Hauptseminar
„Mobile Agenten"

WS 96/97

Thema :

Einsatz und Verwendung mobiler Agenten

am Beispiel des CyberAgent - Systems

Autor: Andreas Roller

Universität Stuttgart
Institut für Informatik
Lehrstuhl Grundlagen der Informatik
- Verteilte Systeme -
Prof. Dr. rer. nat. Kurt Rothermel

Betreuer: Dipl. Inf. Fritz Hohl

Zusammenfassung:

Diese Ausarbeitung soll einen Einblick in die Funktionsweise und Mechanismen des Softwaresystems CyberAgent geben. Dieses von der Firma FTPSoftware entwickelte Programmpaket umfaßt ein Management-Tool zur Verwaltung und eine Programmbibliothek mit verschiedenen Hilfsmitteln zur Erstellung von Agenten zu den unterschiedlichsten Problemen. Anhand eines Beispielagenten soll der Nutzen und Gebrauch von mobilen Agenten dargestellt werden.

Inhaltsverzeichnis

1. Einführung

Programme, die auf einem PC oder einer Workstation ablaufen, sind allgemein bekannt; jeder benutzt Textverarbeitungen oder Tabellenkalkulationen. Im Gegensatz dazu kamen in letzter Zeit immer mehr Anwendungen für Netzwerke auf, viele benutzen regelmäßig Browser um aus dem Internet Daten zu bekommen. Genau hier liegt der Unterschied zu Agentensystemen. Mit einem Agenten kann man nicht nur Daten von anderen Knoten im Netz erhalten, man kann dort auch viele andere nützliche Dinge tun.

Ein Agent bietet die Möglichkeit, selbständig Programme zu starten, Daten aus anderen Anwendungen zu sammeln, aufgrund dieser Daten Entscheidungen zu treffen und entsprechend weiter zu handeln. Das alles ist, falls die entsprechende Software installiert ist, auf beliebigen Plattformen im Netz möglich, egal ob es sich um einen UNIX, NT oder Win95 - Rechner handelt. Daraus resultieren natürlich ganz neue Arten von Anwendungen, die zum Teil noch gar nicht abzuschätzen sind.

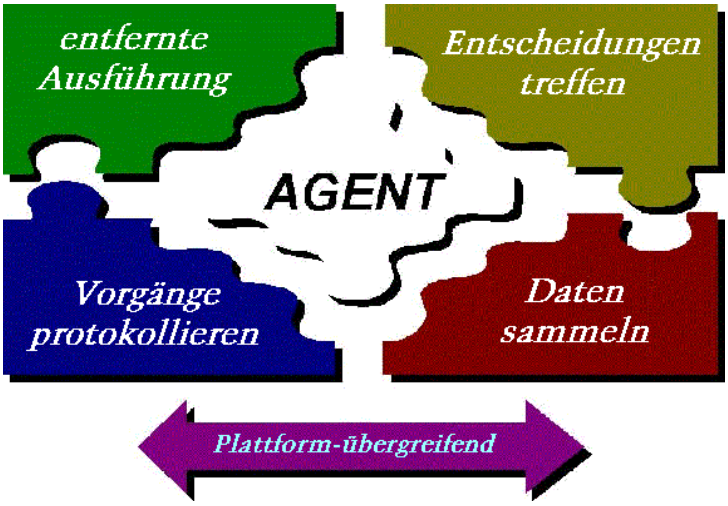

Abbildung 1 : Einsatzmöglichkeiten

Da der Einsatz von Agenten erst in den letzten Jahren mit den immer schneller wachsenden Netzwerken interessant wurde, datieren praktisch alle derzeit verfügbaren Agentensyteme aus der jüngsten Vergangenheit bzw. befinden sich noch im Entwicklungsstadium.

Mit dem CyberAgent - System hat die Firma FTPSoftware, bisher hauptsächlich durch diverse Netzapplikationen im TCP/IP-Bereich in Erscheinung getreten, im Jahre 1995 ein solches, in JAVA implementiertes Programmpaket auf den Markt gebracht. Es umfaßt ein Agentenmanagement - Tool, ein komplettes JAVA-Entwicklungssystem (JDK 1.0.2) und diverse JAVA - Programmbibliotheken mit auf Agenten bezogenen Objektklassen und -Methoden. Unterstützt werden in der derzeitigen Version Windows- und UNIX-Plattformen, wobei sich der Betrieb unter UNIX als einiges schwieriger erweist. Als Verbindungsnetzwerk muß TCP/IP vorhanden sein, das Programm identifiziert andere Lokationen anhand deren DNS-Namen. Installierbar sind drei Versionen, entweder das komplette Softwareentwicklungssystem (SDK), ein Managersystem oder ein Listenersystem. Im folgenden sollen die Möglichkeiten und Funktionen des Systems genauer beleuchtet werden.

2. Aufbau und Funktionen des CyberAgent - Managers

2.1 Komponenten

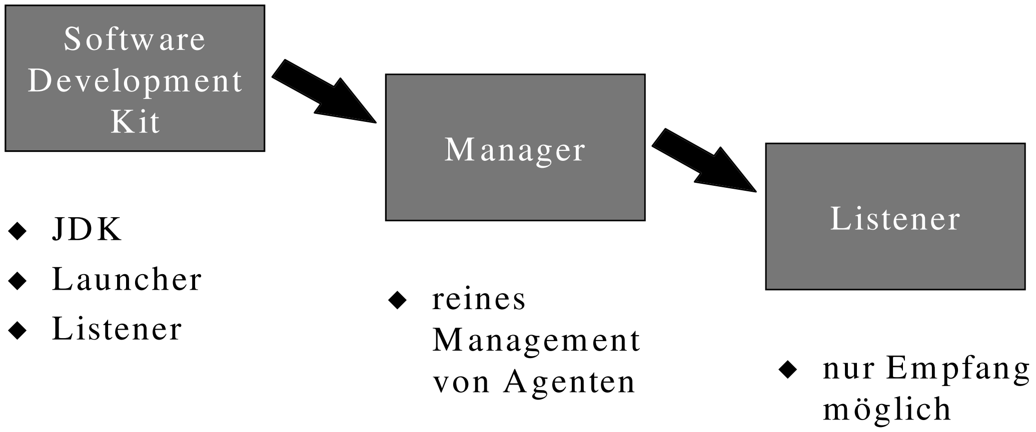

Abbildung 2: Installierbare Komponenten

Die in Abb. 2 gezeigten installierbaren Programmversionen unterscheiden sich in ihrer Leistungsfähigkeit. Im Vergleich zum SDK, welches zusätzlich zur normalen Funktionalität auch eine komplette Programmbibliothek zur Entwicklung eigener Agenten enthält, fällt beim Manager diese Komponente weg. Der Listener bietet keine weiteren Möglichkeiten als den Empfang von Agenten.

Um einen Agenten auf einem Rechner starten oder empfangen zu können muß immer eine Version des Programms aktiv sein, die reine Installation reicht hier nicht aus.

Durch eine Passwortabfrage beim Programmaufruf soll gewährleistet werden, daß nur autorisierte Personen Agenten starten/empfangen können. Da CyberAgent keine Grenzen für Agenten definiert, ein Agent also alles tun darf was ein beliebiges JAVA-Programm kann, ist ein gewisses Maß an Sicherheit auch mehr als notwendig.

2.2 Bedienung des Managers

Die gesamte Steuerung der Agenten wird über den sogenannten CyberAgent-Manager vorgenommen, eine grafische Benutzeroberfläche, welche direkt nach dem Logon erscheint. Abb. 3 zeigt einen Screenshot dieser Oberfläche mit ihren diversen Menüs und Buttons, auf die nun näher eingegangen werden soll.

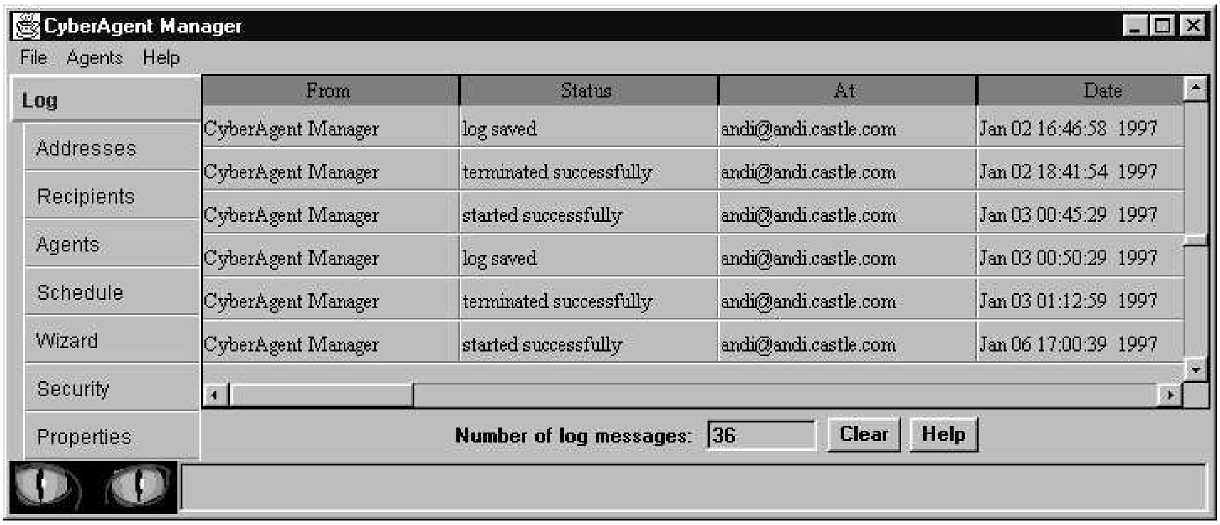

Abbildung 3 : Aufbau des Managers

2.2.1 Pulldown-Menü

Da praktisch die gesamte Bedienung über die acht verschiedenen Tabs erfolgt, wird nur ein Menü mit den wichtigsten Funktionen zur Verfügung gestellt.

- File : Vom *File*menü aus hat man Zugriff auf administrative Funktionen wie der Einstellung einer anderen Konfiguration, oder dem Import von Konfigurationen anderer Manager. Auch ein Exit-Button wurde nicht vergessen.
- Agents : Das *Agents*menü stellt im Grunde die gleichen Möglichkeiten wie der Agent-Tab bereit. Von hier können Agenten gestartet, neue registriert oder das Result-File eines zurückkehrenden Agenten betrachtet werden.
- Help : Im *Help*menü steht der Inhalt der Online-Hilfe zur Verfügung. Beim erstmaligen Aufruf wird man zur Eingabe des Verzeichnisses eines HTML-kompatiblen Browsers aufgefordert, der als Helpviewer benutzt wird. Außerdem kann ein direkter Kontakt zur Homepage der Firma FTPSoftware hergestellt werden.

2.2.2 Log-Tab

Mit Hilfe des *Log*-Tabs läßt sich ein Protokoll aller wichtigen Vorgänge betrachten. Hier finden sich Aktionen wie zum Beispiel der Start bzw. die Terminierung von Agenten, der aktuelle Status des Managers und etwaige Probleme. Angegeben wird immer das Datum der Aktion, die beteiligten Knoten (d.h. Sender/Empfänger) und der jeweilige Status.

2.2.3 Addresses-Tab

Von hier aus können DNS-Namen vergeben oder geändert werden. Die Adressen, anhand derer der Manager die Ziele für Agenten identifiziert, enthalten den Login Namen, Hostnamen und eine optionale Beschreibung. Ein Beispiel für eine voll qualifizierte Adresse wäre `'User@cembalo.informatik.uni-stuttgart.de` *`Testadresse`*`'`.

Außerdem bietet das Menü die Möglichkeit, von anderen Systemen kommende oder als Textfile vorhandene Adressenlisten zu importieren.

2.2.4 Recipients-Tab

Im *Recipients*-Menü werden Funktionen zur Verfügung gestellt, welche die oben definierten Adressen für Agenten sichtbar und somit anlaufbar machen. CyberAgent läßt als Ziele für die Agenten drei verschiedene Möglichkeiten zu (siehe Abb. 4).

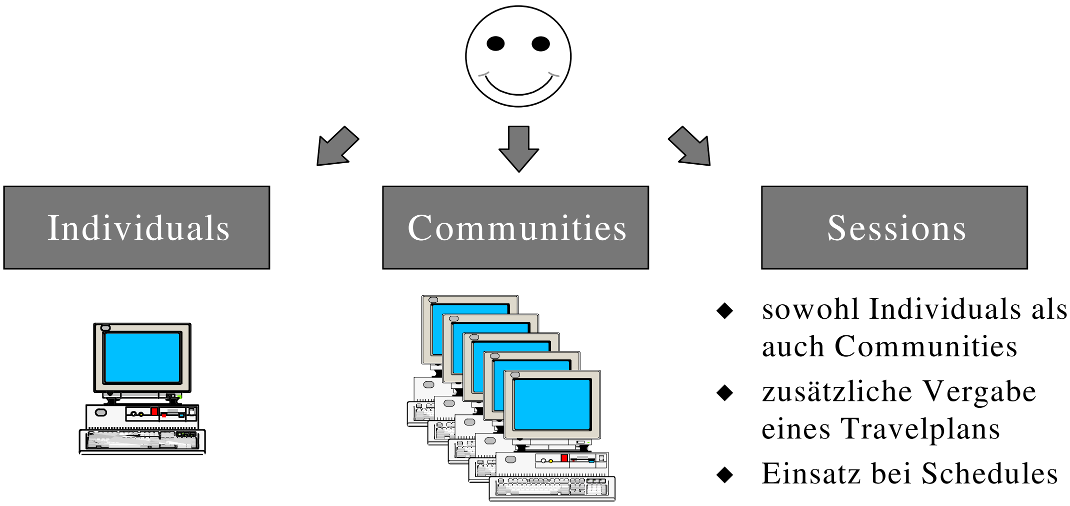

Abbildung 4: mögliche Ziele

Die Vergabe eines *Travelplans* bei Sessions erlaubt den Agenten, die festgelegten Ziele entweder gleichzeitig oder hintereinander anzulaufen. Sessions bieten also eine komplette Ablaufumgebung für Agenten, welche auch über Schedules (s. unten) angesprochen werden können.

Bei allen Empfängern muß zusätzlich ein Sicherheitstyp angegeben werden, mit dessen Hilfe bestimmt wird, ob die Agenten unverschlüsselt oder kodiert gesendet werden. Mögliche Verschlüsselungsarten sind:

- kein Schutz
- Paßwortschutz: durch Eingabe eines Paßwortes wird ein Schlüsselfile generiert, welches der entsprechenden Umgebung zugeordnet wird. Um einen Agenten in eine geschützte Umgebung zu starten, muß sowohl der Empfänger als auch der Sender mit diesem Schlüssel ausgestattet sein (näheres unter 2.2.8).
- DES: nachrüstbar, war in der aktuellen Version nicht implementiert
- RC2: nachrüstbar, war in der aktuellen Version nicht implementiert

2.2.5 Agents-Tab

Die wichtigsten Funktionen stellt der *Agents*-Tab bereit. Hier können alle derzeit registrierten Agenten gestartet oder deren Rückgabefiles betrachtet werden. Auch eine Neuregistrierung von als JAVA-Code abgelegten Agenten ist von hier aus möglich. Dabei muß ein

Agentenname und das Verzeichnis, welches die entsprechende Klassenbibliothek enthält, angegeben werden. Zusätzlich können dem JAVA-Script noch beliebige Parameter mitgegeben werden.

2.2.6 Schedule-Tab

Manchmal besteht die Notwendigkeit, Agenten zu definierten Zeitpunkten immer wieder laufen zu lassen. Vom *Schedule*-Tab aus kann ein Zeitablaufplan mit konkreten Startterminen erstellt werden. Dabei muß der Rhythmus (einmal, stündlich, täglich, wöchentlich), Start- und Stopzeit, der Agentenname und die Zielsession (d.h. die Ablaufumgebung, in welche der Agent gestartet werden soll) angegeben werden.

2.2.7 Wizard-Tab

Hier verbirgt sich ein menügesteuertes System zur Erstellung einfacher Launch-Agenten, mit denen drei verschiedene Aufgabenfelder abgedeckt werden können:

- Start beliebiger ausführbarer Programme (*Executables*), zusätzlich besteht die Möglichkeit zur Spezifikation von Parametern
- Entwurf von Agenten zur Ausgabe von Textnachrichten
- Erstellung von Log-Agenten, welche Daten in spezielle Files schreiben

2.2.8 Security-Tab

Über dieses Menü kann eine Schutzumgebung für die Ausführung von Agenten aufgesetzt werden. Jede Umgebung besitzt bestimmte Schlüssel, falls ein Empfänger definiert wird, kann diesem ebenfalls ein Schlüssel zugewiesen werden. Falls nun ein Agent in eine geschützte Umgebung gestartet wird, läuft er nur, wenn sowohl Ausgangs- als auch Zielsystem denselben Schlüssel besitzen. Soll ein Zielsystem für den Empfang eines gesicherten Agenten ausgestattet werden, so muß der entsprechende Schlüssel auf dem System per Disk-Transfer installiert werden. Vom Security-Tab aus besteht die Möglichkeit, auf dem Host-System vorhandene Schlüssel zu betrachten, oder diese zu importieren.

2.2.9 Properties-Tab

Hier können grundlegende Systemeinstellungen vorgenommen werden, wie zum Beispiel die Anzahl an gespeicherten Logmessages, oder das Verzeichnis des Hilfebrowsers. Außerdem kann eingestellt werden, ob ungeschützte Agenten Zugriff auf das System erhalten können. Wird diese Funktion ausgeschaltet, so wird ein Agent ohne Sicherung beim Eintreffen abgelehnt.

3. Benutzung einfacher Methoden anhand eines Beispielagenten

Ohne die Möglichkeit, selbst Agenten entwerfen zu können wären die bisher vorgestellten Funktionen ziemlich uninteressant. FTPSoftware hat als Implementierungssprache, sowohl für den Manager als auch für die Agenten JAVA gewählt, eine Programmiersprache die vor allem durch ihre Plattformunabhängigkeit in letzter Zeit immer mehr Anhänger gefunden hat. Der CyberAgent beinhaltet eine Reihe von Methoden, die sich auf die implementierte Agent-Klasse beziehen.

Zur Beschreibung des Entwurfs einfacher Agenten soll nun ein Beispiel vorgestellt werden, an dem die grundlegendsten Methoden aufgezeigt werden können.

3.1 Funktionsweise der Beispielagenten

Enwickelt wurden zwei Agenten, ein Dateiserver- und ein Dateisuchagent. Der Dateiserveragent wird auf einem beliebigen Knoten gestartet und trägt Informationen über bestimmte Textdateien, welche auf dem System verfügbar sind. Diese werden ständig in einem Fenster angezeigt. Der Dateisuchagent wird auf einen Knoten gestartet, auf dem ein Serveragent läuft und fordert zur Eingabe eines voll qualifizierten Dateinamens auf. Er hat die Aufgabe, eine Verbindung mit dem Server einzugehen, und festzustellen, ob die eingegebene Datei auf dem System vorhanden ist. Dazu findet ein Nachrichtenaustausch zwischen Server und Sucher statt, wobei der Server auf die Anfrage entweder mit ‘vorhanden’ oder ‘nicht vorhanden’ antwortet. Bei positiver Antwort greift der Sucher auf die Datei zu und zeigt ihren Inhalt an.

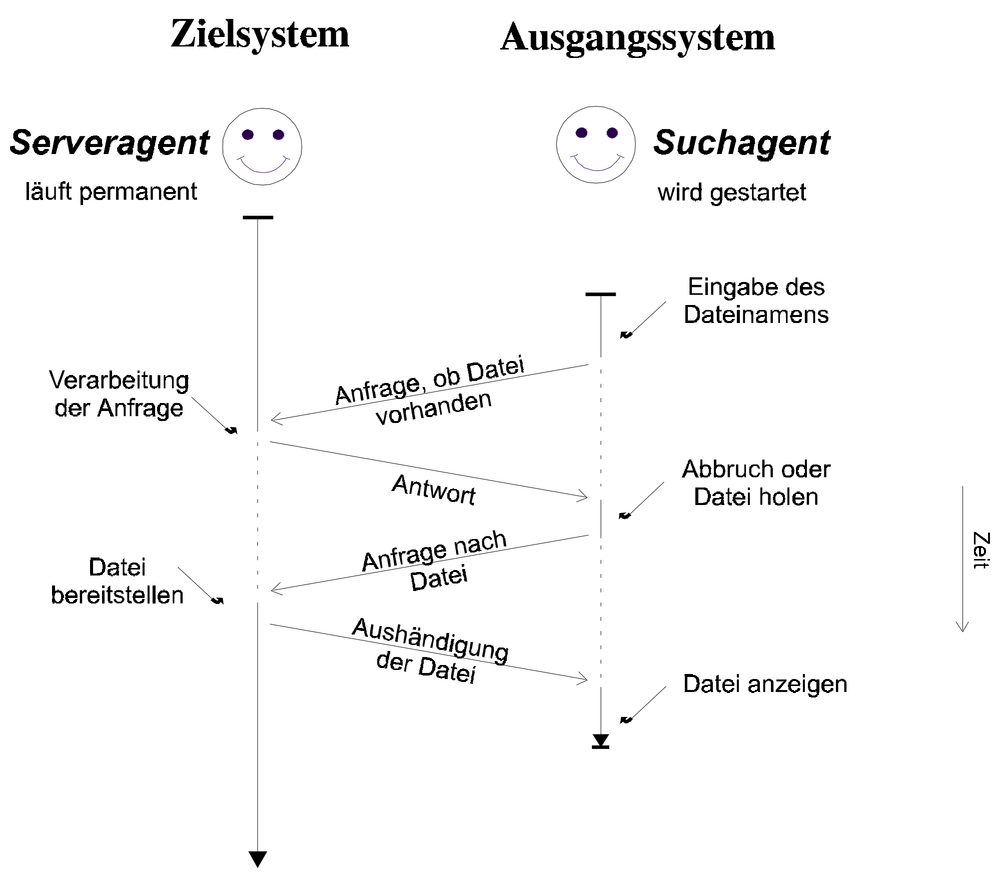

Abbildung 5: Funktionsweise des Beispielagenten

Beide Agenten greifen auf eine Reihe von IO-Klassen zur grafischen Aufbereitung zurück, auf welche nicht näher eingegangen wird. Nachfolgend nun eine kurze Beschreibung der verwendeten agentenspezifischen Methoden (Code der Agenten: siehe Anhang A-C).

3.2 benutzte Methoden

Für dieses Beispiel gibt es zwei wichtige Klassen, die importiert werden müssen. Das ist einmal die Klasse **`FTPSoftware.ia.eng.Agent`**, in der die gesamte Agentenklasse liegt, zum anderen **`FTPSoftware.ia.eng.AgentMessage`**, wo sich alle zum implementieren des Nachrichtenaustausches benötigten Konstrukte befinden.

Anschließend muß mittels **`public class`** ***`Agentname`*** **`extends Agent`** eine Subklasse von **`Agent`** erzeugt werden.

Die wichtigste Methode ist **`doFTPSAgentTask().`** Dies entspricht der main-Methode im Standard-Java, d.h. dem Aufruf des Hauptprogramms. Innerhalb dieser Methode kann beliebiger JAVA-Code stehen, der dann auf jedem Knoten ausgeführt wird.

Innerhalb von **`prepareForLaunch()`** kann Code stehen, der VOR dem Start eines Agenten, d.h. vor der tatsächlichen Migration ausgeführt wird, also etwa Initialisierungen oder wie im Beispiel die Eingabe eines Parameters.

Um eine Nachricht an einen anderen Agenten senden zu können, muß das Objekt **AgentMessage** erschaffen werden. Jede **AgentMessage** enthält die folgenden vier Konstrukte:

- Sender: Agentenname des Absenders
- Receiver: Agentenname des Empfängers
- Subject: String mit einer kurzen Beschreibung, welche Art von Nachricht übermittelt wird
- Content: kann jedes beliebige Konstrukt sein, hier wird der Inhalt der Nachricht definiert

Über **BOOLEAN sendMessage(STRING agentName, AGENTMESSAGE msg)** wird die so erzeugt Nachricht an den gewünschten Knoten geschickt. Falls der Knoten nicht erreichbar ist , wird **FALSE** zurückgeliefert, ansonsten **TRUE**. Der Empfänger kann mit Hilfe von **BOOLEAN doIHaveMessages()** überprüfen, ob Nachrichten in seinem Messagestapel vorhanden sind. Falls ja, kann die oberste Nachricht mittels **msg = getMessage()** geholt werden. Auf diese Nachricht können nun diverse Funktionen angewendet werden, um die gesendeten Daten zu extrahieren:

- **STRING getSender()** liefert den Absender der Nachricht
- **STRING getReceiver()** liefert den Empfänger (d.h. den aktuellen Knoten)
- **STRING getSubject()** liefert die Nachrichtenbeschreibung
- **OBJECT getObject()** liefert den Inhalt der Nachricht

Die so erhaltenen Konstrukte können nun verarbeitet und mittels den entsprechenden set-Befehlen wieder in eine neue Nachricht verpackt werden (d.h. **setSender, setReceiver**,...)

Nach getaner Arbeit muß der Agent noch dazu veranlaßt werden, zum nächsten Knoten zu springen, bzw. zurückzukehren. Dies erfolgt über die Methode **determineNextStop(),** die den nächsten Knoten bestimmt, wobei dies auch der Ausgangsknoten sein kann, falls kein weiteres Ziel mehr verfügbar ist. Mittels **SetNextStop(determineNextStop())** wird der Knoten aktiviert, und letztendlich über **continueJourney()** auch angesprungen. Es gibt jedoch keine Möglichkeit, zu einem anderen Ziel als dem vom Manager aus definierten zu wechseln, Sprünge durch Agenten selber ausgelöst sind also unmöglich.

Um den auf dem aktuellen Knoten noch laufenden Prozeß zu beenden sollte ein **terminateTask()** nicht vergessen werden.

3.3 essentielle Konstrukte

Aus den oben genannten Bestandteilen sind im Prinzip nur wenige für einen funktionierenden Agenten wichtig. Auf diese soll im folgenden noch einmal verwiesen werden.

- importieren der Agent-Klasse
- definieren einer Klasse als Subklasse von Agent
- **main()** - Methode ersetzen durch **doFTPSAgent()**

diese drei Bestandteile werden vom CyberAgent-Manager bei der Neuregistration überprüft, falls sie vorhanden sind wird die entsprechende Klasse als Agent identifiziert. Innerhalb der main-Methode kann beliebiger JAVA-Code stehen, die Zieladressen werden nicht innerhalb des Codes generiert, sondern vom Manager aus.

Offensichtlich ist die Erzeugung eines lauffähigen Agenten kein größeres Problem. Innerhalb von **`doFTPSAgent()`** sollten zwar noch die in 3.2 genannten Methoden zur Rückkehr bzw. zum Sprung auf den nächsten Knoten enthalten sein, aber ausführbar ist der Agent alleine durch die obigen drei Komponenten.

4. Weitere wichtige Methoden

Im folgenden werden einige weitere für den Agentenentwurf wichtige Methoden kurz vorgestellt. Die Aufzählung deckt nur einen Teil der implementierten Funktionen ab, einige davon sind jedoch nur für äußerst seltene Anforderungen interessant, so daß auf deren Einsatz meistens verzichtet werden kann.

- **`doAfterReturnHome()`**
  Methode, die automatisch nach der Rückkehr des Agenten aufgerufen wird.
  Hier kann eine Aufbereitung der eingeholten Daten stattfinden.

- **`goHomeNow()`**
  veranlaßt den Agenten zur sofortigen Rückkehr.

- **`setReport(STRING agentReport)`**
  setzt den Report-Buffer auf den mitgegebenen String. Der Report-Buffer wird mittels appendReport dem aktuellen Report hinzugefügt. Dieser bleibt bei Migration des Agenten erhalten, wodurch Daten von verschiedenen Knoten gesammelt werden können. Bei Terminierung des Agenten wird der Report in ein beim Start definiertes File geschrieben und steht dort zur Auswertung bereit.

- **`appendReport()`**
  fügt den Report-Buffer dem aktuellen Report dazu.

- **`STRING getReport()`**
  liest den Inhalt des Report-Buffers ein.

- **`sendReportHome()`**
  schickt den Inhalt des Report-Buffers zurück und speichert ihn im Report-File.

- **`setData(STRING agentData)`**
  funktioniert ähnlich wie setReport, Daten sind aber nach Terminierung des Agenten verloren.

- **`STRING getData()`**
  funktioniert ähnlich wie getReport, Daten sind jedoch nach Terminierung des Agenten verloren.

- **`BOOLEAN processCommand(STRING sysCommand, BOOLEAN captureOutput, BOOLEAN waitsforCompletion)`**
  führt den Systembefehl in ‘sysCommand’ aus. Falls captureOutput gesetzt ist, wird jede Ausgabe im Report aufgefangen, falls waitsForCompletion gesetzt ist, wird das

Ende des Systembefehls vor einem Fortfahren abgewartet. Liefert TRUE zurück, falls der Befehl erfolgreich ausgeführt wurde.

Außerdem existiert eine ganze Reihe an Check-Methoden, die bestimmte Eigenschaften überprüfen und einen Wert zurückliefern. Hier seien einige davon genannt, ohne konkreter darauf einzugehen, da sie größtenteils für sich selbst sprechen.

`INT getNumberOfFailedStops()`	`INT getNumberOfUnreachedStops()`
`BOOLEAN returnedHome()`	`BOOLEAN isLastStop()`
`BOOLEAN isHome()`	
`STRING getAgentName()`	`STRING getCurrentStop()`
`STRING getSender()`	`STRING getFailedStops()`
`STRING getWorkingDirectory()`	`STRING getPreviousStop()`
`STRING getReportFile()`	`STRING getTravelPlan()`
`STRING getLaunchDate()`	

Tabelle 1: implementierte Check-Methoden

5. Leistungsfähigkeit des Systems

Im folgenden soll eine Bewertung des Systems anhand verschiedener Kriterien einen Eindruck über die Leistungsfähigkeit der CyberAgent-Software vermitteln.

5.1 Kommunikation

Das Versenden und Empfangen von Agenten wurde über TCP/IP realisiert, andere Protokolle werden nicht unterstützt.

Agenten können untereinander kommunizieren, jedoch nur lokal, d.h. wenn sie sich auf dem selben System befinden. Zusätzlich müssen sie vom gleichen Manager aus gestartet worden sein, was eine große Einschränkung darstellt. Zum Handling der Verbindung existiert die Klasse AgentMessage, welche die nötigen Methoden und Konstrukte bereitstellt.

Weiterhin haben Agenten die Möglichkeit zur uneingeschränkten Kommunikation mit dem System, da alle System-Befehle von JAVA direkt absetzbar sind. Auch kann man beliebige DOS- bzw. UNIX-Kommandos in die Agenten einbinden. Dadurch sind überaus mächtige Agenten realisierbar, die praktisch unbegrenzte Änderungen am System vornehmen können. Der Benutzer sollte sich also genau im klaren darüber sein, welche Funktionalität ein gestarteter Agent bietet.

5.2 Namen und Adressen

Ausgangs- und Zielsysteme werden anhand ihres DNS-Namen identifiziert. Diese werden über vom Manager aus definierbare Gruppen zu größeren Einheiten zusammengefaßt.

Bei der Kommunikation zwischen Agenten wird als Ziel der Name des Agenten benutzt. Da nur lokale Kommunikation möglich ist, müssen keine Systemadressen angegeben werden.

Vom Agenten können keine direkten Zieladressen angegeben werden (d.h. es gibt keine Methoden zur Direktmigration), jeder anlaufbare Knoten wird vom Manager aus definiert.

5.3 Migration

Agenten migrieren komplett zum Zielsystem und können vor normaler Terminierung durch bestimmte Methoden zum vorzeitigen Abbruch gezwungen werden. Jeder Agent startet bei Ankunft auf einem neuen System von vorne, d.h. seine main-Methode wird aufgerufen.

Ein Report mit eingesammelten Daten kann jederzeit zum Startsystem zurückgeschickt werden. Bei der Migration von System zu System ist es möglich, zwei Speicherbereiche mitzuführen, denen auf jedem Knoten Daten angehängt, bzw. die verarbeitet werden können. Der Report-Buffer wird bei der Rückkehr automatisch in eine beim Start definierte Datei geschrieben, der Data-Buffer ist bei Terminierung des Agenten verloren.

Die systemübergreifenden Verbindungszeiten bewegen sich im Sekundenbereich, je nach Auslastung kann der erfolgreiche Start auch fast eine Minute dauern.

5.4 Sicherheit

Da Agenten im Unterschied zur normalen Client/Server-Architektur nicht nur Anfragen stellen, sondern sich aktiv auf das Zielsystem begeben, und dort eventuell Änderungen vornehmen ist es natürlich unerläßlich, eine Form von Schutz vor unberechtigtem Gebrauch zu implementieren.

Durch die Mächtigkeit der mit CyberAgent erzeugbaren Agenten ist ein Sicherheitssystem mehr als notwendig.

Es existieren zwei verschiedene Arten von Sicherheitsmechanismen. Zum einen ist der Aufruf des Managers und somit der Start und Empfang von Agenten über eine Paßworteingabe gesichert, die beim ersten Aufruf des Programms vergeben wird. Außerdem können bestimmten Zielgruppen Paßworte zugewiesen werden, über die ein Schlüssel erzeugt wird, der im Zielsystem eingespielt werden muß. Falls ein Agent auf eine geschützte Zielgruppe gestartet wird, wird überprüft, ob der benötigte Schlüssel vorhanden ist. Dabei muß das Paßwort nicht mehr eingegeben werden, es dient nur zur Schlüsselerzeugung. Durch diesen Mechanismus wird Datenintegrität und einfache Authentifikation gewährleistet.

Zusätzlich zum Paßwortschutz bietet FTPSoftware auch ein zusätzlich zu erwerbendes Sicherheitserweiterungspaket an, welches die Standards DES und RC2 unterstützt.

Für einen Einsatz im großen Rahmen ist das Sicherheitssystem nicht effektiv genug, es gibt zu viele Möglichkeiten zum Mißbrauch. Daher ist CyberAgent auf inhouse-Anwendungen beschränkt, bei denen jeder Benutzer über die zur Verfügung stehenden Agenten informiert ist und ihre Auswirkungen kennt.

5.5 Fehlerbehandlung / Debugging

Zur Beobachtung von laufenden Agenten stehen keine größeren Hilfsmittel zur Verfügung, es gibt jedoch einige Methoden, um bestimmte Trace-Kommandos abzusetzen (vom Agent aus aufzurufen), welche von einem aus dem Manager aufrufbaren Debugger verfolgt werden können.

Unerreichbare Knoten können ebenfalls über Methoden überprüft werden. Für Systemabstürze steht kein Recovery zur Verfügung, nichtterminierte Agenten sind verloren.

5.6 Benutzeroberfläche

Die grafische Benutzeroberfläche ist auch für Einsteiger einfach zu bedienen, die Buttons sind übersichtlich angebracht und stellen die zu erwartende Funktionalität zur Verfügung. Bedingt durch die Implementierung des Systems in JAVA sind gelegentliche Bildaufbaufehler festzustellen, auf langsameren Rechnern ist die Bedienung ziemlich träge. Mittels der *Wizard*-Funktion können schnell auch ohne Programmierkenntnisse einfache Agenten gestartet werden, was einen weiteren Anreiz für Einsteiger darstellt.

5.7 Dokumentation

Die komplette Dokumentation wurde über HTML-Files realisiert, d.h. ohne Webbrowser kann das Hilfesystem nicht benutzt werden. Zu jedem Menü stehen Hilfen zur Verfügung, außerdem wird ein Tutorial und eine Übersicht der mitgelieferten JAVA-Klassenbibliothek zur Verfügung gestellt.

6. Schwächen und Probleme

Im folgenden soll stichwortartig auf Probleme im Umgang mit der Software eingegangen und auf Verbesserungsmöglichkeiten hingewiesen werden.

- kein Support mehr von Seiten FTPSoftware, Angebote wurden aus den Web-Seiten entfernt ! Laut FTPSoftware wird die Technik in anderen Produkten noch Verwendung finden, außerdem können Agenten für bestimmte Problemstellungen bei FTPSoftware angefordert werden
- besseres Hilfesystem wäre wünschenswert

- JAVA-bedingt gibt es Probleme beim Bildschirmaufbau, Fenster werden teilweise nicht korrekt dargestellt
- die fehlende globale Kommunikation unter Agenten stellt eine große Einschränkung dar
- durch die Mächtigkeit der erzeugbaren Agenten wäre ein besserer Sicherheitsmechanismus angebracht
- es gibt keine Möglichkeit, laufende Agenten zu beobachten
- der Scheduler sollte frei definierbare Startzeiten zulassen
- hoher Anschaffungspreis: ca. 500 $ für SDK
- unter WINDOWS:
  - jeder Hilfeaufruf erzeugt einen neuen Browser
  - keine Möglichkeit, ohne Netzwerk Agenten auf das eigene System zu schicken
  - auf langsameren Systemen träge Benutzerführung
- unter UNIX:
  - Registrierungsdaemon nur als ROOT ausführbar
  - System läuft instabil, Daemon muß öfters hochgefahren werden
  - Installation relativ aufwendig
  - Absturzgefahr beim Aufruf bestimmter Menüs

7. Zusammenfassung

Der CyberAgent ist ein relativ einfach strukturiertes Agentensystem, das einen guten und leicht verständlichen Einblick in die Materie gewährt. Durch die gelungene Benutzeroberfläche wird auch Einsteigern ein schnelles Vorankommen ermöglicht. Da jedoch gravierende Mängel im Sicherheitsbereich, bei der Kommunikation unter Agenten und bei der Stabilität existieren, kommt das System nicht für den Einsatz in größeren Netzwerken in Frage. Ein weiterer Kritikpunkt ist die Marktpolitik von FTPSoftware, die das Produkt ohne größere Ankündigungen komplett vom Markt genommen, und auch alle Supportseiten im Internet gestrichen haben. Das kann für eine fertige Version, die sicher schon hundertfach verkauft wurde, keine Lösung sein. Trotzdem geht der Ansatz in die richtige Richtung, bei entsprechenden Upgrades hätten mit Sicherheit die vorhandenen Mängel behoben werden können, da das Grundgerüst durchaus vorhanden ist.

Anhang A: Dateiserveragent

```
import java.awt.*;
import java.io.*;
import FTPSoftware.ia.eng.Agent;
import FTPSoftware.ia.eng.AgentMessage;
import AgentIO;

public class ServerAgent extends Agent
{
  static String filename[] = new String[10];
```

```
  // hier werden die vorhandenen Dateinamen gehalten
  static String snd = "ServerAgent";
  static String rcv = "ViewAgent";
  static String sub = "confirm";
  // ... für die Nachricht an den Suchagenten
  static AgentMessage incoming;
  // ... für die Nachricht vom Suchagenten
  static boolean exists = false;
  // Flagvariable, ob die gesuchte Datei vorhanden ist
  static String ret;
  // Returnstring, liefert 'true' oder 'false'
  // an den Suchagenten zurück

  public void doFTPSAgentTask() throws Exception {
    int i;  // Schleifenzähler
    filename[0]="/home/rolleras/readme.txt";
    filename[1]="/home/rolleras/xstart";
    filename[2]="/home/rolleras/cyberagent/.cyber";
    filename[3]="/home/rolleras/cyberagent/.start";
    ShowFile input = new ShowFile();
    input.display(filename);
    // zeigt ein Fenster mit allen vorhandenen Dateinamen
    while(true) {
    // Agent soll immer laufen, bis er manuell beendet wird...
      startAcceptingMessage();
      // Nachrichten können empfangen werden
      while (doIHaveMessages() == false) {} // warte auf MSG
      stopAcceptingMessage(); // Nachrichtenstop
      exists = false;
      incoming = getMessage();
      // holt die Nachricht vom Suchagenten
      String searchname = (String) incoming.getContent();
      // extrahiert den zu suchenden Dateinamen
      for (i=0;i < filename.length;i++) {
        if (searchname.equals(filename[i]))
          exists=true;
      } // prüft, ob der gesuchte Dateinamen vorhanden ist
      if (exists)
        ret="true";
      else
        ret="false";
      AgentMessage outgoing = new AgentMessage(snd,rcv,sub,ret);
      // Nachricht an den Suchagenten wird erzeugt
      sendMessage(rcv,outgoing);
      // ... und versandt
    }
    setNextStop(determineNextStop());
    continueJourney();
    terminateTask(true);
    // das Ende wird nie erreicht, da Server immer laeuft...
  }
}
```

Anhang B: Dateisuchagent

```
import java.awt.*;
```

```
import java.io.*;
import FTPSoftware.ia.eng.Agent;
import FTPSoftware.ia.eng.AgentMessage;
import AgentIO;

public class ViewAgent extends Agent
{
  static String snd = "ViewAgent";
  static String rcv = "ServerAgent";
  static String sub = "ask";
  // ... für die Nachricht an den Serveragenten
  static AgentMessage incoming;
  // ... für die Nachricht vom Serveragenten
  static String name;
  // enthält den Namen der zu suchenden Datei

  public boolean prepareForLaunch()  {
  // wird vor dem Start des Agenten aufgerufen
    getName input = new getName();
    name = input.display();
    // öffnet Eingabefenster für den zu suchenden Namen
    return true;
    // d.h. Agent kann jetzt starten
  }
  public void doFTPSAgentTask() throws Exception {
    AgentMessage outgoing = new AgentMessage(snd,rcv,sub,name);
    sendMessage(rcv,outgoing);
    // eine Suchanfrage wird an den Serveragenten gestellt
    startAcceptingMessage();
    while (doIHaveMessages() == false) {} // warte auf Antwort
    incoming=getMessage(); // hole die Antwortnachricht
    String answer = (String) incoming.getContent();
    // extrahiere die Antwort -> vorhanden oder nicht ?
    if (answer.equals("true")) {
      avail getIt = new avail();
      getIt.getFile(name);
      // File ist da -> benachrichtige Benutzer
      try { Frame f = new view(name); } // zeige Text an
      catch (IOException e) { System.out.println(e); }
    }
    else {
      unavail getNot = new unavail();
      getNot.notify(name);
      // File ist nicht da -> benachrichtige Benutzer
    }
    setNextStop(determineNextStop());
    continueJourney();
    terminateTask(true);
  }
}
```

Anhang C: Input/Output - Klassen

```
import java.awt.*;
import java.io.*;
```

```
 class getName extends Frame {
 // stellt IO für die Eingabe eines Filenamens bereit
  boolean wait = true;
  Button b,b2;
  String name = "ViewTest.java";
  public String display() {
    setLayout(new BorderLayout(15,15));
    setBackground(Color.gray);
    setForeground(Color.black);
    resize(350,150);
    Panel p = new Panel();
    p.setLayout(new FlowLayout(FlowLayout.CENTER,15,15));
    b = new Button("Ok");
    b2 = new Button("Abbruch");
    p.add(b);
    p.add(b2);
    add("South",p);
    Panel p2 = new Panel();
    p2.add(new Label("Filenamen eingeben:"));
    TextField t = new TextField("ViewTest.java",40);
    p2.add(t);
    add("Center",p2);
    move(300,100);
    show();
    while (wait)  {}
    name = t.getText();
    return name;
  }
  public boolean handleEvent(Event evt) {
    if (evt.target == b2) {
      this.hide();
      this.dispose();
      System.exit(0);
      return true;
    }
    else if (evt.target == b) {
      this.hide();
      this.dispose();
      wait = false;
      return true;
    }
    else return false;
  } // handleEvent()
}
class unavail extends Frame {
// stellt IO für nicht vorhandenen Filenamen bereit
  Button b;
  public void notify(String fn) {
    setLayout(new BorderLayout(15,15));
    setBackground(Color.gray);
    setForeground(Color.black);
    resize(300,170);
    b = new Button("Abbruch");
    add("South",b);
    Panel p2 = new Panel();
    p2.add("Center",new Label("Die Datei"));
    p2.add("Center",new Label("' " + fn + " '"));
    p2.add("Center",new Label("ist nicht vorhanden !"));
```

```
    add("Center",p2);
    move(300,100);
    show();
    while (true) {}
  }
  public boolean handleEvent(Event evt) {
    if (evt.target == b) {
      this.hide();
      this.dispose();
      System.exit(0);
      return true;
    }
    else return false;
  } // handleEvent()
}
class avail extends Frame {
// stellt IO für vorhandenen Filenamen bereit
  boolean flag = true;
  Button b,b2;
  public void getFile(String fn) {
    setLayout(new BorderLayout(15,15));
    setBackground(Color.gray);
    setForeground(Color.black);
    resize(300,170);
    Panel p = new Panel();
    p.setLayout(new FlowLayout(FlowLayout.CENTER,15,15));
    b = new Button("Anzeigen");
    b2 = new Button("Abbruch");
    p.add(b);
    p.add(b2);
    add("South",p);
    Panel p2 = new Panel();
    p2.add("Center",new Label("Die Datei"));
    p2.add("Center",new Label("' " + fn + " '"));
    p2.add("Center",new Label("ist vorhanden"));
    add("Center",p2);
    move(300,100);
    show();
    while (flag)  {}
  }
  public boolean handleEvent(Event evt) {
    if (evt.target == b2) {
      this.hide();
      this.dispose();
      System.exit(0);
      return true;
    }
    else if (evt.target == b) {
      this.hide();
      this.dispose();
      flag = false;
      return true;
    }
    else return false;
  } // handleEvent()
}
class view extends Frame {
// stellt IO für die Anzeige einer Textdatei bereit
```

```
  Button close;
  public view(String filename) throws IOException {
    super("Name: " + filename);
    File f = new File(filename);
    int size = (int) f.length();
    int bytes_read = 0;
    FileInputStream in = new FileInputStream(f);
    byte[] data = new byte[size];
    while (bytes_read < size)
      bytes_read += in.read(data,bytes_read,size-bytes_read);
    TextArea textarea = new TextArea(new String(data,0),24,80);
    textarea.setFont(new Font("Helvetica",Font.PLAIN,12));
    textarea.setEditable(false);
    this.add("Center",textarea);
    close = new Button("Schliessen");
    this.add("South",close);
    this.pack();
    this.show();
  }
  public boolean handleEvent(Event evt) {
      if (evt.target == close) {
        this.hide();
        this.dispose();
        System.exit(0);
        return true;
      }
      else return false;
  } // handleEvent()
}
class ShowFile extends Frame {
// stellt IO für die Darstellung der vorhandenen Filenamen
// bereit
  Button b;
  public void display(String fn[]) {
    int i;
    setTitle("vorhandene Dateien");
    setBackground(Color.gray);
    setForeground(Color.black);
    resize(350,400);
    b = new Button("Abbruch");
    add("South",b);
    Panel p = new Panel();
    p.setLayout(new GridLayout(fn.length,1,1,10));
    for (i=0;i < fn.length;i++) {
      p.add(new Label(fn[i]));
    }
    add("Center",p);
    move(200,50);
    show();
  }
  public boolean handleEvent(Event evt) {
    if (evt.target == b) {
      this.hide();
      this.dispose();
      System.exit(0);
      return true;
    }
    else return false;
```

```
    } // handleEvent()
  }
```

Anhang D: Installation

1. Installation unter Windows NT / Windows 95

Unter Windows kann direkt das auf der CD befindliche Setup-Programm aufgerufen werden, welches einen durch die komplette Installation führt. Nach den Lizenzbedingungen wird man zur Auswahl einer der drei Installationsarten aufgefordert.

SDK : komplettes System inklusive Java Development Kit (JDK), ermöglicht die volle Funktionalität, d.h. entwickeln, starten und empfangen von Agenten

Manager : es wird nur ein Teil des JDK installiert, ermöglicht das Starten und Empfangen von Agenten

Listener : ermöglicht nur das Empfangen von Agenten

Nach dieser Auswahl wird man zur Angabe des Zielverzeichnisses aufgefordert, außerdem kann entweder das auf der CD vorhandene JDK eingerichtet, oder ein vorhandener Java-Interpreter benutzt werden. Schließlich wird die Installation mit dem Anlegen der Programmgruppe (enthält Icon für das Agentensystem und die Readme-Datei) beendet.

Beim ersten Aufruf des Systems wird man zu einer Paßworteingabe aufgefordert, die das Programm vor unberechtigter Benutzung schützen soll. Zusätzlich muss für die Onlinedokumentation noch das Verzeichnis des Web-Browsers angegeben werden, da sämtliche Hilfe-Dokumente als HTML-Files vorhanden sind.

2. Installation unter UNIX

Die Unix-Installation gestaltet sich etwas aufwendiger, da einige Schritte manuell auszuführen sind. Zuerst muß das selbstextrahierende `'cyber20s.xtr'` ins Zielverzeichnis kopiert und dort entpackt werden. In das dadurch angelegte `'properties'` - Verzeichnis wird nun eines der Konfigurationsfile kopiert (`'catmgr.10'` für das Komplettsystem, `'catmgr.5'` für den Manager, `'catmgr.1'` für den Listener) und in `'CyberAgentMgr.properties'` umbenannt. Danach muß die CLASSPATH-Variable für die Java-Klassenbibliothek richtig gesetzt werden, zusätzlich zum /LIB - Pfad des JDK wird auch ein Pfad auf /FTPSoftware.zip benötigt.

z.B: CLASSPATH=: /home/user/java/lib/classes.zip:/home/user/Cybergent/FTPSoftware.zip:.

Zum Abschluß muß vom Root noch der Daemon, der die Agentenregistration übernimmt, gestartet werden, und zwar mit

„`nohup java FTPSoftware.ia.eng.Daemon &`"

Dadurch können ankommende Agenten erkannt und ausgehende korrekt gesendet werden.