Hauptseminar : Mobile Agenten
Thema : MOLE

Ausarbeitung zum Vortrag
gehalten am 14.01.97
von Oliver Horster

1. Was sind Mobile Agenten

Mobile Agenten sind Objekte, die innerhalb eines Agentensystems eigenständig arbeiten. Sie sind in der Lage, aus eigenem Antrieb von einem Ort zum anderen zu migrieren. Man kann mobile Agenten auch als eine Art „Softwareroboter“ verstehen, die anstelle des Menschen eine Aufgabe lösen. Agenten sind in der Lage, unabhängig zu arbeiten. Es reicht sie zu starten und nach einer gewissen Zeit das Ergebnis ihrer Arbeit abzuholen. Daher scheinen sie besonders im Bereich der Mobilcomputer geeigneter zu sein als die bisherigen Ansätze, die immer eine dauerhafte Verbindung von Mobilcomputer und Provider verlangen.

2. Mole : Ein Überblick

2.1. Die Sprache Java / Java Bytecode

Java ist eine von SUN propagierte OO-Sprache. Sie wurde entwickelt, um einem WWW-Browser die Möglichkeit zu geben, auch Ausgaben von Programmen anzuzeigen und interaktive Eingaben des Benutzers zu verarbeiten. Aus diesem Grund und da die Java Programme normalerweise über das Netz geladen werden, fehlen ihr Möglichkeiten, auf Rechnern Systemaufrufe durchzuführen, wie z.B. direkte Zugriffe auf das Dateisystem.
Java besitzt unter anderem folgende Eigenschaften :

- keine Pointer
- einen automatischen Garbage Collector
- ein Modulkonzept
- ein Exceptionkonzept
- Synchronisationsmechanismen
- Variablen, Methoden und Klassen können privat oder öffentlich sein.

Als OO-Sprachelemente existieren in Java :

- Konstruktoren und Destruktoren
- Klassenvariablen und Klassenmethoden
- hierarchisches Klassenkonzept mit einfacher Vererbung

Der Java Bytecode ist die Sprache der virtuellen Maschine, die den Zwischencode interpretiert, der vom Javacompiler erzeugt wird. Java Bytecode erbt im Wesentlichen alle Eigenschaften von Java.

Als Agentensprache wird in Mole Java Bytecode verwendet, das System selbst ist in Java geschrieben.

2.2. Agenten

Mole kennt zwei grundlegende Arten von Agenten : Systemagenten und Useragenten

Systemagenten sind fest an eine Location gebunden. Sie dienen als Interfaceagenten, die Systemdienste den Useragenten zugänglich machen. So wird der Zugriff der Useragenten auf das System kontrolliert.

Useragenten untergliedern sich nochmals in mobile und nichtmobile Agenten. Mobile Agenten können Dienste anbieten und in Anspruch nehmen. Sie können von einer Location zu einer anderen wechseln. Nichtmobile Agenten sind an eine Location gebunden, sonst besitzen sie die selben Fähigkeiten wie mobile Agenten.

2.3. Location

Eine Location ist ein Ort, an dem sich Agenten aufhalten können. Dort können sie dann lokal miteinander kommunizieren, Dienste anbieten und benutzen. Die Location kann neue Agenten erzeugen und Agenten zu anderen Locations migrieren. Sie bietet Kommunikationsdienste und Yellow Pages Dienste. Desweiteren bietet sie Agenten die Möglichkeit, sich periodisch vom System aufrufen zu lassen.

2.4. Local Agent

Der Local Agent dient als Ansprechpartner, falls Agenten von anderen Locations mit der Location kommunizieren wollen. In der besprochenen Version von Mole ist nur ein Skelett des Local Agent vorhanden, das keinerlei Funktion bietet.

2.5. Engine

Die Engine ist die eigentlich ausgeführte Einheit. Eine oder mehrere Locations werden von einer Engine ausgeführt. Auf diese Weise minimieren sich die Kosten von Locations, da sie viele Dienste, wie z.B. globale Kommunikation, gemeinsam erbringen können. Jede Location wird als eigenständiger Thread ausgeführt. Kommunikation zwischen Locations einer Engine spielt sich daher in einem gemeinsamen Adressraum ab.

3. Aspekte von Agentensystemen

3.1. Naming

Locations haben einen systemweit eindeutigen Namen. Über einen Namenserver kann jede Location identifiziert werden. Beispielsweise : dunedin.mole.informatik.uni-stuttgart.de

System- und Useragenten werden durch eine acht Byte lange Zahlenfolge systemweit eindeutig identifiziert. Bei der Erzeugung eines Agenten durch eine Location wird ihm von der Location ein Name zugeordnet. Jede Location hat einen Vorrat an freien Namen. In der momentanen Version von Mole wird jeder Location beim Start eine Menge von Namen zugeordnet. Sollte dieser Namensvorrat ausgehen, können keine neuen Agenten mehr erzeugt werden.

Obwohl Local Agents eindeutig durch ihre Location identifiziertbar sind, werden sie wie normale Agenten benannt.

3.2. Kommunikation

Mole bietet verschiedenste Möglichkeiten der Kommunikation.

- Java als Sprache bietet die Möglichkeit der Kommunikation über TCP/IP..
- Es können Messages lokal und global verschickt werden, wobei der Sender den Namen und die Location des Empfängers kennen muß. Messages haben eine Fehlersemantik. Falls eine Message nicht ausgeliefert werden kann, passiert entweder nichts, der Sender erhält eine Fehlermeldung oder die Message wird zu einem späteren Zeitpunkt nochmals verschickt. Agenten können einen Vertreter bestimmen, der ihre Messages erhält, falls sie nicht mehr in dieser Location sind.
- Innerhalb einer Location haben zwei Agenten die Möglichkeit mittels RPC zu kommunizieren. Implementiert wird dieser über Messages.

3.3. Migration

Von allen Agentenarten können nur die mobilen Agenten migrieren. Der Agent muß dazu nur das Ziel der Migration kennen. Der Agent führt einen Methodenaufruf der Location aus, er wird gestoppt, gespeichert und als Nachricht an die Ziellocation verschickt. Dort wird er dann neu gestartet. Migriert werden nur die Daten und der Programmcode. Da der Agent immer durch eine Startmethode gestartet wird, muß sein Zustand nicht migriert werden.

3.4. Sicherheit

In Agentensystemen müssen mehrere Ansichten der Sicherheit betrachtet werden :

- Sicherheit zwischen zwei Agenten

Durch die Wahl von Java / Java Bytecode wird diese Sicherheit in Bezug auf Manipulation garantiert. Durch das Fehlen von Pointern und der Kontrolle des Speicherzugriffs sind Agenten vor Manipulationen durch andere Agenten geschützt. Es existiert allerdings keine Beschwerdeinstanz, falls ein Agent einen angebotenen Service nicht erbringt.

- Sicherheit des Systems vor dem Agenten

Die Aussage über Java gilt auch hier. Das Agentensystem interpretiert den Bytecode und hat dadurch sämtliche Kontrollmöglichkeiten. Der Zugriff auf Systemressourcen wird nur Agenten bzw. Objekten erlaubt, die in einer Liste eingetragen sind. Diese Liste wird für jede Engine definiert. Durch eine Obergrenze der Anzahl der gleichzeitig aktiven Agenten kann das System ein Überfluten verhindern. Da Mole in der aktuellen Version aber nicht von sich aus Agenten terminiert, könnte eine Location auf diese Weise blockiert werden.

- Sicherheit des Agenten vor dem System

Der Agent ist dem System völlig ausgeliefert.

3.5. Dienste & Trading

Die Location bietet einen Yellow Pages Dienst. Jeder Agent kann sich dort als Dienstanbieter eintragen lassen. Allerdings gilt dieser Dienst für jede Location alleine, es gibt keinen globalen Dienst. Ein Agent kann sich eine Liste aller angebotenen Dienste geben lassen oder eine Liste aller Agenten, die einen speziellen Dienst anbieten. Dazu muß er allerdings genau wissen, welchen Dienst er haben möchte.
In den Yellow Pages stehen Zeichenketten, die den Dienst benennen und Namen von Agenten, die diesen Dienst anbieten.
Es existiert keine Trading Komponente die in der Lage ist, allgemein formulierte Anforderungen zu verarbeiten und dazu Dienstanbieter zu nennen.

3.6. Fehlerbehandlung

Engines können kontrolliert heruntergefahren werden. Dann speichern sie alle im Moment aktiven Agenten persistent und starten sie beim nächsten Start.
Falls eine Engine aufgrund eines Fehlerzustands beendet wird, gehen alle aktuellen Daten und Agenten verloren.

4. Installation

Zum Betreiben einer Engine benötigt man :

- Einen UNIX Rechner
- Suns Java Development Toolkit JDK 1.0.2
- JavaSofts Object Serialization package (Version 1.9)

Der Installationsvorgang besteht aus folgenden Schritten

- Installation des Java Development Toolkit JDK
- Installation des Object Serialization package (OSP)
- Installation der Mole Software
- Konfigurieren der Engines
- Erzeugen von Startfiles für jede Location

4.1. Installation der Software

- Installation des JDK

Nach dem Entpacken des JDK muß nur die Variable CLASSPATH um alle benötigten Einträge erweitert werden.

- Installation des Object Serialization package

Nach dem Entpacken des Object Serialization package müssen die Variablen CLASSPATH und LD_LIBRARY_PATH angepaßt werden.

- Installation der Mole Software

Nach dem Entpacken der Software muß nur die Variable CLASSPATH erweitert werden.

Das Setzen der Variablen ist Bedingung für die Benutzung von Mole.
Als Beispiel :
JDK-Verzeichnis /home/horsteor/jdk
OSP-Verzeichnis /home/horsteor/serial
Mole-Verzeichnis /home/horsteor/mole

CLASSPATH /home/horsteor/serial/lib/objio.zip:/home/horsteor/mole/classes:
/home/horsteor/jdk/java/lib
LD_LIBRARY_PATH /home/horsteor/serial/lib/sparc

4.2. Konfigurieren der Engines

Jede Engine braucht eine Konfigurationsdatei. Diese enthält zum einen die Engine Definitionen und zum anderen die Definitionen der Locations der Engine.
Der Engine Definitionsteil besteht aus folgenden Komponenten :

- ENGINEADDRESS
  Enthält die IP Adresse des ausführenden Rechners.
- ENGINEPORT
  Enthält die Portnummer der Mole Engine.
- ENGINESYSTEMLIST
  Name eines Textfiles, in dem alle Klassen aufgezählt werden, die in dieser Installation von Mole zur Verfügung stehen. Alle Klassen die hier aufgeführt sind, haben Systemrechte. D.h. Useragenten die in dieser Liste stehen haben dieselben Rechte wie Systemagenten und sollten deshalb hier nicht stehen. Neue Systemagenten müssen hier eingetragen werden.
- ENGINECLASSES
  Zeigt auf ein leeres Verzeichnis, in das neue und temporäre Klassen gespeichert werden können. Im Normalfall sollte jede Engine ihr eigenes Verzeichnis haben.
- ENGINEHOME
  Zeigt normalerweise auf das Mole Verzeichnis. Da der Zugriff nur lesend erfolgt können beliebig viele Engines auf ein Verzeichnis zugreifen.

Nun folgt für jede Location ein Definitionsteil :

- LOCATIONNAME
  Hier steht der DNS Name der Location.
- LOCDESCRIPTION
  Hier steht eine informelle Textzeile.
- STARTUPFILE
  Hier steht das Startfile der Location, in dem die Agenten gespeichert sind, die beim Start gestartet werden.
- NSSTART
  Beginn des Namensraumes der Location.
- NSEND
  Ende des Namensraumes der Location.

Beispiel : siehe Anhang (cembalo.cfg)

4.3. Erzeugen von Startfiles für jede Location

Am einfachsten werden die Startfiles durch ein Programm erzeugt : Makefiles.java.
Für jede Location wird ein Outputstream definiert, dann werden die Agenten über ihre normalen Konstruktoren erzeugt und über den Outputstream persistent gemacht.
Anschließend wird das Programm compiliert und gestartet. Dadurch werden die Startfiles erzeugt.

Beispiel : siehe Anhang (Makefiles.java)

5. Zeitbedarf

	V1	V2	V3	V4	V5	Avg.
Serviceabfrage	16	36	10	11	10	**16**
Wandern	5788	1690	1198	1967	2034	**2535**
Dateiserver	214	94	94	96	101	**119**
Rückkehr	3807	3682	3219	3324	3794	**3565**
Ausgabe	16	19	21	18	26	**20**
Gesamt	**9841**	**5521**	**4542**	**5416**	**5965**	**6257**

6. Zusammenfassung

MOLE ist ein Agentensystem, das verschiedene Arten von Agenten unterstützt. Zur Kostenverringerung der Locations werden Engines eingesetzt. Die Namensvergabe ist systemweit eindeutig. Kommunikation ist sowohl lokal alsauch global möglich. Kommuniziert wird über Messages oder RPCs.
MOLE ist kein fertiges Produkt sondern ein Forschungsprojekt in dem aktiv geforscht wird.

7. Ausblick

Zur Zeit laufen verschiedenste Studien- und Diplomarbeiten die sich mit MOLE auseinandersetzen. Speziell ein verbesserter Installationsvorgang und ein Visualisierungstool. Dieses Tool soll dann den Inhalt einer Location anzeigen.

8. Literatur

Hohl,Fritz
Konzeption eines einfachen Agentensystems und Implementation eines Prototyps
Universität Stuttgart, Fakultät Informatik, Diplomarbeit Nr. 1267 (1995)

Projekt-Homepage
http://www.informatik.uni-stuttgart.de/ipvr/vs/projekte/mole/release.html

9. Anhang

Konfiguratiosdatei einer Engine : cembalo.cfg

```
# cembalo.cfg
#  config file for a Mole engine
# Author: Oliver Horster
# Date: 27.11.96

# definition of the engine
ENGINEADDRESS 129.69.210.24
ENGINEPORT 8009
ENGINESYSTEMLIST systemclasses.dat
ENGINECLASSES /home/horsteor/mole/classes/mole
ENGINEHOME /home/horsteor/mole/engine

# definition of the first location
LOCATIONNAME regensburg.mole.informatik.uni-stuttgart.de
LOCDESCRIPTION  Regensburg unknown city
STARTUPFILE regensburg.dat
NSSTART 0.0.0.0.0.2.0.1
NSEND 0.0.0.0.0.2.255.255

# definition of the second location
LOCATIONNAME wuerzburg.mole.informatik.uni-stuttgart.de
LOCDESCRIPTION Wuerzburg why not
STARTUPFILE wuerzburg.dat
NSSTART 0.0.0.0.0.3.0.0
NSEND 0.0.0.0.0.3.255.255
```

Startupfile der Locations : Makefiles.java

```
// Makefiles.java
//   generate the initial agents persistency file for
//   the engines at start-up time
// System: Java JDK 1.0
// Author: Fritz.Hohl@informatik.uni-stuttgart.de
// Date: 20.6.96
// angepasst von Oliver Horster
```

```
package mole;

import java.io.*;

public class Makefiles
  extends Object {

static public void main(String argv[]) {
  try {
    String moledir = "/home/horsteor/mole/classes/mole/";
    FileOutputStream f1 = new FileOutputStream(moledir + "dunedin.dat");
    ObjectOutputStream s1 = new ObjectOutputStream(f1);
    FileOutputStream f2 = new FileOutputStream(moledir + "greymouth.dat");
    ObjectOutputStream s2 = new ObjectOutputStream(f2);
    FileOutputStream f3 = new FileOutputStream(moledir + "regensburg.dat");
    ObjectOutputStream s3 = new ObjectOutputStream(f3);
    FileOutputStream f4 = new FileOutputStream(moledir + "wuerzburg.dat");
    ObjectOutputStream s4 = new ObjectOutputStream(f4);

  // Dunedin  spinett
    DateiServer ds1 = new DateiServer("fileservice",new AgentName(1,2,3,4,5,6,7,8));
    s1.writeObject(ds1);
    s1.flush();

  // Greymouth  spinett

  // Regensburg  cembalo
     DateiSuchAgent dsa3 = new DateiSuchAgent("ich suche",new
AgentName(1,2,3,4,5,6,7,66),new String("versuch.txt"), new
LocationName("dunedin.mole.informatik.uni-stuttgart.de"));
      s3.writeObject(dsa3);
      s3.flush();

  // Wuerzburg   cembalo
}
  catch (java.lang.Exception e) {
    System.out.println("Makefiles.java: Error: " + e.toString());
  }}}
```

Beispielagent : DateiServer

```
// DateiServer.java
// System: Java JDK 1.0
// Autor: Oliver Horster
// HS VS  Beispielagent

package mole;

import java.io.*;

public class DateiServer
  extends SystemAgent {
  private String Path = "/home/horsteor/mole/Dateien/";
// ----------------------------------------
// constructor
// ----------------------------------------
```

```
public DateiServer() {
}
// ----------------------------------------

public DateiServer(String einString, AgentName aName) {
  // standard constructor

  selfdescription = einString;
  myname = aName;
}
// ----------------------------------------
//      start
// ----------------------------------------
public void start() {
  // awake after migration or creation

AgentName a =null;
Engine.debug("=========   starts");
  actuallocation.registerMeForService(myname, "DateiServer");
Engine.debug("=========   registered");

// sanity check / check for second server
a=(actuallocation.serviceProvidersOf("DateiServer"))[0];
Engine.debug("=========   : Name "+ myname.toString());
Engine.debug("=========   : Provider  "+ a.toString());
  if (!a.equals(myname)) {
    actuallocation.die(this);
  }
}
// ----------------------------------------
//      end
// ----------------------------------------
public void end() {
  // leave system

Engine.debug("=========   endet");
  actuallocation.unregisterMeForService(myname, "DateiServer");
Engine.debug("=========   unregister");
}
// ----------------------------------------
//           File Service
// ----------------------------------------
public String[] getFile(String Filename)
  throws Exception {
  // returns an File

  String doc[] = new String[1000];
  String doc2[];
  String line = "";
  int counter = 0;
  FileInputStream fis = null;

Engine.debug("=========  entering getFile with : "+Filename);
 // Open and Read the File
  try {
    fis = new FileInputStream(Path + Filename);
    line = getline(fis);
```

```
    while (line != null) {
      doc[counter++] = line;
      line = getline(fis);
    }
  }
  catch (IOException e) {
Engine.debug("=========  : can't open " + Path+Filename);
    throw(new Exception());
  }

  // after work, close the File
  try {
    fis.close();
  }
  catch (IOException e) {
Engine.debug("=========  : can't close " + Path+Filename);
    throw(new Exception());
  }

  // copy file
doc2 = new String[counter];
  for (int i = 0; i < counter; i++) {
    doc2[i] = doc[i];
  }
  return(doc2);
}

// ----------------------------------------------------
public String askforFile(String Filename)
  throws Exception {
  // return "file 'Filename' exists" if File exists
  // else "no file 'Filename'"

  String line = "";
  String answer = new String();
  int counter = 0;
  FileInputStream fis = null;

Engine.debug("=========  entering askforFile with : "+Filename);
  // Open and Read the File

  try {
    fis = new FileInputStream(Path + Filename);
    line = getline(fis);
    answer = "file "+Filename+" exists";
}
  catch (IOException e) {
Engine.debug("=========  : can't open " + Path+Filename);
    answer = "no file "+Filename;
Engine.debug("=========  : askforfile returns exept."+answer);
  return(answer);
  }
  // after work, close the File
  try {
    fis.close();
  }
```

```
  catch (IOException e) {
Engine.debug("=========  : can't close " + Path+Filename);
    throw(new Exception());
  }
Engine.debug("=========  : askforfile returns "+answer);
  return(answer);
}
// ----------------------------------------------------
private String getline(InputStream ci)
  throws java.io.IOException {
  // adds the missing InputStream.readln() method

  int ai = 0;
  int counter = 0;
  byte ba[] = new byte[1000];

  ai = ci.read();
  while((ai != 10) && (ai != -1)) {
    ba[counter++] = (byte)ai;
    ai = ci.read();
  }
  if (ai == -1) {
    return(null);
  }
  else {
    return(new String(ba, 0, 0, counter));
  }
}
// ----------------------------------------
//            messages
// ----------------------------------------
public void receiveMessage(Message m) {
  // this procedure is called by the system when messages
  // for this agents arrive

  String tmp = "";

Engine.debug("============  get message");
  try {
    if (m.content instanceof String) {
     tmp = (String)(m.content);
Engine.debug("========== Messagetext :::"+tmp+":::");
     if (tmp != "") {
       if (tmp.endsWith("?")) {
       actuallocation.message(new Message(
        myname,
        actuallocation.locationName(),
        m.sender,
        m.senderlocation ,
        0,
        0, // Message ID
        askforFile(tmp.substring(0,tmp.length()-1))));
        }
       else {
       actuallocation.message(new Message(
        myname,
        actuallocation.locationName(),
        m.sender,
```

```
        m.senderlocation ,
        0,
        1, // Message ID
        getFile(tmp)));
       }
      }

      else {
      Engine.debug("=========  : empty filename");
      }
    }
    else {
      Engine.debug("=========  : unknown message");
    }
  }
  catch (Exception e) {
    // do nothing
  }
}
// ----------------------------------------
}
```

Beispielagent : DateiSuchAgent

```
// DateiSuchAgent.java
//   sucht eine spezielle Datei
// System: Java JDK 1.0
// Autor: Oliver Horster
// Datum: 3.12.96

package mole;

import java.net.*;
import java.io.*;

public class DateiSuchAgent
  extends UserAgent
  implements MobileAgent {

  AgentName DServer = null;             // der Dateiserver dieser Location
  String Dateiname = "";          // Name der Datei
  int Status = 0;               // Datei fehlt noch
  String Datei[] = null;          // Die gesuchte Datei
  LocationName home = null;        // Startloc.
  LocationName target = null;      // Ausweichloc.
    // folgende Variablen dienen der Zeitmessung
  long tstart=0l;
  long twander=0l;
  long tanfrage=0l;
  long tantwort=0l;
  long twander2=0l;
  long tende=0l;

// ----------------------------------------
// constructor
// ----------------------------------------
public DateiSuchAgent() {
}
```

```
// ----------------------------------------
public DateiSuchAgent(String einString, AgentName aName, String gesucht ,
                                LocationName t) {
  // the standard constructor



  selfdescription = einString;
  myname = aName;
  Dateiname = gesucht;
  target=t;
}
// ----------------------------------------
//          start
// ----------------------------------------
public void start() {
  // awake after migration or creation

    AgentName dlist[];

  // home Varible belegen
  if (home==null) {
    home=actuallocation.locationName();
    tstart=System.currentTimeMillis();
  }

  if (Status == 0 ) {    // erstmal File holen

    // Testet, ob DateiServerdienst verfuegbar
    Engine.debug(">>>>>>>>> DateiSuchAgent start");

    dlist= (actuallocation.serviceProvidersOf("DateiServer"));

for (int i=0;i<dlist.length;i++){
  Engine.debug(">>>>>>>>> DateiSuchAgent provider :"+dlist[i]); }

    if (dlist.length != 0) {
      DServer=dlist[0];
    }

    if (DServer != null) {
Engine.debug(">>>>>>>>> DateiSuchAgent server gefunden");
Engine.debug(">>>>>>>>> Frage Server : "+DServer.toString());
      // DateiServer existiert !!
      tanfrage=System.currentTimeMillis();
      actuallocation.message(new Message(
      myname,
      actuallocation.locationName(),
      DServer,
      actuallocation.locationName(),
      0,
      0, //Message ID
      Dateiname+"?"));
    }
    else {
Engine.debug(">>>>>>>>> DateiSuchAgent kein server da");
```

```
      // Kein Server hier !!!! also Wandern
      Engine.debug(">>>>>>>>  DateiSuchAgent: no Server here");
      Engine.debug(">>>>>>>>  DateiSuchAgent: suche jetzt in Dunedin");
      twander=System.currentTimeMillis();
      actuallocation.goTo(this,target);
    }
  } else {
    twander2=System.currentTimeMillis();
    System.out.println("das Ergebnis des DateiSuchAgenten");
    System.out.println("---------------------------------");
    for (int i=0 ; i<Datei.length; i++) {
      System.out.println(Datei[i]);
    }
    System.out.println("---------------------------------");

    // Nach erledigter Arbeit terminieren wir uns
    actuallocation.die(this);
  }
}
// ----------------------------------------
//          messages
// ----------------------------------------
public void receiveMessage(Message m) {
  // this procedure is called by the system when messages
  // for this agents arrive

Engine.debug(">>>>>> DateiSuchAgent: Message from " + m.sender.toString());
Engine.debug(">>>>>> DateiSuchAgent: Message "+m.content);
Engine.debug(">>>>>> DateiSuchAgent: Message ID "+m.messageid);

  if (m.messageid==0 ) {

Engine.debug(">>>>>> DateiSuchAgent: FILE FOUND");

    // fordere Datei an
    // die laut Aufgabenstellung da sein muss -> Fehlerbehandlung fehlt
    actuallocation.message(new Message(
    myname,
    actuallocation.locationName(),
    DServer,
    actuallocation.locationName(),
    0,
    1, //Message ID
    Dateiname));
  }
  else {

    // Datei erhalten, nach Hause und abliefern !!!
    Datei= (String[])m.content;

    Status=1; // Datei erhalten
    tantwort=System.currentTimeMillis();
    actuallocation.goTo(this,home);  // ab nach Hause

  }
}
```

```
// ----------------------------------------
//           end
// ----------------------------------------
public void end() {
  // prepare to be removed from the system

tende=System.currentTimeMillis();



Engine.debug(">>>>>> DateiSuchAgent: Zeitmessung");
Engine.debug(">>>>>> DateiSuchAgent: ");
Engine.debug(">>>>>> DateiSuchAgent: start      : " + String.valueOf(tstart));
Engine.debug(">>>>>> DateiSuchAgent: wander     : " + String.valueOf(twander));
Engine.debug(">>>>>> DateiSuchAgent: anfrage    : " + String.valueOf(tanfrage));
Engine.debug(">>>>>> DateiSuchAgent: antwort    : " + String.valueOf(tantwort));
Engine.debug(">>>>>> DateiSuchAgent: wander2    : " + String.valueOf(twander2));
Engine.debug(">>>>>> DateiSuchAgent: ende       : " + String.valueOf(tende));

}
// ----------------------------------------
}
```