Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Seminararbeit - Master Informatik

Ilia Revout

Verteilte Anwendungen für lokale Netzwerke auf der Grundlage der Technologien mobilen Agenten

Fakultät Technik und Informatik
Studiendepartment Informatik
Betreuer: Prof. Dr. von Luck
Datum: 30. Dezember 2005

Ilia Revout

Thema der Seminararbeit - Master Informatik

Verteilte Anwendungen für lokale Netzwerke auf der Grundlage der Technologien mobilen Agenten

Stichworte

Server, Client, Peer-To-Peer, Mobile Agenten, FIPA, MASIF

Kurzzusammenfassung

In diesem Artikel wird ein kurzer Überblick über die Arten der Kommunikation in verteilten Anwendungen gegeben. Etwas ausführlicher wird die Technik der mobilen Agenten vorgestellt. Es werden zwei wichtige Standards für die Agentensysteme präsentiert und miteinander verglichen. Am Ende wird eine Idee vorgestellt, wie diese Technik in der leichten Form in einem lokalen Netzwerk eingesetzt werden könnte.

Inhaltsverzeichnis

1 Einführung **4**

2 Client-Server Architektur **4**

3 Peer-To-Peer **5**

4 Kommunikationsarten **5**
4.1 Remote Procedure Call 5
4.2 Verteilte Objekte 6
4.3 Remote Evaluation 6
4.4 Prozessmigration 6
4.5 Code on Demand: Java Applets 6
4.6 Mobile Agenten 7

5 Mobile Objekte vs. Mobile Agenten **7**

6 Vorteile von mobilen Agenten **8**

7 Agentenplattformen **8**

8 Standards für Agentenplattformen **9**
8.1 MASIF 10
8.2 FIPA 11
8.3 Vergleich von MASIF und FIPA 11

9 Die Technologie der mobilen Agenten für verteilte Anwendungen in lokalen Netzwerken **11**

1 Einführung

Verteilte Anwendungen gewinnen immer mehr an Bedeutung, da die Komplexität heutiger Anwendungen meistens die Möglichkeiten der einzelnen Rechner überschreitet. Auch aus der Sicht der Wiederverwendbarkeit macht es mehr Sinn bestimmte Services von anderen in Anspruch zu nehmen und nicht selbst für die Implementierung Zeit zu investieren. Eine Definition soll nun etwas deutlicher machen, was genau eine verteilte Anwendung ist: 'Eine verteilte Anwendung ist eine Anwendung, deren Funktionalität in eine Menge von kooperierenden Teilkomponenten zerlegt ist. Jede Teilkomponente hat einen internen Zustand. Teilkomponenten sind autonome Verarbeitungseinheiten, die auf verschiedene Rechner abgebildet werden können. Jede Teilkomponente läuft als Prozess auf ihrem eigenen Rechner ab. Teilkomponenten tauschen untereinander Informationen über ein Netzwerk aus.' [1]
Die folgende Liste gibt einen Überblick über die Vorteile von verteilten Anwendungen gegenüber einer nicht verteilten:

- Effizienz durch parallele Verarbeitung
- Fehlertoleranz, Ausfallsicherheit, Verfügbarkeit
- Gemeinsame Nutzung von Ressourcen
- Integration bestehender Lösungen, Erweiterbarkeit

In dieser Kurzfassung werden die Technologien vorgestellt, die für die Realisierung der verteilten Anwendungen meistens eingesetzt werden. Etwas ausführlicher wird die Technologie der mobilen Agenten erklärt. Es werden zwei Standards für Agentensysteme präsentiert und verglichen. Am Ende wird eine Idee vorgestellt, wie eine verteilte Anwendung mit Hilfe von Technologien der mobilen Agenten realisiert werden könnte.

2 Client-Server Architektur

Der am meisten verbreitete Ansatz für eine verteilte Anwendung ist die Client-Server Architektur. Die Grundidee besteht darin, dass ein zentraler Rechner die Ressourcen besitzt und sie für die anderen Rechner zur Verfügung stellt. Die Anzahl der Ressourcen, die ein Server zur Verfügung gestellt hat, ändert sich ständig mit der Zeit. Früher, als die Ressourcen noch relativ teuer waren, hat der Server die meisten Aufgaben übernommen. Die Clients wurden praktisch nur als Terminals verwendet. Der Hauptgrund der Verteilung war das Einsparen von Kosten. Die Ressourcen werden immer günstiger, so dass viele Clients nun selbst in der Lage sind, die Aufgaben zu bewerkstelligen. Aus dem Server mit vielen Ressourcen wurde meist der Datenbankserver. Die dabei verfolgte Idee ist der gemeinsame Datenbestand.

3 Peer-To-Peer

Anders wird bei dem *Peer-To-Peer Ansatz* vorgegangen. Hier übernimmt jeder Rechner sowohl die Funktionalität des Clients als auch des Servers. Die Vorteile dieses Ansatzes sind offensichtlich:

- Unabhängigkeit von einem Server
- Ausfallsicherheit
- Geringe Kosten
- Leichte Installation

Mit vielen Vorteile werden leider auch die Nachteile eingekauft. So ist es zum Beispiel viel schwerer einen Sowtwareupdate durchzuführen. Auch die Synchronisation oder gemeinsame Datensicherung wird damit erschwert. Allgemein kann man nicht sagen, welcher Ansatz besser ist. Die Entscheidung muss in Abhängigkeit vom konkreten Anwendungsfall getroffen werden. An dieser Stelle muss aber gesagt werden, dass Peer-To-Peer als Technologie noch nicht ausreichend erforscht ist. "Das ist so, wie wenn sie einen Ferrari in der Garage haben, aber sich nur zum Musikhören hinein setzen"[2]

4 Kommunikationsarten

Bei der Betrachtung der Kommunikation zwischen zwei Rechnern spielt es keine Rolle mehr, welche Architektur (Client-Server oder Peer-To-Peer) verwendet wird. Der Rechner, der die Anfrage startet, übernimmt für diese Zeit die Rolle des Clients. Der Rechner, der auf eine Anfrage wartet, ist dann ein Server. An dieser Stelle ist irrelevant, welcher Rechner die Verbindung aufgebaut hat. Wichtig ist nur, dass die Verbindung vorhanden ist. Hier wird die Möglichkeit einer Anwendung untersucht, die entfernte Ressourcen in Anspruch nimmt. Nachfolgend werden ein Paar Technologien vorgestellt, die die Verteilung der Ressourcen übernehmen.

4.1 Remote Procedure Call

Die am meisten verbreitete Technologie ist der *Remote Procedure Call*. Wie die Bezeichnung schon verrät, handelt es dabei um den entfernten Aufruf einer Methode. Der Client muss die genaue Beschreibung der Methode kennen: den Namen der Methode, die erwarteten Parameter und den Rückgabewert. Bei dem Aufruf verhält sich die Anwendung genau so, als ob die Methode lokal vorhanden wäre: synchron und blockierend. Für den Anwender ist dieser Vorgang absolut transparent. Der Nachteil liegt hier in der Schnittstelle, welche die Kommunikation genau beschreibt. Wird auf dem Server die Signatur der Methode verändert, so müssen alle Clients auch entsprechend angepasst werden.

4.2 Verteilte Objekte

Bei diesem Ansatz wird der Remote Procedure Call um die Objektorientierung erweitert. Es existieren unterschiedliche Objekte auf den unterschiedlichen Rechnern, die eine Menge von Operationen anbieten. Diese Objekte sind überall im System eindeutig und können von jedem Client in Anspruch genommen werden. Dazu kommt noch, dass die Objekte von einem Rechner zu einem anderen verschoben werden können. Anstatt von Stubs, die bei dem Remote Procedure Call fest vorgegeben sind, werden hier die Proxies verwendet, die auch zur Laufzeit dynamisch erstellt werden können. Die Nachteile der fest definierten Schnittstellen sind hier nicht vorhanden. Zu den weiteren Vorteilen kann auch die Zustandbehaftbarkeit der Objekte gezählt werden. So ist z.B. vorstellbar, dass an einem Objekt mehrere User gemeinsam arbeiten können, obwohl sie sich physikalisch an unterschiedlichen Orten befinden können.

4.3 Remote Evaluation

Oft ist es sinnvoll, die Methoden, die rechenintensiv sind, nicht lokal, sondern auf einem leistungsfähigeren Rechner ausführen zu lassen. Da kommt der *Remote Evaluation* Ansatz ins Spiel. Hierbei handelt es sich um eine Technologie, mit deren Hilfe eine Prozedur mit allen notwendigen Daten von einem Rechner zu einem anderen verschickt, dort ausgeführt und das Ergebnis zurückgesendet wird. Diese Methodik kann auch dann eingesetzt werden, wenn die Lösung eines Problems zerlegt werden kann. In solch einem Fall können mehrere Rechner eingesetzt werden, die einzeln nicht leistungsfähig genug sind, um gemeinsam das Problem lösen zu können. Existiert eine Umgebung, die diesen Ansatz unterstützt, so ist es möglich, beliebige Methoden auf den entfernten Rechnern ausführen zu lassen. Die Kommunikationsschnittstelle im Sinne der Remote Procedure Call existiert hier nicht.

4.4 Prozessmigration

Wenn es um die Lastverteilung geht, dann ist die Homogenität des gesamten Systems als Vorteil zu sehen. Es existiert eine Reihe von Betriebssystemen, die in der Lage sind, vollständig autonom die Prozesse von einem Rechner zu einem anderen zu verlagern, abhängig von der Lastverteilung. Dabei geschieht dies transparent, sowohl für Benutzer als auch für Entwickler. Die homogene Umgebung kann aber nicht nur über das Betriebssystem erreicht werden. Diese Aufgabe kann auch von einer Umgebung (eine Art Virtueller Maschine) übernommen werden. In diesem Fall muss allerdings die Anwendung für diese Umgebung angepasst werden.

4.5 Code on Demand: Java Applets

Java Applets sind sehr verbreitet und fast jedem bekannt. Sie sind ein typisches Beispiel für die *Code on Demand* Technik. Dabei wird die gesamte Anwendung von dem Server zum Client übertragen und dort ausgeführt. Der Nachteil dieser

Technologie liegt in der Tatsache, dass der Server erst dann den Code senden kann, wenn der Client ihn angefordert hat. Normalerweise wird dabei nur der Code der Anwendung übertragen. Mit etwas mehr Aufwand ist es aber auch möglich die Daten zu übertragen. Damit haben wir absolute Unabhängigkeit von der Schnittstellenbeschreibung zwischen dem Server und dem Client.

4.6 Mobile Agenten

Die Nachteile, die bei dem Code on Demand vorliegen, werden mit der Technologie der *mobilen Agenten* letztendlich behoben. Mobile Agenten sind Programme, die in der Lage sind, sich von einem Rechner zu einem anderem zu bewegen. Dabei wird das Programm und die Daten mit dem Zustand übertragen. Dieser Prozess wird Migration genannt. Eine weitere entscheidende Eigenschaft der mobilen Agenten ist ihre Fähigkeit, selbständig die Entscheidung zu treffen, ob eine Migration stattfinden soll oder nicht. Auch wohin migriert wird, entscheiden die mobilen Agenten selbst. Für diese Fähigkeiten ist eine spezielle Umgebung zuständig. Diese Umgebung wird die Agentenplattform genannt und ist ähnlich einer virtuellen Maschine (VM) aufgebaut. Der Agent wird innerhalb der Agentenplattform ausgeführt und überwacht. Über die Agentenplattformen wird etwas detaillierter im Abschnitt „Agentenplattformen“ berichtet. Sobald ein Agent die Entscheidung getroffen hat zu migrieren, sendet er einen Wunsch an die Agentenplattform. Diese Plattform packt den Agenten zusammen mit den kompletten Daten und schickt ihn zu dem gewünschten Rechner weiter. Es werden zwei Arten der Migration unterschieden: Pull- und Push-Migration. Bei der Push-Migration wird, wie oben beschrieben, die gesamte Applikation mit den Daten und dem Zustand verschickt. Ein Nachteil ist dabei, dass auch die Programmteile übertragen werden, die niemals benutzt werden. Der Vorteil ist, dass die Verbindung nach der Übertragung nicht mehr gebraucht wird, und damit getrennt werden kann. Bei der Pull-Migration werden nur notwendige Daten übertragen. Sobald weitere Komponenten gebraucht werden, werden diese nachgeladen. Hier wird zwar an dem erstmaligen Datenvolumen gespart, allerdings wird eine ständige Verbindung zum Homerechner gebraucht.

5 Mobile Objekte vs. Mobile Agenten

Da die mobilen Agenten gewisse Ähnlichkeiten zu mobilen Objekten aufweisen, werden hier diese beiden Technologien verglichen. Die mobilen Objekte werden meistens für die Lastverteilung eingesetzt, dagegen haben die mobilen Agenten ihren Schwerpunkt in der Nutzung der verteilten Ressourcen. Die Migration wird durch das Betriebssystem bzw. Middleware bei den mobilen Objekten ausgelöst, bei den mobilen Agenten sind es die Agenten selbst, die die Migration anstoßen. Die mobilen Objekte stellen nur ein Teil der Applikation dar, die mobilen Agenten dagegen sind die komplette Applikation selbst. Auch bei der Kommunikation ist der Unterschied groß. So kommunizieren die mobilen Objekte mit Hilfe von

Methodenaufrufen, dagegen senden die mobilen Agenten die Nachrichten.

6 Vorteile von mobilen Agenten

„Eine Lösung“ für alle möglichen Fälle existiert nicht. Auch der Einsatz von mobilen Agenten ist nur in bestimmten Situationen vorteilhaft. In diesem Abschnitt sind ein Paar Vorteile aufgelistet, die für die Entscheidung, ob diese Technologie eingesetzt werden sollte oder nicht, nützlich werden können. An ersten Stelle steht die Belastung des Netzes. Nachdem ein Agent übertragen wurde, kann die Verbindung auch getrennt werden. Die Menge von Informationen, die für das Ergebnis irrelevant sind, wird nicht hin und her verschickt. So kann z.B. ein Agent direkt am Rechner, auf dem die Datenbank läuft, eine Abfrage starten, das Ergebnis analysieren und nur das gesuchte Tupel zurücksenden. Die Tatsache, dass Agenten vollständig asynchron arbeiten, ist ein weiterer Vorteil. So braucht der Anwender nicht mehr zu warten, bis die Arbeit von einem mobilen Agenten vollständig ausgeführt worden ist. Sobald der mobile Agent den Rechner verlassen hat, werden die Ressourcen des Rechners komplett freigegeben. Erst, wenn der mobile Agent zurückkommt, bekommt der Benutzer das Ergebnis. Bevor die Agenten migrieren, können sie auch die Verbindung zu unterschiedlichen Rechnern prüfen, falls mehrere in Frage kommen. Damit sind sie in der Lage, Netzwerklatenz zu vermeiden.

Das Verhalten von mobilen Agenten in Fehlerfällen soll auch betrachtet werden. Die Fehler, die mit der Nutzung von Ressourcen zusammenhängen (Anfragen und Antworten), sind praktisch ausgeschlossen, da die gesamte Kommunikation lokal stattfindet. Folglich können Fehler nur durch einen Serverabsturz verursacht werden. Dabei kommen zwei Fälle in Frage. Bei einem absehbaren Absturz kann die Agentenplattform den Agenten einfrieren und nach dem Neustart des Rechners wieder laufen lassen. Eine andere Möglichkeit ist es dem Agenten selbst die Handlungsentscheidung zu überlässen. Er hat in diesem Fall die Möglichkeit, vor dem Absturz von diesem Rechner weiter zu migrieren. Wenn der Absturz nicht vorhersehbar ist, kann natürlich nichts unternommen werden. Dieses Problem ist weiterhin offen. Keine Probleme bereitet dagegen der Absturz des Homerechners. Der Agent wartet so lange, bis der Homerechner wieder zur Verfügung steht und kommt dann mit dem Ergebnis zurück.

7 Agentenplattformen

Alle tollen Fähigkeiten des Agenten sind keine wirkliche Leistung der Agenten selbst. Wie schon kurz erwähnt wurde, sind es die Agentenplattformen, die alles dafür notwendige zur Verfügung stellen. Hier ist nur ein kurzer Auszug daraus, was die Agentenplattformen leisten sollen:

- Agenten aufnehmen und sie ausführen

- Migration der Agenten ermöglichen
- Agenten eindeutig identifizieren
- Kommunikation bereitstellen
- Sicherheitsprobleme vermeiden
- Entfernte Steuerung für den Verwalter ermöglichen

Manche Bereiche sind sehr umfangreich und stellen eine echte Herausforderung für die Entwickler dar. So z.B. die Frage der Sicherheit. Dabei müssen zwei Seiten betrachtet werden: Die Agentenplattform selbst soll von bösartigen Agenten geschützt werden, aber auch die Sicherheit der Agenten soll gewährleistet werden. Ein weiterer Schwerpunkt ist die Kommunikation. Die erste Problemstellung ist das Finden eines gesuchten Agenten. In diesem Bereich gibt es unterschiedliche Verfahren, die an dieser Stelle nur kurz erwähnt werden sollen:

- Broadcast: Es werden alle Agentenpattformen gefragt, ob ein bestimmter Agent zur Zeit registriert ist.
- Blackboard: Jeder Agent meldet sich bei der 'Zentrale', wenn er seinen Aufenthaltsort wechselt und teilt der Zentrale seine neuen Koordinaten mit.
- Spur: Jeder Agent hinterlässt eine Information darüber, wohin er gehen wird.

Dazu kommt ein weiteres Problem. Da die Agenten sich mit der gleichen Geschwindigkeit über das Netzwerk bewegen wie die Nachrichten, kann es durchaus vorkommen, dass die Nachrichten den Agenten aus physikalischen Gründen nicht erreichen werden.

Es wird zwischen drei Migrationsarten unterschieden: starke Migration, schwache Migration mit festem Einsprung und schwache Migration mit beliebigem Einsprung. Bei der starken Migration wird die Ausführung des Agenten genau an der Stelle fortgesetzt, wo er seine Arbeit unterbrochen hat. Bei der schwachen Migration mit festem Einsprung wird beim Starten des Agenten immer die gleiche Methode aufgerufen. Bei beliebigem Einsprung können unterschiedliche Methoden aufgerufen werden. Auch hier ist es verständlich, dass diese Aufgabe von der Agentenplattform übernommen werden muss. Deswegen unterscheiden sich die Agentenplattformen auch nach diesem Kriterium. Die Beispiele dafür sind D'Agents [3] für die starke Migration, Aglets [4] für die schwache Migration mit festem Einsprung und Voyager [5] für die schwache Migration mit beliebigen Einsprung.

8 Standards für Agentenplattformen

Spätestens jetzt sollte deutlich werden, dass nicht jeder beliebige Agent überall seine Arbeit durchführen kann. Die Agenten können nur auf den Agentenplattformen ausgeführt werden, für die sie geschrieben wurden. Diese Homogenität macht den

gesamten Ansatz der mobilen Agenten praktisch zur Nichte. Diese Schwachstelle wurde von zwei Organisationen erkannt. Eine Organisation ist die OMG (Object Management Group) [6] und die zweite die FIPA (Foundation for Intelligent Physical Agents) [7]. Beide Organisationen haben sich als Ziel das Herausbringen eines Standards für die Agentenplattformen gesetzt und damit das oben benannte Problem der Homogenität zu lösen. Dieser Standard sollte dafür sorgen, dass jeder beliebige Agent auf jeder beliebigen Agentenplattform laufen kann, wenn beide den Standard implementiert haben. Obwohl beide Organisationen sich etwa gleiche Ziele gesetzt haben, liegen die Schwerpunkte der beiden entwickelten Standards in unterschiedlichen Bereichen. In den folgenden Abschnitten sollen beide Standards kurz vorgestellt und verglichen werden.

8.1 MASIF

Der Standard MASIF wurde von der OMG herausgebracht. Die folgenden Bereiche wurden von der OMG standardisiert:

- die Terminologie, um ein einheitliches Verständnis von Begriffen zu erreichen
- die Namensgebung für Agenten und Agentensysteme, damit eine eindeutige Identifizierung möglich wird
- die Lokalisierung von Agenten und Agentensystemen, um eine Kommunikation zu etablieren
- der Agententransfer zur Unterstützung der Mobilität von Agenten
- das Agentenmanagement, um den Lifecycle eines Agenten zu verwalten

Ein Agent wird im MASIF-Standard wie folgt definiert: Ein Agent ist ein ablauffähiges Programm, welches autonom im Auftrag einer Authority (Person oder Organisation) handelt.

Die Agentenplattform besteht aus mehreren Bereichen, die Places genannt werden. Jeder Place hat bestimmte Rechte für die Ressourcen und kann mehrere Agenten aufnehmen. Damit wird gewährleistet, dass unterschiedliche Agenten nach den Rechten in Gruppen unterteilt werden. Für die Kommunikation zwischen unterschiedlichen Plattformen ist die 'Communication Infrastructure' zuständig. Sie wird mit Hilfe von CORBA bereitgestellt. Mehrere Plattformen einer Authority können zu einer Region zusammengefasst werden.

Für die Implementierung des Standards werden zwei Schnittstellen bereitgestellt:

- MAFAgentSystem-Interface stellt Operationen zum Transport und Management der Agenten zur Verfügung.
- MAFFinder-Interface ist für die Lokalisierung von Agenten und Agentensystemen gedacht und bringt den „naming service“ für die Agenten mit.

8.2 FIPA

Etwas anderes versteht der FIPA-Standard unter einem Agenten: Ein Agent ist ein Akteur, der auf einer Agentenplattform ausgeführt wird und intelligent im Sinne der KI ist. Der FIPA-Standard besteht hauptsächlich aus vier Bereichen:

- Ein Agent Management System ist auf der Agentenplattform integriert und ist für das Management des Agenten-Lifecycle verantwortlich.
- Directory Facilitator ist ein Verzeichnisdienst, bei dem ein Agent seine Dienste registrieren lassen bzw. nach Diensten anderer Agenten suchen kann.
- Ein Agent Communication Channel ist für die Kommunikation der Agenten zuständig, die sich auf unterschiedlichen Plattformen befinden.
- FIPA-ACL ist eine Sprache, mit der die Agenten Informationen austauschen können.

FIPA-ACL basiert auf KQML (Knowledge Querying and Manipulation Language) [8]. Die Sprache wurde für zum Ermöglichen der Wissenverteilung entwickelt. Dank Ontologien, die in den ACL-Nachrichten eingegeben werden, ist es möglich, den semantischen Gehalt der Nachricht abzuleiten.

8.3 Vergleich von MASIF und FIPA

Obwohl die beiden Standard etwa gleiche Grundprinzipien aufweisen, existieren doch ein Paar gravierende Unterschiede. So legt der FIPA-Standard den Wert auf die Künstliche Intelligenz der Agenten. Die Agenten sollen in der Lage sein, ihr Wissen mit anderen Agenten zu teilen und damit gemeinsam Probleme zu lösen. Der MASIF-Standard dagegen legt den Wert auf die Migration der Agenten.

9 Die Technologie der mobilen Agenten für verteilte Anwendungen in lokalen Netzwerken

Es sollte jetzt die Mächtigkeit der Technologie der mobilen Agenten klar sein. Leider ist es oft so, dass gerade die Mächtigkeit im Wege steht. Die Komplexität der Anwendung kann schnell die Grenze der Machbarkeit in Bezug auf Zeit oder Geld überschreiten. Ganz anders sieht es aus, wenn bestimmte Freiheiten weggenommen werden. Z.B. wenn die Agenten nicht selbst die Migrationsentscheidung treffen sollen, oder wenn die Topologie des Netzwerkes genau bekannt ist.

Hier soll eine Idee vorgestellt werden, wie die Technologie der mobilen Agenten für eine verteilte Anwendung im lokalen Netzwerk angewendet werden kann.

Die Ausgangsbedingungen sind wie folgt: Es ist ein lokales Netzwerk gegeben. Die Topologie dieses Netzwerkes ist genau festgelegt und bekannt. Auch alle Ressourcen der Rechner im Netzwerk sind bekannt. Auf jedem Rechner soll eine

spezielle Umgebung installiert werden, die gewisse Ähnlichkeiten zu einer Agentenplattform aufweist. Da aber diese Umgebung im lokalen Netzwerk eingesetzt wird, sind viele Eigenschaften der Agentenplattformen für diese Umgebung nicht relevant und können unberücksichtigt weggelassen werden. So kann z.B. die Sicherheit fast komplett weggelassen werden. Es soll lediglich sichergestellt werden, dass die Programme, die in der Umgebung ausgeführt werden, nicht verloren gehen.

Der Wunsch ist, ein Programm zu schreiben, welches in der Lage ist, von einem Rechner zu einem anderen zu migrieren um dessen Ressourcen lokal nutzen zu können. Die Entscheidung, zu welchem Rechner das Programm gehen soll, wird nicht vom Programm, sondern vom Programmierer getroffen. Für den Entwickler soll das gesamte Netzwerk nichts anderes als ein großer Rechner sein. Die Umgebung soll dabei alle dafür notwendigen Schritte übernehmen. Der Pseudocode soll ungefähr wie folgt aussehen:

- macheEtwas1();
- geheZuRechner(2);
- macheEtwas2();
- geheZuRechner(1);
- macheEtwas3();

Damit sollen die Vorteile der mobilen Agenten (asynchroner Aufruf, lokale Nutzung der Ressourcen) genutzt werden und gleichzeitig die Art der Programmierung möglichst unverändert bleiben.

Unter Ressourcen werden nicht nur andere Programme wie eine Datenbank, eine Hardware wie z.B. eine WebCam oder ISDN-Karte verstanden, sondern auch die Menschen, die auf dem Rechner arbeiten. So wird es z.B. möglich ein Programm zu schreiben, welches in einer Firma von einem Rechner zu anderem geht und bei jedem den User (Angestellten) nach den Urlaubsplänen fragt. Das Programm kennt dabei folgende Randbedingungen: In der Firma sollen mindestens 10 Mitarbeiter anwesend sein, davon müssen mindestens 3 aus der Abteilung X, 2 aus der Abteilung Y usw. sein. Wenn Person A Urlaub hat, dann kann Person B nicht fehlen. Der Leiter hat die Aufgabe, die Regeln zu setzen. Nachdem das Programm konfiguriert und gestartet wird, kann der Leiter weitere Aufgaben erledigen, da sein Rechner während der Programmausführung unbelastet bleibt. Erst dann, wenn das Programm die Arbeit vollständig erledigt hat, was durchaus mehrere Tage dauern kann wenn z.B. jemand nicht erreicht werden konnte, kommt es mit dem Ergebnis zurück und der Leiter bekommt einen fertigen Urlaubsplan.

Literatur

[1] http://www11.informatik.tu-muenchen.de/lehre/lectures/ss1994/va/chap_1/verte.html

[2] http://www.zdnet.de/news/tkomm/0,39023151,39134180,00.htm

[3] http://agent.cs.dartmouth.edu/

[4] http://www.trl.ibm.com/aglets/

[5] http://www.recursionsw.com/

[6] http://www.omg.org/

[7] http://www.fipa.org/

[8] http://www.csee.umbc.edu/kqml/papers/

[9] http://www11.informatik.tu-muenchen.de/lehre/lectures/ss1994/va/chap_1/verte.html

[10] http://de.wikipedia.org/wiki/Verteilte_Systeme

[11] NET.ObjectDAYS2000 Tutorial 2 :'Von verteilten Objekten zu mobilen Agenten'

[12] http://www.cs.berkeley.edu/projects/sprite/sprite.html

[13] http://lampwww.epfl.ch/~zenger/papers/partyd.pdf

[14] http://www.omg.org/docs/orbos/98-03-09.pdf

[15] http://www.fipa.org/specs/fipa00001

[16] http://www.informatik.hu-berlin.de/Institut/struktur/systemanalyse/diplom/dorn04.pdf

[17] J. Bradshaw (Ed.); Software Agents; The MIT Press, 1997

[18] W. Brenner, R. Zarnekow, H. Wittig; Intelligente Softwareagenten - Grundlagen und Anwendungen; Springer-Verlag

[19] http://www.cordis.lu/infowin/acts/analysys/products/thematic/agents/ch4/ch4.htm