Aktuelle Entwicklungsumgebungen für mobile Agenten und Multiagentensysteme

Mathias Petsch

1 Einleitung

Seit den Anfangszeiten der theoretischen Erforschung von mobilen Agenten und Multiagentensystemen (MAS) hat es immer wieder Versuche gegeben, die in der Theorie erzielten Erkenntnisse in Entwicklungen und Anwendungen für Agentensysteme umzusetzen. Spielten zunächst Programmiersprachen wie *Lisp* (*Ver-Flex-BB* [Alba92]), *C/C++* (*MAGSY* [Fisc93]), *Smalltalk* (*Agent Factory* [O'Har98]) und *TCL/TK* (*D'Agents* [Gray95]) eine entscheidende Rolle bei der Erstellung agentenbasierter Anwendungen, hat die Entwicklung der Programmiersprache *Java* zu einer starken Zunahme und Verbreitung agentenbasierter Softwareapplikationen geführt. Insbesondere auf dem Gebiet der Entwicklungsumgebungen und Frameworks spielen Anwendungen, die auf anderen Sprachen (wie *C++, Smalltalk*) beruhen, heute kaum noch eine Rolle. Java besitzt für die Entwicklung agentenbasierter Systeme vor allem deshalb Vorteile, da dort wichtige Konzepte zur Erstellung verteilter Anwendungen zur Verfügung gestellt werden, wie integrierte Netzwerk- und Datenbankunterstützung, *RMI* (Remote Method Invocation – Remote Procedure Call unter Java), *JINI* (Java Intelligent Network Infrastructure; mittels *JINI* soll es möglich sein, Hard- bzw. Software ohne spezifische Treiber oder Administration in ein Netzwerk zu integrieren) und eine durchgängige Ausrichtung auf Objektorientierung. Da *Java* zudem einfach zu erlernen und zu erweitern ist, wird agentenbasierte Software heute meist auf deren Basis entwickelt.

In diesem Beitrag werden Entwicklungsumgebungen vorgestellt, welche durchgängig unter Verwendung von *Java* die Erstellung agentenbasierter Anwendungen unterstützen. Dabei werden, ausgehend vom Grundstock der *Java*-Bibliotheken, häufig Konzepte und Klassen verwendet, die den Entwurf von Agentensystemen mit den jeweilig spezifischen Eigenschaften (Kommunikation/Kooperation) erlauben. Oft existieren auch spezielle *Server* bzw. *Agencies*, so genannte Laufzeitumgebungen, in denen die Agenten operieren.

Zu beobachten ist, dass der Entwurf agentenbasierter Systeme und Entwicklungsumgebungen nicht mehr allein Gegenstand akademischer Forschung ist. Zunehmend widmen sich auch kommerzielle Unternehmen der Entwicklung agentenbasierter Software, u. a. IBM (*Aglets* [Agle00]), Toshiba (*Plangent* [Plan00]), British Telecommunication Labs (*Zeus* [Zeus00]), Alcatel (*Live Agent* [Live00]) und Mitsubishi Electric (*Concordia* [Conc00]). So stammen derzeit ca. 30 Entwicklungsumgebungen aus dem kommerziellen und ca. 40 aus dem akademischen Bereich, wobei die Zahl der zur Verfügung stehenden Produkte zum gegenwärtigen Zeitpunkt immer noch stark anwächst. Die Entwicklung mobiler Agentensysteme stellt hierbei einen besonderen Schwerpunkt dar. Cirka ein Drittel der verfügbaren Entwicklungsumgebungen stellt Hilfsmittel zur Verfügung, die den Entwurf von mobilen Agentensystemen erlauben.

Im Folgenden werden einige ausgewählte Umgebungen mit ihren jeweiligen Fähigkeiten kurz beschrieben und analysiert. Insbesondere werden dabei Betrachtungen zu den Entwicklern, der aktuellen Version und Verfügbarkeit des Systems angestellt. Im Weiteren wird die Architektur mit den jeweiligen Komponenten (*Places, Agencies*) und ihren Besonderheiten (Art der Migration, Persistenz) untersucht. Im weiteren Verlauf werden die zur Verfügung stehenden Arten der Kommunikation, unterstützte Protokolle sowie eventuell bestehende Besonderheiten in der Kommunikation betrachtet. Außerdem wird untersucht, ob im jeweiligen Fall Klassen oder Komponenten existieren, die ein kooperatives Arbeiten unterstützen, d. h. ob spezielle Objekte oder Algorithmen innerhalb der Architektur bestehen, die die Planung, Koordination und Kontrolle des kooperativen Arbeitens unterstützen. Im Abschnitt Sicherheit wird schließlich untersucht, ob und gegebenenfalls welche Sicherheitskonzepte bzw. -architekturen in den jeweiligen Entwicklungsumgebungen existieren. Dabei liegt der Schwerpunkt der Betrachtung auf den Arten der Bedrohung eines Systems (i. d. R. des Hostrechners, auf dem die Agenten zur Ausführung kommen) und der darauf operierenden Agenten (Agent bedroht Host, Host bedroht Agent, Agent bedroht Agent). Des Weiteren wird untersucht, inwieweit die Kommunikation der Agenten unter-

Dipl.-Wirt.-Inf. Mathias Petsch,
Technische Universität Ilmenau,
Fakultät für Wirtschaftswissenschaften,
Institut für Wirtschaftsinformatik,
Postfach 100 565, D-98684 Ilmenau,
E-Mail: mathias.petsch@wirtschaft.tu-ilmenau.de

einander und mit anderen Komponenten geschützt ist.

2 Entwicklungsumgebungen für Mobile Agentensysteme

Mobile Agentensysteme sind dadurch charakterisiert, dass die Agenten in der Lage sind, zwischen den in einem Netzwerk angeschlossenen Hosts zu migrieren. Die Agenten können sich also von Rechner zu Rechner bewegen und unter Zugriff auf die lokalen Ressourcen die ihnen gestellten Aufgaben jeweils vor Ort verrichten. Dabei lässt sich zwischen schwacher Migration, bei der der Agent im Anschluss an den Migrationsvorgang seinen internen Zustand verliert, und starker Migration, bei der der Agent nach der Migration seinen internen Zustand behält, unterscheiden.

Mobile Agenten besitzen derzeit noch keine modularen Architekturen und können nur einfache Funktionen realisieren. Problemlösendes Verhalten dagegen bleibt intelligenten Agenten vorbehalten. Es ist allerdings gleichwohl möglich, mobile Agenten in ein Multiagentensystem zu integrieren.

2.1 Aglets

2.1.1 Überblick

Die von IBM Japan entwickelte *Aglets Workbench* (AWB) [Agle00] stellt eine Entwicklungsumgebung für mobile Agenten dar und liegt derzeit in der Version *Aglet Software Development Kit* (ASDK) 1.1 Beta 3 vor. Diese ist allerdings nicht mit Java 2 kompatibel. Der Source Code ist offen gelegt, um den *Open Source* Gedanken zu unterstützen, der eine ständige Erweiterung und Verbesserung des Systems durch die Entwicklungsunterstützung der Benutzer ermöglicht.

Der Entwickler des Werkzeuges Danny Lange gab als einen der Gründe für die Entwicklung von Aglets an [Venn97]: „To me the aglet was an answer to some 15-years-old question: What comes after object-oriented programming? … aglets allow me to think in terms of the network computer“.

Das Wort *Aglet* ist ein Kunstwort gebildet aus den Begriffen Agent und *Applet*. *Aglets* lassen sich als mobile *Applets* charakterisieren. Der angestrebte Einsatzbereich der Technologie liegt in der Unterstützung von Electronic Commerce Anwendungen [LaOs00]. Mit dem Referenzsystem *Tabican* hat IBM die Eignung des Systems in einem agentenbasierten Markt für Übernachtungen und Zimmerbuchungen nachgewiesen [LaOs00].

2.1.2 Architektur/Konzept

Die AWB (*Aglets Workbench)* besteht aus dem *Development Kit*, durch welches die Entwicklung der Agenten unterstützt wird und einer Ausführungsumgebung, die als Plattform für die Ausführung der Funktionalitäten der Agenten dient. Die Entwicklungsumgebung basiert auf dem *Aglet Object Model*, in dem zwischen drei grundlegenden Elementen unterschieden wird: *Aglets* (Java Objekte, die autonom agieren und zwischen speziellen Hostrechnern migrieren können), *Context* (Arbeitsumgebung für *Aglets*; es können gleichzeitig auf einem Rechner mehrere *Context* parallel gestartet werden) und *Messages*. Der Zugriff auf die Funktionalität der Agenten erfolgt über *Proxies*. Dadurch ist der Zugriff Dritter auf den Agenten und ein transparenter Datenzugriff gegeben.

Durch das *AWB* werden für die Entwicklung von *Aglets* bestimmte Entwurfsmuster vorgegeben, die die weitere Gestaltung agentenbasierter Anwendungen unterstützen [ArLa00].

Die grafische Benutzeroberfläche (*Tahiti*) repräsentiert den jeweiligen *Context* und dient der Erzeugung, Steuerung und Kontrolle der *Aglets*. Die Migration des Agenten kann entweder dadurch erfolgen, dass der Agent selbstständig entscheidet, ob er auf einen anderen *Context* migriert (*dispatch*), oder ob der Agent durch eine externe Anweisung zur Migration aufgefordert wird (*retract*). Im Anschluss an die Migration kann der Agent seine Abarbeitung nicht exakt an dem Punkt fortsetzen, an dem die Ausführung durch den Migrationsvorgang unterbrochen wurde. Statt dessen erzeugt die Laufzeitumgebung der Zielplattform ein Ereignis (*Exception*) und der Agent setzt die Verarbeitung in der entsprechenden Behandlungsroutine des Ereignisses fort. Der Zustand eines Agenten kann durch die Methode *deactivate* persistent gespeichert und durch *activate* reaktiviert werden.

2.1.3 Kommunikation/Kooperation

Kooperatives Verhalten wird durch die *Aglets*-Entwicklungsumgebung nicht explizit unterstützt. Die Kommunikation der *Aglets* erfolgt durch *Messages*, die als Objekte repräsentiert sind. Dabei kann die Kommunikation sowohl synchron als auch asynchron erfolgen. *Aglets* können durch einen *Multicast* Aufruf jedem Objekt Nachrichten übermitteln, die dieser in seinem individuellen Context interpretiert.

2.1.4 Sicherheitskonzept

Agenten im System bekommen einen eindeutigen Bezeichner zugewiesen, der sich während der gesamten Laufzeit nicht ändert. Das integrierte Sicherheitsmodell unterstützt vor allem Maßnahmen zum Schutz der Host-Rechner gegen Angriffe von Agenten und gegen Angriffe von Agenten auf Agenten. Die Sicherheitsrichtlinien werden dabei in Form von Regeln aufgestellt und in einer *Policy*-Datenbank hinterlegt. In dieser werden u. a. die Konditionen festgelegt, unter welchen ein Agent Zugriff auf bestimmte Objekte hat. Weiterhin ist festgeschrieben, welche Authentifizierung die Benutzer bzw. der in ihrem Auftrag arbeitenden Agenten für die jeweiligen Aktionen benötigen. Außerdem ist der Grad an Sicherheit der Kommunikation (z. B. Verschlüsselung), der zwischen den jeweiligen *Aglets* und ihrem *Context* benötigt wird [KaLO97, 71], in der *Policy*-Datenbank gespeichert. Darüber hinaus kann der Besitzer eines Agenten für diesen gewisse Sicherheitspräferenzen hinterlegen, die durch den jeweiligen *Context* zu beachten sind. Zum Beispiel kann bestimmt werden, zu welchen Rechnern die Migration des Agenten und ob dessen Klonen erlaubt ist.

2.1.5 Würdigung

Die Java *Aglets* gehörten zu den ersten Java basierten Entwicklungsumgebungen für mobile Agentensysteme. Durch eine fast zweijährige Pause in der Weiterentwicklung des Systems sind leider auch aktuelle Entwicklungen (z. B. Inkompatibilität zu Java 2) nicht berücksichtigt. Seit August 2000 wird auf Basis des *Open Source* Ansatzes versucht, diese Lücke zu füllen.

Vorteile des *Aglet* Systems bestehen in dem zur Verfügung stehenden Sicherheits-

modell und der grafischen Benutzerführung. Durch die nur schwache Migration der Agenten (der interne Zustand des Agenten bleibt im Verlauf der Migration nicht erhalten) liegt der Einsatzbereich der Agenten bei Tätigkeiten, die fest definiert und wiederholt durchgeführt werden müssen, d. h. auf intelligentes Arbeiten (und Lernen im Sinne der Künstlichen Intelligenz), Informationssammlung und -bearbeitung ist die *Aglet* Entwicklungsumgebung nicht ausgerichtet. Kooperatives Problemlösen wird durch das System nicht explizit unterstützt, kann jedoch durch Erweiterung der *Aglet*-Klassen realisiert werden.

2.2 Concordia

2.2.1 Überblick

Die Entwicklungsumgebung *Concordia* [Conc00] für mobile Agenten stammt aus dem Mitsubishi Electric Information Technology Center America und liegt aktuell in der Version 1.1.7 vor. Die Software ist als *Evaluation Kit* auf den Webseiten der Entwickler erhältlich. Ausgelegt ist *Concordia* insbesondere auf die Suche von Informationen in Datenbanken, die über das Internet verteilt sind.

Die Entwickler von *Concordia* unterscheiden in ihrer Definition von Agenten zwischen intelligenten und mobilen Agenten [Wong97, 87]. Unter intelligenten Agenten werden Agenten verstanden, die stationär und mit Intelligenz ausgestattet sowie zu problemlösendem Verhalten befähigt sind, um spezifische Aufgaben zu erfüllen. Mobile Agenten dagegen realisieren einfache Funktionen und können sich durch Migration in einem Netzwerk bewegen, problemlösendes Verhalten und wissensbasiertes Schlussfolgern jedoch ist nicht möglich.

2.2.2 Architektur/Konzept

Der *Concordia Server* ist, wie in Bild 1 dargestellt, der zentrale Bestandteil des Systems, in dem neben der Steuerung der Lebenszyklen der Agenten auch die Laufzeitumgebung für diese bereitgestellt wird. Im Server sind interagierende Komponenten integriert (*Event Manager, Persistence Manager, Queue Manager, Directory Manager, Security Manager, Agent Manager*), auf die der Agent über eine so genannte *Service Bridge* Zugriff auf die zur Verfügung gestellten Dienste und Anwendungen (auch außerhalb des Systems) hat.

Jeder Agent innerhalb des *Concordia* Systems verfügt über eine Liste (*Itinery Object*) von Migrationspunkten, in der die jeweilig zu erfüllenden Aufgaben vermerkt sind. Der Agent kann diese Liste während der Laufzeit verändern. Der Transfer des Agenten erfolgt durch den *Queue Manager*, der den Agenten solange in einer Warteschlange hält, bis der Zielhost verfügbar ist. Der innere Zustand des Agenten bleibt nach der Migration erhalten, d. h. alle bislang gespeicherten Daten des Agenten migrieren gemeinsam mit dem Agenten zwischen den Hosts. Der Agent wird vor dem Versenden und nach der Ankunft automatisch durch den *Persistence* Manager gespeichert. Damit wird ein möglicher Verlust des Agenten, z. B. durch einen Netzwerkfehler, vermieden.

2.2.3 Kommunikation/Kooperation

Concordia nutzt zur Kommunikation zwei unterschiedliche Vorgehensweisen. Bei verteilten asynchronen Ereignissen erhalten die Agenten entsprechende Ereignisse zugesandt. Die asynchronen verteilten Ereignisse lassen sich in gruppenorientierte und Einzelereignisse, die einer vorherigen Anmeldung bedürfen, unterscheiden. In beiden Fällen hinterlässt der Agent eine Referenz auf ein verteiltes Objekt, an das die Ereignisse weitergeleitet und von dem aus die Nachrichten abgerufen werden können.

Eine weitere Möglichkeit der Kommunikation besteht in der Kollaboration. Dafür stellt *Concordia* eine Schnittstelle für die synchrone kooperative Arbeit zur Verfügung, welche die Grundlage für das so genannte *Cooperative Information Gathering (CIG)* [Cast98] bildet. Dabei nutzen die Agenten einen Synchronisationspunkt, um ihre Arbeitsergebnisse zu hinterlegen. Diese werden durch eine anwendungsspezifisch zu erstellende Softwaremethode analysiert und ausgewertet. Auch die notwendige Synchronisation muss individuell entwickelt werden. Erst wenn alle Gruppenmitglieder ihre Ergebnisse am Synchronisationspunkt (*Collaboration Point*) hinterlegt und durch eine *arrive*-Methode bestätigt haben, wird der Kollaborationsvorgang gestartet. Dieser Vorgang wird als *Strong Collaboration Modell* bezeichnet [Wong97, 95]. Im W*eak Collaboration Modell* darf der Agent hingegen die Gruppe verlassen. Die durch den Agenten erzielten Ergebnisse werden im Gesamtergebnis dann jedoch nicht berücksichtigt.

2.2.4 Sicherheitskonzept

In der frei zugänglichen Version (*Evaluation Kit*) sind die Zuverlässigkeits- und Sicherheitsmodule sowie einige Administrationsfähigkeiten nicht enthalten. Hingegen

Kernpunkte für das Management

Im Beitrag werden die Plattformen IBM Aglets, Grasshopper und Concordia als Entwicklungsumgebung für mobile Agentensysteme und FIPA OS und Zeus als Plattformen für die Entwicklung von Multiagentensystemen untersucht. Der Schwerpunkt der Betrachtung liegt dabei auf den folgenden Punkten:

- Überblick (Entwickler, Version, Verfügbarkeit),
- Architektur/Konzept (Aufbau),
- Kommunikation/Kooperation (Art und Weise der Kommunikation, Möglichkeiten der Kooperation),
- Sicherheitskonzept (welches Sicherheitskonzept ist vorhanden und was wird geschützt),
- Zusammenfassende Würdigung.

Stichworte: Mobile Agenten, Multiagentensysteme, Entwicklungsumgebungen, Java

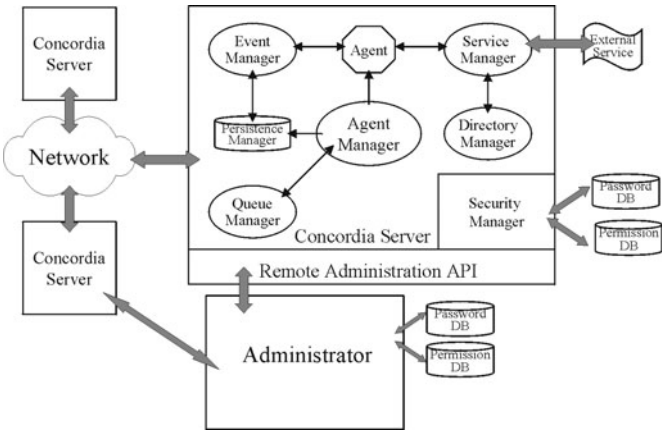

Bild 1 Concordia Architektur [WaPW98]

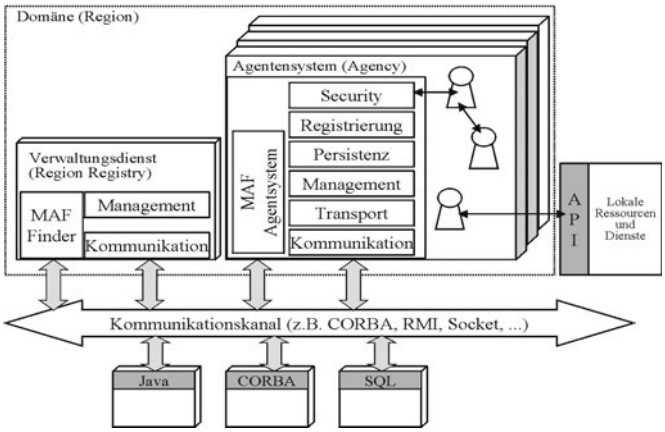

Bild 2 Grasshopper Architektur [Gras00b]

verfügt die Version für den kommerziellen Einsatz (*Full Feature Set*) über einige Sicherheitskonzepte.

In Abhängigkeit von den Rechten des Anwenders steuert *Concordia* mittels *Security Manager* die Zugriffe auf die Ressourcen des jeweiligen Systems. Der Schutz der Agenten während der Migration wird durch Einsatz des *SSL (Secure Socket Layer Protokoll)* realisiert. Die persistent gespeicherten Daten (gespeicherte Zustände der Agenten) werden durch einen symmetrischen Schlüssel codiert. So werden Angriffe des Agenten auf den Host und der Agenten untereinander verhindert, jedoch wird die Möglichkeit des Angriffs eines Hosts auf den Agenten im bisherigen Sicherheitskonzept nicht berücksichtigt.

2.2.5 Würdigung

Concordia ist ein System zur Erstellung mobiler Agenten, welches, im Gegensatz zu *Aglets* und *Grasshopper,* kooperatives Handeln durch Synchronisation von Ergebnissen in der Agentenumgebung unterstützt.

Im Unterschied zu den anderen hier vorgestellten Entwicklungsumgebungen für mobile Agentensysteme verfügt *Concordia* über die Möglichkeit, Zugriffsrechte abhängig vom Nutzer des Agenten und der Anwendung zu vergeben.

2.3 Grasshopper

2.3.1 Überblick

Grasshopper [Gras00a], entwickelt von GMD Focus und IKV++ GmbH, wird derzeit in der Version 2.2 vertrieben. Die Entwicklungsumgebung steht auf den Webseiten zum Download bereit, nur für den professionellen Einsatz sind Lizenzgebühren zu entrichten. Das System wurde konform zum *OMG-MASIF* Standard (OMG = Object Management Group; MASIF = Mobile Agent System Interoperability Facility) entwickelt, der als Zielsetzung die Interoperabilität verschiedener mobiler Agentensysteme hat [Breu99, 326]. Weiterhin wird seit dem Release 2.1 ein Add-On zur Kompatibilität zum *FIPA* Standard (*Foundation for Intelligent Physical Agents*) angeboten [Mage00, 16].

Referenzanwendungen mit *Grasshopper* wurden u. a. im *MARINE* Projekt (*Mobile Agent Environments in Intelligent Networks*) [Mari00] und *Camelon*-Projekt (Mobile Agenten im UMTS) [Came00] realisiert.

2.3.2 Architektur/Konzept

In Bild 2 sind die Grundkomponenten des *Grasshopper DAE (Distributed Agent Environment)* abgebildet, die aus Regionen, Agentensystemen und Infrastrukturdiensten (*Region Registry*) bestehen. Im System wird zwischen stationären und mobilen Agenten unterschieden, die in ihren Ausführungsumgebungen (*Agencies*) agieren. Diese *Agencies* bestehen aus einem Kernsystem und einem oder mehreren Plätzen, die nach funktionalen Aspekten (logische Bündelung von Aufgaben bzw. Fähigkeiten) zusammengefasst werden. Die Kontrolle und Steuerung der Agenten erfolgt über ein grafisches Benutzerinterface, der *Agency Console*, das benutzerspezifisch angepasst werden kann [HöMQ99, 7]. Die Migration der Agenten wird durch den Transportdienst unterstützt, wobei der Agent seine Aufgabenabarbeitung exakt an der Stelle fortsetzt, an der er vor der Migration unterbrochen wurde. Ein Persistenzdienst speichert periodisch oder nach explizitem Aufruf (*Explicit Persistence*) den aktuellen Zustand der Daten eines Agenten, die später durch die Bildung einer neuen Instanz des Agenten wieder aktiviert werden können. Durch einen Mechanismus (*Implicit Persistence*) können darüber hinaus alle gestarteten Plätze des Systems im Hintergrund gespeichert werden.

2.3.3 Kommunikation/Kooperation

Die Kommunikation in *Grasshopper* wird durch verschiedene Protokolle unterstützt. Üblicherweise erfolgt die Kommunikation über eine Socketverbindung, jedoch werden darüber hinaus *RMI*, das *IIOP (Internet Inter-ORB Protocol)* und eine Spezialisierung des *CORBA IIOP* für agentenbasierte Interaktion, basierend auf dem *MASIF*-Standard, *MAF IIOP* unterstützt. Der Nachrichtenaustausch erfolgt über einen *Communication Service* des Kernsystems. Alternativ kann eine zum OMG *MASIF*-Standard kompatible *CORBA*-Schnittstelle für Fernzugriffe verwendet werden [Breu99, 331]. Der Nachrichtenaustausch erfolgt innerhalb und zwischen Regionen synchron, asynchron oder über *Multicast*-Aufrufe.

Zur Unterstützung der Kooperation können in *Grasshopper* Gruppen gebildet

werden, jedoch existieren keine expliziten Verfahren zur Koordination und Synchronisation.

2.3.4 Sicherheitskonzept

Das *Grasshopper* Sicherheitskonzept unterscheidet zwischen interner und externer Sicherheit. Die externe Sicherheit, der Schutz von *Remote* Verbindungen zwischen Komponenten des Systems, wird durch den *X.509* Standard und das *SSL*-Protokoll realisiert. Das interne Sicherheitskonzept sichert den Schutz von Ressourcen der Ausführungsumgebungen gegenüber dem unerlaubten Zugriff von Agenten sowie den Schutz der Agenten untereinander. Dazu werden an die Agenten in Abhängigkeit von ihrer Herkunft und ihren Aufgaben Rechte vergeben. Angelehnt ist das Konzept an das *Policy*-Prinzip von *JDK 1.2* (Java Development Kid – Entwicklungsumgebung für Java von SUN Microsystems [Sun00]). Die Vergabe von Rechten erfolgt einzeln oder an Gruppen von Agenten. *Policy*-Objekte überwachen zur Laufzeit die Zugriffe von Agenten auf die jeweils angeforderten Ressourcen und erlauben deren Freigabe entsprechend der gewährten Rechte [Breu99, 335].

2.3.5 Würdigung

Grasshopper bildet eine flexible Entwicklungsumgebung für mobile Agentensysteme, die durch die Unterstützung der *MASIF* und *FIPA* Standards (nur für stationäre Agenten) eine Interoperabilität mit anderen Multiagentensystemen und *Legacy Systems* unterstützt. Durch die Bildung von Gruppen kann kooperatives Arbeiten unterstützt werden.

Im Fall einer Migration eines Agenten unter *Grasshopper* wird der interne Zustand des Agenten beibehalten. Durch ein Sicherheitskonzept, welches eine sichere Kommunikation (durch Verschlüsselung) und eine Vergabe von Rechten beinhaltet, wird ein sicheres Arbeiten der mobilen Agenten unterstützt.

3 Entwicklungsumgebungen für Multi-Agenten Systeme

Multiagentensysteme sind dadurch charakterisiert, dass die Agenten im System über Autonomie und Intelligenz verfügen und sich untereinander kooperativ verhalten. Multiagentensysteme sind „… concerned with coordinating intelligent behavior among a collection of (possibly pre-existing) autonomous intelligent ‚agents' how they can coordinate their knowledge, goals, skills, and plans jointly to take action or to solve problems. The agents in a Multiagent system may be working toward a single global goal, or toward separate individual goals that interact" [BoGa88, 3].

3.1 FIPA OS

3.1.1 Überblick

FIPA-OS [Fipa00b] ist eine Entwicklung von Nortelnetworks und steht in der aktuellen Version 1.3.2 zum Download zur Verfügung. Bei *FIPA-OS* handelt es sich um ein *Open Source* Produkt. Das Tool benötigt als Grundlage zur Entwicklung *JDK 1.2*, einen WWW-Server (zur Interoperabilität mit anderen *Fipa-OS* Plattformen) und einen *ORB* (*Object Request Broker*). Sind alle diese Komponenten vorhanden (Web-Server optional), kann mittels der zur Verfügung stehenden Java-Klassen ein Multiagentensystem entworfen werden, welches kompatibel zum FIPA97 Standard (in Teilen auch zu FIPA98 und FIPA99) ist.

3.1.2 Architektur/Konzept

FIPA-OS verwendet beim Entwurf eines MAS das *FIPA* Referenz Modell (Bild 3).

Dieses besteht aus dem *DF* (*Directory Facilitator)*, der als eine Art „gelbe Seiten" des Systems agiert, indem die zur Verfügung stehenden Dienste der Agenten des Systems hinterlegt werden. Ein weiterer Bestandteil ist das *AMS* (*Agent Management System)*, dass die Funktion des zentralen Managements im Agentensystem erfüllt. Dabei kann vom *AMS* das Verhalten der Agenten kontrolliert und gesteuert, der Agenten *Lifecycle* überwacht und zentrale Administrationsdienste durchgeführt werden. Zur Kommunikation im System werden der *MTS* (*Message Transport Service)* und der *ACC (Agent Communication Channel)* verwendet. Dabei kann der Nachrichtenaustausch zwischen Komponenten innerhalb und zu Plattformen außerhalb des Systems erfolgen. Diese Komponenten werden zu einer *AP* (*Agent Platform*) zusammengefasst [Nort00].

FIPA-OS stellt eine Entwicklungsumgebung für MAS dar, welche die grundlegenden Bestandteile zum Entwurf eines Agenten und dessen Kommunikation mit externen und internen Komponenten enthält. Komplexere Methoden zur Sicherheit, Kooperation und Koordination bzw. zum „intelligenten Verhalten" (Planen und Lernen) stehen nicht zur Verfügung. Durch die Offenheit des Codes und der verwendeten Programmiersprache *Java* kann der Entwickler jedoch die Eigenschaften des Systems eigenständig erweitern.

3.1.3 Kommunikation/Kooperation

Die Kommunikation der Agenten im *FIPA-OS* untereinander und mit Komponenten außerhalb des MAS erfolgt über den *MTS*. Dieser unterscheidet zwischen *Internal* und *External MTP* (*Message Transport Protocol*). Zum Austausch der Nachrichten wird der *ACC* verwendet, der die Nachrichten entsprechend der *FIPA Agent Message Transport Service Specification* [Fipa00a] versendet. Die Semantik/Syntax der Nachrichten ist in *FIPA ACL* (*Agent Communication Language*) definiert.

Die Kommunikation mit externen Agentenplattformen erfolgt mittels *ACC* über einen Web Server. Dabei wird die *MTP* Adresse über den der Plattform bekannten Web Server an alle mit diesem Server verbundenen Plattformen versendet.

Zur Kooperation und Koordination stellt *FIPA-OS* keine expliziten Verfahren (wie Planen, Verhandeln) zur Verfügung. Koordinatives Verhalten kann nur durch Implementierung eigener Algorithmen erreicht werden.

3.1.4 Sicherheitskonzept

Für die *Agent Platform* in *FIPA-OS* besteht kein Sicherheitsmodell. Es können lediglich Klassen des *JDK* für Verfahren der Verschlüsselung und Sicherheit verwendet werden. Bislang existieren keine Pläne, das von der *FIPA* vorgeschlagene Sicherheitsmodell (Erweiterung der Architektur um einen zentralen *Agent Platform Security Manager* [Fipa98]) in *FIPA-OS* zu integrieren.

Mechanismen, Architekturen oder Konzepte zur Erweiterung der Sicherheit innerhalb der Plattform von *FIPA-OS* be-

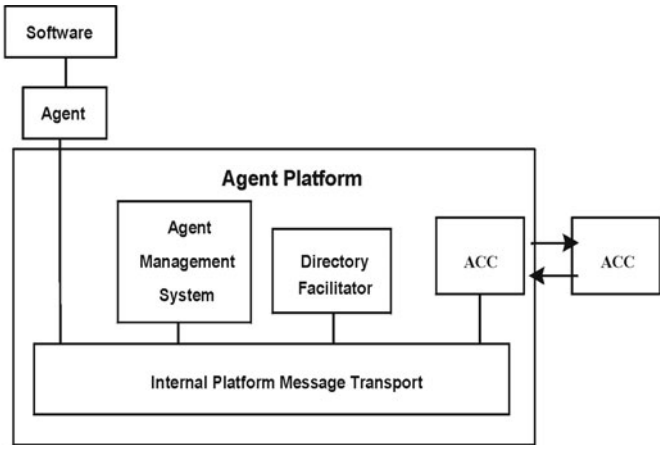

Bild 3 FIPA Referenzmodell [Fipa00c]

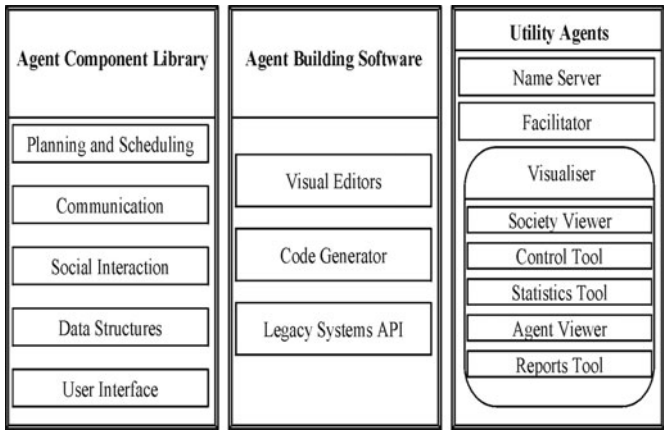

Bild 4 Komponenten des Zeus Agent Building Toolkit [Nwana et al. 99, 135]

dürfen der anwendungsspezifischen Entwicklung.

3.1.5 Würdigung

FIPA-OS ist eine Entwicklungsumgebung für Multiagentensysteme, die den *FIPA* Standard unterstützt. Die Umsetzung des *FIPA* Standards in eine Entwicklungsumgebung hat den entscheidenden Vorteil, dass dadurch die Interoperabilität zwischen bestehenden MAS-Plattformen unterstützt wird, nachteilig ist allerdings der zentralistische Ansatz des *FIPA*-Referenzmodells (zentrale Kommunikation und zentrales Management). Auch mangelt es an einem Sicherheitskonzept. Explizite Methoden zur Kooperation und Koordination werden nicht unterstützt.

3.2 Zeus

3.2.1 Überblick

Das Multiagentensystem-Toolkit *Zeus* [Zeus00] ist eine Entwicklung der British Telecommunication Labs. Momentan steht die Version 1.04 zum Download zur Verfügung. Beim Zeus *Agent Building Toolkit* handelt es sich um *Open Source* Software. Es existieren zwei Versionen. Eine Version setzt *JDK 1.3* voraus und eine zweite arbeitet mit den älteren Versionen von Java zusammen. Zielstellung von Zeus ist die Unterstützung der Entwicklung von „… deliberative and goal-directed multi-agent systems for task-oriented domains such as service provisioning, resource/process management, and supply chain management …“ [Ndum99, 6]. Beispielhaft wurden bislang u. a. einfache Anwendungen für einen elektronischen Markt (*Fruit Market*) und eine Simulation einer einfachen *Supply Chain* einer PC Herstellung realisiert.

3.2.2 Architektur/Konzept

Das *Zeus Agent Building Toolkit* besteht aus verschiedenen Komponenten (Bild 4), die die Entwicklung eines MAS unterstützen [Coll98. 61]. Die *Agent Component Library* beinhaltet eine Sammlung von Klassen und Methoden, die zur Bildung der Agenten und deren Kommunikation notwendig sind. So sind dort unter anderem Klassen zur Planung und dem Scheduling, der sozialen Interaktion, der Kommunikation, den Datenstrukturen und zur Bildung von User Interfaces hinterlegt. Die Komponente *Agent Building Software* enthält Module, die die visuelle Entwicklung des MAS unterstützen, den Code zur Ausführung generieren, und eine API, die die Anbindung an bestehende Systeme (*Legacy Systems*) erlaubt. Die Komponente *Utility Agents* enthält den *Name Server*, den *Facilitator* (über den *Facilitator* können Agenten mit den jeweilig benötigten Eigenschaften identifiziert und über den *Name Server* ermittelt werden) und Tools zum Monitoring des Agentenverhaltens. Weiterhin erfolgt durch die *Utility Agents* das Management, die Administration und die Erstellung von Statistiken über das Verhalten der Agenten im MAS.

Bei der Entwicklung eines MAS wird zunächst das Agenten Modell mit seinem Verhalten in verschiedenen Stufen (*definition layer, organisation layer, co-ordination layer*) definiert. Dabei wird dem Agenten das interne Verhalten, das Verhältnis zu anderen Agenten und das Koordinations- und Verhandlungsverhalten implementiert. Hierzu wird der *ZEUS Agent Generator* verwendet, der den visuellen Entwurf der Agenten erlaubt.

Über ein mehrstufiges Verfahren werden die Agenten im MAS entworfen. Zunächst werden mit Hilfe eines *Ontology Editor* und eines *Fact/Variable Editor* die spezifischen Variablen, ihre Beschreibungen und Zustände der Agenten definiert. Anschließend erfolgt im *Agent Definition Editor* die Festlegung der Aufgaben, Ressourcen und Pläne eines Agenten. Weiterhin werden im *Task Description Editor* die globalen und individuellen Aufgaben der Agenten beschrieben und im *Organisation Editor* die interne Organisation und Beziehungen der Agenten im MAS festgelegt. Im *Co-ordination Editor* erfolgt die Auswahl des spezifischen Koordinationsprotokolls. Abschließend wird der Code für die Agenten aus der *Agent Component Library* generiert.

Obwohl für die Entwicklung und den Entwurf der Agenten nur ein Agenten Modell zur Verfügung steht, lässt sich dieses durch spezifische Eingriffe in den zur Verfügung stehenden *Java* Code erweitern.

Die *Zeus Utility Agents* beinhalten den *Nameserver Agent*, der der Protokollierung und Lokalisation von Agenten im System dient, und den *Facilitator Agent,* der Informationen über Fähigkeiten der Agenten im System besitzt. Weiterhin sind der *Visualiser Agent,* der über Fähigkeiten zur Analyse und Kontrolle der Agenten im

Tabelle 1 Überblick der untersuchten Entwicklungsumgebungen

	IBM-AGLETS	CONCORDIA	GRASSHOPPER	FIPA-OS	ZEUS
Version	1.1 Beta 3	1.1.7	2.2	1.3.2	1.04
Open Source	X			X	X
Verfügbarkeit	Uneingeschränkt, per download (Nicht zu Java 2 kompatibel)	Evaluation Kit (eingeschränkt um Sicherheits- und Zuverlässigkeitsmodule)	Uneingeschränkt, per download, für den professionellen Einsatz kostenpflichtig	Uneingeschränkt, per download	Uneingeschränkt, per download
Besonderheiten			MASIF/FIPA kompatibel	FIPA kompatibel	Grafische Entwicklungsumgebung
Mobilität	Schwache Migration	Starke Migration	Starke Migration	–	–
Kommunikation	Synchron/asynchron	Asynchron, Kollaboration	RMI, IIOP, CORBA, MAF IIOP	FIPA ACC (ACL)	Asynchron über TCP/IP Sockets, KQML, FIPA ACL
Kooperation	–	Collaboration Model	Gruppenbildung	–	General Purpose Planning
Sicherheitskonzept	Schutz der Ressourcen (Host), Schutz Agent-Agent, Schutz der Kommunikation	Schutz der Ressourcen (Host), Schutz Agent-Agent, Zugriffsrechte abhängig von den Rechten der Benutzer, Schutz der Kommunikation (SSL)	Schutz der Ressourcen (Host), Schutz Agent-Agent, Schutz der Kommunikation (SSL, X.509)	–	–

System verfügt, und Tools zur Kontrolle und Steuerung des Agentensystems, Bestandteil der *Zeus Utility Agents*. Dadurch ist es zu jedem Zeitpunkt möglich, über Statistik- und Monitoringfunktionen das Verhalten des Systems zu beeinflussen.

3.2.3 Kommunikation/Kooperation

Die Kommunikation erfolgt asynchron über *TCP/IP Sockets*, über die Nachrichten, die auf *KQML* (*Knowledge Query Management Language*) bzw. *FIPA ACL* basieren, versandt werden.

Für die Planung und Koordination der Agenten im MAS ist ein *General Purpose Planning* und *Scheduling* System implementiert, welches auf einem graphbasierten Problemlösungsverhalten beruht. Die Koordination erfolgt über den zentralen *Planner/Scheduler* in der *Co-ordination Engine* des Systems. Entsprechend der gegebenen Zielstellung erfolgt hierbei eine Plandekomposition. Die Vergabe der Aufgaben an die einzelnen Agenten erfolgt über einen Kontraktmechanismus, der entsprechend den „Kosten" und den Fähigkeiten des Agenten die Aufgaben zuteilt. Anschließend werden die Teillösungen von der *Co-ordination Engine* zu einer Gesamtlösung zusammengefasst. Sollte ein Teilplan nicht ausgeführt werden können, wird dieser erneut vergeben. Sollte auch dabei keine Lösung erzielt werden, kann der Plan in seiner Gesamtheit nicht ausgeführt werden.

3.2.4 Sicherheitskonzept

Explizite Sicherheitsmechanismen existieren nicht, da es sich beim Entwurf eines MAS in *Zeus* um ein geschlossenes System handelt, d. h. Agenten aus anderen MAS sind derzeit nicht in der Lage, Teil eines mit *Zeus* gebildeten MAS zu werden. Dadurch sind allerdings auch die Risiken für das System (Host) bzw. die Agenten im System begrenzt. Nicht auszuschließen ist jedoch die Gefährdung des Hosts bzw. die unerlaubte Inanspruchnahme von Ressourcen.

3.2.5 Würdigung

Der große Vorteil der MAS Entwicklungsumgebung *Zeus* liegt in der Möglichkeit, das System mit Hilfe einer grafischen Entwicklungsumgebung zu gestalten. Dabei können mit Hilfe von grafischen Werkzeugen die Agenten mit ihren Kommunikations- und Kooperationsbeziehungen sowie das gesamte System entworfen werden. Allerdings ist auch für die grafische Gestaltung des Systems eine größere Einarbeitungszeit notwendig. Ein Sicherheitskonzept existiert nicht. Weiterhin ist der Entwickler im Wesentlichen auf die zu Grunde liegende *BDI*-Architektur (*Belief-Desire-Intention*-Modell – Agenten verfügen demnach über Annahmen über ihre Umwelt, Aufträge, Wünsche und Absichten, aus denen sich Verpflichtungen für ihr zukünftiges Handeln ergeben. Diese Verpflichtungen binden in der Zukunft Ressourcen.) der Agenten beschränkt.

4 Überblick

In Tabelle 1 soll ein kurzer Überblick über die grundlegenden Eigenschaften der untersuchten Entwicklungsumgebungen gegeben werden. Dabei zeigen sich die teilweise erheblichen Unterschiede in der Unterstützung der Kommunikation, Kooperation und der verwendeten Sicherheitskonzepte. Die Werkzeuge zur Erstellung von mobilen Agenten unterscheiden sich weiterhin bei der Art der Migration.

Insbesondere der Unterschied zwischen den Entwicklungsumgebungen für mobile Agenten und denen für Multiagentensysteme, zeigt sich in der Verwendung der Kommunikationsunterstützungen und -sprachen, sowie der Verwendung von internen Sicherheitskonzepten. Weiterhin ist

der Grad der Bereitstellung von Verfahren zur Kooperation der Agenten stark unterschiedlich ausgeprägt.

5 Ausblick und Zusammenfassung

In den letzten Jahren entstand eine große Zahl von Entwicklungsumgebungen für mobile Agenten und Multiagentensysteme. Vor allem auf dem Gebiet der Entwicklung von mobilen Agenten ist eine Vielzahl neuer Werkzeuge entworfen wurden. Diese lassen sich hinsichtlich ihrer spezifischen Eigenschaften charakterisieren. Mobile Agenten weisen vor allem in der Art der Migration Unterschiede auf. Ein Agent kann während des Migrationsvorgangs seinen internen Zustand behalten oder aber er verliert ihn. Es ist erkennbar, dass bei mobilen Agenten stärker Sicherheitsaspekte berücksichtigt werden als bei MAS. Dies ist bedingt durch Migration der Agenten zwischen verschiedenen Hosts und die daraus resultierenden Gefahren.

Besitzen Java *Aglets*, *Concordia* und *Grasshopper* ausgereifte Sicherheitskonzepte, so wurden bei den MAS Entwicklungsumgebungen *Zeus* und *FIPA-OS* auf solche Architekturen und Ansätze verzichtet. Bei den Kommunikationsbeziehungen der Agenten wird zunehmend versucht, auf bestehende Standards aufzubauen. Dabei spielt insbesondere *Corba* eine entscheidende Rolle.

Die Frage der Sicherheit von Multiagentensystemen über die reine Kommunikationssicherheit hinaus (z. B. Verschlüsselung, Authentifizierung) mit den spezifischen Problemen der resultierenden Gefahren für das System, die durch agentenspezifisches Verhalten auftreten können, wurde jedoch bislang auch in der wissenschaftlichen Literatur nur unzureichend gewürdigt. So sind unter anderem Gefährdungen denkbar, die z. B. aus der „Intelligenz" der Agenten und dem Missbrauch des kooperativen Verhaltens der Agenten in einem MAS resultieren.

FIPA-konforme Systeme werden durch *FIPA-OS* und *Grasshopper* unterstützt. Der Standard *MASIF* für mobile Agentensysteme spielt (abgesehen von *Grasshopper*) hingegen kaum ein Rolle bei bestehenden Entwicklungsumgebungen.

Die untersuchten Systeme unterscheiden sich aber auch hinsichtlich des Umfangs der Unterstützung der Softwareentwicklung. Stellt *FIPA-OS* im Wesentlichen nur ein Framework zur Kommunikation und die dazugehörige Architektur zur Verfügung, so bietet *Zeus* eine geschlossene grafische Entwicklungsumgebung, die Konzepte zur Kooperation, Kommunikation und zum zentralen Planen (die Planung erfolgt ausgehend von einer zentralen Stelle, die für die Plandekomposition und anschließende Generierung einer Gesamtlösung verantwortlich ist) bereitstellt.

Literatur

[Agle00] http://www.trl.ib.co.jp/aglets/. Abruf am 2000-10-06.

[Alba92] *Albayrak, S.:* Kooperative Lösung der Aufgabe Auftragsdurchsetzung in der Fertigung durch ein Mehr-Agenten-System auf der Basis des Blackboard Modelles. Dissertation, Fachbereich der Informatik der TU Berlin, 1992.

[ArLa00] *Aridor, Y.; Lange, D.B.:* Agent Design Patterns: Elements of Agent Application Design. http://www.genmagic.com/asa/danny/Patterns.pdf, Abruf 2000-10-06.

[BoGa88] *Bond, A. H.; Gasser, L.:* Readings in Distributed Artificial Intelligence. Morgan Kaufmann, San Mateo 1988.

[Breu99] *Breugst, M.; Choy, S.; Hagen, L.; Höft, M.; Magedanz, T.:* Grasshopper – An Agent Platform for Mobile Agent-Based Services in Fixed and Mobile Telecommunications Environments. In: *Hayzelden, A. L. G.; Bigham, J.:* Software Agents for Future Communication Systems. Springer Verlag, Berlin 1999, S. 326 – 357.

[Came00] http://www.comnets.rwth-aachen.de/~cameleon. Abruf am 2000-10-06.

[Cast98] *Castillo, A.; Kawaguchi, M.; Paciorek, N.; Wong, D.:* Concordia as Enabling Technology for Cooperative Information Gathering. In: Proceedings of Japanese Society for Artificial Intelligence Conference Tokyo, June 17 – 18, 1998.

[Coll98] *Collis, J.C.; Ndumu, D.T.; Hwana, H.S.; Lee, L.C.:* The Zeus Agent Building Toolkit. In: BT Technology Journal 16 (1998) 3, S. 60 – 68.

[Conc00] http://www.meitca.com/HSL/Projects/ Concordia/. Abruf am 2000-10-06.

[Fipa98] FIPA 98 Specification Part 10, Version 1.0, Agent Security Management. http://www.fipa.org/specs/fipa00020/0C00020.doc. Abruf am 2000-10-06.

[Fipa00a] FIPA Agent Message Transport Service Specification. http://www.fipa.org/specs/fipa00067/XC00067C.doc. Abruf am 2000-10-06.

[Fipa00b] http://www.nortelnetworks.com/products/announcements/fipa/. Abruf am 6. Oktober 2000.

[Fipa00c] FIPA 98 Part 1 Version 1.0: Agent Management Specification. http://www.fipa.org/specs/fipa00002/OC00002.pdf. Abruf am 2000-10-06.

[Fisc93] *Fischer, K.:* Verteiltes und kooperatives Planen in einer flexiblen Fertigungsumgebung (Distributed and Cooperative Planning in the Environment of a Flexible System). DISKI, Dissertationen zur Künstlichen Intelligenz. Infix, 1993.

[Gras00a] http://www.ikv.de/products/grasshopper/. Abruf am 6. Oktober 2000.

[Gras00b] Grasshopper – Eine Plattform für mobile Software-Agenten. http://www.grasshopper.de/download/GHEinfuehrung.pdf. Abruf 2000-10-06.

[Gray95] *Gray, R.S.:* Agent TCL: A transportable agent system. In: Proceedings of the CIKM Workshop on Intelligent Information Agents, Fourth International Conference on Information and Knowledge Management (CIKM 95), Baltimore, Maryland, December, 1995.

[HöMQ99] *Höft, M.; Magedanz, T.; Quantz, J.:* Grasshopper – The Agent platform. In: AgentLink News 2 (1999) 3, S. 6 – 9.

[KaLO97] *Karjoth, G.; Lange, D.B.; Oshima, M.:* A Security Model for Aglets. In: IEEE Internet Computing 1 (1997) 4, 68 – 77.

[LaOs00] *Lange, D.B.; Oshima, M.:* Mobile Agents with Java: The Aglet API. http://www.genmagic. com/asa/danny/Wwwj.pdf, Abruf am 2000- 10-06.

[Live00] http://www.agentsoft.com/. Abruf am 2000-10-06.

[Mage00] *Magedanz, T.:* Grasshopper 2 – The Next Generation Agent Platform. In: AgentLink News 3 (2000) 5, S. 16 – 17.

[Mari00] http://www.itatel.it/drsc/marine/marine.html. Abruf am 2000-10-06.

[Ndum99] *Ndumu, D.; Collis, J.; Owusu, G.; Sullivan, M.; Lee, L.: Zeus:* A Toolkit for Building Distributed Multi-Agent Systems. In: AgentLink News 2 (1999) 2, S. 6 – 9.

[Nort00] FIPA-OS V1.3.2 Distribution Notes. http://download.sourceforge.net/fipa-os/FIPA-Osv1-3-2.pdf. Abruf am 2000-10-06.

[Nwan99] *Nwana, H.C.; Ndumu, D.T.; Lee, L.C.; Collis, J.C.:* ZEUS: A Toolkit for Distribute Multi-Agent Systems. In: Applied Artificial Intelligence Journal 13 (1999) 1, S. 129 – 186.

[O'Har98] *O'Hare, G.M.P.; Collier, R.; Conlon, J.; Abbas, S.:* Agent Factory: An Environment for Constructing and Visualizing Agent Communities. In: 9th Irish Artificial Intelligence and Cognitive Science (AICS) Conference, UCD, 1998.

[Sun00] www.java.sun.com. Abruf am 2000-10-06.

[Plan00] http://www2.toshiba.co.jp/plangent/index.htm. Abruf am 2000-10-06.

[Venn97] *Venners, B.:* Solve real problems with Aglets, a type of mobile agents. In: Javaworld (97) 5, http://www.javaworld.com/jw-05-1997/jw-05-hood.html, Abruf 2000-10-06, ohne Seite.

[WaPW98] *Walsh, T.; Paciorek, N.; Wong, D.:* Security and Reliability in Concordia. In: Proceedings of 31st Annual Hawai,i International Conference on System Sciences (HICSS31), 1998.

[Wong97] *Wong, D.; Paciorek, N.; Walsh, T.; DiCelie, J.; Young, M.; Peet, B.:* Concordia: An Infrastructure for Collaborating Mobile Agents. In: *Rothermel, K. (Hrsg.):* Mobile Agents. First International Workshop, MA 97, Springer Verlag, Berlin 1997, S. 86 – 97.

[Zeus00] http://www.labs.bt.com/projects/agents.htm. Abruf am 2000-10-06.