Seminar

„Techniken Intelligenter Agenten"

Mobile Agenten

Michael Jaeger

mjaeger@cs.uni-frankfurt.de

Johann Wolfgang Goethe-Universität
Fachbereich Informatik (20)

Inhaltsverzeichnis

1 Einleitung **3**

2 Grundlagen **3**

2.1 Anwendungsbereiche mobiler Agenten 5

3 Technik **8**

3.1 Eine Agenten-Programmiersprache 8

3.2 Ein Agenten Host 9

3.3 Das Agenten-Paradigma - ein Umdenken bei der Programmierung 11

4 Agentensysteme **12**

4.1 Telescript 12

4.2 AMETAS 14

4.2.1 Die Struktur des Systems 14

4.2.2 Sicherheit 16

4.2.3 Beispiel: NetDoctor 16

5 Schlussbemerkung **18**

1 Einleitung

Agenten sind längst nicht mehr nur von theoretischem Interesse. Es besteht eine große Hoffnung darauf, dass mobile Agenten dem Benutzer wiederkehrende Aufgaben abnehmen und mittels Kooperation auch sehr komplizierte Probleme lösen können.

Diese Ausarbeitung hat zum Ziel, dem Leser sowohl die theoretischen als auch die technischen Grundlagen zum Verständnis der Materie „mobile Agenten" zu vermitteln und stellt zu Veranschaulichung zwei Agentensysteme - eines aus der Forschung ein anderes aus dem kommerziellen Bereich - vor.

2 Grundlagen

Wie jeder Themenbereich, so hat auch dieser seine eigene Fachsprache. Fangen wir also gleich beim Titel an: Was hat man sich unter einem mobilen Agenten vorzustellen? In der Literatur findet man dazu viele, größtenteils im Detail unterschiedliche Ansichten. Natürlich kann auch ich keine allgemein gültige Definition für diesen Begriff geben, ich werde jedoch versuchen, anhand einer Auswahl von Literatur zu diesem Thema meine Vorstellung des Begriffs zu umreißen.

Als Grundlage sollte zu Anfang geklärt werden, was ein Agent ist, da ja (wie schon aus dem Begriff ersichtlich ist) ein mobiler Agent eine Spezial-Form eines Agenten darstellt (worüber sich auch fast alle Autoren der von mir gelesenen Texte einig sind). Da über die Definition eines Agent in der Fachliteratur keine Einigkeit besteht, möchte ich eine Definition verwenden, die meiner Ansicht nach allgemein genug ist, um einen Großteil anderer Definitionen einzuschließen:

> *„Ein* **autonomer Agent** *ist ein System, das in einem Teil einer Umgebung situiert ist. Es nimmt seine Umgebung wahr und handelt in ihr gemäß einer angestrebten Ordnung über die Zeit hinweg, und beeinflusst so, was es in Zukunft wahrnehmen wird." [8]*

Nach Ansicht der Autoren ist ein Agent ein System, das in eine Umgebung eingebettet ist, die es wahrnimmt und nach eigenen Vorstellungen manipuliert, so dass die zukünftige Wahrnehmung davon wieder beeinflusst wird (ich setze an dieser Stelle die Begriffe „autonomer Agent" und „Agent" gleich, da Autonomie in allen von mir gelesenen Texten zu den grundlegenden Eigenschaften von Agenten gehört).

Nach dieser Definition wäre also der Mensch ein Agent. Er nimmt seine Umgebung (=Umwelt) über seine Sinnesorgane war und manipuliert sie (baut z.B. ein Haus), um seiner angestrebten Ordnung[1] gerecht zu werden. Ebenso wie der Mensch kann aber auch ein

[1] Hier habe ich den englischen Begriff *agenda* wie im Wörterbuch vorgeschlagen mit *Ordnung* übersetzt und damit unterstellt, dass die Menschen nach einer vorgegebenen Ordnung handeln - ein Punkt, den ich hier nicht weiter ausdiskutieren möchte.

einfacher Thermostat zur Regelung der Raumtemperatur ein Agent sein. Dieses Gerät nimmt über Wärme-Sensoren die Raumtemperatur war und beeinflusst sie, indem es je nach Zielvorgabe die Heizung höher oder niedriger stellt - was wieder auf die Wärmesensoren zurückwirkt.

Diese zwei Beispiele sollen zeigen, wie allgemein der Agenten-Begriff in oben angeführter Definition zu verstehen ist. Natürlich geht es hier nicht um Menschen oder Meß- und Regeltechnik, sondern um **Software Agenten**. Das sind Programme, die die oben geforderten Eigenschaften erfüllen, deren Umwelt demnach der Computer mit all seinen Schnittstellen ist. Im weiteren Text sind, wenn von *Agenten* die Rede ist, immer *Software Agenten* gemeint.

Zur weiteren Klassifikation von Agenten wird in [8] die in der Psychologie bekannte **Matrix-Organisation** vorgeschlagen, nach der jedes Merkmal eines Agenten als eine Dimension einer n-dimensionalen Matrix verstanden wird, so dass eine Zelle dieser Matrix einer Auswahl von Attributen entspricht, die eine Art von Agenten beschreibt. Solche Merkmale sind nach [9, 17, 22]:

- **Kooperationsfähigkeit** (Zusammenarbeit mit anderen Agenten)
- **Mobilität**
- **(Re-)Aktionsfähigkeit**
- **Flexibilität/Lernfähigkeit** (Anpassungsfähigkeit an die Umweltbedingungen)
- **Kommunikationsfähigkeit** (mit anderen Agenten)
- **Ziel-Orientiertheit** (Ein Agent reagiert nicht nur auf seine Umgebung, sondern er arbeitet auf ein bestimmtes Ziel hin)
- **Persistenz** (Dauerhaftigkeit: das Programm kann an jeder beliebigen Stelle seiner Ausführung angehalten und mitsamt seinem Zustand, der sowohl aus den Variablenwerten, als auch aus Programmzähler, Stack-Zeiger etc. besteht, „eingefroren" und zu einem beliebigen Zeitpunkt später wieder „aufgetaut" und fortgesetzt werden[2])

Ein **mobiler Agent** (oder auch **transportable agent** [12]) ist demnach ein Agent, der die Fähigkeit der „**Mobilität**" besitzt. Darunter ist zu verstehen, dass der Agent nicht an einen Ort gebunden ist, sondern mitsamt seinem Code und seinen Daten (sowohl denen, die

[2]Diese Eigenschaft ist vor allem bei mobilen Agenten wichtig, da diese vor dem Transport „eingeschläfert" und danach wieder „aufgeweckt" werden müssen. In dieser Zeit sind natürlich die „Sinnesorgane" des Agenten lahmgelegt, so dass Signale und Nachrichten von außen u.U. nicht von dem Agenten wahrgenommen werden.

seinen aktuellen Zustand beschreiben, als auch denen, die z.B. durch Messungen gesammelt wurden) umher wandern kann[3].

In diesem Zusammenhang drängt sich die berechtigte Frage auf, wo der Agent herum wandert. Um dies zu klären, möchte ich den Begriff der **Stelle** (auch **place** genannt) einführen. Unter diesem Begriff versteht man eine Umgebung (oder weniger abstrakt einen „Ort" oder „Raum"), an dem sich ein Agent aufhalten kann. Dabei handelt es sich im wesentlichen um ein Programm, das eine Umgebung zur Verfügung stellt, in der Agenten operieren können. Dank Mobilität können die Agenten nun die aktuelle Stelle wechseln - also von einer zur anderen wandern [10].

Eine Stelle könnte z.B. wie in [12] ein Mail-Server in Verbindung mit einem Interpreter[4] sein. Der Server kann die Agenten als Datei-Anhänge verschicken und empfangene E-Mail darauf überprüfen, ob ein Agent in Form eines Programms in der Sprache des Interpreters angehängt ist, und falls dies der Fall ist, selbigen in dem Interpreter ausführen (in diesem Fall muss es eine Schnittstelle von dem Interpreter zum Mail-Server geben, damit sich Agenten selbst weiter verschicken - also wandern - und dabei womöglich auftretende Fehler abfragen können). In diesem Beispiel „leben" die Agenten also in Interpreter-Prozessen (die die Stellen darstellen) und können mittels E-Mail-Kommunikation migrieren. Der Interpreter stellt zudem Schnittstellen für Agenten zur Verfügung, mittels der sie mit der Außenwelt kommunizieren können.

Der Raum, in dem sich Agenten bewegen können, lässt sich somit topologisch in Stellen aufteilen, die Schnittstellen zur Verfügung stellen, damit ein Agent seine Umgebung manipulieren kann.

2.1 Anwendungsbereiche mobiler Agenten

Bevor nun auf die Technik und konkrete Agentensysteme eingegangen wird, möchte ich einige mögliche Anwendungsbereiche mobiler Agenten erläutern. Zunächst soll aber noch geklärt werden, welche Arten von Anwendungen besonders für den Einsatz mobiler Agenten sprechen (siehe auch [21]):

[3] An dieser Stelle möchte ich betonen, dass einige Autoren [14] der Meinung sind, dass „Intelligenz" im Sinne der künstlichen Intelligenz nicht unbedingt eine Eigenschaft von Agenten sein muss - genausowenig wie die Mobilität. Insofern bleibt die Frage offen, ob dieser Beitrag wirklich zum Thema „Techniken intelligenter Agenten" passt - wobei natürlich auch der Begriff der Intelligenz ein sehr dehnbarer ist.

[4] Nach [7] ist ein Interpreter *„... ein Programm, welches ein Programm einer anderen Programmiersprache nach den notwendigen syntaktischen Überprüfungen sofort ausführt. Im Gegensatz zu einem Übersetzer [Compiler] muss das Quellprogramm nicht erst in eine andere Programmiersprache übersetzt werden, sondern der Interpreter analysiert nacheinander jede Anweisung und Deklaration des Quellprogramms und führt diese unmittelbar aus."* Programme, die in einer Interpreter Sprache wie PERL, Python oder Tk geschrieben sind, müssen also zur Ausführung nicht in Binär-Code übersetzt werden, sondern wenn überhaupt nur in einen plattformunabhängigen Byte-Code. Das spart Arbeit und fördert die plattformunabhängigkeit, da so nur der Interpreter selbst auf eine andere Plattform portiert werden muss, damit ein Programm auf dieser Plattform lauffähig ist. Nachteilig ist ein signifikanter Geschwindigkeitsverlust in der Ausführung.

- **Passiver Benutzer, zeitunkritische Daten.** Anwendungen, bei denen eine sofortige Reaktion auf ankommende Daten gefordert ist, unabhängig davon, ob der Benutzer anwesend ist oder nicht. Als Beispiel kann man sich eine verteilte Suche vorstellen, bei der Suchergebnisse aus verschiedenen Quellen sofort verarbeitet werden müssen, auch wenn der Auftraggeber der Suche gerade nicht verfügbar ist.

- **Verteilte Berechnungen.** Komplizierte Berechnungen können oft auf kleinere Probleme heruntergebrochen werden, die dann auf verschiedenen Computern gelöst werden. Mobile Agenten könnten diese Teilprobleme zu anderen Stellen tragen und sie dort bearbeiten und so zum Beispiel eine Art „load balancing" (Lastverteilung) durchführen, indem sie sich immer die Computer mit der meisten freien Rechenzeit aussuchen.

- **Vertrauensunwürdige Zusammenarbeit.** In einem auf mobilen Agenten basierenden System könnten Agenten verschiedener Parteien sich auf „neutralem Boden" treffen und zusammenarbeiten, ohne dass dies ein Sicherheitsrisiko für eine der Parteien darstellen würde, da kein Agent Zugang zum Netzwerk seiner Auftraggeber hat bis er zurückkehrt.

- **Schlechte Netzwerkverbindungen/zeitweilig vom Netz abgehängte Systeme.** Unter Bedingungen, wie sie z.B. bei der mobilen Arbeit mit Notebooks gegeben sind, kann der Benutzer einen Agenten mit einer Aufgabe ins Netz schicken und sich abhängen. Wenn er sich später wieder anmeldet, kann er sich die Ergebnisse, die der Agent produziert hat, geben lassen oder unter Umständen den Agent wieder auf seinen Computer holen, um ihn dort weiterarbeiten zu lassen.

Da ich nur wenige Hinweise auf Systeme gefunden habe, die wirklich konsequent auf dem Einsatz mobiler Agenten aufbauen und kommerziell im Einsatz sind oder waren[5] (gemeint sind z.B. der Personalink Service von AT&T, siehe [5], der NetDoctor von [10] und Sony's Magic Link PDA, siehe [17]), kann ich für den Großteil der hier angeführten Anwendungsgebiete leider keine Erfahrungswerte liefern. Nichtsdestotrotz soll nun eine Aufzählung möglicher Anwendungsbereiche mobiler Agenten und deren Erläuterung folgen:

- Wie schon oben erwähnt wurde bei AT&T ein Dienst namens Personalink angeboten. Hierbei wurden mobile Agenten für **intelligente Mail-Verarbeitung** [5] eingesetzt: Ein sog. „courier agent" kümmert sich dabei um die korrekte Auslieferung der E-Mail direkt an den Benutzer, falls dieser online ist, oder sonst in seinen Briefkasten. An dieser Stelle übernimmt ein intelligenter Agent die Nachricht und verarbeitet sie aufgrund von Schlüsselinformationen, die er vom „courier agent" erfragt. Dieses Vorgehen steigert im Gegensatz zum herkömmlichen E-Mail-Protokoll SMTP die Sicherheit und Erweiterbarkeit (z.B. um einen Übersetzungsservice oder ein Mail-Routing aufgrund von Prioritäten an andere Geräte, z.B. Pager oder Handy) und bietet überdies die allgemeinen Vorteile objektorientierter Kapselung.

[5] Leider habe ich keine Hinweise darauf gefunden, ob die betreffenden Systeme noch im Einsatz sind.

- Diese Anwendung kann allgemeiner in den Bereich **Netzwerkmanagement** eingeordnet werden. So wird in [10] z.B. ein Netzwerküberwachungssystem auf Basis von mobilen Agenten vorgestellt, die den Status von Rechnern im Netzwerk überprüfen, und falls dieser nicht normal ist, Konfigurationsänderungen vornehmen bzw. den Administrator benachrichtigen, falls keine Besserung eintritt. Mehr dazu in Abschnitt 4.2.3.

- Ein weites Anwendungsgebiet für mobile Agenten wird vor allem im Bereich **Electronic Commerce** gesehen ([5, 15, 14]). Erfahrungen in diesem Bereich wurden in [4] gesammelt, als man während eines eintägigen Kongresses einen Agenten-Marktplatz installierte, auf dem jeder Benutzer vorgegebene Güter kaufen und verkaufen konnte. Dabei musste für jede Verkaufs- bzw. Kaufaktion ein Agent erzeugt werden, der versuchte, diese Aufgabe im Sinne des Benutzers zu erledigen, indem er mit anderen Agenten einen optimalen Preis durch Handeln erzielte. Zu diesem Zweck wurde er mit einem maximalen und einem minimalen Preis versehen, den der Besitzer zu zahlen bereit war. Außerdem konnte der Benutzer zwischen den Verhandlungsstrategien *„ängstlich"*, *„kühl-berechnend"* und *„sparsam"* wählen, die festlegten, wie schnell der Agent seinen Preis erhöhte bzw. erniedrigte, wenn er mit anderen Agenten verhandelt[6].

- Genauso könnte man sich eine Anwendung denken, bei der Agenten durch das Internet reisen und nach dem besten Preis für ein Produkt suchen, das der Besitzer gerne kaufen möchte. Diese Anwendung wäre sowohl dem Bereich Electronic Commerce, als auch dem Bereich **Informationsbeschaffung und -management** zuzuordnen [18]. Dabei wäre es auch vorstellbar, dass sich Agenten gegenseitig befragen und so eine verteilte Wissensbasis schaffen [18], oder dass Agenten zur **Überwachung** von z.B. Aktienkurse eingesetzt werden, die dann beim Eintritt bestimmter Ereignisse (z.B. wenn eine Aktie einen Werte über- oder unterschreitet) den Benutzer informieren.

- Auch könnte man mobile Agenten einsetzen, um **Kommunikation in heterogenen Umgebungen** zu ermöglichen. So würde man einen mobilen Agenten erstellen, der alle von den verschiedenen Systemen „gesprochene Sprachen" versteht und übersetzen kann. Möchte nun ein System einem anderen eine Nachricht zukommen lassen, so erzeugt es einen solchen Agenten, übergibt ihm die Botschaft und schickt ihn zu dem Zielsystem [11].
  In [6] ist als Beispiel ein virtuelles Unternehmen angeführt, das sich aus mehreren einzelnen Unternehmen mit unterschiedlichen Prozessen und Infrastrukturen zusammensetzt. Da ein Angleichen der Systeme zu aufwändig ist (gerade vor dem Hintergrund, dass viele Joint-Ventures nur über einen sehr begrenzten Zeitraum existieren),

[6] Die Firma SAP beschäftigt sich gerade mit dem Thema *„business to business"* und plant in den Bereichen Autoproduktion und Chemie eine Plattform für *„virtuelle Marktplätze"* zu entwickeln, in denen dann Agenten, ähnlich wie in dem angeführten Beispiel, den Geschäftsprozess unterstützen könnten. Weitere Informationen sind unter [1] zu finden.

werden Schnittstellen für mobile Agenten geschaffen, die eine höhere Abstraktionsebene der Geschäftsabläufe bieten sollen.

3 Technik

Nach Klärung der wichtigsten Grundbegriffe soll jetzt eine Einführung in die verschiedenen Techniken folgen, die der Einsatz mobiler Agenten voraussetzt. Zu Anfang möchte ich verschiedene technische Vorbedingungen und Lösungsansätze erläutern, um dann später in Abschnitt 4 deren Realisierung beispielhaft vorzustellen.

3.1 Eine Agenten-Programmiersprache

Mobile Agenten, so wie sie hier behandelt werden, sind ein Stück Software, dass in einer Programmiersprache erstellt wurde. Damit man einen mobilen Agenten programmieren kann, muss die verwendete Programmiersprache gewisse Grundvoraussetzungen erfüllen, insbesondere

- **Plattformunabhängigkeit.** Ein Agent soll in einem Netzwerk von einem Rechner zu einem anderen migrieren können. Damit aber der Agent auf diesem anderen Rechner weiterlaufen kann, muss sein Code auch hier ausgeführt werden können. Besonders geeignet sind hierfür sog. Interpreter Sprachen (siehe auch S. 5), deren Code zur Ausführung lediglich den Interpreter benötigen. Dementsprechend gibt es Agentensysteme z.B. in PERL („Penguin") und TCL („Agent TCL"). Gerade in den letzten Jahren wurde auch immer häufiger Java als Agenten Sprache eingesetzt[7]. Beim Entwurf dieser Programmiersprache wurde großer Wert auf Netzwerkfunktionalität und Plattformunabhängigkeit gelegt. Die große Akzeptanz in Industrie und Forschung scheint sie zur Programmierung von Agenten zu prädestinieren. Beispiele für Agentensysteme, die komplett in Java programmiert wurden sind AMETAS (siehe Abschnitt 4.2) und die Aglet-Technologie von IBM.
- **Sicherheit.** Gerade in Zusammenhang mit Agenten ist das Merkmal „Sicherheit" besonders wichtig, eignen sich doch mobile Agentensysteme hervorragend zur Programmierung von Viren oder Würmern. Zu diesem Zweck muss die Umgebung, in der das Agentenprogramm abläuft ein ausgefeiltes Sicherheitssystem zur Verfügung stellen (wie z.B. die Java-Sandbox), damit Agenten nur die ihnen zugestandene Aktionen durchführen dürfen. Besonders ist dabei zu beachten, dass Agenten nicht ihren Host (s.u.), und umgekehrt Hosts die Agenten ausspionieren können. Gerade im Bereich Electronic Commerce ist es essentiell, dass auch die Übertragung von Agenten nicht

[7] In der Tat wird bereits über eine einheitliche Agentenschnittstelle in Java nachgedacht, da es bereits sehr viele Agentensysteme in dieser Programmiersprache gibt [14].

abgehört werden kann - man stelle sich nur vor, ein Agent trägt sensible Informationen wie Kreditkartennummern oder ähnliches mit sich herum!

- **spätes Binden.** Unter diesem Begriff wird die Fähigkeit eines Programms verstanden, erst zu Laufzeit und bei Bedarf bestimmte Code-Teile (z.B. Klassen) nachzuladen (am besten aus dem Netzwerk). Für mobile Agenten bedeutet dies, dass der Agenten-Code recht klein gehalten werden kann, wenn der Aufbau des Agenten streng modular konzipiert wird.

Bei Agenten-Programmiersprachen (auch **MCL - Mobile Code Language** genannt) wird zwischen zwei Arten von Mobilität unterschieden [3, 10]:

- **Starke Mobilität** (auch **transparente Migration**). Bei dieser Art wandert der Agent nach Aufruf eines Befehls mitsamt seines Codes und seines Ausführungszustandes zu einer Stelle, so dass er am Ziel genau an der Position weiter ausgeführt werden kann, an der er gestoppt wurde. Aus Sicht des Programmierers wird also der Programmablauf nicht unterbrochen.
- **Schwache Mobilität** (auch **nicht transparente Migration**). Im Gegensatz zur transparenten Migration wandert hier der Agent nicht mitsamt seines ganzen Ausführungszustandes (z.B. ohne den Programmzähler) zum Zielort. Stattdessen hat der Programmierer dafür zu sorgen, dass wichtige Daten lokal in Objektvariablen gespeichert werden. Der Agent wird dann an der Zielstelle wieder neu gestartet. Diese Tatsache muss bereits bei der Programmierung des Agenten bedacht werden (siehe Abschnitt 3.3)!

3.2 Ein Agenten Host

Ein in einer Agenten-Programmiersprache geschriebener Agent läuft in einer auf diese Programmiersprache zugeschnittenen Umgebung, die auf einem Computer installiert ist, dem **Agenten Host**. Diese Umgebung stellt dem Agenten Schnittstellen zur Verfügung, mit denen er auf Ressourcen des Host zugreifen und mit anderen Agenten kommunizieren kann (wenn der Agent diese Fähigkeit besitzt). Auf einem Agenten Host können durchaus mehrere Stellen laufen.

Gerade bei der Ressourcen-Nutzung durch Agenten kann es aber Probleme geben, denn was soll passieren, wenn ein Agent eine Ressource an sich bindet und dann zu einer anderen Stelle auf einem anderen Host wechselt? Zum einen kann man sich vorstellen, dass der Agent einfach den Zugriff auf die Ressource verliert, wenn er den Host wechselt. Eine wesentlich elegantere Lösung stellen aber die beiden folgenden Lösungsansätze dar:

- Bei der Verwendung sog. **Ressource-Proxies** greift der Agent nie direkt auf die Ressource selbst zu, sondern immer nur auf ein Proxy-Programm, das dann den Zugriff auf die eigentliche Ressource steuert. Dabei ist es dem Agenten egal, wo sich die Ressource befindet, denn auf den eigentlichen Zugriff kümmert sich ja der Proxy.

- Eine andere Idee ist der Einsatz eines Systems mit verteilten Objekten, wie es zum Beispiel CORBA[8] eines ist. Wie bei den Ressource-Proxies oben wird auch hier immer nur über sogenannte ORBs[9] auf die eigentlichen Ressourcen (Objekte) zugegriffen, die sich darum kümmern, die Ressourcen im Netz mit dem Agenten zu verbinden.

Auch bei der Kommunikation zwischen Agenten werden zwei Arten unterschieden:

- Bei der **synchronen Kommunikation** kommuniziert ein Agent mit dem anderen, indem er diesem eine Nachricht schickt, die der Empfänger sofort zu bearbeiten hat. Eine Nachricht landet also sofort beim Empfänger und auch eine Antwort lässt nicht lange auf sich warten. Im Grunde genommen ruft also ein Agent über die Ferne eine Methode eines anderen Agenten auf, was oft als Einschränkung der Autonomie des anderen Agenten gesehen wird.
- Aus diesem Grund wurde die **asynchrone Kommunikation** als Ansatz entwickelt. Im Gegensatz zur synchronen wird hier (ähnlich wie beim E-Mail-Versand) die Nachricht eines Agenten A an einen Agenten B in dem Postfach von Agent B abgelegt, so dass dieser sich die Nachricht abholen und bearbeiten kann, wann er will.

In Abschnitt 4 werden für beide Kommunikationsarten beispielhaft Systeme vorgestellt.

Abschließend soll eine Graphik den Aufbau der oben beschriebenen Komponenten verdeutlichen:

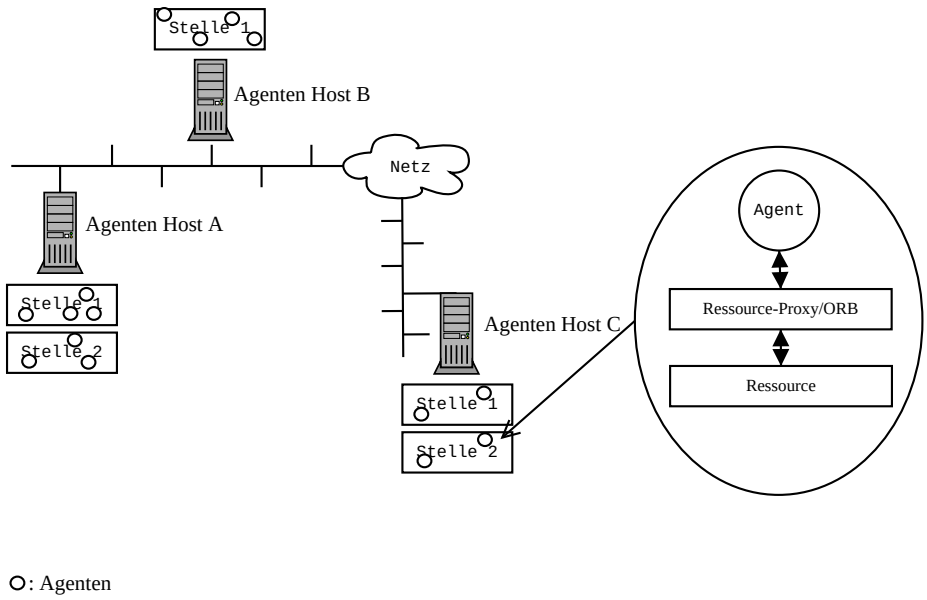

Abbildung 3.1: Aufbau eines Agentensystems

[8] Common Object Request Broker Architecture
[9] Object Request Broker

3.3 Das Agenten-Paradigma - ein Umdenken bei der Programmierung

Die Erstellung von Anwendungen, die auf dem Einsatz mobiler Agenten beruhen, erfordert ein Umdenken, da gewisse Grundannahmen bisheriger Programmierparadigmen nicht mehr gültig sind. Gemeint ist, dass der Programmierer z.B. nie eine dauerhafte Annahme über die Beschaffenheit der Ablaufumgebung seines Agenten treffen darf, da diese sich bei jeder Migration drastisch ändern kann. Auch kann es durchaus passieren, dass Agenten verloren gehen - in diesem Bereich wird noch geforscht [20], eine allgemein gültige Lösung, die für alle Umgebungen optimal ist, wurde aber noch nicht gefunden. Eine Idee, die in [19] beschrieben wird, ist zum Beispiel, dass ein Agent bei seiner Reise „Sicherheitskopien" (sog. *„checkpoints"*) von sich anlegen kann, so dass er, falls sein aktueller Host abstürzt, wieder mit seinem alten Zustand hergestellt werden kann. Die Vor- und Nachteile dieser Lösung liegen auf der Hand - schon für den Einsatz transaktionsbasierter Systeme ist sie völlig ungeeignet.

Aus diesen Gründen wird die Programmierung von Agenten gar als Paradigmenwechsel angesehen, ähnlich dem beim Wechsel von der prozeduralen zur objektorientierten Programmierung[23]. Zur Verdeutlichung möchte ich ein kleines Beispiel für ein Problem anführen, das bei der Programmierung eines mobilen Agenten in einem System ohne transparente Migration (s. Abschnitt 3.1) auftritt:

Es soll ein Agent programmiert werden, der eine vorgegeben Auswahl von Stellen besucht. Der Agent könnte folgendermassen aussehen:

```
TestAgent1() {
  int i;                                  /* Definition der Laufvariablen */
  String[] Stellen=["Stelle 1","Stelle 2"];         /* Liste der Stellen */

  for (i=0;i<len(Stellen);i++) {   /* Die Stellen-Liste wird durchlaufen */
    go(Stellen[i]);                     /* Der Agent migriert zu Stelle i */
  }
}
```

Algorithmus 3.1: Einfacher Agent, der auf der Stelle tritt.

Bei diesem Beispiel-Code wird der Agent immer an Stelle 1 hängenbleiben. Das liegt daran, dass das System keine transparente Migration unterstützt, demzufolge wird der Agent nach jedem Aufruf der `go`-Methode neu gestartet. Das hat in dem obigen Beispiel aber zur Folge, dass jedes Mal wieder die Schleife von vorne gestartet wird. Ein wenig modifiziert kann der Agent aber trotzdem umherwandern:

```
TestAgent2() {
  int i=-1;                  /* Definition der lokalen Zustandsvariablen,*/
                            /* dieses Mal mit einem Initialisierungswert */
  String[] Stellen=["Stelle 1","Stelle 2"];        /* Liste der Stellen */

  if ( i<len(Stellen) ) {
    i = i+1;                  /* Den Stellenzähler um eins hochzählen ... */
    go(Stellen[i-1]);        /* ... und zu der gewünschten Stelle wandern */

  } else {
             /* Wenn alle Stellen besucht wurden, beendet sich der Agent */
    return;
  }
}
```

Algorithmus 3.2: Einfacher Agent, der verschiedene Stellen besucht.

Der Hauptunterschied bei dem modifizierten Agent ist, dass die Laufvariable einen Initialisierungswert bekommen hat. Da nun der Agent zwar auf jeder Stelle neu gestartet wird, die Werte der lokalen Variablen aber gesichert werden, bleibt der Wert von `i` nach jeder Migration erhalten.

Bei der Programmierung mit mobilen Agenten tun sich, wie oben bereits angedeutet, völlig neue Fragestellungen auf, die von Ausfallsicherheit über Sicherheit im Allgemeinen bis zu der Frage reichen, wie sich verschiedene Agentensysteme koppeln lassen und welche Kommunikationsprotokolle verwendet werden (denn erst dann werden sich mobile Agenten auf breiter Front durchsetzen - mit Sicherheit wird es nämlich nicht nur *ein* Agentensystem auf dem Markt geben).

An dieser Stelle belasse ich es bei den angegebenen Beispielen, da dieses Thema in einem anderen Seminarvortrag behandelt wird.

4 Agentensysteme

In diesem Abschnitt werden verschiedene Agentensysteme und deren Philosophie vorgestellt. Dabei soll klar werden, dass das Thema von durchaus verschiedene Richtungen angegangen und auch unterschiedlich interpretiert werden kann.

4.1 Telescript

Gleich zu Beginn soll das **Telescript**-System von General Magic vorgestellt werden, ohne dessen Erwähnung keine Arbeit mit dem Thema „Mobile Agenten" vollständig wäre. Es

handelt sich dabei um das erste kommerzielle Mobile-Agenten-System, das auf dem Markt war. Die Firma wurde 1990 mit dem Ziel gegründet, ein Agentensystem für portable Computer zu entwickeln und zu vermarkten. Die erste Version war schon 1994 erhältlich und wurde bald von verschiedenen Firmen wie AT&T, Motorola, Sony, Fujitsu und Nippon Telephone & Telegraph eingesetzt [14].

Das System besteht aus einer eigenen Programmiersprache (Telescript) und einer Laufzeitumgebung (Telescript Engine). Das gesamte Agentensystem besteht im Prinzip aus den drei Komponenten „Agent", „Stelle" und „`go`" (der Befehl, um die Stelle zu wechseln). Die Programmiersprache erlaubt transparente Migration und direkte Kommunikation ist möglich - Agenten können also Methoden von anderen Agenten aufrufen. Dabei gilt allerdings die Regel, dass bevor Agent A eine Methode eines anderen Agenten B aufrufen kann, A erst B „treffen" muss. Zu diesem Zwecke gibt es extra einen Operation `meet` bei jeder Stelle, die eine Referenz auf Agent B zurückliefert. Als Argument für die `meet`-Operation wird ein Petitions-Objekt übergeben, so dass A nur dann B treffen kann, wenn A eine gültige Petition besitzt.

Konflikte, die entstehen können, wenn zwei Agenten auf der gleichen Stelle das selbe Objekt referenzieren und einer die Stelle verlässt, werden nach dem *„object ownership concept"* gelöst: Wenn ein Agent ein Objekt erzeugt, so ist er Besitzer des Objektes. Ein Objekt verbleibt immer bei seinem Besitzer - wechselt dieser also die Stelle, so wandert das Objekt mit ihm ab, und alle Referenzen auf dieses Objekt, die unter Umständen andere Agenten noch haben, werden gelöscht. Agenten, die auf so eine gelöschte Referenz zugreifen bekommen einen Ausnahmefehler, was bei der Programmierung immer bedacht werden muss [16]. Agenten können aber auch z.B. durch *„events"* asynchron miteinander kommunizieren. Zu diesem Thema wird noch einiges in Abschnitt 4.2 bei der Vorstellung von AMETAS gesagt werden.

Das Sicherheitssystem setzt sich aus *„interpretation"*, *„credentials"* und *„permits"* zusammen. *„Interpretation"* bedeutet, dass die Telescript Engine einen Interpreter darstellt, der eine Laufzeitüberprüfung des Agenten-Codes vornimmt. *„Credentials"* erlauben es, dass sich Agenten und ihre Besitzer authentifizieren (dieses Vorgehen soll vor Virus-ähnlichen Agenten schützen) und *„permits"* definieren Restriktionen betreffend die Fähigkeiten eines Agenten (z.B. Lebensspanne, maximale Größe und Kosten, die verbraucht werden dürfen) [12].

Obwohl Telescript das erste System seiner Art auf dem Markt war, hat es sich bis heute nicht richtig durchsetzen können. Hauptkritikpunkte sind die proprietäre Programmiersprache, von der es keine freie Implementierung gibt [11] und deren schwere Erlernbarkeit der gegenüber bereits eine Menge von Agentensystemen stehen, die in Java programmiert werden, was den Lernaufwand für Programmierer, die bereits Java-Kenntnisse besitzen, minimiert. Auch die Einführung von **Tabriz AgentWare** zum Management von Agenten-basierten Applikations-Servern und **Tabriz AgentTools** für die Erstellung von Websites mit einer Schnittstelle für Telescript-Agenten, konnte an dieser Situation bis heute nichts ändern.

Gegenüber der wohldefinierten Programmierschnittstelle von Telescript steht momentan noch eine nicht standardisierte Java-API, wobei sich Java aber einer großen Beliebtheit erfreut. Der Trend scheint momentan sehr in Richtung Java zu gehen, da die Verbreitung der Sprache und die Verfügbarkeit der Java Virtual Machine auf allen wichtigen Plattformen in Kombination mit der zunehmenden Standardisierung, Hoffnung auf eine offene, standardisierte und plattformunabhängige Mobile-Agenten-Plattform verspricht.

4.2 AMETAS

Nach einem kommerziellen System mit einem proprietären Ansatz möchte ich nun ein System, das auf offenen Standards beruht und aus der Forschung stammt präsentieren. **AMETAS** steht für Asynchronous MEssage Transfer Agent System und wurde an der Goethe-Universität Frankfurt entwickelt. Entstanden ist es aus dem System **pl1** und wird seit 1996 von Helge Müller, Klaus Hermann und Michael Zapf weiterentwickelt. Das System wurde mit Hilfe des JDK1.1 entwickelt und ist komplett in Java geschrieben. Oberstes Ziel war es, die Autonomie der Agenten zu wahren und eine sichere Plattform zur Verfügung zu stellen, die möglichst plattformunabhängig ist.

Da AMETAS ein sehr modernes und gut durchdachtes Agentensystem darstellt, bei dem versucht wurde ein möglichst konsistentes System zu schaffen, möchte ich auf es an dieser Stelle genauer eingehen.

4.2.1 Die Struktur des Systems

Wie auch bei Telescript, gibt es in AMETAS „Agenten", „Stellen" und eine „go"-Operation. Dabei ist ein Agent wie folgt definiert:

> *„AMETAS-Agenten sind mobile, autonome Klienten, die in sich abgeschlossen sind und keine Referenz auf andere Agenten halten. Sie kommunizieren über einen asynchronen Nachrichtenaustausch-Mechanismus, um die Agentenautonomie zu wahren und die Kontrolle der Kommunikationsaktivitäten zum Zwecke ihrer Sicherheit und Typisierung zu gewährleisten. Ein direkter Zugriff auf das unterliegende System (Dateisystem, Grafikausgabe usw.) ist AMETAS-Agenten nicht gestattet." [10]*

Auffallen sollte, dass AMETAS keine synchrone Kommunikation von Agenten erlaubt. Vielmehr gibt es auf jeder Stelle ein *„Postamt"*, in dem Nachrichten für jeden Agent in sogenannten *„Postfächern"* abgelegt werden können. Der Agent kann sich bei der Stelle für Ereignisse registrieren, so dass er zum Beispiel ein Signal erhält, wenn eine neue Nachricht für ihn angekommen ist. Die Migration erfolgt im Gegensatz zu Telescript nicht-transparent, so dass wie in Abschnitt 3.1 beschrieben, alle wichtigen Daten vom Programmierer lokal in Objektvariablen gespeichert werden müssen.

Die Struktur des Systems soll folgende Grafik veranschaulichen:

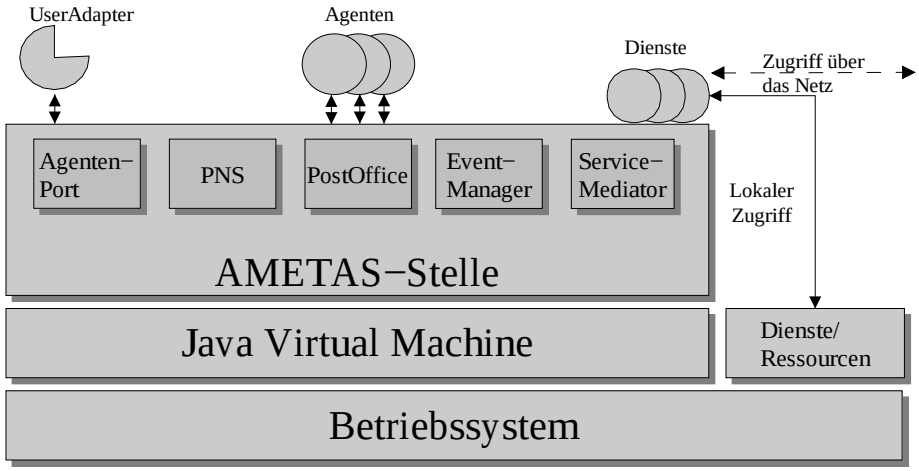

Abbildung 4.1: Struktur von AMETAS [10]

Wie man sieht läuft einer AMETAS-Stelle in einer Java Virtual Machine - auf einem Computer können also durchaus mehrere Stellen existieren. Auf der AMTAS-Stelle selbst laufen verschiedene sog. **Basisdienste**:

- Der **AgentenPort** stellt eine Schnittstelle für Anfragen von fremden Stellen dar. Über ihn können werden Agenten versendet und empfangen. Das Empfangen von Agenten setzt natürlich eine erfolgreiche Authentifizierung seitens der Absender-Stelle voraus und kann auch durchaus bei Sicherheitsbedenken abgelehnt werden.
- Für die Verwaltung des Namensraums ist der sog. Place Name Service (**PNS**) zuständig. Er verwaltet die Namen aller Stellen und verhindert so, dass es zu Inkonsistenzen bei der Namensvergabe kommt. Seine Arbeitsweise ist vergleichbar mit der eines DNS[10]. Wie dort auch können mehrere PNS hierarchisch zu einer verteilten Namens-Datenbank gekoppelt werden.
- Wie bereits oben beschrieben, können sich Agenten bei der Stelle für bestimmte Ereignisse registrieren. In diesem Falls sorgt der **EventManager** dafür, dass jeder Agent über die Ereignisse informiert wird, für die er sich registriert hat, und diese dann auch übergeben bekommt.
- Der **ServiceMediator** sammelt die Beschreibungen aller installierter Dienste und nimmt Anfragen von Agenten entgegen. Auf eine Anfrage hin wird im bestehenden Dienstangebot nach passenden Diensten gesucht und deren IDs dann an den Fragesteller zurückgeliefert.

[10] Domain Name Server - ist verantwortlich für die Namensauflösung im Internet.

4.2.2 Sicherheit

Besonders interessant (und mindestens ebenso kompliziert und umfassend) ist das Sicherheitssystem, das AMETAS zugrunde liegt. Im System werden verschiedene Entitäten unterschieden: Der Agent selber, der Benutzer (User), der Benutzer-Adapter, der Dienst und die Stelle. Ein **Dienst** stellt auf einer Stelle bestimmte Dienste Agenten zur Verfügung, wenn diese genügend Rechte besitzen, und **Benutzer-Adapter** stellen ein Frontend zu Steuerung des Agentensystems dar. Sowohl Benutzer-Adapter und Dienste als auch die Agenten selber kommunizieren nur über sog. **Treiber** mit der Stelle. Diese Architektur soll die **Stellennutzer** (Agenten, die auf einer Stelle laufen) und Dienste vor Sicherheitsangriffen schützen und eine einheitliche, sauber definierte Schnittstelle zur Stelle bringen. Jede Entität im System besitzt eine eindeutige Identität in Form eines privaten und eines öffentlichen Schlüssels, so dass auch hier Anonymität vermieden wird. Jede Migration erfolgt verschlüsselt, so dass ein Abhören der Leitung kein Sicherheitsrisiko darstellt.

Das Treibermodell von AMETAS kann wie folgt veranschaulicht werden:

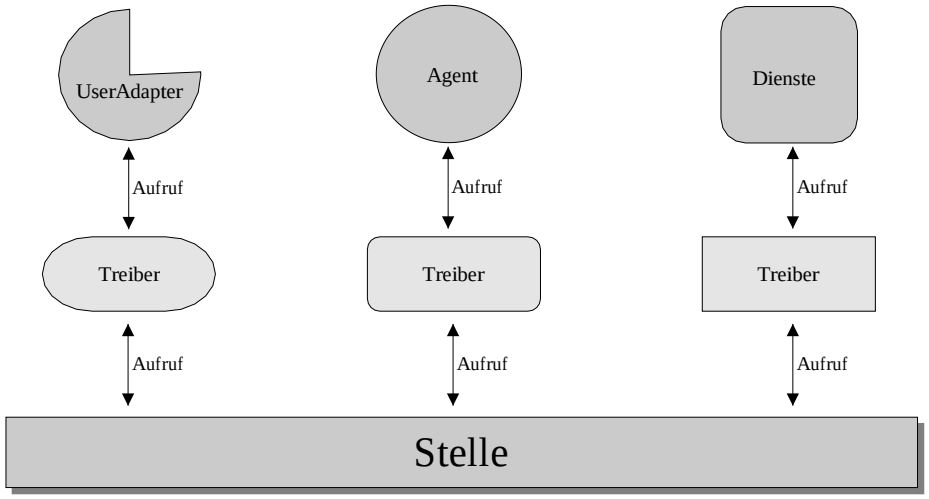

Abbildung 4.2: Die Treiberschicht zwischen Stellennutzern und der AMETAS-Stelle [10]

4.2.3 Beispiel: NetDoctor

Bei NetDoctor handelt es sich um ein System, das Server im Netzwerk überwachen soll. Dabei „reisen" mobile Agenten (sog. **Health Agents**) von Rechner zu Rechner und überprüfen anhand von sog. Health-Funktionen, ob der Rechner ordnungsgemäß läuft. Ist dies nicht der Fall, so versuchen die Agenten, die Situation durch Konfigurationsänderungen zu verbessern. Weil die Abfrage der Zustandsdaten vom Server über SNMP[11] erfolgt, wird

[11] Simple Network Management Protocol - Protokoll, mit dem Statusinformationen über Rechner ausgetauscht und sogar Konfigurationsänderungen vorgenommen werden können.

dieser Ansatz (die Verbindung des Agentenparadigmas mit dem Client/Server-Paradigma) in [10] *hybrid* genannt.

Ein **Health Agent** wird auf seiner Heimatstelle gestartet und migriert dann zur Zielstelle, auf der er sich für verschiedene Dienste registriert. Sodann beginnt er mit der Überwachung der Stelle mittels SNMP. Anhand eine **Health-Funktion** kann der Agent nun berechnen, wie es um die „Gesundheit" des Rechners bestellt ist. Dieser Ansatz hat für das Netzwerkmanagement folgende Vorteile:

- Verringerung der Netzwerklast.
- Erhöhung der Abtastrate, ohne dass das Netzwerk belastet wird.
- Verteilung der Rechenlast auf alle Komponenten im Netz.
- Schnelle mögliche Reaktion auf kritische Situationen.
- Gute Skalierbarkeit.

Die Health Agents haben in dem System folgende Eigenschaften:

- **Zielorientiertheit:** Nämlich die Erhaltung der Rechner„gesundheit".
- **Glaube:** Der Agent glaubt, dass ein kritischer Zustand vorliegt, wenn eine Schwellwert überschritten wurde.
- **Sensoren:** Mittels SNMP kann sich der Agent ein Bild vom Status des Rechners machen.
- **Effektoren:** Mittels dem SNMP-`set`-Befehl kann der Agent bestimmte Konfigurationsänderungen an dem Rechner vornehmen

Healthfunktionen stellt im Prinzip ein Mittel semantische Kompression von n Messwerten in *einen* Index dar, anhand derer der Health Agent den Status des Rechners ermitteln kann. Da es sich hierbei meist um sehr komplexe Zusammenhänge handelt, ist der Einsatz von Perzeptren[12] abzuwägen, die auf den jeweiligen Einsatz trainiert werden müssen und dann Problemsituationen aufgrund des Erlernten selbst erkennen. Der Nachteil bei dieser Methode ist allerdings eine neue Trainingsphase bei jeder Umkonfiguration der Hardware. Auch das finden eines geeigneten Perzeptrons kann sich als nicht immer einfach erweisen.

[12]Bei einem Perzeptron handelt es sich um um das einfachste Modell aus dem Bereich künstlicher neuronaler Netze. In einem n-dimensionalen reellwertigen Eingabevektor i werden mittels eines ebenfalls n-dimensionalen Gewichtungsvektors allen Komponenten von i durch Multiplikation Gewichtungen zugeordnet. Nach weiteren Berechnungen kommt am Ende als Ergebnis eine 0 oder eine 1 (das Perzeptron *feuert*) heraus[2].

Da die Health Agents und die Belastung (Speicher und Prozessorzeit) auf den zu überwachenden Rechnern möglichst klein gehalten werden sollen, kann kann man die Intelligenz, die es benötigt, um Fehlersituationen zu beheben, aus den Health Agents herausnehmen und komplett in sog. **Mobile Manager** verlegen. Diese werden dann im Notfall gerufen, um Umkonfigurationen an den Rechnern vorzunehmen und wandern dann wieder von selbst ab. Dadurch ist gewährleistet , dass die Rechner nur so wenig wie möglich belastet werden:

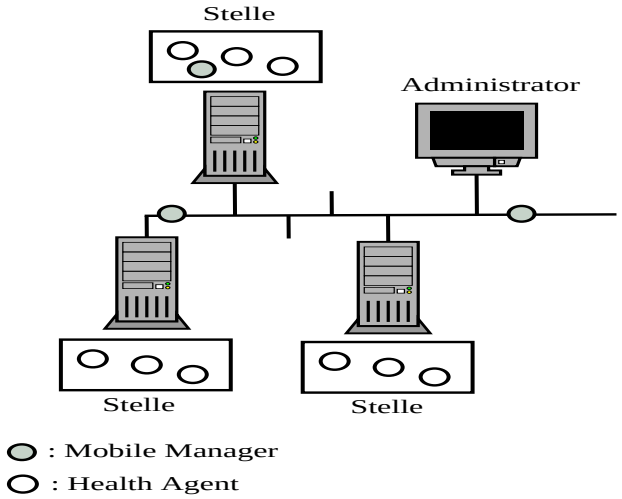

Abbildung 4.3: Das NetDoktor-Konzept [10]

5 Schlussbemerkung

Natürlich gibt es auch Zweifel am nutzbringende Einsatz mobiler Agenten. In [5] werden verschiedene Anwendungsbereiche für mobile Agenten diskutiert und für jeden Fall eine Methode (meist RPC[13]) gefunden, die im Einzelfall mobilen Agenten vorzuziehen wäre. Die Autoren kommen aber zu dem Schluss, dass Agenten ihren Nutzen nicht in Einzelapplikationen, sondern in ihrem Zusammenwirken erbringen.

In Bezug auf die zukünftige Entwicklung mobiler Agenten ist festzustellen, dass der Einsatz mobiler Agenten zunehmen wird, da die Vernetzung von Computer-Systemen stetig zunimmt, was den Wunsch nach neuen Technologien für wiederkehrende Aufgaben laut werden lässt. Die Aussicht auf eine Reduzierung der Netzwerklast und vor allem die Einsatzmöglichkeiten asynchron und autonom arbeitender Programme, die dann hoffentlich ausfallsichere und robuste Systeme bringen, treibt die Forschung in diesem Bereich voran. Allein die Arbeit an einer Standard *„Java Agent API"* [13] zeigt, was für ein Gewicht mobile Agenten mittlerweile haben. Eine Standardisierung in diesem Bereich wird aber essentiell

[13]Remote Procedure Calls - der entfernte Aufruf von Prozeduren auf einem Server.

für die Weiterverbreitung dieser Technik sein, da es jetzt schon sehr viele verschiedene Agentensysteme gibt, deren Kopplung erst gewünschten Synergien freisetzen wird.

Zu guter Letzt sei noch einmal auf den Punkt „Sicherheit" hingewiesen. Gerade die Gefahr, die von Würmern oder trojanischen Pferden ausgeht, ist in letzter Zeit in der Öffentlichkeit publik geworden, und es muss in Agentensystemen ein Weg gefunden werden, um derart bösartige Programme zu verhindern. Auch die Sicherheit der Agenten selber und auch der Agentenhosts ist ein vieldiskutierte Frage. Wie am Beispiel AMETAS gezeigt wurde, gibt es hier durchaus schon gute Ansätze, das Thema „Sicherheit" bleibt aber trotzdem immer ein ganz zentrales, wenn es um den Einsatz mobiler Agenten geht.

Wahrscheinlich wird die Einführung von Agentensystemen für den „normalen" Benutzer transparent geschehen, so dass dieser gar nicht weiß, wie seine Aufträge erfüllt werden. So revolutionär der Begriff „mobiler Agent" zu Anfang klingen mag, so unspektakulär wird die Integration dieser Technik in den Alltag wahrscheinlich verlaufen (auch der Einsatz und die Entwicklung moderner Komponentenmodelle mag für den Entwickler revolutionär sein - der Benutzer nimmt dies jedoch höchstens in Form von (hoffentlich) besserer Software wahr.)

Literatur

[1] http://www.mysap.com/.

[2] Das perzeptron. http://www.cl-ki.uni-osnabrueck.de/ nnmsaure/perzeptr.html.

[3] Antonio Carzaniga, Gian Pietro Picco, and Giovanni Vigna. Designing Distributed Applications with Mobile Code Paradigms. Research report.

[4] A. Chavez, D. Dreillinger, R. Guttman, and P. Maes. A Real-Life Experiment in Creating an Agent Marketplace. In *Lecture Notes in Computer Science*, volume Software Agents and Soft Computing of *Lecture Notes in Artifical Intelligence*. Springer Verlag, 1997.

[5] David Chess, Colin Harrison, and Aaron Kershenbaum. Mobile Agents: Are They a Good Idea? Research report, IBM, 1995. http://www.infosys.tuwien.ac.at/Research/Agents/archive/mobagtibm.ps.gz.

[6] P. E. Clements, Todd Papaioannou, and John Edwards. Aglets: Enabling the Virtual Enterprise. In *Managing Enterprises - Stakeholders, Engineering, Logistics and Achievement*. Loughborough University, 1997. http://luckyspc.lboro.ac.uk/Docs/Papers/Mesela97.html.

[7] Lektorat des B.I.-Wissenschaftsverlags, editor. *Informatik-Duden*. DUDEN-Verlag, 1993.

[8] Stan Franklin and Art Graesser. Is it an Agent, or just a Program? In *Lecture Notes in Computer Science*, volume Third International Workshop on Agent Theories, Architectures, and Languages of *Lecture Notes in Artifical Intelligence*. Springer Verlag, 1996.

[9] Michael R. Genesereth and Steven P. Ketchpel. Software Agents. Technical report, Stanford University.

[10] Klaus Herrmann. Netz- und Systemmanagement mit mobilen Agenten im AMETAS-Agentensystem, 1998.

[11] Frederick Knabe. An Overview of Mobile Agent Programming. Research report.

[12] Keith D. Kotay and David Kotz. Transportable Agents. Research report, Department of Computer Science, 1994.

[13] Danny B. Lange. Present and Future Trends of Mobile Agent Technology. In *Lecture Notes in Computer Science*, volume Mobile Agents of *Porceedings*. Springer Verlag, 1998.

[14] George Lawton. Agents to roam the Internet. *SunWorld*, 1996. http://www.sunworld.com/swol-10-1996/swol-10-agent.html.

[15] Anselm Lingnau, Roger Brand, Andreas Möbs, and Oswald Drobnik. Produktrecherche mit mobilen Agenten in einem elektronischen Markt. Forschungsbericht, Fachbereich Informatik, Telematik (ABVS/Telematik). ftp://ftp.tm.informatik.uni-frankfurt.de/pub/papers/agents/kek98-paper.ps.gz.

[16] Anselm Lingnau, Peter Dömel, and Oswald Drobnik. Mobile Agent Interaction in Heterogeneous Environments. Forschungsbericht, Fachbereich Informatik, Telematik (ABVS/Telematik). ftp://ftp.tm.informatik.uni-frankfurt.de/pub/papers/agents/ma97.ps.gz.

[17] H.S. Nwana and D. T. Ndumu. An Introduction to Agent Technology. In *Lecture Notes in Computer Science*, volume Software Agents and Soft Computing of *Lecture Notes in Artifical Intelligence*. Springer Verlag, 1997.

[18] H.S. Nwana and M. Wooldridge. Information Agents for the World Wide Web. In *Lecture Notes in Computer Science*, volume Software Agents and Soft Computing of *Lecture Notes in Artifical Intelligence*. Springer Verlag, 1997.

[19] Holger Peine and Torsten Stolpmann. The Architecture of the Ara Platform for Mobile Agents. In *Lecture Notes in Computer Science*, volume Mobile Agents of *First International Workshop, MA '97*. Springer Verlag, 1997.

[20] Flavio M. Assis Silva and Radu Popescu-Zeletin. An Approach for Providing Mobile Agent Fault Tolerance. In *Lecture Notes in Computer Science*, volume Mobile Agents of *Proceedings*. Springer Verlag, 1998.

[21] Bret Sommers. Agents: Not just for Bond anymore. *JavaWorld*, 1997. http://www.javaworld.com/javaworld/jw-04-1997/jw-04-agents.html.

[22] Todd Sundsted. An Introduction to Agents. *JavaWorld*, 1998. http://www.javaworld.com/javaworld/jw-06-1998/jw-06-howto.html.

[23] Bill Venners. Solve real problems with aglets, a type of mobile agent. *JavaWorld*, 1997. http://www.javaworld.com/javaworld/jw-05-1997/jw-05-hood.html.