Mobile Agenten

Matthias Rohr (mi4295)
Informatik Seminar SS2004
FH Wedel

Inhalt

I. Einleitung

Abstrakt

Mobile Agenten – Ende der 90er Jahre noch als revolutionäre Softwarediszipliin gefeiert, hat sich in den letzten Jahren vielfach Ernüchterung breit gemacht. Einige Gründe hierfür, sowie Möglichkeiten, wie Mobile Agenten doch die Bedeutung erlangen könnten, werde ich im hinteren Teil dieser Arbeite beleuchten, nicht jedoch ohne vorher die nötigen Grundlagen beleuchtet zu haben.

Hierzu will ich mich in erster Linie auf (die wenigen) vorhandenen Standards (z.B. MAFO) stützten, die im letzten Kapitel dann näher ausgeführt werden.

Was sind Mobile Agenten ?

Der Begriff „Mobiler Agent“ sieht sich glücklicherweise nicht ganz so vielen Definitionsversuchen ausgesetzt, wie es bei dem Begriff eines Software-Agent der Fall ist. Grundsätzlich kann man hier zwei unterschiedliche Ansätze antreffen:

1. Mobiler Agent als Kurzform für „mobiler Software-Agent“, also eine um die Eigenschaft Mobilität erweiterte Softwareentität
2. Mobiler Agent als Form eines Software-Agenten der mobil ist, jedoch dadurch auch geprägt von anderen Eigenschaften oder Ausprägungen ist

Der MAFO-Standard [vgl MAF], die wohl bedeutendste Spezifikation zu diesem Thema, definiert einen Mobilen Agenten nach dem ersten Muster, wonach dieser zunächst ein (Software-)Agent und damit ein *„ablauffähiges Programm, welches autonom im Auftrag einer Person oder Organisation handelt*“ darstellt und darüber hinaus die Eigenschaft besitzt zwischen verschiedenen Agenten-Systemen – welche die Laufzeitumgebungen des Agenten darstellen, wechseln kann.

Entsprechend ist hier ein *Stationärer Agent* als Agent zu betrachten, der nur auf einem Agentensystem ausgeführt wird.

In dieser recht kurzen Definition befinden zwei entscheidende Aussagen: 1. Ein Agent handelt immer im Auftrag von jemandem (einer sog. Autorität) und 2. er existiert nur innerhalb eines Agentensystems (AS). In den meisten Fällen wird dieser mit Multiagenten-Systemen (MAS) gleichgesetzt, mehrere Agenten zur gleichen Zeit enthalten können.

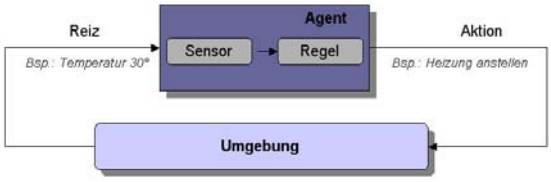
Auch wird statt Agentensystem oft der Begriff Umgebung (engl. Environment) verwendet, die jedoch einer abstrakteren Betrachtung zugrunde liegt und auch einen bestimmten Bereich innerhalb eines Agentensystems darstellen.
Die MAFO-Definitionen will ich als Grundlage nehmen, komme jedoch nicht darum, diese nach dem zweiten Muster[1] entsprechend anzupassen. Dies soll im Folgenden anhand der Eigenschaften eines mobilen Agenten geschehen, wie sie von den meisten Autoren in diesem Zusammenhang genannt werden:

Eigenschaften eines Mobilen Agenten:

1. Mobilität
   Der Agent kann zwischen verschiedenen Agentensystemen wechseln. Man sagt: er *migriert* auf ein anderes Agentensystem. Hierfür müssen zwei Vorraussetzungen erfüllt sein:
   - Der Agent kann als Datenstrom dargestellt werden
   - Der Agent kann von der Zielplattform entfangen und interpretiert / ausgeführt werden

2. Autonomität
   Der Agent entscheidet selbstständig anhand bestimmter Kriterien über seine nächste Aktion . D.h er arbeitet unabhängig von einer kontrollierenden Instanz (z.B. einem Anwender).

3. Soziale Kompetenz
   Der Agent kommuniziert mit anderen Agenten, (Agenten-)Systemen oder auch Menschen. Vorraussetzung hierfür:
   - gemeinsame Protokolle
   - gemeinsame Ontologien (siehe Kapitel 2)
   - gemeinsame Interaktionsvereinbarungen (Sprechakte)



4. Reaktivität
   Der Agent reagiert auf Änderungen seiner Umgebung. Hierzu besitzt er Sensoren mit denen er diese wahrnehmen kann (z.B. Temperaturerhöhung) und Regeln die eine bestimmte Aktion (z.B. Heizung anstellen) auslösen, oder den empfangenen Reiz ignorieren.

Abb. 1.1: Reaktivität (Auf Basis von Wooldridge2002)

5. Proaktivität (auch Zielorientiertheit)

Agenten reagieren nicht nur auf Reize ihrer Umgebung, sondern besitzen zudem einen internen *Zustand* (engl. state) und sind zu zielgerichtetem Planung und Handeln fähig. Man sagt: sie ergreifen die Initiative[2].

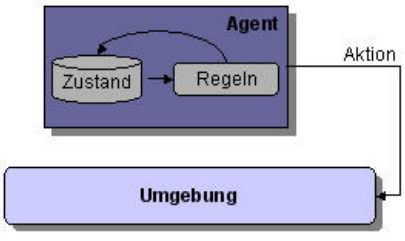


Abb. 1.2: Proaktivität
(Auf Basis von Wooldridge2002)

Über diese recht simple Betrachtung von Zustand und Regeln lassen sich zudem weitere Eigenschaften wie Intelligenz oder Persönlichkeit über eine entsprechende Komplexität beider Komponenten, sowie der Beziehungen zwischen ihnen abbilden.

[1] ich will deshalb die Bezeichnung „Mobiler Agent“ als eigenen Bezeichner der Bezeichnung „mobiler Agent“, vorziehen, wo mobil lediglich ein Verb für Agenten darstellt.

[2] Le Do sagte hierzu: “The difference between an automation and an agent is a somewhat like the difference between a dog and a butler. If you send your dog to buy a copy of the New York Times every morning, it will come back with its mouth empty if the news stand happens to have run out one day. In contrast, the butler will probably take the **initiative** to buy you a copy of the Washington Post, since he knows, that sometimes you read it instead.”

6. Persistenz

   Der Agent bleibt auch nach einen Umgebungswechsel mitsamt seines Zustandes erhalten, d.h. er wird nicht schlicht kopiert, sondern setzt seine Ausführung innerhalb der neuen Umgebung fort.

Sicherlich ließen sich hier noch weitere Eigenschaften aufzählen die mobile Agenten besitzen können, die innerhalb dieser Betrachtung allerdings weniger Relevanz besitzen.

Abgrenzung zu anderen Softwareparadigmen

Was Mobile Agenten für die Softwareentwicklung bedeuten, lässt sich am anschaulichsten darstellen, wenn man dieses mit anderen Konzepten (auch: Paradigmen) vergleicht, die gemeinüblich zur Programmierung von Netzwerkanwendungen eingesetzt werden.

Mobiler Code

Hierunter soll jegliche Software verstanden werden, die auf einem entfernten System aufgeführt wird.

Beispiele für Mobilen Code sind:

- Postscript
- Applets
- Mobile Agenten

Mobile Agenten stellen somit eine spezielle Form von Mobilem Code dar.

Bösartiger Mobiler Code

Oft werden unter Mobilen Agenten Software-Schädlinge wie Würmer, Viren oder Trojaner verstanden.

Diese stellen bezogen auf die eingangs dargestellte Definition Mobiler Agenten jedoch keinen solchen dar, da diese nicht auf einem Agentensystem aufsetzen. Stattdessen handelt es sich dabei um sogenannten bösartigen Mobilen Code, der hier nicht weiter Beachtung finden soll.

Client-Server (CS)

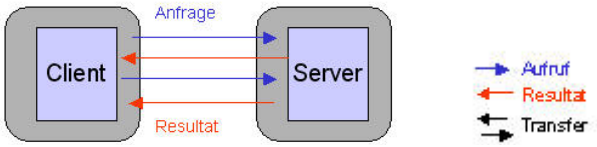


Abb. 1.3: Client Server

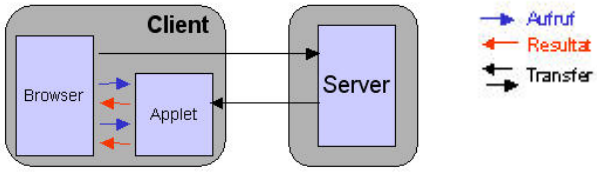
Hierbei stellt der Client an den Server Anfragen (requests), die dieser ausführt und seinerseits mit entsprechenden Antworten (responses) erwidert. Beide Übertragungen basieren dabei auf ein vom Server vorgegebenem Format (Protokoll).

Abstrahiert kann man sagen, dass hier beim Server die Funktionalität, aber auch der Grossteil des Ressourcenverbrauchs einer Anfrage liegt.

Beispiele: HTTP oder FTP.

Code-on-Demand (CoD)

Anstatt Anfragen vollständig auf der Serverseite zu bearbeiten, dient diese hier lediglich zur Auslieferungen eines Stücks Software an den Client. Der – nun in der Lage dieses zu interpretieren – führt es darauf mit eigenen, lokalen Ressourcen aus.



Der Vorteil neben der Entlastung der Server-Ressourcen, besteht bei diesem Ansatz auch in der Reduzierung der Netzwerkkommunikation, die während der Ausführung der Anwendung sogar gleich null ist.

Beispiel: Java Applets

Remote Evaluation (REV)

Einen etwas anderen und auch komplizierteren Ansatz bietet die *Entfernte Ausführung* (Remote Evaluation). Der Grundgedanke: Eine Clientanwendung ruft eine Prozedur auf, wobei es für ihn transparent ist, ob diese lokal, oder auf einem entfernten Rechner ausgeführt wird.

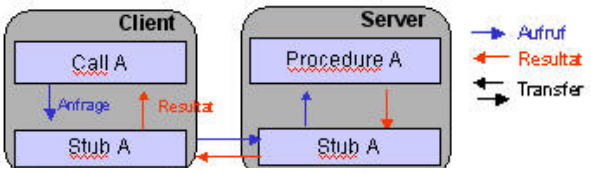


Abb. 1.4: RPC Funktionalität

Realisiert wird dies durch den Einsatzlokaler Stellvertreter, sogenannter *Stubs* (engl. für Stummel). Der Client ruft dabei nicht die „echte“ Prozedur auf dem entfernten Rechner, sondern den lokalen Stub dieser Prozedur auf.

Über den *Stub* wird nun die direkte Netzwerkkommunikation mit dem *Stub* auf der Gegenseite hergestellt[3], worüber schließlich die „echte“ Prozedur aufgerufen wird und deren Ergebnisse über den gleichen Weg an den ursprünglichen Aufruf zurückgibt.

Es gibt zahlreiche Implementierungen dieses Konzepts, Beispiele sind RPC, RMI[4] (Java spezifisch), DCOM (W32 spezifisch) und Corba[5], die diesen Ansatz auf unterschiedliche Weise (teilweise unter Einsatz eines zusätzlichen Namensdienstes) umsetzen.

Grundsätzlich lässt sich jedoch ein Aufruf von drei Methoden (A(),B() und C()) auf allen Implementierungen, immer wir folgt darstellen:

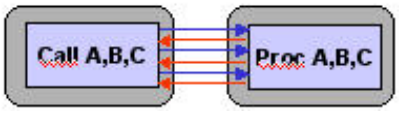


Abb. 1.5: ReV Ablauf

Mobile Agenten (MA)

Bezogen auf die vorgestellten Konzepte, besitzen Mobile Agenten in diesem Kontext teilweise Ähnlichkeiten mit CoD (Code on Demand), nur dass hierbei die Softwareentität eben nicht vom Client abgerufen, sondern von diesem erzeugt und auf den Server zur Verarbeitung übertragen wird.

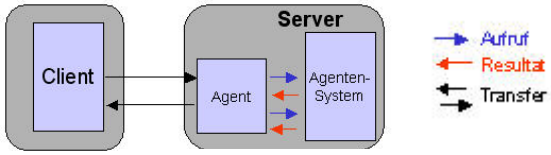


Abb. 1.6: Mobile Agenten

[3] In manchen Implementierungen (z.B. RMI) wird diese Komponente auch *Skeleton* bezeichnet
[4] Da in dieser Arbeit stark auf Java bezug genommen wird, wo RPC nur in Form von RMI eingesetzt wird, soll im Folgenden RPC mit RMI assoziiert werden, auch wenn beide Technologien unterschiede aufweisen.
[5] mehr zu Corba ist im Kapitel 3 nachzulesen

Die Ressourcen werden somit beim Server verbraucht, während die Funktionalität (jedenfalls theoretisch) vom Client vollständig vorgegeben werden.

Insofern besitzen beide Konzepte (CoD und MA) die gleiche Motivation: den eigenen Ressourcenverbrauch und die Netzwerkbelastung minimal zu halten. Einmal aus Sicht des Servers, einmal aus Sicht des Clients.

Mobile Agenten im Vergleich zu RPCs

Da viele Autoren den Grundgedanken hinter Entwicklung Mobiler Agenten hauptsächlich in der Schaffung einer Alternative zu *Remote Procedure Calls* (RPCs) sehen, liegt der direkte Vergleich beider Konzepte an dieser Stelle nahe. Der Idee hierhinter war, dass Mobile Agenten durch ihre Architektur, prinzipiell die gleichen Möglichkeiten wie RPCs bieten, dabei jedoch ohne deren Nachteile auskommen. Die da wären:

1. Bandbreite:

Jeder Prozeduraufruf und dessen Antwort vom Server wird über das Netzwerk übertragen. Bei hundert Aufrufen während der Ausführung eines Programms ergäbe sich somit eine beträchtliche Netzwerkbelastung.
Mobile Agenten hingen werden nur einmal versendet und empfangen.

Auf der andren Seite können RPCs bei nur geringer Kommunikation aber auch weitaus effektiver als Mobile Agenten sein (schließlich muss hier kein Agentencode übertragen werden. Strasser und Schwem [Strasser1997] haben dies wie folgt ermittelt:

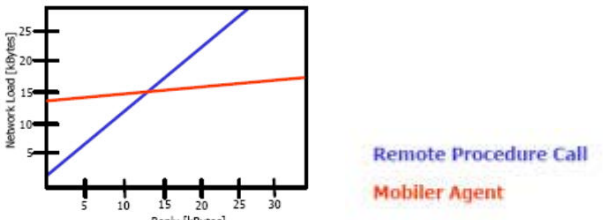


Abb. 1.6: RPC vs. Mobile Agenten (nach Strasser91997)

2. Latenzzeit

Die Dauer zwischen Prozeduraufruf und Antwort liegt bei der lokalen Kommunikation des Mobilen Agenten gewiss deutlich unter der von RPCs, welche gewöhnlich über das Netzwerk erfolgen.

Allerdings kann auch hier ein RPC durchaus lokal oder in räumlicher Nähe zum Client erfolgen, was dann nur noch sehr geringe Unterschiede zu Mobilen Agenten bedeuten würde.

Hingegen wäre für eine beträchtliche Entfernung beider Seiten – etwa wie bei der zwischen Hamburg und Taipeh, RPCs sicherlich eine denkbar ineffektive Variante und Mobilen Agenten klar unterlegen.

3. Ausfallsicherheit

Bei Netzwerkübertragungen besteht stets die Gefahr, dass die übertragene Nachricht nicht oder nur unbrauchbar ankommt.

In beiden Fällen müsste eine entsprechende Funktionalität implementiert (z.B. Timeouts) oder genutzt werden, wodurch ein höherer Aufwand und potenziell längere Übertragungsdauer, bestehen würde. Bei der rein lokalen Kommunikation Mobiler Agenten besteht dieser Nachteil nicht.

Zusammenfassung

Vereinfacht kann man die vorgestellten Konzepte wie folgt voneinander abgrenzen:

	Ressourcenverbrauch	Funktionalität
CS	Server	Server
CoD	Client	Server
ReV	Server	Client / Server*
MA	Server	Client

* Server gibt die Prozeduren vor

Mobile Agenten muten hier leicht als veridealisierte Disziplin an: Der Client darf Funktionalität der Anwendung selbst vollständig bestimmten, ohne dabei lokale Ressourcen zur Verfügung stellen zu müssen.

Der direkte Vergleich zu RPCs macht jedoch deutlich, dass der zusätzliche Aufwand für die Erstellung eines Mobilen Agenten als Ersatz für ein bestehendes RPC-Konzept, stark vom jeweiligen Szenario abhängt. Sind viele Aufrufe über weiter Entfernung oder ist ein großes Datenaufkommen zu erwarten bieten sich Mobile Agenten an, bei räumlich naher Entfernung beider Seiten oder nur wenig Kommunikation mit geringem Datenaufkommen dagegen sind klassische RPCs wiederum im Vorteil.

Was hier allerdings bisher noch nicht betrachtet wurde, ist die Möglichkeiten Mobiler Agenten gänzlich neue Anwendungsfelder, die für konventionelle Konzepte[6] bisher kaum vorstellbar waren, zu erschließen.

Anwendungsszenarien Mobiler Agenten

1. Intelligente Suche in verteilten Datenbanken

Ein Benutzer Bob möchte eine Internet Recherche betrieben.

Bisher:

Bob fängt bei einem Suchdienst an, wobei er meist nur wenige Worte als Suchanfrage angibt und überfliegt dann die ersten paar Ergebnisse, spezialisiert darauf eventuell seine Anfrage oder wechselt auf die angebotenen Inhalte, wobei er immer wieder Verweisen folgt. Ergebnis: Die Recherche dauert sehr lange, wobei viele relevante Informationen von Bob nicht wahrgenommen werden.

Durch Mobile Agenten:

Bob definiert (etwa durch eine GUI) seine Suchanfrage. Er sagt was er will und in welchem Kontext (z.B. Kohl im Kontext Politik) stehen soll. Er schickt einen Suchagenten auf verschiedene Datenbanken. Trifft der Agent nun auf eine relevante Information (die er durch seine KI selektiert), so speichert er diese. Trifft er auf einen Verweis zu einer weiteren Quelle, klont er sich und schickt seinen Klon zu der angegebenen Ressourcen. Der Klon verfährt genauso, usw..
Die Klone kehren nach einiger Zeit zu ihrem Erschaffer zurück, übergeben ihm die gefundenen Informationen und terminieren. Schließlich tut dies auch der einst von Bob erzeugte Agent, der nun die Information sämtlicher Klone an Bob gibt.
Ergebnis: für die Recherche muss Bob nur seine Suchanfrage formulieren, und erhält sehr umfangreiche Resultate, die er lokale schnell selektieren kann.

[6] hierbei soll unter konventionellen Konzepten, die drei vorgestellten Softwareparadigmen (CS, CoD und ReV) verstanden werden

2. Überwachung und Steuerung

Ein Agent wird auf ein Börsen System geschickt, überwacht dabei sämtliche Kursverläufe und vollzieht Aktionen entsprechend der ihm mitgegebenen Parameter (z.B. Verkauf wenn Aktie um 0,2 Prozent fällt).

3. Mobile Computing

Bob möchte von unterwegs eine sehr umfangreiche Formel berechnen. Da er in seinem PDA hierfür nicht über die nötigen Ressourcen verfügt, schickt er einen Mobilen Agenten an einen leistungsfähigen Rechner, auf dem er Formel ausrechnen lässt und zusammen mit dem Ergebnis auf den PDA zurückkehrt.

4. Electronic Commerce

Bob möchte eine günstige Reise nach Kuba buchen. Dafür erstellt er einen Shopping Agenten, der zwischen verschiedenen Reise-Anbietern wechselt, die günstigsten Preise ermittelt, eventuell diese sogar bucht und mit den Ergebnissen an Bob zurückkehrt.
Wie problematisch speziell dieser Einsatz ist, soll im Abschnitt zu Sicherheit von Mobilen Agenten, näher dargestellt werden.

Abschließend bleibt hier noch zu sagen, dass bisher keine Killer-Applikation für den Einsatz Mobiler Agenten existieren, die also etwa den Aufbau von weltweiten Agentensystemen ökonomisch rechtfertigen würde. Wie später gezeigt, sprechen auch andere Gründe gegen ein solches Vorhaben.

II. Aspekte eines Agentensystems

Zahlreiche[7] Implementierungen von Agentensystemen (= Agentensystem-Typen) wurden mittlerweile entwickelt, darunter Grasshopper, Voyager, IBM Aglets, usw. Die teilweise höchst unterschiedlichen Konzepte die dabei integriert werden, sollen in diesem Abschnitt näher betrachtet werden.

Agentensprache

Eine Programmiersprache muss theoretisch sowohl für den Mobilen Agenten selbst, als auch für deren Agentensystem festgelegt werden. Da allerdings insbesondere aus Portabilitätsgründen bestehende Systeme vor allem auf interpretierenden Sprachen wie TCL oder Java basieren, gibt das Agentensystem dadurch gleich auch die konkrete Agentensprache vor.

Waren die ersten Systeme noch auf Basis von TCL (bzw. AgentTCL) realisiert, konnte sich in den letzten Jahren insbesondere Java als dominierende Agentensprache durchgesetzt.

Einige Gründe hierfür sind:

Portabilität	ein Java-Interpreter (JDK) ist inzwischen vom Mobilem Handy bis hin zum Mainframe, für große Anzahl and Plattformen erhältlich.
Sicherheit	Java-Programme laufen in einer Virtuellen Maschine ab und haben deshalb keinen direkten Zugriff auf systemkritische Komponenten (wie etwadem Framestack). Zusätzlich befindet sich jedes Java Programm in einem eigenen Kontext (sog. Sandbox), für den spezifische Rechte festgelegt werden können
Dynamisches Binden	Dynamisches Laden von Klassen während der Laufzeit
Multithreading	Sogenannte leichtgewichtige Prozesse gewährleisten einen ressourcensparende und performanten Betrieb

[7] www.inf.uno-stuttgart.de

Hingegen existieren jedoch auch Argumente, die gegen den Einsatz von Java sprechen, etwa die fehlende Kontrolle des Ressourcenverbrauchs eines Threads / Objektes und insbesondere die nur eingeschränkte Serialisierbarkeit, auf die später genauer eingegangen wird.

Agentenmanagement

Agenten werden aus der Sicht eines Agentensystems über einen sogenannten Agentenlebenszyklus (vgl. Abb. 2.1) abgebildet:

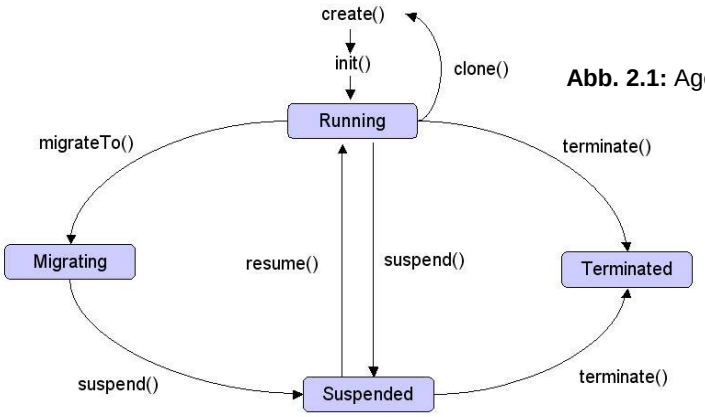


Abb. 2.1: Agentenlebenszyklus

Ähnlichkeiten zu einem lebenden Wesen sind hier offensichtlich: der Agent wird „geboren“, lebt, schläft und wacht auf, pflanzt sich fort[8] und, wenn seine Zeit gekommen ist stirb er.

Die Phasen des Agentenzyklus sind letztlich die bestimmenden Faktoren, auf die ein Agentensystem zur Steuerung eines Mobilen Agenten zugreift. Aus diesem Grund ist dieser auf manchen Systemen in leicht abgewandelter Form vorzufinden.

Weiterhin wird in diesem Zusammenhang auch eine Form von eindeutiger Adressierung für den Agenten benötigt. Übliche Form für eine globale Agentenadressierung ist dabei **Agent-Identifier@host/context/place** wobei konkreter auf dieses Thema in Kapitel Standardisierung behandelt wird.

Mobility

Die Konzepte die dem Agenten seine Mobilität verleihen, lassen sich aus netzwerktechnischer (*Protokolle*) wie aus softwaretechnischen Sicht (*Migration*) betrachten.

Protokolle

Je nach Agentensystem, werden hier verschiedene Technologien angeboten. In den meisten Fällen basiert die Übertragung auf Plain-Sockets oder RPC/RMI, manchmal jedoch auch ganz ungewöhnlichen Techniken, wie Mail (z.B. Voyager) oder HTTP (z.B. Grasshopper[9]).

Migration

Bereits zuvor wurde ein Agent im übertragenen Sinne mit einem lebenden Wesen verglichen. Dementsprechend soll durch die Migration eines Agentenprozesses erreicht werden, dass dieser nicht etwa beendet und auf dem Zielsystem neu gestartet, sondern vielmehr in seiner Ausführung fortfährt -

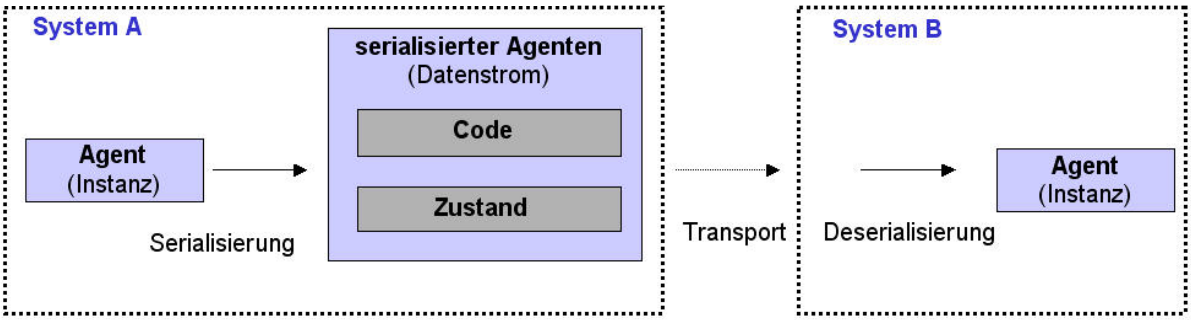
[8] Da der Agent dies bisher nur mit sich selbst tut und dazu jederzeit Fähig ist, könnte man sogar sagen er wäre dem Menschen evolutionär im Vorteil :)
[9] Grasshopper bietet hierfür eigene Webserver-Module (z.B. für den Apache-HTTPD)

also angehalten und reaktiviert wird. Der Umfang, in dem dies ein Agentensystem ermöglicht bestimmt die Stärke der Migration, auf die etwas weiter unten eingegangen wird.

Migrationsablauf

Die einzelnen Schritte, die von einem Agentensystem durchgeführt werden müssen um einen Agenten(-Prozess) von einem System auf ein anderes zu bewegen, wird durch die MAFO-Spezifikation (vgl. Kapitel Standardisierung) wie folgt empfohlen:

Abb. 2.2: Migrationsablauf (allgemein)



System A (versendet Agenten)

1) Ausführung des Agenten wird unterbrochen
2) Identifikation der zu übertragenen Teile (z.B. Klassen) des Agenten
3) Agent wird serialisiert
4) Enkodierung des Agenten entsprechend des verwendeten Transport Protokolls (z.B. HTTP)
5) Authentifizierung bei System B
6) Übertragung von Agent mit Code u. Zustand

System B (empfängt Agenten)

7) Authentifizierung von System A
8) Dekodierung des Agenten
9) Deserialsierung d.A.
10) Instanzzierung d.A.
11) Wiederherstellung des Agentenzustandes
12) Fortsetzen der Agentenausführung

Bestandteile einer Migration

Welche Informationen letztlich in den serialisierten Agenten-Datenstrom gespeichert und auf dem Zielsystem rekonstruiert werden, ist maßgeblich durch die verwendete Agentensprache, aber auch durch das Agentensystem selbst, bestimmt.

Neben einem Codeteil (der die benötigten Klassen enthält) betrifft dies insbesondere den Kontextteil[10]. Hier sind verschiedene Informationen zum Agenten (Name, Identifikation) - vor allem aber auch der bereits diskutierte Zustand des Agenten - enthalten

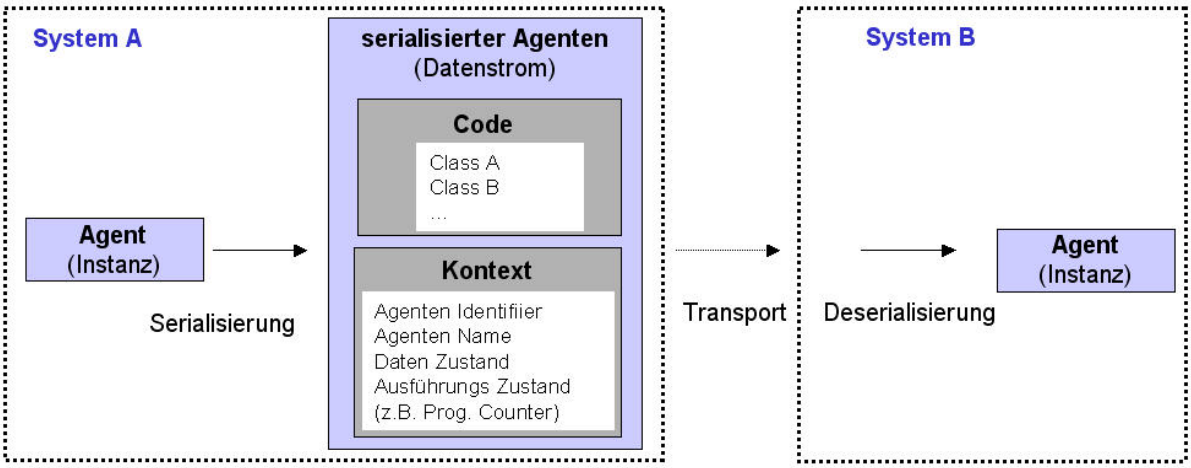
Diesen unterteilt man in Daten-Zustand (z.B. Variablen-Werte) und Ausführungszustand (u.a. Programm-Counter). Unterstützt ein Agentensystem auch die Serialisierung nach der zweiten Form, so spricht man von starker- ansonsten von schwacher Migration.

Grundsätzlich gilt in diesem Zusammenhang nach [Braun2000] die Regel:

=> je schwächer Migrationsart, desto mehr Aufwand für den Programmierer
=> je stärker die Migrationsart, desto komplizierter für das Agentensystem

[10] Der Kontext soll hierbei den bisher verwendeten Begriff Zustand erweitern

Abb. 2.3: Migrationsablauf (detailliert)



Schwache Migration

Diese Form ist in der Praxis fast ausschließlich anzutreffen; auch hier wieder in zwei unterschiedlichen Konzepten: mit festem und beliebigen Einstiegspunkt.

Bei der schwache Migration mit **festem Einstiegspunkt** wird beim instanziierten Agentenprozess auf dem Zielsystem an einen fest vorbestimmten Punkt gesprungen. Bei Aglets ist dies beispielsweise durch Callback-Methoden realisiert.

```
public class Display
      implements java.io.Serializable
{
   Boolean isRemote;
   public class Example1 extends Aglet {
       public void onCreation (Object init) {
       addMobilityListener( new MobilityAdapter() {
      public void onArrival( MobilityEvent e) { Einstiegspunkt auf System B
            System.out.println(„i am remote“);
            isRemote = true;
      }
       }
   };
public void run() {
   if (! isRemote) {
      moveTo(„Host2“);   Ausstiegspunkt auf System A
   }
}
```

Dem hingegen gibt ein Konzept der schwachen Migration mit **beliebigen Einstiegspunkt** dem Agenten (bzw. der aufrufenden Instanz) die Möglichkeit, vor dem Verlassen eines Systems, den Punkt (z.B. eine Methode) zu bestimmen, an dem dieser auf dem Zielsystem, seine Ausführung fortsetzt:

Bei Voyager sieht das etwa wie folgt aus:

```
public class Display
      implements java.io.Serializable
{
    public void display(String message )  Einstiegspunkt auf System B
    { System.out.println( message); }
}
public class AgentClient {
    public static void main( String[] args )
   {
      Display agent1 = Proxy.of( new
                  Display() );
```

```
      Agent.of( agent1 ).moveTo(     Ausstiegspunkt auf System A
        "//host:8000", "display",
        new Object[]{
      „show on remote");
}
```

Starke Migration

Durch starke Migration (beispielsweise in AgentTCL umgesetzt) kann ein Prozess an exakt der gleichen Stelle (z.B. innerhalb einer Schleife) auf dem Zielsystem rekonstruiert werden, an der er sich bei seiner Ausführung zuletzt befand.

```
      String Hosts[] = { „Host1“, ... „Hostn“ };
      for  (int i=0; i < Hosts.length(); i++) {
            moveTo( Hosts[i] );  Ausstiegspunkt auf Host 1
                                 Einstiegspunkt auf Host 2
      }
```

Die Form der starken Migration ist insbesondere durch interpretierende Sprachen aber nur schwer umsetzbar, da hierbei auf den Framestack des Betriebssystems zugegriffen werden muss um schließlich u.a. den Programm-Counter auszulesen.

Jedoch lässt sich durch die vorangegangene Betrachtung der Möglichkeiten der schwachen Migration hier auch Fragen, ob starke Migration wirklich benötigt wird.

Push-Pull Migration

Ein anderer Aspekt der Migration stellt die Verfahrensweise da, in dem diese durchgeführt wird.

Bei der **Push-Migratio**n (eingesetzt z.B. in IBM Aglets), die am geläufigsten ist, wird der Agentenkontext mit sämtlichen benötigten Klassen auf einmal an das Zielsystem übertragen.

Die **Pull-Migration** (eingesetzt z.B. in Voyager 2.0), sieht dagegen zunächst nur die Übertragung des Kontextes vor. Das Zielsystem fordert dann je nach Bedarf, einzelne Klassen wiederum beim ersten System an.

Die Vorteile die für die zweiten Variante sprechen, sind insbesondere bei Agenten mit zahlreichen Klassen denkbar. Auf der anderen Seite erzeugt diese wohlmöglich unnötigen Netzwerk-Traffic, vor allem bei Agenten mit nur wenigen Klassen.

Je nach Anwendung bietet sich somit die eine – oder andere Möglichkeit ein.

Migration unter Java

Um ein Objekt in Java serialisieren zu können braucht lediglich das Interface *java.io.Serializable* eingebunden zu werden. Dieses enthält zwar zwei Methoden (*writeObject()* und *readObject()*), die auch überschrieben werden können, etwa um die Serialiserung anzupassen, was jedoch nicht unbedingt erforderlich ist; bereits die bloße Einbindung dieses Interfaces reicht hierfür aus, damit die Virtuelle Maschine (JVM) weis, dass dieses Objekt serialisiert werden muss.

Java unterstützt die beiden Formen der schwachen Migration, indem alle Klassen-Attribute (im objekt- und statischen Kontext) berücksichtigt werden. Sollen einzelne Variablen von der Serialisierung ausgenommen werden, können diese über den Bezeichner *transistent* entsprechend gekennzeichnet werden

Beispiel:

```
class MyAgent extends java.io.Serializable {
      int myvar1; // wird serialisiert
      static int myvar2; // wird serialisiert
      transistent int myvar3; // wird nicht serialsiert
      transistent static int myvar4; // wird nicht serialisiert
```

Variablen innerhalb von Methoden, sind hier von der Serialisierung ausgenommen, schließlich sind diese nur sinnvoll würde auch der Ausführungszustandes serialisiert werden können, was dann für starke Migration erforderlich wäre., von Java jedoch nicht direkt unterstützt wird, was durch die Architektur Java's bestimmt und durchaus auch begründbar ist.

Zwei bekannte Möglichkeiten existieren hier jedoch um starke Migration indirekt zu realisieren:

1. durch Anpassung der JVM (vgl.: CIA)
   => Nachteil: Inkompatibilität zu anderen Systemen
2. durch Anpassung von Sourcecode (vgl Frühstöcken 98)
   => Nachteil: sehr komplex; Vergrößerung des Sourcecodes

Kommunikation

Unter Kommunikation soll in diesem Kontext, die Möglichkeiten verstanden werden, unter denen die einzelnen Komponenten eines Agentensysteme miteinander in Kontakt aufnehmen und Daten austauschen.

Hierbei sind die folgenden Kommunikations-Beziehungen relevant:

- zwischen Agenten (zur Kooperation)
- zwischen Agentensystemen (z.B. Agentenlokalisierung)
- zwischen Agent und Agentensystem (zur Auftragerfüllung)

Da der letzte Punkt in den einzelnen Systemen insbesondere durch eigene Interfaces realisiert ist, soll dieser hier nicht näher betrachtet werden.

Zwischen Agenten

Grundsätzlich lässt sich jedoch in Bezug auf Mobile Agenten die folgende Unterscheidung davon treffen, wie zwei Agenten miteinander kommunizieren:

1. im Sinne Stationäre Agenten

   Dabei bedienen sich diese verschiedener Kommunikationsmechanismen, die das Agentensystem zur Verfügung stellt (z.B. RPC/RMI).

2. im Sinne Mobiler Agenten

   Dass bedeutet, dass der Agent auf das System des Kommunikationspartners (Kommunikator) migriert und mit diesem dort lokal kommuniziert.

   Für beide Fälle, also für lokale wie entfernte Kommunikation, existieren wenigstens drei Konzepte, die je nach Agentensystem, in unterschiedlichem Umfang und Ausführungen vorzufinden sind:

   1. direkte benannte Kommunikation

      Dem Kommunikator (Agent als Nachrichtensender) ist der Standort des Kommunikanten (Agent als Nachrichtenempfängers) bekannt. Somit schickt direkt an seine Adresse: *Agent1 -> Agent2@host/context...*

2. indirekte benannte Kommunikation

Der Standort des Kommunikanten muss hierbei dem Kommunikator nicht bekannt sein. Stattdessen bedient sich dieser einer vermittelnden Stelle (sog. **Broker**), über die eine Nachricht indirekt an den Kommunikanten ausgeliefert wird.
Ist ein Stellvertreter mit einem bestimmten Agenten direkt assoziierbar so wird dieser als dessen **Mailbox** bezeichnet. Eine solche stellt empfangene Nachrichten entweder direkt an den Agenten aus (reaktives Konzept) oder wartet darauf, dass der Agent diese abfragt (aktives Konzept).

Nicht geleerte Mailboxen können dem Agentenkontext zugeführt werden, wenn dieser auf ein anderes System migrieren will, oder als lokale Stellvertreter (als sog. **Proxy-Objekte**[11]) auf einem Agentensystem verbleiben um die Erreichbarkeit des Agenten sicherzustellen.

3. indirekte anonyme Kommunikation

Prinzipiell entspricht diese dem vorangegangenem Konzept, nur dass hierbei der Stellvertreter keinen Rückschluss auf einen bestimmten Agenten ermöglicht. Man spricht hier von sog. **Blackboards** („Schwarzen Brettern"), die dem Prinzip von IRC ähneln:

Zwei Agenten braucht dabei nur eine gemeinsame **Queue** (z.B. „Queue1421 auf Host3) bekannt sein, an die sie nun Ihre Nachrichten senden. Ein Agentensystem braucht die benötigte Queue mit dem entsprechenden Namen erst anzulegen, wenn es die erste Nachricht an eine diese erhält. Eine hohe Dynamik ist dadurch sichergestellt.

Agenten Lokalisierung (Agent Tracking)

Hierunter sind die Techniken zu verstehen, die dazu dienen den Standort eines bestimmten Agenten zu ermitteln, was natürlich elementar notwendig für eine Kommunikation mit diesem ist.

Zwei Methoden sind hierbei zu unterscheiden:
Broadcast, bei dem ein Dienst alle Agentensysteme eines Netzwerk (oder alle ihm bekannten) befragt ob sich auf diesen der gesuchte Agent befindet. Bei einer Vielzahl potenzieller Aufenthaltsorte, ist eine derartige Vorgehensweise natürlich höchst ineffizient.
Etwas umfangreicher zu implementieren aber deutlich effektiver ist eine **zentrale Registrierung**, bei der sich Agenten vieler Systeme an- bzw. wieder abmelden. Ein Dienst braucht deshalb lediglich die Registrierung zu befragen und erhält dadurch die aktuelle Adresse des gesuchten Agenten.

Auch ein **dezentralisierter Registrierungsdienst** ist hierbei vorstellbar, bei dem ein Agentensystem etwa die Verantwortung für ein bestimmtes Netzwerk besitzt. Verlässt ein Agent nun dieses, so hält es das Zielsystem des Agenten gespeichert, um entsprechende Suchanfragen an dieses zukünftig weiterzuleiten.

Und noch ein Vorteil besitzt ein solcher Dienst: er erlaubt eine zentrale Kontrolle einzelner Agenten, wodurch sich diese beispielsweise terminieren lassen, sollten diese einmal nicht mehr zu ihren Erschaffern zurückfinden. Solche Agenten werden auch **Weisen** genannten.

Sicherheit

Aufgrund der hohen Relevanz, die dieses Thema gerade für Mobile Agenten Systeme darstellt würde eine vollständige Behandlung hier sicherlich den Rahmen dieser Arbeit sprengen. Deshalb soll hier nur auf einige relevante Aspekte eingegangen werden. Für eine umfassende Betrachtung sei hier auf [Hohl2001] und [Jansen1999] verwiesen.

[11] Proxy-Objekte werden in der Literatur oft auch als Möglichkeit beschrieben mit der eine zusätzliche Sicherheitsschicht, z.B. in Form eines *Access Control Wrappers*, vom Agenten zwischen Agent und Agentensystem gezogen wird.

Sicherheit für Agentensysteme

Agentensysteme müssen sich vornehmlich vor anderen Agenten selbst – sogenannter **bösartigen Agenten** – schützen.

Schon die Wahl der Programmiersprache, kann für den Entwickler eines solchen Systems bereits insofern relevant sein, wenn diese über ein eigenes Sicherheitskonzept verfügt. Java beispielsweise integriert mit ihrer virtuellen Maschine und dem „Sandbox"-Konzept bereits einige mächtige Sicherheitsfunktionen.

Dennoch reicht ein solches sprachspezifisches Konzept natürlich nicht aus, um Sicherheitsrichtlinien auch für die Anwendungsschicht vorzugeben.

Hierzu ist das Agentensystem selbst aufgefordert, eine entsprechende Sicherheitsarchitektur bereitzustellen.

Beispiel: Voyager

Voyager beinhaltet einen *AccessControlWrapper*. Durch diesen lassen sich verschiedene Agenten-Operationen (z.B. create(), migrate(), listenOnPort(), clone()) für unbekannte (*foreign agents*) und bekannte (*known agents*) Agenten festlegen. Um einen Agenten dem Voyager-System bekannt zu geben, muss dessen Signatur beim jeweiligen System eingetragen sein.

Sicherheit für Mobile Agenten

Mobile Agenten zu schützen ist weitaus komplizierter, als es etwa bei einem stationäres Agentensystem der Fall ist. Schließlich migriert dieser autonom zwischen diversen (potenziell unbekannten) Systemen, auf die der Erstellter in den vielen Fällen keinen Zugriff haben wird.

Folglich ergeben sich für Agenten zumindestens die drei folgenden Gefahrenquellen:

Transport

Hierunter fallen vorwiegend sogenannte „Man-In-The-Middle"-Angriffe, bei denen ein Dritter die Netzwerkübertragung abhört und ggf. auch manipuliert.

Viele Agentensysteme bieten hier bereits die Möglichkeit zur Verschlüsselung der Übertragung durch SSL. Weiterhin ist eine Identifikation von Agentensystemen durch installierte Signaturen oder etwa durch eine Zertifizierungsstelle denkbar.

Bösartige Agenten

Bestimmte Agenten könnten versuchen andere Agenten zu manipulieren, oder deren Informationen auszuspähen. Hierfür ist es erforderlich, dass das Agentensystem eine entsprechende Sicherheitsarchitektur bietet (z.B. getrennte Adressräume/Threadgruppen) und der Agent nur so viele sensible Informationen wie nötig speichert.

Java bietet hier – wie bereits gesehen – hierzu beispielsweise die Möglichkeit ausgewählte Informationen (z.B. Passwörter) explizit von der Serialisierung explizit auszuschließen:

```
transistent string Passwort
```

Eine weitere Gefahr, die von anderen Agenten ausgehen kann, besteht darin, dass diese die Identität eines anderen Agenten vorgeben. Dies lässt sich insbesondere durch Signaturen vermeiden, die natürlich vom entsprechenden Agentensystem unterstützt werden müssen.

Bösartige Agentensysteme

Die weit größte Gefahr für Mobile Agenten geht von Agentensystemen selbst aus. Das Problem wurde bereits angesprochen: der Agent – ein Stück Software mit potenziell sensiblen Informationen – bewegt sich auf fremden Systemen, auf die der Absender meist keinen Zugriff e hat. Wie kann er hier sichergehen, dass ein

Agentensystem nicht auf geschützte Daten des Agenten zugriff erhält, oder etwa die Abfragen des Agenten manipuliert ?

Für den letzten Fall, wäre ein Szenario vorstellbar, in dem das Agentensystem eines Reiseanbieters, die Preisabfrage des Agenten in der Form manipuliert, so dass dieser fälschlicherweise genau den betreffenden Anbieter an den Absender als Günstigsten ausweisen würde.

Fazit

Gerade der letzte Punkt macht deutlich, wie problematisch immer noch die Sicherheitslage Mobiler Agenten ist. Wer würde freiwillig einen Agenten auf ein fremdes System versenden, wenn er nicht sicher gehen kann, dass dieses den Agenten nicht manipulieren oder ausspähen wird ?

Sollen Agenten nicht nur in einem geschlossenen System, sondern gar internetweit eingesetzt werden, so ist eine Zertifizierungsstelle notwendig. Diese hätte nicht nur die Aufgabe, Authentitäten zu bestätigen, sondern eben auch einzelne Agentensysteme zu testen und nur wirklich sicheren Systemen ein entsprechendes Zertifikat auszustellen. Ob dies jedoch überhaupt umzusetzen ist, bleibt fraglich.

III. Standardisierung

Ein anderer Punkt um Agentenaktivität auch zwischen heterogenen Systemen zu ermöglichen stellen die verschiedenen Standardisierungsversuche dar. Zahlreiche Agentensystem-Typen existieren, doch deren Möglichkeit untereinander zu interoperieren ist nur begrenzt, meistens sogar gänzlich ausgeschlossen.

Die beiden entscheidenden technischen Aspekte von Interoperabilität stellen dabei die Methodik zur Kommunikation und zum Transport dar.

FIPA

Die *Foundation of Intelligent Phisical Agents* (FIPA)[12] ist eine nicht-gewinnorientierte Organisation die von z.Z. 24 Unternehmen gebildet wird und die sich seit 1997 mit der Standardisierung von Software Agenten beschäftigt.

Grundsätzlich hat die FIPA neben der weitmächtigeren OMG, nur insofern Bedeutung, als dass sie einen Bereich betrachtet, den die OMG komplett auslässt: die Kommunikation zwischen Agenten. Hierzu definiert sie eine eigene *Agenten Kommunikations Sprache* (FIPA-ACL).

FIPA-ACL

Die Agentensprache der FIPA basiert grundsätzlich auf den Konzepten von KQML (eine andere Form von ACL), betrachtet diese jedoch in weit allgemeinerer Form und erlaubt so z.B. die Einbindung verschiedener Formen zur Wissensrepräsentationen.

Wichtige Bestandteile der FIPA-ACL sind:

Ontologien Legen den Bezug eines Begriffs eindeutig fest. So kann der Begriff „Kohl“ sowohl in den Ontologien Agrarwirtschaft, wie Politik vorkommen.

Sprechakte Innerhalb der FIPA-ACL sind 24 Grammatiken definiert, die einen bestimmten Kommunikationsverlauf festlegen. So legt beispielsweise der Sprechakt „Information“ eindeutig fest, das der Absender eine Frage stellt und von dem Empfänger darauf eine bestimmte Antwort erwartet. Sicherlich relativ trivial, dennoch aber unbedingt erforderlich.

[12] www.fipa.org

Wissens-repräsentations-sprachen	Die Darstellung von Informationen (Wissen), wie also z.B. mitgeteilt wird, dass es gerade in Hamburg regnet, wird über sogenannte Wissensrepräsentationssprachen dargestellt. FIPA unterstützt hier gleich einen ganzen Satz solcher Sprachen (z.B. KIF,,FIPA-SL,CCL), die dies in unterschiedlicher Syntax tun.

Beispiel FIPA-ACL Sprechakt:

Bob möchte von Scott wissen, wie das Wetter bei ihm ist ist. Scott antwortet, dass es bei ihm regnet:

```
(inform
       :sender (agent-identifier :name Scott)
       :receiver (set (agent-identifier :name Bob))
       :content („weather(today,raining)”)
       :language Prolog
       :ontology weather)
```

inform	Der Sprechakt *inform* definiert Anfrage/Antwort-Schema
content	Wissensrepräsentation (hier in FIPA-SL)
ontology	Kontext in dem diese Anfrage steht (hier: *wetter*)

MASIF

Vom OMG[13] Entwickelt, dass mit seinen über 800 beteiligten Unternehmen auch. für Standards wie Corba oder UML verantwortlich ist, besitzt die *Mobile Agent System Interoperability Facilities Spezifikation* (z.B. KIF), eine weit höhere Bedeutung und Durchsetzungsfähigkeit als die Vorschläge der FIPA.

Jedoch bezieht sich MASIF hauptsächlich auf das Agenten-Management und vernachlässigt den Bereich Kommunikation vollständig, so dass sich beide Standards hier ergänzen lassen.

Wesentliche Bestandteile von MASIF sind:
- Terminologie einer Agenteninfrastruktur
- Namensgebung
- Lokalisierung von Agenten und Agentensystemen
- Agententransfer
- Agentenmanegement

MASIF Terminologie

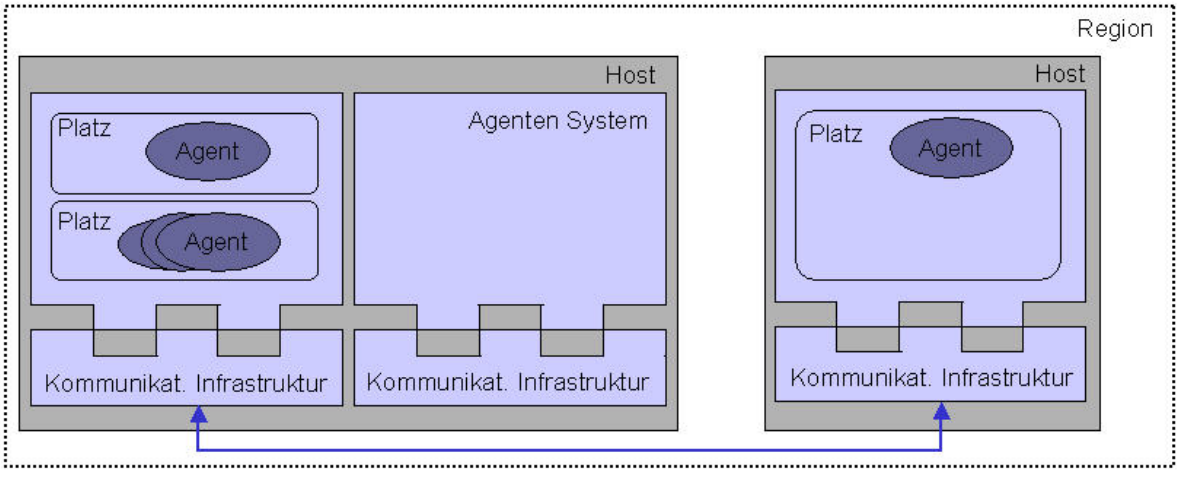


Abb. 3.1: MASIF Terminologie (basierend auf [MAF])

[13] Object Management Group (OMG), unter www.omg.org

Autoritäten (authorities)	Besitzer, der für Agent und Agentensystem existieren muss
Plätze (places)	Bezeichnet den Kontext, in dem sich einzelne Agenten befinden. Mehrere Agenten können sich innerhalb eines Platzes aufhalten von denen wiederum mehrere innerhalb eines Agentensystems existieren können. Agenten werden über Plätzen dann etwa bestimmte Rechte und Ressourcen zugewiesen.
Agentensystem	wie besprochen, jedoch mit der Anmerkung, dass auch mehrere Systeme auf einem Host bestehen können.
Regionen	Mehrere Agentensysteme der gleichen Autorität
Agenten Profile	Definiert die (Migrations-) Kompatibilität zweier Agentensysteme. Maßgebliche Parameter hierfür sind Agentensystemtyp + Serialisierungsmethode
Kommunikations-Infrastruktur	Bezeichnet den Kommunikationskanal (Schnittstellen+ Protokolle) über den zwei Agentensysteme miteinander kommunizieren
Location	Dient zur konkreten Adressierung eines Agenten. Bestandteile hierfür sind Agentensystem + Platz + Agenten-Identifikation

MASIF Architektur

Interoperabilität stellt ein MASIF-konformes System maßgeblich durch den Einsatz von Corba sicher – welches ebenfalls von der OMG standardisiert wurde.

Hintergrund Corba

Corba entspricht grundsätzlich der *Remote Evaluation Architektur* (vgl. Kapitel 1). Entgegen anderer Architekturen, ist Corba unabhängig von Plattform- und Implementierung einsetzbar. So wird beispielsweise ermöglicht, dass ein C++-Programm unter Windows über Corba, Java-Methoden unter Unix aufrufen kann.

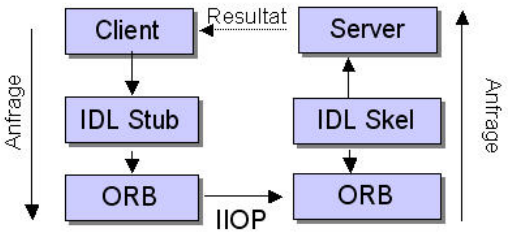


Abb. 3.2: Corba Architektur

Gewährleistet wird dies durch drei Bestandteile:

ORB[14]	eine sprachspezifische Implementierung
IIOP[15]	über das zwei verschiedene ORBs miteinander kommunizieren können
IDL [16]	sprachunabhängige Schnittstellen, über die entsprechende Methoden aufgerufen und sprachspezifische Coderahmen erstellt werden können

Die Integrierung von Corba in MASIF findet auf Anwendungssicht durch zwei IDL-Schnittstellen statt:

MAFAgentSystem	wird vom korrekten Agentensystem eingebunden und stellt entsprechende Transport-Funktionalitäten, u.a. zum Erstellen und Versenden von Agenten bereit
MAFFinder	definiert eine Form von Registrierungsdienst, über den sich Agenten registrieren, abmelden und von externen Diensten lokalisiert werden können. Dabei muss nicht jedes Agentensystem ein solchen Dienst anbieten, MASIF sieht hier einen pro Region vor

[14] Object Request Broker (ORB)
[15] Internet Inter-ORB Protocol (IIOP)
[16] Interface Defination Language (IDL)

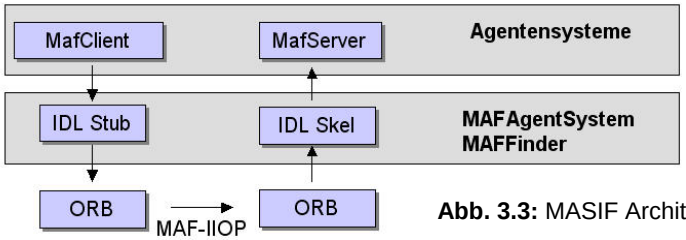


Abb. 3.3: MASIF Architektur

Der konkrete Migrationsablauf, der hierdurch entsteht lässt sich wie folgt beschreiben:

1) Agent A stellt Migrationsanfrage an sein Agenten-System (A)
2) System A stellt Migrationsanfrage (Profil, Klassenliste) an System B
3) Wenn System B Profil unterstützt:
   - (Stoppen von Agent A durch System A)
   - Abrufen der Agenten-Klassen über System A
4) (Starten von Agent A auf System B)
5) Umregistrierung des Agenten bei MAFFinder 1+2

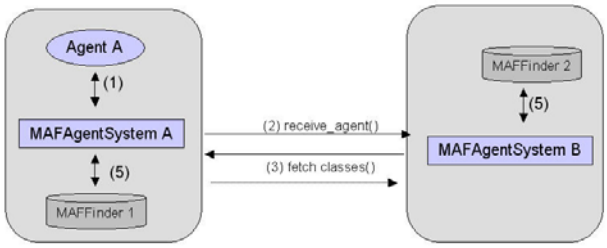


Abb. 3.4: Migrationsablauf (nach MASIF)

Grenzen bestehender Standards

Agenten

Definiert MAFO noch relativ genau, die Funktionsweise eines Agentensystems und wie etwa ein Agententransport auszusehen hat, so schweigt es sich doch vollständig darüber aus, wie nun konkret beispielsweise ein Voyager-Agent auf einem Aglet-System migrieren und ausgeführt werden kann.

Eine hierfür nötige Agenten-Spezifikation fehlt nämlich gänzlich. Eine Gruppe der Universität München [vgl Kemtper2001] hat beim Versuch ein MAFO-konformes Agentensystem zu entwickeln dies schließlich durch agentensystem-spezifische Gateways (am Beispiel Voyager) umgangen [vgl. MAF]. Eine praktizierbare Lösung steht hier somit immer noch aus.

Sicherheit

Standards regeln ebenfalls nicht, wie die Sicherheit von Agenten und Agenten–Systemen sichergestellt werden kann. Möglich wäre dies etwa, könnten ein Agent vom Agentensystem bestimmte Sicherheitskriterien zugesichert bekommen (und umgekehrt). Eine vertrauenswürdige Zertifizierungsstelle müsse in einem solchen Fall zudem für die Glaubwürdigkeit des Systems oder eines Agenten garantieren können.

Anwendung

Zur Erstellung eines Mobilen Agenten, etwa für eine Datenbank-Recherche, muss weiterhin festgelegt sein, welche Funktionen dieser auf einem fremden System dazu aufrufen kann (z.B. *findAuthor()* oder *findAllWords()*). Entsprechende anwendungsspezifische Schnittstellen werden hier benötigt.

Fazit und Ausblick

Wurde die Forschung um Mobile Agenten noch bis vor wenigen Jahren voller Enthusiasmus betrieben, so hat sich in letzter Zeit doch vielfach Ernüchterung breit gemacht.

Die vielen offenen Fragen – insbesondere im Bezug auf die Sicherheit Mobiler Agenten – lassen deren Einsatz in anonymen Netzen (wie dem Internet) bislang noch in weite ferne gleiten.

Auch ist bislang noch kein unschlagbares Anwendung (Killer-Applikation) erdacht worden, die eine entsprechende Anstrengung in diesen Bereichen ökonomisch rechtfertigen würde.

Dennoch ist der Einsatz Mobiler Agenten innerhalb geschlossener Systeme, in Netzen also, wo sich Agentenplattformen eindeutig vorgeben und sich deren Sicherheit garantieren lässt, durchaus denkbar.

So ist die Entwicklung Mobiler Agenten inzwischen stark in Richtung mobiler Endgeräte (vgl. Kapitel 1) gegleitet (vgl. Enago unter www.grasshopper.de), wo sich ein entsprechend geschlossenes System vorfinden lässt und auch sinnvolle Anwendungen vorstellbar sind.

Literatur

[Wooldridge2002] Wooldridge, M., *An Introduction to MultiAgent Systems*, Wiley 2002, ISBN: 0-471-49691-X

[Wooldridge1996] Wooldrige, M und Jennings, N., *Intelligent agents: Theory and practice*. In Knowledge Engineering Review *10*, Jan. 1996.

[MAF] Joint Submission, *Mobile Agent Facility Specification*, OMG, 1997, unter www.omg.org/technology/documents/formal/mobile_agent_facility.htm

[OMG] OMG Agent Technology Green Paper, unter www.objs.com/agent/index.html

[FIPA] FIPA Spezifikationen, 1997-2000, unter www.fipa.org

[Braun2000] Braun, P. und Erfurth und C., Rossak, W., *Von Verteilten Systemen*, Net.Objects Days 2000, unter XXX

[Milic2002] Milic, S. , Diplomarbeit *Sichere Kommunikation zwischen Mobilen Agenten*, Johann Wolfgang Goethe-Universität Frankfurt, 10.2002

[Hohl2001] Hohl. F.. *Sicherheit in Mobile-Agenten-Systemen*. PhD thesis, Fakultät für Informatik der Universität Stuttgart, April 2001.

[Cabri1997] Cabri, G. und Leonardi und L., Zambonelli, F., *Coordination in Mobile Agent Applications*, Universität Modena, Oktober 1997

[Labrou1999] Labrou, Y. und Finin und T., Peng, Y., *The Interoperability Problem: Bringing together Mobile Agents and Agent Communication Languages*, In Procceedings of the 32nd Hawai International Conference on System Sciences, 1999

[Shoeman2003] Schoeman, M. und Cloete, E., *Architectural guidelines for Mobile Agent Systems*, University South Africa, März 2003

[Collier1998] Collier, M., *Mobile Agents and Active Netowrks: Complementrary or Competing Technologies?*, School of Electronic Engineering, Dublin City University, Ireland, 1998

[Lange1999] Lange, D., *Seven Good Reasons For Mobile Agents*, Communications of the ACM, , Vol. 42, No.3, März 1999

[Birell1984] Birell, A. D. und Nelson, B. J.: *Implementierung Remote Procedure Calls*, ACM Transactions on Computer Systems, Vol.2, 1984, pp. 39 – 59

[Steinecke2003] Steinecke, K. *Die FIPA, Agenten & Verteilte Anwendungen*, 2003

[White1997] White, J.: *A Common Agent Platform*, unter: http://www.genmagic.com/internet/cap/w3c-paper.htm

[Jansen1999] Jansen, W. und Karygiannis, T., *Mobile Agent Security*, NIST Special Publication 800-19, Nation Insitute of Standards and Technology, unter http://csrc.nist.gov/publications/nistpubs/800-19/sp800-19.pdf

[Kempter2001] Kempter, B. und Reiser, M. und Rölle, H., *Implementierung eines MASIF konformen Agentensystems, Die Mobile Agent System Architecture (MASA)*, Munich Network Management Tream, Technische Universität München