorithmen in einer unbeschränkten Umgebung ergeben.

Im Seminar wurde mir die Frage gestellt, ob ein Konzept nur dann erfolgversprechend sei, wenn es das Modell der realen Welt nicht darstellen müßte, da in meinem Vortrag all diese Verfahren als unqualifiziert dargestellt wurden. Ich bin Brooks Meinung, wenn er ein Wissen über die ganze Welt zur Lösung von Aufgaben für unnötig hält. Jedoch stelle ich die Hypothese auf, daß alle die für die Lösung einer Aufgabe notwendigen, über Subsumption aus der Beschreibung eines Teils der Welt und Aktionsbeschreibungen ableitbaren Informationen die jeweils wichtige Teilmenge der reale Welt Beschreibung sind.

In der Abschlußdiskussion des Seminars kritisierten Stefan Mintert und Rainer Stoll die Klassifizierung der Agenten anhand von Mobilität und Lokalisation. Ihrer Meinung nach ist diese Art der Typisierung im Hinblick auf transparent verteilte Systeme nicht zukunftssicher. Trotz dieses Einwandes halte ich die Klassifikation von Woolridge dennoch für sinnvoll, da Agenten zumeist in einer abgegrenzten speziellen Umgebung agieren. Diese Agenten-Umgebungen müßten auch auf transparent verteilten Systemen geschaffen werden, so daß Begriffe wie Orte und Mobilität erhalten bleiben. Ein Beispiel hierfür ist das SUN Network File System (SUN-NFS). Mit SUN-NFS ist das transparente Einbinden von Speicherressourcen zur Laufzeit möglich. Die verteilten Speicherressourcen werden durch eine Directory-Struktur gegliedert die mit einer Agenten-Umgebung vergleichbar ist.

5 Literaturverzeichnis

[FLANAGAN96] David Flanagan, „Java in a Nutshell“, O’Reilly & Associates Inc., 1996.

[WOOL95] Michael Wooldrige and Nicholas R. Jennings „Intelligent Agents: Theory and Practice“, Knowledge Engineering Review. October 1994, Revised January 1995.

[WOOL96a] Michael Woolridge, „Internet Agents“, Spring 1996.

[WOOL96b] Michael Woolridge, „Intelligent Agents“, Spring 1996.

[WOOL96c] Michael Woolridge and Nicholas R. Jennings, „Software Agents“, IEE Review, January 1996.

[WAHLSTER96] Wolfgang Wahlster, „Internet Agenten“, KIFS 1996.

[PFLEEGER97] Charles P. Pfleeger, „Security in Computing“, Prentice Hall Inc., 1997

[PLUG] http://browserwatch.iworld.com/plug-in.html
Beschreibung der Netscape Plug-Ins.

[SCHNEIER96] Bruce Schneier, „Applied Cryptography“, Second Edition, Wiley, 1996.

[SILBER94] Abraham Silberschatz and Peter B. Galvin, Addison-Wesley, 1994.

[STCL] http://www.cs.uoregon.edu/kortuem/Colloquium/abstract2.html
Beschreibung des Safe-TCL.