Universität Stuttgart
Fakultät Informatik

AIDA I - Abschlußbericht

Fritz Hohl,Joachim Baumann,Kurt Rothermel,Markus Schwehm,Markus Straßer

Email: `Fritz.Hohl@informatik.uni-stuttgart.de`

Institut für Parallele und Verteilte
Höchstleistungsrechner (IPVR)
Fakultät Informatik
Universität Stuttgart
Breitwiesenstr. 20 - 22
D-70565 Stuttgart

AIDA I - Abschlußbericht

*Fritz Hohl, Joachim Baumann, Kurt Rothermel,
Markus Schwehm, Markus Straßer*

Bericht Nr. 1998/03
Februar 1998

Kurzfassung

In diesem Bericht geht es um die Zusammenfassung der Erkenntnisse, die im Verlauf der ersten Phase des AIDA-Projektes bis September 1997 gewonnen wurden. AIDA ist ein Projekt, das von der Deutschen Forschungsgemeinschaft (DFG) finanziert wird. Das Thema dieses Projektes sind *mobile Agenten*, also Einheiten, die aus Code, Daten und Zustand bestehen und sich selbständig in einem Netzwerk bewegen können. Das Ziel von AIDA I war es, auf der Grundlage eines allgemeinen Verarbeitungsmodells flexible Systemmechanismen für verteilte, agentenbasierte Systeme zu entwickeln.

1 Einführung

Mobile Agenten sind aktive und autonome Verarbeitungseinheiten, die in der Lage sind, Funktionalität für eine Anwendung zu erbringen und selbständig von Systemknoten zu Systemknoten migrieren können. Mobile Agenten stellen zum einen ein *Programmiermodell* dar, dessen Einheiten, die mobilen Agenten, wie Softwareroboter in einer (künstlichen) Umgebung interagieren können. Diese Interaktion schließt Kommunikation mit anderen Agenten und Rechnern mit ein, ebenso wie Einwirkung auf diese Umwelt, solange diese Einwirkung definiert ist. Mobile Agenten stellen als Technik auf der anderen Seite jedoch auch eine *Middleware* dar, die eine Plattform für Applikationen bietet, und deren Anforderungen in die Kommunikationsaufrufe und andere Dienstleistungen der darunterliegenden Rechnersystemschicht umsetzt. Technisch gesehen schließlich sind mobile Agenten Einheiten aus Programmcode und Zustandsdaten, die es einem Agenten erlauben, Berechnungen, die auf einem Knoten angefangen wurden, auf einem anderen Knoten weiterzuführen.

Mobile Agenten sind immer noch ein junges Gebiet, das allerdings immer mehr Aufmerksamkeit nicht nur von seiten der Forschung, sondern auch von Seiten der Industrie erfährt, wie sich durch die Zunahme der Gruppen, die sich mit diesem Thema beschäftigen, deutlich wird. Dieses Interesse wird nicht nur durch ein Gebiet geweckt, dessen Eigenschaften interessante Forschungsthemen erschließen, sondern vor allem durch die Eigenschaften der Technik der mobilen Agenten, die zumindest für einige Anwendungen wesentliche Vorteile gegenüber der herkömmlichen Client-Server-Technik bietet.

Das AIDA-Projekt versucht, auf der Ebene des Agentensystems Verfahren und Mechanismen zu entwickeln, die es Anwendungen erlauben, die Vorteile mobiler Agenten auszunutzen. Da vor allem zum Zeitpunkt des Beginns von AIDA I kaum Systeme und Modelle in diesem Bereich vorhanden waren, wurde als erster Schritt auf der Basis möglichst einfacher Modelle ein offenes Agentensystem erarbeitet, um damit die Forschung der Systemmechanismen zu ermöglichen.

2 Entwicklung des Gebietes

Die Forschung auf dem Gebiet der mobilen Agenten ist derzeit zum einen geprägt von der sprunghaften Zunahme von Gruppen, die beginnen, auf diesem Gebiet zu arbeiten und damit einhergehend, von neuen Agentensystemen auf Prototypenebene. Diese Vermehrung von gegeneinander inkompatiblen Systemen ist für eine Technik schädlich, die auf einer für viele Anwendungen wichtigen breiten Installationsbasis beruht. Daher gibt es zunehmend Bestrebungen, die Interopabilität der Systeme durch Standardisierungsbemühungen zu ermöglichen. Die zwei wichtigsten Institutionen auf diesem Gebiet sind die Object Management Group, die einen "Call for Proposals" bereits 1995 veröffentlichte, sowie die Foundation for Intelligent Physical Agents (FIPA), deren Vorschläge nicht nur Aspekte mobiler, sondern auch intelligenter Agenten umfassen. Zum anderen ist das Gebiet geprägt von der Suche nach Anwendungen, die diese Technologie auch kommerziell interessant machen, den "Killerapplikationen" für mobile Agenten.

Die Entscheidung, für AIDA, Java als Programmiersprache zu verwenden, hat sich im Nachhinein als sehr vorteilhaft erwiesen, da sich zum einen Java sehr stark verbreitet hat und viele Firmen Java für ihre Rechnerplattformen anbieten und es immer mehr Entwicklungsumgebungen und Werkzeuge für diese Sprache gibt. Zum anderen eignet sich diese Sprache in ihrem Aufbau sehr gut für diese Zwecke, und es gibt zunehmend neue Bibliotheken, die zum Teil in die Sprachspezifikation eingebaut werden und die wichtige Funktionalität wie Cryptoverfahren oder Datenbankanbindung einbringen. Die Hinwendung zu Java spiegelt sich auch in den neuen Agentensystemen wieder, die fast komplett Java verwenden. Dadurch scheint auch die Argumentationslinie für Agentensysteme schwächer zu werden, die einen Mehrsprachenansatz propagieren. Das Problem ist, daß man die Eigenschaften, die Java bieten kann, wie Threads, Sicherheit und dynamisches Laden von Code, nicht von allen anderen Programmiersprachen erwarten kann. Das bedeutet, daß entweder nur "geeignete" Sprachen eingebaut werden können, oder daß sehr massiv in die Implementation der Laufzeitumgebungen eingegriffen werden muß. Zusammen mit dem Problem der Datenkonversion zwischen verschiedenen Programmiersprachen sind Mehrsprachensysteme sehr viel schwerer zu realisieren und bieten daher auch nur wenige Programmiersprachen an. Zudem ist es unter den derzeitigen Gegebenheiten allenfalls unklar, ob mehrere Sprachen auch wirklich gebraucht werden.

3 Aufbau der Forschungsgruppe am IPVR

Beginnend mit dem Start von AIDA I wurde am IPVR eine Gruppe aufgebaut, die sich schwerpunktmäßig mit dem Bereich "Mobile Agenten" beschäftigt. Diese Gruppe besteht z.Zt. aus fünf Forschern und einer Vielzahl an Studenten. Gleichzeitig mit dem Aufbau der Gruppe konnten Forschungskontakte ins In- und Ausland geknüpft werden. So initiierte die IPVR-Gruppe die Bildung des "Deutschen Mobile-Agenten-Treffen" (DeMAT) mit, das sich alle sechs Monate trifft und Forschungsergebnisse austauscht. Von den bisher fünf Treffen der DeMAT wurde eines mitorganisiert, und an allen teilgenommen. Durch die Mitorganisation und Teilnahme an Workshops zum Thema (Mitorganisation des Workshops auf der ECOOP '96 in Linz, der ECOOP'97 in Jväskylä und des Dagstuhlseminars "Mobile Software-Agents" im Oktober 1997, Teilnahme außerdem am W^{3}C/OMG Workshop zum Thema "Mobile Code", an der DAIS'97, sowie der PDPTA'97) konnten zahlreiche Forschungskontakte zu anderen Gruppen geknüpft werden, die sich mit diesem Thema beschäftigen.

Diese Kontakte konnten genutzt werden, um im April 1997 in Berlin den ersten internationalen Workshop zum Thema "Mobile Agents" zu initiieren und mitzuorganisieren. Da von den über 100 Teilnehmer aus der ganzen Welt sehr positive Reaktionen kamen, wurde von Programmkommittee ein Folge-Workshop 1998 in Stuttgart am IPVR beschlossen, den die Forschergruppe organisieren wird.

Neben einer Reihe von Artikeln konnten auch Proceedings der mitveranstalteten Workshops mitherausgebracht werden (MA'97 Proceedings bei Springer, Workshopanteil an den ECOOP'96 und ECOOP'97 Workshop Proceedings).

Entscheidend mitgeprägt wurde die Arbeit der Mitarbeiter in der Forschergruppe durch Studien- und Diplomarbeiter, sowie die Mithilfe der wissenschaflichen Hilfskräfte. Betreut werden konnten etwa 20 Studien- bzw. Diplomarbeiten ([Ahe97a], [Ahe97b], [Bec96], [Bec97], [Joc97], [Kir96], [Kla96], [Kla97], [Kub97], [Kuh97], [Nit97], [Pin96], [Rie96], [Rö96], [Röh97], [Sch96], [Sza97], [Voi96], [Wie97], [Zep96]), deren Ausarbeitungen im WWW elektronisch verfügbar sind.

Schließlich konnte die Arbeit der Forschungsgruppe, nicht zuletzt auch die des AIDA I - Projektes, in mehreren Präsentationen für Gäste des Instituts, an anderen Hochschulen wie der Universität Genf oder in der Industrie (Siemens, Deutsche Bank, Tandem usw.) vorgeführt werden.

Durch die Teilnahme am Stand der Baden-Württembergischen Hochschulen auf der CeBIT

1997 in Hannover konnte das Projekt darüber hinaus in der Öffentlichkeit präsentiert werden. Neben der Wissensvermittlung an ein breites Publikum stand die Kontaktaufnahme mit über 100 zum Teil hochkäratigen Kontakten im Vordergrund.

4 Definition des Verarbeitungsmodells

Mobile Agenten sind vor allem ein Programmiermodell, in dem Begriffe wie Agent, Platz oder Dienst auftaucht. In diesem Kapitel werden daher diese Begriffe erklärt. Nähere Informationen hierzu lassen sich in [SBH96] finden.

4.1 Agenten

Es existiert zur Zeit keine allgemein anerkannte Definition von mobilen Agenten; die meisten Definitionen sind eher Umschreibungen, weil es schwerfällt, über einer gegebenen Definition Systeme auszuschließen, die nicht in diese Kategorisierung fallen (siehe [FG96] für eine Diskussion der Definitionsproblematik). Einen ersten Eindruck mag aber die folgende Beschreibung liefern: Agenten sind die aktiven Verarbeitungseinheiten in einem Mobile-Agenten-System. Wie eine Art "Softwareroboter" sind sie in der Lage, sich autonom von einem Ort des Agentensystems zu einem anderen zu bewegen (falls nichts dazwischenkommt), sie können mit der "Umwelt" an einem Ort interagieren und mit anderen Agenten kommunizieren. Sie arbeiten autonom in dem Sinne, daß sie keine ständige Verbindung zu einem menschlichen Benutzer benötigen.

Eine für die Programmierer sehr wichtige Forderung an die Verarbeitung mittels Agenten ist, daß sie nicht plötzlich verschwinden, etwa weil ein Knoten abgestürzt ist. Es ist daher beim Design des Agentensystems darauf zu achten, daß Agenten im Bezug auf ihre Existenz persistent sind, auch weil sie z.B. wertvolle Daten transportieren (z.B. elektronisches Geld).

Im Mole-Agentenmodell gibt es zwei Arten von Agenten: Benutzeragenten und Systemagenten. Benutzeragenten sind mobil, und zunächst einmal Fremde an jedem Ort. Sie können daher ohne weiteres nicht auf sicherheitssensitive Ressourcen wie Festplatten oder das Netzwerk zugreifen. Systemagenten dagegen haben vollen Zugriff auf alle Systemressourcen (bzw. soviel, wie ihnen die Benutzerkennung, unter dem der Knoten läuft, zugesteht). Ihre Hauptaufgabe ist es daher auch, Systemressourcen des unterliegenden Rechnersystems in die Agentenwelt zu hinein anzubieten. Das kann z.B. bedeuten, daß Agenten in eine Datenbank schreiben bzw. aus ihr lesen können, indem sie mit dem "Datenbank-Gateway-Agenten" kommunizieren. Da Systemagenten systemabhängige Einzelheiten kennen müssen, macht es keinen Sinn, daß sie mobil sind. Da also Systemagenten auch stationäre Agenten sind, ist es einfach, ihnen den vollen Zugang zum System zu gewähren, da sie vom Verwalter eines Ortes installiert werden, also keinen neuen Code enthalten. Wenn Benutzeragenten Zugang zu sicherheitssensitiven Ressourcen haben wollen, müssen sie die Systemagenten davon "überzeugen", ihnen entweder Zugang zu verschaffen oder ihnen Dienstleistungen auf diesen Ressourcen zu erbringen. Ein Beispiel für den ersten Fall könnte ein Datensammelagent eines Netzwerkmanagementsystems sein, der die Daten eines Netzknotens erfahren will und sich daher mittels eines Ausweises beim zuständigen Systemagenten als Berechtigter authentifiziert. Ein Beispiel für den zweiten Fall könnte ein "Speicheragent" sein, ein Systemagent, der Benutzeragenten in einem Abrechnugsverfahren anbietet, ihre Daten zu speichern bzw. bereitzustellen.

4.2 Plätze

Eine der wesentlichen Fähigkeiten eines mobilen Agenten ist die der Migration, also der selbständigen Bewegung zwischen zwei "Orten". Diese Fähigkeit setzt die Existenz von etwas voraus, zu dem ein Agent hinmigrieren kann. Für diese Migrationsziele gibt es im wesentlichen drei Möglichkeiten:

1. abstrakte "Plätze" als Migrationsziel. Da Plätze auf einem Rechner liegen, gibt es keine volle Ortstransparenz, sondern eine Transparenz vom physikalischen Ort.
2. physikalische Einheiten (z.B. Rechner) als Migrationsziel. Es gibt keine Ortstransparenz.

Die zweite Möglichkeit ist eine der Eigenschaften, die von verteilten Systemen zu überwinden versucht wird, ist doch eine der wichtigsten Erkenntnisse hier, daß Ortstransparenz sehr wichtig ist. Diese Möglichkeit, der Einsatz von Plätzen als Migrationsziel, kann folgendermaßen charakterisiert werden:

- ein Platz ist ein Gebiet gemeinsamer Fehler, wenn er auf einem Rechnerknoten installiert ist, d.h. alle Agenten auf diesem Platz sind entweder erreichbar oder nicht
- auf einem Platz kostet die Kommunikation zwischen allen Agenten gleichviel
- die Kommunikation auf einem Platz ist mindestens genauso billig, wie die zwischen Agenten auf verschiedenen Plätzen

Wir verwenden daher die zweite Möglichkeit. Wie in anderen Agentensystemen (z.B. in Telescript) auch, gibt es addressierbare Plätze als Migrationsziele.

Aus technischer Sicht sind Plätze nicht nur Migrationsziele. Sie stellen auch die Ausführungsumgebungen für Agenten, die ja spezielle Programme sind, dar. Sie bestehen in der Implementierung daher meist aus einem oder mehreren Interpretern und den Komponenten, die die Systemmechanismen erbringen. Da Plätze stationär sind, lassen sich einige Eigenschaften zwischen Plätzen messen, z.B. die Netzwerkverzögerung und, über der Zeit, die Übertragungsgeschwindigkeit.

Schaubild 1 verdeutlicht noch einmal die Zusammenhänge zwischen Plätzen und Agenten.

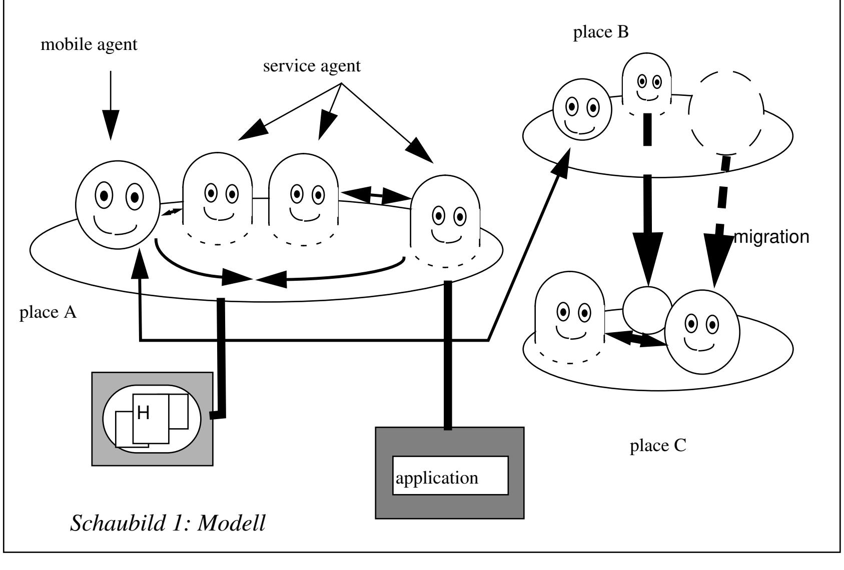


Schaubild 1: Modell

4.3 Dienste

Man muß Dienste in einem Agentenmodell vorsehen, um a priori die Agenten klassifizieren zu können, die für eine Interaktion in Frage kommen und um das Agentensystem als Rahmen für das Anbieten bzw. Inanspruchnehmen von Dienstleistungen zu benutzen zu können. Ein Dienst ist dabei eine Funktionalität, die ein Agent anbietet, und ein anderer Agent in Anspruch nehmen kann. Agenten, die bestimmte Dienste anbieten, können über eine Systemkomponente gefunden werden, die Einzelheiten der Dienstinanspruchnahme ist in der Dienstbeschreibung festgelegt. Ein Agent kann mehrere Dienste anbieten. Details übber dieses Thema lassen sich ebenfalls in [SBH97] finden.

4.4 Agentengruppen

Dadurch, daß Agenten autonome Einheiten sind, die eine größtmögliche Flexibilität bezüglich der Abhängigkeit zu anderen Agenten, bezüglich möglicher Kommunikationspartner und der Kommunikationsart haben, ist es nicht ohne weiteres möglich, die Relationen und Kommunikationsbeziehungen von Agenten herauszufinden, die zur selben Anwendung gehören oder zusammenarbeiten. Um einer Anwendung nicht alle Arbeit zuzumuten, ihre Teilnehmer und deren Kommunikations selbst verwalten zu müssen, wurde das Konzept der *Agentengruppe* eingeführt.

Eine Agentengruppe besteht aus einer Menge von Agenten oder Unteragentengruppen, die zusammen an einer Aufgabe arbeiten. Innerhalb dieser Menge haben einzelne Agenten bestimmte Rollen. Der *Group Initiator* erzeugt die Agentengruppe initial und vergibt die Rollen. Der *Group Receiver of Results* ist die Instanz, die das "Gruppenergebnis" bekommt. Der *Group Administrator* ist für die Kontrolle der Lebenszeit der Gruppe zuständig und damit auch für die Terminierung und eine eventuelle Waisenerkennung. Die *Group Members* sind die Träger der eigentlichen Berechnung; sie unterliegen von ihrer Lebenszeit her der Gruppenkoordination.

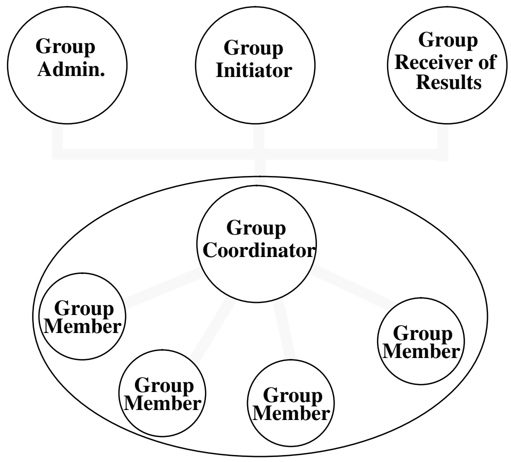
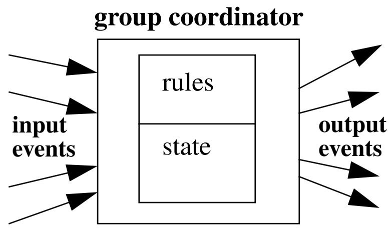


Schaubild 2 Group Model

Der *Group Coordinator* schließlich koordiniert die Gruppe. Dazu werden die Abhängigkeiten der Gruppenmitglieder und des Gruppenziels durch Events und einen Regelsatz modelliert, der auf den Events arbeitet und wiederum Events erzeugt. Eine solche Regel besteht aus einer Bedingung und einem Aktionsteil, der angewandt wird, wenn die Bedingung erfüllt ist. Der Aktionsteil besteht aus einfachen Anweisungen, die z.B. Ausgabe-Events erzeugen, den Zustand innerhalb des Koordinators ändern usw.



Internal Rule: if (collected offers are better than offers held so far) {
send(new offers);
adjust held offers list;
}
External Rule: if (member has completed its search) {
inc (members completed);
if (members completed == no members) then
send (result to receiver of results);
}

Schaubild 3 Group Coordinator

Obwohl der Gruppenkoordinator logisch eine Einheit ist, ist er "physikalisch" verteilt in jedem Group Member implementiert, so daß z.B. Netzwerkpartitionierungen u.U. aufgefangen werden können.

Nähere Informationen zum Gruppenkonzept lassen sich in [BHR97] und [BR97] finden.

5 Agenteninteraktion und -migration

5.1 Agenteninteraktion

5.1.1 Namen

Namenskonzepte sind die Grundlage für die Interaktion von Agenten, Plätzen und Diensten. In diesem Abschnitt sollen diese Konzepte kurz erläutert werden.

Namen von Plätzen

Namen von Plätzen sollen transparent sein bezüglich der Netzwerkadresse des verwendeten Rechnerknotens. Daher wurde ein Verzeichnisdienst benötigt, der in der Lage ist, zum Teil frei wählbare Strings auf (Netzwerkadresse, Portnummer)-Paare abzubilden. Da DNS für diesen Zweck geeignet erschien, und da mit diesem Mechanismus ein erprobter und performanter Dienst zur Verfügung steht, der darüberhinaus in TCP/IP-Netwerken auf jeden Fall vorhanden ist, wurde diese Abbildung über DNS-Einträge vorgenommen. DNS eignet sich für diese Abbildung deshalb, weil die Dynamik der Abbildung sehr ähnlich der Dynamik des eigentlichen Einsatzgebietes erscheint.

Um Plätze addressieren zu können, mussten daher jeweils DNS-Einträge vorgenommen werden. Um die Einträge zur Adressierung benutzen zu können, musste eine eigene DNS-Resolver-Funktion für Java geschrieben werden, da die Portnummer in einem zusätzlichen Attribut (TXT) kodiert ist. Dieses Vorgehen, obwohl funktionierend, ist in der Benutzung zu umständlich, da Betreiber von Plätzen erst den DNS-Adminstrator finden müssen (was vor allem in großen Organisationen problematisch zu sein scheint), und ihn dann bitten müssen, die richtigen

DNS-Einträge zu machen (was aufgrund der Unkenntnis vieler Benutzer von DNS ein Problem ist). Daher wäre bei Verwendung von DNS eigentlich eine Komponente notwendig, die es erlaubt, DNS-Einträge automatisch vornehmen zu lassen, was aber eine sehr plattform-abhängige Installation dieser Komponente beim DNS-Server notwendig machen würde und erfahrungsgemäß von vielen DNS-Administratoren nicht sehr gerne gesehen wird, da DNS eine sicherheitssensitive Systemkomponente ist. Die Alternative zu DNS wäre ein System mit gleicher Funktionalität (eventuell sogar unter Verwendung gleicher Protokolle), das unabhängig vom Netzwerk-DNS ist, und automatische Verwaltung gestattet.

Um zumindest den lokalen Test von Agentenbasierten Anwendungen zu ermöglichen, ohne einen echten DNS-Eintrag zu benötigen, wurde in Mole eine lokale Abbildungstabelle von Platznamen implementiert.

Namen von Agenten

In Mole ist jeder Agent eine eindeutige Einheit, eine Sicht, die durch eine global eindeutige und auf die Lebenszeit des Agenten bezogen gleiche Bezeichner für jeden Agenten unterstützt wird. Die Eindeutigkeit des Namens kann dabei nur dann garantiert werden, wenn das Agentensystem dieses Bezeichners erzeugt. Der Prozeß der Erzeugung neuer Namen ist dabei am einfachste, wenn dies ohne globales Wissen geschehen kann. Tabelle 1 zeigt die Struktur dieser Bezeichner.

Tabelle 1: Die Komponenten der Agenten_Bezeichner

# Bytes	Meaning
4	Dynamic Counter, incremented for every new agent id
4	Crash counter, incremented every time the system is started. Also incremented if dynamic counter overflows.
12	IP Version 6 address of the system on which the engine runs
2	The port number of the engine
2	Reserved for future use (set to 0)

Der Bezeichner teilt sich auf in den Teil, der die namensgebende Einheit kennzeichnet und einen Teil, der die individuelle Nummer des Agenten enthält. Die namensgebende Einheit, die Engine, an der der Agent erzeugt wurde, wird durch die Adresse im IPv6-Format gekennzeichnet sowie durch die Portnummer des Engine-Prozesses. Falls diese Adresse für einen Knoten nicht ermittelt werden kann, weil er keine global eindeutige IP-Adresse hat, dann wird dieser Teil aus einer Zufallszahl gebildet, wobei der mögliche Raum eine Kollision unwahrscheinlich macht. Pro Engine gibt es dann noch einen "Crash Counter", der bei jedem Hochfahren des Engineprozesses erhöht wird und deshalb persistent auf Platte gespeichert werden muß. Der letzte Teil ist ein Zähler, der pro neuem Agentenbezeichner um eins erhöht wird und nicht persistent gehalten werden muß. Sollte der (seltene) Fall eintreten, daß dieser letzte Teil nicht ausreicht, wird der Crash Counter um eins erhöht.

Da aus dem Namen keine Informationen über den Aufenthaltsort des Agenten folgen, und auch keine sonstigen Informationen enthalten sind, bedarf es eines Verzeichnisdienstes, um diese Informationen zu Agentenbezeichnern zu assoziieren.

Ein weiteres Konzept, Agenten, mit denen interagiert werden soll, zu benennen, sind die Badges (Namensschilder). Agenten können sich beliebige Badges, die aus einem einfachen String bestehen, "anheften". Andere Agenten können dann mit einem oder mehreren Agenten, die einen bestimmten Badge haben, interagieren. Detailliertere Informationen hierzu finden sich in [SBH97].

Namen von Diensten

Da der Bereich "Trading" nicht verfolgt werden konnte, wurden Dienste in Mole nur insoweit bereitgestellt, als daß sich Agenten lokal an einem Platz unter einem String, der als Dienstname fungiert, registrieren lassen können. Andere Agenten können dann zu einem String die Liste der registrierten Agenten abfragen.

5.1.2 Kommunikation zwischen Agenten

Das initiale Modell der Kommunikation bestand in der Möglichkeit, an andere Agenten Nachrichten zu schicken, bzw. Prozeduren anderer Agenten aufzurufen.

Nachrichten

Eine Nachricht besteht aus einem beliebigen Objekt, das -wie eine Email- an eine Adresse geschickt wird, die aus einem Agentennamen und einem Platznamen aufgebaut ist. Falls der Empfänger an diesem Ort existent ist, wird die Prozedur `receiveMessage` mit der Nachricht als Parameter in einem eigenen Thread aufgerufen. Falls der Agent nicht anwesend ist, wird -in Abhängigkeit von der anzugebenden Fehlersemantik- die Nachricht entweder einfach gelöscht oder der Absender wird durch eine andere Nachricht von diesem Umstand informiert

Remote Procedure Call (RPC)

Ähnlich wie bei einer Nachricht wird der RPC an eine Kombination aus Agenten- und Platznamen adressiert. Falls der Agent dort vorhanden ist , wird ein Thread erzeugt und über eine Dispatch-Prozedur die eigentliche Prozedur aufgerufen. Das Ergebnis der Prozedur, das ebenso wie die Parameter beliebige Objekte sein können, wird dem Aufrufer als Resultat seines Aufrufs übergeben[1]. Falls der adressierte Agent nicht am spezifizierten Platz vorhanden ist, wird eine Fehlermeldung zurückgegeben.

Dieser Ansatz hatte, obwohl für für einige Anwendungen durchaus ausreichend, zwei wesentliche Probleme: Die Adressierung von Agenten und die Tatsache, daß mit diesen Mechanismen effizient nur zustandslose Interaktion möglich war. Das Problem bei der Adressierung war, daß man im Voraus den Agenten kennen musste, mit dem man kommunizieren wollte. Wollte man nur ein Merkmal des Kommunikationspartners angeben, oder diesen gar nicht spezifizieren wollen, war das nicht möglich. Das Problem bei Interaktionen, die Zustand haben, war, daß es keine Möglichkeit gab, diesen Zustand zu adressieren, es sei denn man hätte den Zustand explizit mitgeschickt, was ab einer gewissen Größe nicht mehr tragbar ist. Nun sind aber gerade zustandsbehaftete Interaktionen (z.B. mit einem Cursor in einer Datenbank) in realen Anwendungen oft notwendig. Um diese Probleme zu beheben, wurden zwei weitere Mechanismen eingeführt: Session-orientierte Kommunikation und anonyme Kommunikation mittels Events.

Session-orientierte Kommunikation

Mit diesem Mechanismus wird für eine bestimmte Zeit eine Interaktionsbeziehung zwischen

1. In Mole mussre die Dispatch-Prozedur von Hand geschrieben werden, da es zum damaligen Zeitpunkt nicht möglich war, Prozeduren generisch, also unter Angabe des Prozedurnamens als String, aufzurufen. Die RPC-Konstruktion musste selbst implementiert werden, da das Java-eigene RPC-Paket, RMI, nicht in der Lage war, Agentenobjekte zu adressieren.

zwei Agenten explizit aufgebaut. Sobald diese Beziehung besteht, kann dann wie bisher mit Nachrichten und RPC kommuniziert werden, aber die Adressierung ist über den Session-Bezeichner möglich und ein Kontext kann von den Partnern aufgebaut werden,

Der Aufbau der Sitzung wird durch zwei neue Kommandos möglich. Mit *PassiveSetUp* erklärt ein Partner seine Bereitschaft zum Aufbau einer Interaktionsbeziehung. Dieses Kommando ist nicht-blockierend und vor allem für Agenten vorgesehen, die einen Dienst nach außen anbieten wollen. Mit *ActiveSetUp* kann ein Agent blockierend versuchen, eine Interaktion aufzubauen. Der jeweilig mögliche Interaktionspartner wird dabei entweder durch die Angabe einer konkreten Agenten-Id spezifiziert oder durch die Angabe eines Badge-Prädikats. Dieses Prädikat erlaubt die (Teil-)Spezifikation der Beschriftung von Badges (Badges sind "Namensschilder", sie wurden kurz im letzten Teilkapitel erläutert). Eine bestehende Interaktionsbeziehung kann unilateral wieder beendet werden.

Anonyme Kommunikation mittels Events

Ist es nicht notwendig, den Kommunikationspartner zu spezifizieren, oder sind zukünftige Kommunikationspartner nicht im Voraus bekannt, wird ein Verfahren benötigt, um anonyme Kommunikation zur ermöglichen. In stationären verteilten Systemen kommen für diesen Zweck normalerweise zwei Verfahren in Frage: TupleSpaces und Eventsysteme. Wir verfolgten den Event-Ansatz weiter, weil es bei TupleSpaces fraglich schien, daß diese für große Zahlen an Teilnehmern und Interaktionen in verteilten Systemen skalieren.

EventManager gibt es bereits als Produkte, etwa im Umfeld von CORBA. Da diese auf stationäre Systeme ausgerichtet sind, ist es notwendig, diese an mobile Teilnehmer anzupassen. Weiterhin sind EventManager so zu modifizieren oder zu konzipieren, daß sie für große Zahlen von Teilnehmern und Events skalieren, weil man annehmen kann, daß ein größeres Agentensystem diese Charakteristiken hat.

Nähere Informationen zum Kommunikationsbereich finden sich in [SBH97].

5.1.3 Finden von Agenten

Interaktionen von Agenten oder dem Agentensystem mit Agenten erfordern (zumindest auf Systemebene) die Kenntnis über den Aufenthaltsort des Agenten, mit dem interagiert werden soll. Dieser Aufenthaltsort ist dabei nicht im Voraus zu bestimmen, da Agenten in Mole keine Restriktionen bezüglich ihrer "Reiseroute" auferlegt bekommen. Das Finden von Agenten ist also eine Vorstufe zur Interaktion mit den Agenten.

Die Probleme, die dabei auftauchen sind:

- die Zeit für eine Migration bewegt sich in der Zeit für eine beliebige Nachricht
- die minimale Zeit für den Finden-Algorithmus liegt bei einer Nachricht
- die minimale Zeit für eine Sequenz aus Finden und Interagieren liegt bei der Zeit für die Übertragung zweier sequentieller Nachrichten
- die Zeit, die eine für die Übertragung einer Nachricht gebraucht wird, ist groß in Relation zur Ausführungsgeschwindigkeit von Code, z.B. benötigt eine Nachricht innerhalb
    - eines Rechners in der Größenordnung von 100ns
    - eines LANs in der Größenordnung von 10 ms
    - eines MANs in der Größenordnung von 25ms
    - eines WANs in der Größenordnung von 250 ms
- falls das Auffinden und Interagieren seriell stattfinden, kann es zu veralteten Daten über den Aufenthaltsort kommen

Eine Möglichkeit, den Auffinden-Interagieren-Zyklus effektiver zu machen ist daher, beide Aspekte miteinander zu verschmelzen, und bei den Mechanismen, die das Auffinden ermöglichen Parameter vorzusehen, die die Interaktion (z.B. einen Rückkehrbefehl) angeben.

Wenn man sich ansieht, welche Eigenschaften die Informationen über den Aufenthaltsort von Agenten haben, findet man folgendes:
- es gibt typischerweise eine Informationsquelle und eventuell mehrere Informationssenken
- die Informationsquelle sind nacheinander verschiedene Orte
- die Anzahl an Schreibeoperationen ist typischerweise viel größer als die Anzahl an Leseoperationen (d.h. der Agenten bewegt sich öfter als daß eine Entität dessen Aufenthaltsort wissen will)
- die Informationen können im Extremfall nur sehr kurz aktuell sein
- die Menge an Aufenthaltsdaten steigt linear mit der Anzahl von Agenten, die gefunden werden können
- die Informationen müssen automatisch aktualisiert werden

Weitere Aspekte des Auffindens von Agenten sind:
- Die Voraussetzung dafür, einen Agenten aufzufinden, ist die Kenntnis der Existenz eines Agenten, und nicht alle existenten Agenten z.B. einer Anwendung müssen vorher bekannt sein.
- Die "Moving Target"-Problematik, d.h. Agenten können sich eventuell so schnell bewegen, daß die Informationen über den Aufenthaltsort beim Ausliefern der Informationen oder bei der Reaktion veraltet sind.

Wenn man nun, wie geschehen, mehrere Auffindemechanismen untersucht, sind die folgenden Fragen zu beantworten:
- Wie verhält sich der Auffindemechanismus bei einer Partitionierung des Netzes?
- Wieviel kostet ein Auffindemechanismus?
- Wie skalierbar ist der Auffindemechanismus?
- Wie hoch ist dessen Antwortzeit?
- Wie gut eignet er sich für die Interaktionsintegration?
- Wie robust ist er gegen unwillige Agenten oder Agentenserver?

Unter diesen Gesichtspunkten wurden vier Einzelverfahren und ein kombiniertes Verfahren konzipiert bzw. der Literatur entnommen und evaluiert.

Heimatregister
Dieses Verfahren wird bei GSM eingesetzt und bei IPv6 zur Unterstützung mobiler Endgeräte. Es besteht darin, daß jeder Agent ein festes Heimatregister hat, von dem der jeweilige Aufenthaltsort abgefragt werden kann. Bei jeder Migration schicken Sende- und Empfangsknoten eine Nachricht an das Heimatregister. Das Problem ist, daß dieses Verfahren bei "moving targets" ständig veraltete Informationen liefert und daß es nicht zur Interaktionsintegration geeignet ist.

Broadcast
Dieses Verfahren fragt einfach alle Agentenserver per Broadcast. Der Nachteil ist die schlechte Skalierbarkeit bei vielen Anfragen, zudem muß es die Möglichkeit geben, einen schnellen (Netzwerk-)Broadcast durchzuführen.

Energiekonzept
Das Energiekonzept ist eigentlich ein Mittel der Ressourcenkontrolle. Ein Agent bekommt von seinem Eigentümer eine gewisse Menge "Energie" zugeteilt, die vom Agenten durch jede Aktion verbraucht wird. Ist die Energie verbraucht, kann der Agent gelöscht werden. Daher muß

sich der Agent vor dem Verbrauch der Energiemenge wieder bei seinem Eigentümer melden, um neue Energie anzufordern. Dieses Verhalten kann man ausnutzen, um beim Eigentümer nach dem Aufenthaltsort des Agenten zu fragen. Da sich der Agent nicht bewegen kann, bis er eine Antwort seines Eigentümers hat, ist diese Information aktuell. Der Nachteil ist, daß im Zweifelsfall auf die Meldung des Agenten gewartet werden muß.

NameService
Hier wird ein bestehender Verzeichnisdienst als Informationsquelle benutzt. Allerdings lassen sich zumindest DNS und X.500 aufgrund ihrer anderen Ausrichtung nicht gut für diesen Zeck einsetzen, und die Implementierung eines eigenen Dienstes ist sehr aufwendig.

Kombination aus Heimatregister und NameService
Hier wird angenommen, daß sich das Agentensystem in Regionen partitionieren läßt. Innerhalb der Regionen wird mit Broadcast gesucht, die Region, in der sich der Agent aufhält, wird mithilfe eines Heimatregisterverfahrens ermittelt. Dieses Verfahren funktioniert nur dann besser als z.B. das Heimatregisterverfahren, wenn Migrationen eher "lokal" bezüglich der Region sind.

Tabelle 2 zeigt schematisch die Eigenschaften der Verfahren über den oben aufgeführten Kriterien.

Tabelle 2: Verfahren zum Auffinden von Agenten

	Heimatregister	**Broadcast**	**Energiekonzept**	**NameService**	**HR + NS**
Aktualität des Ergebnisses	bestmöglich ohne Agenten anzuhalten	mittel (schlecht im Fehlerfall)	sehr gut	wie HR + Zugriff auf NS	gut
Verhalten bei Partitionierung	schlecht, bei Replikation: besser	gut	schlecht, bei Replikation: besser	hängt vom NS ab	schlecht, bei Replikation: besser
Kosten des Mechanismus	2N/Migration * #Replikate	vielleicht 1N/ Server	skalierbar, Accounting muß exist.	hängt vom NS ab	2N/Zonenwechsel * #Replikate
Skalierbarkeit des Verfahrens	gut (bis auf # Nachrichten)	schlecht für große # Anfragen/Orte	gut	hängt vom NS ab	gut (bis auf # Nachrichten)
Antwortzeit des Verfahrens	sehr gut	2N seriell + Ausbreitung des BC	skalierbar: gut-schlecht	hängt vom NS ab	gut (1N)
Eignung für Interaktionsintegration	nicht geeignet	gut geeignet	gut geeignet	nicht geeignet	geeignet
Robustheit gegen unwillige Agenten	gut, falls Server übernimmt	gut	gut	gut, falls Server übernimmt	gut, falls Server übernimmt
Robustheit gegen unwillige Server	nicht robust	nicht robust	nicht robust	nicht robust	nicht robust

Es zeigt sich, daß eine Kombination aus Heimatregister und NameService geignete Ergebnisse erwarten läßt, allerdings setzt es das Vorhandensein von größeren "Regionen" voraus, die im vorgesehenen Organisationsmodell nicht vorhanden sind. Implementiert wurden daher das Heimatregisterverfahren und das Energieverfahren, allerdings noch ohne Interaktionsintegration.

5.2 Agentenmigration

Migration ist das Kriterium, das mobile von anderen Agenten unterscheidet, und das wesentlich zu den Eigenschaften mobiler Agenten beiträgt.

5.2.1 Agenten als Objektinseln

In herkömmlichen verteilten objektorientierten Systemen interagieren zwei Objekte häufig über ein gemeinsames Objekt, also eines, auf das beide eine Referenz besitzen. Das ist effizient, solange beide alle Objekte im selben Adressraum sitzen und wird schon auf der Programmiersprachenebene unterstützt. Gemeinsame Referenzen sind allerdings ein Problem, sobald eines der Objekte aus dem Adressraum herausmigriert. Ansätze, die dieses Problem angehen, kopieren entweder die referenzierten Objekte beim migrierenden Objekt, oder sie entscheiden pro gemeinsamen Objekt, ob es "mitmigriert" und damit aus dem ursprünglichen Adressraum herausgenommen wird, oder es dort verbleibt. Das Objekt, in dessen Adressraum sich das gemeinsame Objekt nun nicht mehr befindet, hält dann eine ungültige Referenz, was sich im Laufzeitfehlern äußern kann, sofern dieses Objekt über diesen Umstand nicht informiert wird oder sich bei jedem Zugriff von der Gültigkeit der Referenz überzeugt.

Um diese Transparenz der Gültigkeit von Objektreferenzen auch bei Migrationen von anderen Objekten zu erhalten, wurde angedacht, die Idee von Hogg [Hogg91] zu verfolgen, bei der Objekte zu "Inseln" gruppiert sind, und nur Referenzen innerhalb dieser Inseln haben. Die Kommunikation zwischen den Inseln erfolgt über Objekte, die entweder als Kopie übergeben werden oder als "vollständige Übergabe", bei der das übergebende Objekt keine Referenz darauf zurückbehält. Da Agenten sehr gut mit diesen Inseln identifiziert werden konnten, hätte der Einsatz dieses Modells das gewünschte Ziel erreicht.

Leider konnte das Modell bei der Verwendung von Java als Agentensprache nicht zwingend implementiert werden, da dies eine Modifikation der Laufzeitumgebung von Java zur Folge haben würde, was aufgrund der Dynamik der Javaentwicklung als unvorteilhaft angesehen wird. Daher ist das Inselmodell noch Teil des Verarbeitungsmodells, kann aber von zwei kooperierenden Agenten durch Austausch einer gemeinsamen Referenz unterlaufen werden.

5.2.2 Schwache Migration

Ein Agent besteht aus Code und Daten, wobei sich die Daten wiederum aufteilen lassen in "statische" Daten wie globale Variablen und Instanzenvariablen und "dynamische" Daten wie Threads, lokale Variablen sowie Parameter und Rücksprungadressen. Eine Migration, die Code und alle Daten transportiert erlaubt eine transparente, die sogenannte "starke" Migration, bei der das Programm des Agenten mit der nächsten dem Migrationsbefehl folgenden Programmzeile auf dem Migrationsziel fortgesetzt wird. Der Programmierer muß keine weiteren Maßnahmen treffen, um die Migration zu ermöglichen.

Demgegenüber steht die "schwache" Migration, die nur Code und statische Daten transportiert. Da alle die Daten fehlen, die eine Wiederaufnahme des letzten Ausführungszustandes ermöglichen würden, muß der Agent bei einem festen Punkt, etwa durch das Anstoßen einer Startprozedur, wieder gestartet werden, wobei die statischen Daten, die als migrationswürdig deklariert wurden, automatisch mittransferiert werden. Indem man den Ausführungszustand, solange dies durch die Verarbeitung notwendig ist, manuell in den statischen Daten kodiert, läßt sich in der Startprozedur ein gewisser Ausführungszustand wieder restaurieren. Durch die manuelle Kodierung hat der Programmierer außerdem die Kontrolle über den Umfang der Daten, die mittransportiert werden, während bei der transparenten Migration alle Datenelemente, die nicht auf jeden Fall gelöscht weden können (und damit dem Garbage Collector unterliegen), mitgenommen werden müssen.

Obwohl die starke Migration sehr viel komfortabler ist, ist sie wesentlich schwerer zu implementieren, da dafür ein implementierungsunabhängiges Format für den Ausführungszustand konzipiert werden und in die Laufzeitumgebung von Java implementiert werden müsste. Dieses Vorgehen würde aber wiederum die Portierbarkeit des Agentensystem beeinträchtigen und müsste bei jeder neuen Version der Laufzeitumgebung wiedereingebaut werden, da eine entsprechende Funktionalität im Java-System nicht vorhanden ist. Da die schwache Migration höchstens genausoviele Datenelemente transportiert (und damit schneller ist) und die schwache Migration nach unseren Erfahrungen vom Programmierer gut zu beherrschen ist, entschlossen wir uns, keine starke Migration zu implementieren, sondern uns auf die schwache Form zu beschränken, ein Standpunkt, dem auch fast alle anderen Java-basierten Agentensysteme gefolgt sind (bis auf ARA, das allerdings pro Agent eine eigene Laufzeitumgebung einsetzt, und bei dem daher Agentenkommunikation innerhalb eines Platzes auch immer eine Adreßraumgrenze überschreitet). Nähere Informationen zur Migration lassen sich in [RHR97] und [SBH97] finden.

5.2.3 Effiziente Migration von Code

Die Effizienz der Migration eines Agenten ist -aufgrund der Bedeutung dieses Aspektes für mobile Agenten- wesentlich für die Performanz von Anwendungen, die auf mobilen Agenten beruhen. Aus technischer Sicht muß bei einer Migration der Code und die Daten des Agenten transportiert werden. Da die Daten von Agent zu Agent verschieden sind, läßt sich dieser Transport bis auf Datenkompression nicht beschleunigen. Der Transport von Code allerdings läßt sich effizienter gestalten, wenn er aus Teilen besteht, die auch von anderen Agenten verwendet werden. Da auch verschiedene Agententypen zum Teil gleiche Funktionalität benötigen, ist diese Annahme durchaus realistisch. Weiterhin ist zu erwarten, daß mehrere Agenten eines Typs gleichzeitig im Agentensystem arbeiten.

Aus diesen Überlegungen wurde ein Mechanismus konzipiert und implementiert, der aus zwei Teilverfahren besteht, die unterschiedlichen Anforderungen genügen. Dazu wurde das Konzept des *Code-Servers* eingeführt. Ein Code-Server speichert Programmcode und stellt ihn auf Anfrage zur Verfügung. Das Problem der Codemigration kann vereinfacht auf die Suche nach einem günstigen Code-Server zurückgeführt werden. Das Problem wurde in zwei unterschiedliche Aufgaben aufgeteilt. Ein Basismechanismus garantiert das korrekte Ergebnis der Suche, während ein schneller Dienst nur manche Klassen auffindet, dafür aber besonders schnell arbeitet. Für den Basismechanismus werden eine Reihe von alternativen Algorithmen untersucht und verglichen. Der schnelle Dienst setzt sich aus verschiedenen Konzepten der Kooperation von Servern und dem Agentensystem zusammen. Die Kombination beider Suchmechanismen sorgt für eine zuverlässige und schnelle Codemigration. Das Agentensystem muß die Sicherheit der Agenten gewährleisten, da manipulierte Agenten im Namen ihres Benutzers großen Schaden anrichten können. Das Agentensystem ist dabei darauf angewiesen, daß die Codemigration gegen Manipulationen geschützt wird. Ein Mechanismus zur automatischen Signierung der Klassen ist Teil der Implementierung des Code-Servers.

Nähere Informationen zu diesem Thema finden sich in [HKB97].

5.2.4 Performanzmodelle für die Kommunikation/Interaktion

Im Verarbeitungsmodell von Mole hat ein Agent grundsätzlich immer die Wahl, entweder zu einem anderen Agenten hinzumigrieren und lokal zu kommunizieren oder nicht zu migrieren und entfernt zu kommunizieren, falls er mit einem Agenten auf einem anderen Platz interagieren will. Diese Entscheidung kann der Agent selbst treffen, falls er alle notwendigen Parameter wie z.B. die aktuelle Übertragungslatenz zu dem anderen Knoten kennt, aber die Berechnung ist aufwendig und verlangt u.U. Code, der komplexer ist, als die eigentliche Verarbeitung.

Daher ist es sinnvoll, dem Agenten die Berechnung dieser Entscheidung als Systemmechanismus anzubieten. Zur Berechnung dieser Enscheidung ist ein Performanzmodell notwendig, das die folgenden Parameter berücksichtigt:

- Netzwerkverzögerung
- Übertragungsrate zwischen den zwei Knoten
- Dauer des Marshallings von Daten über der Größe der Daten
- Größe des Agenten bei der Migration
- Größe der Parameter der Interaktion
- Größe der Resultate der Interaktion
- Selektivität der Verarbeitung, d.h. Größe des Verhältnisses von Resultaten zu Daten, die vom Agenten weiterverarbeitet werden

Die letzten drei Parameter lassen sich nicht einfach vom System messen, man kann entweder versuchen, diese statistisch aus der bisherigen Verarbeitung vorherzusagen, oder man überlässt dem Agenten (bzw. dessen Programmierer) die Angabe dieser Werte.

Ein Modell, das diese Parameter berücksichtigt, kann eine Entscheidung entweder nur im Hinblick auf eine Interaktion betrachten, oder sogar über einer Folge von Interaktionen Aussagen treffen, wobei das Problem ist, daß diese Folge nicht unabhängig voneinander sein muß.

Eine Messung des Performanzmodells in Mole unter vereinfachenden Annahmen (eine Interaktion) in einem kleineren Szenario aus drei Rechnerknoten in Deutschland und Schweden ergab, daß ein solcher Systemdienst meist die richtige Entscheidung empfiehlt. Eine Erweiterung des Modells auf mehrere Interaktionen erscheint dagegen zum jetzigen Zeitpunkt zu komplex zu sein. Nähere Informationen zum Performanzmodell lassen sich in [SS97] finden.

6 Sicherheitsaspekte

Der Aspekt der Sicherheit in Mobile-Agenten-Systemen ist nicht nur extrem wichtig für viele der für diese Technik vorgesehenen Einsatzgebiete wie z.B. Electronic Commerce, er ist auch von großem Interesse für die Forschung, weil durch die Mobilität und durch die Tatsache, daß eine Partei eine andere ausführt, neue Aspekte vorliegen. Dieses Kapitel schlägt zuerst eine Strukturierung des Sicherheitsgebietes vor und identifiziert Aufgaben, die in jedem Teilgebiet gelöst werden müssen. Anschließend werden daraus Anforderungen an Komponenten eines zu erstellenden Sicherheits-Rahmenwerks erarbeitet. Schließlich werden zwei verschiedene Sicherheitsmodelle auf ihre Eignung für dieses Rahmenwerk hin untersucht. Weitere Informationen zu diesem Aspekt lassen sich in [Hoh97] finden.

6.1 Strukturierung des Gebiets

Der Bereich Sicherheit läßt sich nach zwei Aspekten strukturieren: nach Interaktionsbereichen und nach den funktionalen Komponenten, die benötigt werden. Die Interaktionsbereiche von Agenten, Interpreter und dritten Parteien in einem Mobile-Agenten-System sind:

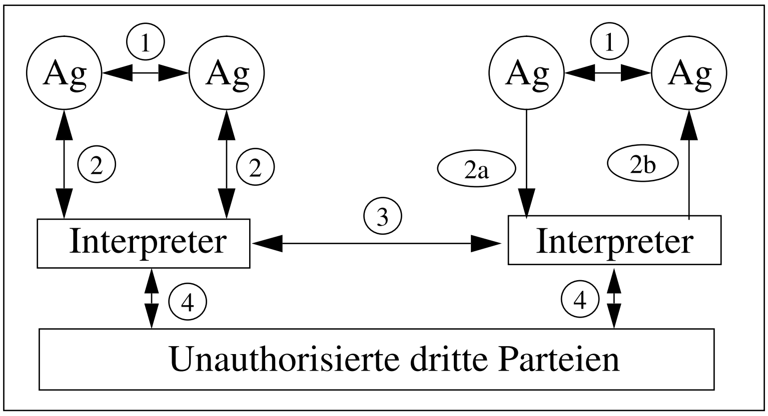


Schaubild 4: Interaktionsbereiche

1. Sicherheit zwischen Agenten
2. Sicherheit zwischen Agent und Interpreter
   a. Sicherheit von Agenten gegenüber böswilligen Interpretern
   b. Sicherheit von Interpretern gegenüber böswilligen Agenten
3. Sicherheit zwischen Interpretern
4. Sicherheit zwischen Interpreter und unauthorisierten dritten Parteien

6.1.1 Sicherheit zwischen Agenten

Dieser Bereich umfasst alle Angriffe, die zwischen zwei Agenten in derselben oder einer anderen Kaufzeitumgebung möglich sind. Einige dieser Angriffe lauten:

- (Physische) Manipulation des anderen Agenten
- Verschleierung der Identität eines Agenten (Maskierung)
- Betrug bei der Inanspruchnahme bzw. dem Anbieten von Diensten
- "Denial-of-Service"-Attacken

Der erste Angriff läßt sich durch eine geeignete Wahl der Agentensprache lösen, die eine solche Manipulation z.B. über Zeiger nicht zulassen darf. Das ist z.B. bei allen Systemen der Fall, die Java verwenden, also auch in Mole. Daneben darf es im System auch keine Sicherheitslöcher geben, die einen solchen Zugriff ermöglichen. Es ist generell unmöglich, die Existenz solcher Löcher bei einer bestimmten Implementierung auszuschließen, und auch bei einigen Java-Versionen sind Zugriffe dieser Art durch das Ausnutzen von Implementierungsfehlern möglich gewesen. Der zweite Angriff verlangt nach einem Authentifizierungskonzept für Agenten, das vom Agentensystem, oder wenn diesem nicht vertraut werden kann, vom anderen Agenten selbst durchgeführt werden muß. Falls der Agent selbst die Identität eines anderen Agenten prüft, muß sichergestellt werden, daß das Agentensystem diesen Agenten nicht so manipulieren kann, daß die Authentifizierung unterlaufen werden kann. Der dritte Angriff verlangt nach einem Dienstinanspruchnahmemodell, das für beide Parteien, Anbieter und Benutzer, sicherstellt, daß für Leistungen Gegenleistungen erbracht werden und daß es Mittel gibt, die Fakten einer Dienstnutzung so aufzuzeichnen, daß sie bei Streitigkeiten nachvollziehbar sind. Solche Dienstemodelle gibt es, siehe z.B. [SL95]. "Denial-of-Service"-Attacken schließlich sind auch in existierenden verteilten Systemen kaum von vornherein auszuschließen geschweige denn einfach zu verhindern. Immerhin benötigt hier die Mobilität der Agenten keine neuen Mechanismen, so daß Gegenmaßnahmen, so sie denn existieren, auch hier angewandt werden können.

6.1.2 Sicherheit zwischen Interpretern

Dieser Bereich umfasst alle Angriffe, die zwischen zwei Interpretern möglich sind. Dazu gehören: Maskierung, "Denial-of-Service"-Attacken u.a. Da Interpreter nicht mobil sind und durch das bisherige Modell verteilter Systeme abgedeckt sind, kommt an dieser Stelle nichts Neues hinzu. Immerhin verlangt die Möglichkeit eines Maskierungsangriffes die Authentifizierung von Interpretern durch einen anderen Interpreter.

6.1.3 Sicherheit zwischen Interpretern und unauthorisierten dritten Parteien

Dieser Bereich umfasst alle Angriffe, die von dritten Parteien gegen Interpreter gerichtet sind (Agenten sind nur über den Umweg Interpreter anzugreifen). Dazu gehört auch hier die Maskierung in dem Sinne, daß sich dritte Parteien für authorisierte Interpretern ausgeben sowie die Möglichkeit, die Kommunikation zwischen zwei Interpretern abzuhören oder zu manipulieren. Wie im letzten Abschnitt wird auch dieser Bereich von den bisherigen verteilten Systemen abgedeckt. Aus der Möglichkeit des Angriffes auf die Kommunikation folgt nicht nur wieder die Notwendigkeit der Authentifizierung von Interpretern, sondern auch die der Verschlüsselung der Kommunikation. Diese Notwendigkeit wiederum impliziert das Vorhandensein einer Schlüsselverteilstruktur im gesamten Agentensystem, so daß Verschlüsselung (und auch Authentifizierung jenseits von ebenso unpraktikablen wie unsicheren Passwortverfahren) überhaupt möglich wird.

6.1.4 Sicherheit von Interpretern vor böswilligen Agenten

Dieser eine Teil des Bereiches der Sicherheit zwischen Interpreter und Agenten umfasst die möglichen Angriffe von Agenten gegenüber dem Interpreter. Diese Angriffe sind u.a.:

- unauthorisiertes Verbrauchen von Ressourcen des Interpreters (CPU, Speicher usw.)
- unauthorisierter lesender/schreibender/modifizierender/benutzender Zugriff auf Ressourcen des Interpreters
- Verschleierung der Identität eines Agenten (Maskierung)
- Mißbrauchen der Identität des Interpreters (Trojanisches Pferd)
- usw.

Um diese sehr zahlreichen Arten des Angriffs auszuschließen und um nicht in Gefahr zu laufen, daß Agenten Löcher in Subsystemen des Interpreters ausnutzen (z.B. beim Betriebssystem), kann man als Basis einen Mechanismus verwenden, der auch in anderen Mobile-Code-Systemen verwendet wird, da diese mit denselben Problemen zu kämpfen haben. Dieser Mechanismus nennt sich "Sandbox" und baut auf dem Umstand auf, daß einige Mobile-Code-Systeme sehr restriktiv mit dem sein können, was den mobilen Programmen erlaubt wird. So können Java Applets in vielen Browsern nicht auf das Dateisystem zugreifen, Systemprogramme oder Bibliotheken benutzen und dürfen meist auch nur zu bestimmten Rechnern Netzwerkverbindungen aufbauen. Was sie dürfen, ist rechnen, solange keine externen Ressourcen (Plattenplatz, netzwerk) benötigt werden, und das Öffnen von Fenstern samt dem dazugehöhrenden Benutzungsdialog.

Bei mobilen Agenten hingegen muß dieser Mechanismus zum einen restriktiver, und zum anderen freizügiger sein. Restringiert werden muß der unkontrollierte Zugriff auf interne Ressourcen (CPU, Speicher), der bei Applets kaum ein Problem darstellt, da diese unter der "Aufsicht" des Benutzers ablaufen. Dagegen müssen viele Agenten Zugriff auf externe Ressourcen bekommen, da ein Agent, der z.B. keine Datenbanken abfragen kann, nur in wenigen Anwendungsfeldern Verwendung finden kann. Daher muß also zum einen ein Abrechnungs- und Kontrollsystem für den quantitativen Verbrauch von Ressourcen durch den Agenten geschaffen werden (wie das z.T. in Telescript der Fall ist, wo Ressourcenverbrauch 'Ticks' kostet), zum anderen muß ein Authorisierungsmechanismus authorisierten Agenten Zugang zu Systemressourcen geben. Für letzteres bieten sich in großen Agentensystemen eher Ausweise (Capabilities) als Zugriffskontrollisten an, da die Zahl der Subjekte, also der Agenten, sehr groß sein kann.

6.1.5 Sicherheit von Agenten gegenüber böswilligen Interpretern

Dies ist der andere Teibereich der Sicherheit zwischen Agent und Interpreter. In ihn fallen alle Angriffe, die ein Interpreter gegen einen Agenten ausführen kann. Dies sind:

- lesen/kopieren von Agentendaten
- lesen des Agentencodes
- modifizieren von Daten
- modifizieren von Code (Viren, Würmer, trojanische Pferde)
- “denial of correct execution”
- maskieren von Interpretern

Die Modifikation von Code läßt sich dadurch verhindern, daß der Autor des Codes diesen digital signiert (auch dafür werden kryptographische Schlüssel benötigt). Dies geht solange der Code sich durch die Ausführung nicht ändert (kein selbstmodifizierender Code). Das Signieren von Codeteilen ist z.B. in der Sprache Java als Teil der Standardbibliotheken möglich (da auch dieser Aspekt bei Applets von Bedeutung ist).

Falls Daten oder Code eines Agenten auf einem Interpreter nicht benötigt werden, kann man sie durch Verschlüsselung unlesbar machen, falls sie konstant sind, signieren, um so zumindest die Modifikation zu verhindern, aber alle Teile, die von der Anwendung auf diesem Interpreter verändert werden können, können mit dieser Technik nicht geschützt werden.

Das unauthorisierte Modifizieren von Daten durch den Interpreter läßt sich u.U. nachweisen (siehe [Vig97]), wenn auch andere Angriffe mit diesem Ansatz nicht nachgewiesen oder gar abgewehrt werden können.

Es gibt aber derzeit keine Ansätze, die restlichen Angriffe zu verhindern, außer man benutzt spezielle, "vertrauenswürdige" Hardware, wie im CITADEL-Projekt [Pal94], wo gegen Manipulation geschützte Komponenten eingesetzt werden, die einen Prozessor und zumindest einen geheimen Schlüssel enthalten. Der Ansatz, für mobile Agenten auf den Interpretern spezielle Hardware einzusetzen ist allerdings sehr unattraktiv, umso mehr, als die gesamte Technik noch nicht sehr weit verbreitet ist.

Es gibt einige Ansätze, die Möglichkeit von Angriffen des Interpreters dadurch zu umgehen, daß Agenten nur auf vertrauenswürdige Interpretern migrieren, die per definitionem nicht angreifen werden. Diese Ansätze ([FGS97a],[FGS97b],[Ord96]) haben den Nachteil, daß Vertrauen entweder ein zu grobes Maß ist (weil z.B. einem Lufthansa-Interpreter dann nicht vertraut werden kann, wenn es um das Vergleichen von Flugpreisen geht), oder die Zahl der Interpreter, die benutzt werden können, so gering ist, daß einige der Vorteile mobiler Agenten, wie z.B. der der effizienten Interkation mit einem Server, nicht genutzt werden können. Eine Variante dieses Ansatzes ist es, das Agentensystem nicht-offen bezüglich des Einbringens von

Interpretern zu gestalten, sondern den Bentrieb des Agentensystems in die Hände "vertrauenswürdiger" Firmen wie AT&T zu legen. Diese in dieser Hinsicht geschlossenen Systeme benötigen allerdings einen immensen initalen Aufwand, um ein erstes Agentensystem zu installieren, und für einige Anwendungen kann das Vertrauen in die Betreiberfirmen wie gezeigt, sehr klein sein.

Obwohl die meisten der oben erwähnten Angriffe also mit existierenden Ansätzen nicht verhindert werden können, ist dieser Teilbereich für viele Anwendungsgebiete emminent wichtig. Falls diese Angriffe nicht verhindert werden können, kann z.B. ein kryptographischer Schlüssel offenbar werden, elektronisches Geld kann kopiert und ausgegeben werden, die Funktion des Agenten kann durch falsches Ausführen des Codes unterlaufen werden usw. Daneben ist es auch für andere Teilbereiche der Sicherheit wichtig, daß der Interpreter den Agenten nicht völlig kontrollieren kann. Wenn z.B. der Interpreter die Daten eines Agenten lesen kann und dessen Code kennt, kann er verhindern, daß die falsche Identität eines anderen Agenten entlarvt werden kann, weil der Interpreter das entscheidende "if" falsch ausführt.

Blackbox Security - ein neuer Lösungsansatz

Im Rahmen des AIDA-Projekts wurde ein neuer Ansatz erarbeitet, der dieses Problem zum großen Teil lösen kann. Er beruht auf der Beobachtung, daß obwohl ein Interpreter jedes Byte des Speichers und jede Zeile des Quellcodes lesen und manipulieren kann, er u.U. Zeit dafür benötigt herauszufinden, was dieses Byte eigentlich enthält und welchen Beitrag die Programmzeile zur Anwendung beisteuert. Damit diese Analyse nicht vor dem Eintreffen eines Agenten vorgenommen werden kann, ist es daher notwendig, daß jeder Agent eine andere Struktur hat, anders "aussieht", auch wenn er dasselbe wie ein anderer Agent leistet. Diese Aufgabe wollen "Verwürfelungsalgorithmen" leisten, die einen Originalagenten, d.h. dessen Code und Daten, in eine neue Form bringen, eben "verwürfeln", wobei die neue Form dasselbe wie die alte leistet. Dieser Schutz, der den Agenten eine Zeitlang wie eine "Blackbox" aussehen läßt, ermöglicht es dem Agenten zwar nicht, für immer unanalysierbar zu sein, erlaubt ihm aber zumindest ein Zeitintervall, in der man davon ausgehen kann, daß der Agent -wenn er ausgeführt wird- nicht durch einen lesend oder manipulierend angegriffen werden kann. Dieses "Schutzintervall" hat zur Folge, daß der Agent ein "Verfallsdatum" trägt, nach dem er weder von anderen Interpretern angenommen wird oder er mit anderen Agenten oder Interpretern interagieren kann.

Da man also davon ausgehen muß, daß ein Agent nach dem Schutzintervall angreifbar ist, müssen auch die Daten, die er mit sich trägt, und die aus sich heraus einen Wert darstellen, dieses Verfallsdatum tragen, nach dessen Ablauf sie nicht mehr verwendet werden können. Ein Beispiel für diese Art von Daten sind elektronische Geldscheine oder geheime Schlüssel, während Daten, die nur für den Agenten von Bedeutung sind, nicht in dieser Art geschützt werden müssen. Der beschriebene Ansatz lässt hoffen, daß das Problem der böswilligen Interpreter im wesentlichen gelöst werden kann. Daher soll er weiter verfeinert werden, eine Implementierung soll die Machbarkeit zeigen und Messungen ermöglichen.

Nähere Informationen zu diesem Ansatz lassen sich in [Hoh97] finden.

6.1.6 Benötigte Komponenten und Anforderungen

Nachdem die Interaktionsbereiche identifiziert wurden, sollen jetzt die Komponenten benannt werden, die ein Rahmenwerk benötigt, das Sicherheit in einem Mobile-Agenten-System gewährleisten soll. Weiter sollen diese Komponenten grob skizziert werden.

Authentifikation von Agenten, Interpretern, Code und Benutzern

Um die Identität dieser Parteien sicherzustellen, müssen Agenten, Interpreter und Benutzer authentifiziert werden können. Um Manipulationen an Code auszuschließen, muß die Integrität von Code sichergestellt werden, oder, aus einer anderen Sicht, muß auch hier die Identität gewahrt sein. Betrachten wir die Teilaspekte einzeln:

Authentifikation von Agenten
Agenten müssen gegenüber Interpreterm und gegenüber anderen Agenten authentifiziert werden. Abhängig davon, ob ein Agent einem Interpreter vorliegt, oder der Agent von einem anderen Interpreter aus kommuniziert, kann im ersten Fall die Authentifizierung aus der Datenform des Agenten geschlossen werden, die in den konstanten Teilen (konstante Daten und konstanter Code) vom Benutzer oder einer authorisierten Seite aus signiert sein muß. Dazu ist die Signatur über dem Datenzustand des Agenten zu prüfen, wozu wiederum Kontrolldaten der signierenden Institution (z.B. ein öffentlicher Schlüssel) vorhanden sein müssen. Falls ein Agent einem Interpreter nicht vorliegt, sondern von einem anderen Interpreter aus kommuniziert, muß die Authentifizierung den "normalen" Weg über die Kenntnis eines Geheimnisses, gehen. Dieses Geheimnis, im Allgemeinen ein geheimer Schlüssel des Agenten oder des Besitzers, wird durch das Stellen einer Anfrage und die Prüfung der Antwort erfolgen (siehe z.B. [NS78] für einen gängigen Authentifizierungsalgortihmus). Dabei ist zu beachten, daß wenn es böswillige Interpreter gibt, auf denen der Agent war, und der Agent seine Daten nicht schützen kann, dieses Geheimnis auch anderen Seiten bekannt sein kann.

Die Authentifizierung von Agenten gegenüber anderen Agenten kann entweder über die Authentifizierung von Agenten gegenüber dem Interpreter des anfragenden Agenten erfolgen, wenn der anfragende Agent diesem Interpreter vertrauen kann. Falls dieses Vertrauen nicht gegeben ist, muß der Agent diese Authentifizierung selbst übernehmen, wobei auch hier die "normale" Authentifizierung über das Nachweisen der Kenntnis eines Geheimnisses erfolgen wird. Da der Agent in diesem Fall seinem Interpreter nicht vertraut, muß er die Möglichkeit haben, die Authentifizierung vollziehen zu können, ohne von seinem Interpreter beeinflusst zu werden. Falls der Interpreter z.B. Zugriff auf die Kontrolldaten der Authentifizierung (z.B. den öffentlichen Schlüssel des zu authentifizierenden Agenten) hätte, könnte er diese Daten so ändern, daß die Authentifizierung erfolgreich ist. Selbst, falls das der Fall ist, muß der Agent sicher diese Authentifizerungsdaten bekommen, d.h. er trägt sie bereits bei sich (und kann sie vor dem Interpreter verbergen), oder er kann sie vom Schlüsselverteilmechanismus bekommen, ohne dabei vom Interpreter manipuliert zu werden.

Authentifikation von Interpretern
Interpreters müssen gegenüber anderen Interpretern und Agenten authentifiziert werden. Die Authentifizierung gegenüber anderen Interpretern ist dasselbe wie die Authentifizierung zweier Rechner in einem herkömmlichen verteilen System, und kann daher wieder den Nachweisen der Kenntnis eines Geheimnisses benutzen. Auch hier ist es dazu wichtig, von der Schlüsselverteilkomponente die Kontrolldaten (öffentlicher Schlüssel) sicher zu bekommen.

Die Authentifizierung gegenüber Agenten kann im Prinzip genauso funktionieren, solange das oben bei der Authentifizierung von anderen Agenten zutrifft, nämlich, daß der Interpreter den Authentifizierungsprozeß nicht manipulieren kann und daß der Agent die Kontrolldaten sicher bekommen kann. Zu beachten ist allerdings, daß es unmöglich sein muß, daß der Interpreter mittels eines anderen Agenten auf dem "richtigen" Interpreter, die Authentifizerung umgehen und sich selbst maskieren kann, da einer Authentifizierung zwischen Agent und Interpreter wahrscheinlich nicht die Einrichtung eines sicheren Kommunikationskanals folgen wird, da die Kommunikation lokal und damit sicher ist.

Authentifikation von Benutzern
Um die Abrechnung und eine Haftbarkeit des Besitzers eines Agenten sicherzustellen, muß gewährleistet sein, daß der Besitzer eines Agenten authentifiziert werden kann. Dieser Vorgang ist allerdings mehr organisatorischer Natur und betrifft die Institution, die die Organisation der Abrechnung und der juristischen Aspekte übernimmt. Ohne eine solche Authentifikation sind alle Agenten im rechtlichen Sinne anonym und eine Abrechnung muß anders bewerkstelligt werden (etwa über elektronisches Geld).

Authentifikation von Code
Es muß sichergestellt werden, daß der Code eines Agenten nicht unauthorisiert verändert wurde. Falls der Code sich während der Verarbeitung nicht ändert, kann diese Zusicherung auf die Prüfung gegen eine vor der Migration eines Agenten erstellten Signatur des Codes geprüft werden. Diese Signatur kann sich auf einen Teil des Codes beziehen, etwa eine Klasse. Dann ist es sinnvoll, wenn der Autor dieses Teil signiert. Sie kann aber auch den gesamten Code eines Agenten signieren, dann kann die Signatur vom Eigentümer des Agenten vorgenommen werden. Dazu muß allerdings der Umfang des Codes des Agenten von vorneherein feststehen. Dazu ist es nicht unbedingt notwendig, den Code jeweils in einem Stück zu migrieren, falls es effizienter ist, den Code dynamisch aus anderen Quellen als dem Absendehost zu bekommen (vgl. [HKB97] für eine Diskussion eines solchen Systems), dann kann die Signatur des Eigentümers auch nur die Liste der Kontrollinformationen der Signaturen der Codeteile umfassen. Dieses Vorgehen hat den Vorteil, daß nur eine Kontrollinformation, die der Signatur des Eigentümers, besorgt werden muß, was wahrscheinlich sowieso der Fall ist, wenn der Eigentümer bekannt sein muß.

Verschlüsselung der Kommunikation zwischen Interpretern und Agenten

Um die Kommunikation zwischen zwei Interpreter zu schützen, muß diese verschlüsselt werden. Dieser Aspekt ist in verteilten Systemen seit langem erforscht, so daß auch hier auf ein bekanntes Verfahren benutzt werden kann. Da auch Agenten Interpreter zum Transport von Kommunikation benutzen, ist damit auch diese Kommunikation vor Dritten geschützt, wenn den Interpretern vertraut werden kann. Ist das nicht der Fall, müssen die Agenten wiederum selbst verschlüsseln, was eine sichere Übermittlung von Schlüsseln an die Agenten bedingt.

Schlüsselverteilung

Um den Beteiligten benötigte Schlüssel für die Authentifizierung, die sichere Kommunikation und andere Zwecke geben zu können, muß es eine Komponente geben, die diese Aufgabe übernimmt. Diese Komponente muß

- sicher gegen Manipulation sein
- ausfallsicher sein
- performant genug sein
- Schlüssel an alle Beteiligten sicher ausgeben können, insbesondere an Agenten
- in das "Betriebsmodell" des Agentensystems passen, also z.B. veteilt sein, wenn es keine zentrale Authorität gibt usw.

Es ist zu prüfen, ob hier ein existierender Mechanismus, etwa X.509, benutzt werden kann, oder ob ein existierender Mechanismus modifiziert werden oder gar neu entwickelt werden muß.

Authorisierung

Um Agenten Zugriff auf Systemressourcen zu geben, muß es eine Komponente geben, die diesen Zugriff kontrolliert. Diese Authorisierung kann entweder auf der Basis der Identität des Agenten erfolgen, oder sie kann Ergebnis einer Befragung der Authorisierungspolitiken sein, die z.B. den Typ des Agenten miteinbeziehen. Diese of "Security Policies" genannten Mechanismen erlauben es, auf verschiedenen Ebenen Regeln zu formulieren, die beschreiben, welche Rechte welchen Agenten zu welchen Bedingungen eingeräumt werden. Dieser Aspekt wird im Abschnitt "Sicherheitsmodelle" ausführlicher diskutiert.

Abrechnung (für Electronic Commerce)

Abrechnung von erbrachten Leistungen des Agentensystems, des Interpreters oder anderer Agenten kann zu zwei Zwecken erhoben werden: entweder *intern*, um damit z.B. den Ressourcenverbrauch eines Agenten quantitativ kontrollieren zu können, oder *extern*, wenn mit abzurechnenden Leistungen Gegenwerte in der realen Welt, z.B. Geld, verbunden ist.

Wenn in einem Agentensystem externe Abrechnung vorgesehen ist, muß gewährleistet werden, daß diese nicht manipulierbar ist und daß keine der Parteien dabei betrogen werden kann. Sie sollte daher von einem neutralen Dritten vorgenommen werden oder objektiv nachvollziehbar sein. Dieser Aspekt ist allen Anwendungen gemein, die den Charakter von Electronic Commerce haben; daher können Mechanismen aus diesem Umfeld auch hier eingesetzt werden, solange diese mit mobilen Teilnehmern zurecht kommen.

Da die interne Abrechnung nur administrativen Zwecken dient, und von einer neutralen Anwendung durch den Interpreter ausgegangen werden kann, muß diese nicht so großen Ansprüchen genügen. Hier können die Mechanismen angewandt werden, die zur Abrechnung von Leistungen in einem Betriebssystem benutzt werden.

Ressourcenkontrolle

Ressourcenkontrolle ist die quantitative Kontrolle des Ressourcenverbrauchs durch die Agenten. Auch dieser Aspekt ist Teil des Sicherheitsmodells und wird dort behandelt.

"Denial-of-service"-Gegenmaßnahmen

Um Denial-of-service-Attacken zu verhindern, müssen alle möglichen Angriffe dieser Art in Betracht gezogen werden. Grob kann man diese zwar in

- Angriffe gegen Agenten und
- Angriffe gegen Interpreter einteilen,

eine komplette Aufzählung ist allerdings kaum möglich und verlangt in jedem Fall die genaue Kenntnis eines Agentensystems, auf das es sich bezieht. Daher kann man auch nicht unbedingt auf die Existenz eine Komponente schließen, die alle Denial-of-service-Attacken verhindert, vielmehr scheint es so zu sein, daß verschiedene Maßnahmen im Einzelfall dazu führen können, daß einige dieser Attacken unterbleiben. Dieser Aspekt ist aber in jedem Fall Teil der Sicherheitsproblematik jeden normalen verteilten Systems, und Problemlösungen aus diesem Gebiet sollten sich auch hier anwenden lassen.

6.2 Sicherheitsmodelle

In diesem Abschnitt sollen kurz drei Sicherheitsmodelle vorgestellt werden, sowie deren Anwendbarkeit im Bezug auf mobile Agenten. Ein Sicherheitsmodell modelliert dabei alle relevanten Parteien, ihre erlaubten Aktionen sowie den Regelformalismus, der eine Sicherheitspolitik sicherstellt.

6.2.1 OMG Security Service [OMG97]

Das Sicherheitsmodell der OMG ist ein abstraktes Dokument, das - wie bei der OMG üblich - keine Implementierung vorschreibt und im wesentlichen einen Rahmen definiert, innerhalb dessen sich Produkte, die diesen Vorschlag implementieren, bewegen können. Er ist nicht auf mobile Agenten ausgerichtet, sondern beschreibt Funktionen, die sehr viele Anwendungen benötigen. Dazu gehören:

- Identifikation und Authentifikation von Benutzern und Objekten
- Authorisierung und Zugriffskontrolle
- Abrechnung
- Sicherung der Kommunikation inklusive Authentifikation, Integritätsschutz und Verschlüsselung von Nachrichten
- Nicht-Abstreitbarkeit von Interaktionen
- Administration von Sicherheitsaspekten

Die OMG Security Spezifikation enthält somit fast alle Komponenten, die auch für mobile Agenten gebraucht werden, der Bereich, der fehlt, ist der Schutz vor Denial-of-Service-Attakken, der aber ein Forschungsthema ist. Trotzdem muß ein Sicherheitsmodell für mobile Agenten mehr umfassen, als in der OMG Spezifikation beschrieben, weil die Mobilität z.T. neue Aspekte aufbringt, z.B. das erwähnte Problem der böswilligen Interpreter. Weiter ist das OMG Modell abstrakt und will eine Vielzahl von verschiedenen Implementierungen einbeziehen, die verschiedene Crypto-Mechanismen benutzen. Diese Ausrichtung ist bei einem Modell für mobile Agenten, nicht gegeben. Schließlich beruht das Sicherheitsmodell auf der generellen OMG-Architektur, insbesondere dem Object Request Broker (ORB), der in einem Mobile-Agenten-System nicht vorhanden sein muß. Ein anderes Sicherheitsmodell, das auf dem OMG-Modell aufbaut, aber auf ein Mobile-Agentensystem ausgerichtet ist, ist das des MAF Proposals.

6.2.2 MAF Proposal Security Model [MAF97]

Auf den Request for Proposals zu einer "Mobile Agent Facility" (MAF) reichte eine Gruppe bestehend aus Vertretern von IBM, General Magic, GMD Fokus, The Open Group und Crystaliz einen Vorschlag ein, der auch ein abstraktes Sicherheitsmodell umfasst.

Dieses Sicherheitsmodell geht über den Security Service von CORBA hinaus, da er nicht alle Bedürfnisse von Mobile-Agenten-Systemen befriedigen kann. Hier wird er aber zumindest als Grundlage benutzt, und es existiert eine (informale) Umsetzung der Funktionen des Proposal Modells auf Funktionen des CORBA Security Services.

Das Sicherheitsmodell umfasst die Existenz von Sicherheitspolitiken, definiert verschiedene Kommunikationsparameter, die ein Agent wählen kann und macht Aussagen über Authentifikation und Delegation.

Sicherheitspolitiken beinhalten Regeln für:

- das Einschränken oder Gewähren von bestimmten Eigenschaften wie dem Erzeugen neuer Agenten, der Migration oder dem Ausgeben von elektronischem Geld
- das Setzen von Limits für den Verbrauch von Systemressourcen
- das Einschränken oder Gewähren von Zugang zu bestimmten Zielen

Sie können vom Agent und vom Agentensystem definiert werden.

Falls ein Agent migriert, oder sonstige Kommunikationsdienste des Interpreters in Anspruch nimmt, kann er folgende Sicherheitsmaßnahmen fordern:

- Verschlüsselung der Kommunikation
- Überprüfung der Integrität der Kommunikation
- Authentifizierung des Kommunikationspartners
- Prüfung auf mehrfaches Vorhandensein des gleichen Agenten

Authentifikation von Interpretern
geschieht über die existierenden Mechanismen

Authentifikation von Agenten
aufgrund des Problems böswilliger Interpreter dürfen Agenten keine Schlüssel mit sich tragen. Daher werden Agenten mit anderen Mitteln, nämlich mithilfe der sogenannten "Authenticators" authentifiziert. Diese sind Algorithmen, von denen es mehrere Typen gibt, und die z.B. die Identität des Ausgangsortes oder des Eigentümers des Agenten in Betracht ziehen. Ein Typ, die "one-hop authenticators" betrachten einen Agenten als authentifiziert, sobald der Agent dieselbe Autorität als Besitzer hat wie der Ausgangsort und der Ausgangsort authentifiziert werden kann, sonst nicht. Authenticators, die mehrere Stationen (Hops) in Betracht ziehen können, werden erst für zukünftige Überarbeitungen des Proposals vorgesehen.

Delegation
Falls ein Agent einen entfernten Methodenaufruf macht, werden die Rechte des Agenten diesem Aufruf mitgegeben, damit er mit diesen Rechten ablaufen kann

Das Sicherheitsmodell des MAF Proposals ist ein erster Ansatzpunkt für ein Modell, das ein Mobile-Agenten-System schützt. Allerdings deckt es nicht alle Bereiche der Sicherheit ab, es macht keine Aussagen über Lösungen der Probleme, die auftauchen, wenn böswillige Interpreter auftauchen und es führt das abstrakte Modell vollständig nicht auf die Ebene der Implementierung, so daß zwei Agentensysteme, deren Sicherheitskomponente diesem Modell folgen würde, nicht unbedingt kompatibel wären.

6.2.3 Aglets Security Model [KLO97]

Dieses Modell ist im Umfeld des Aglets-Systems der IBM Japan zu sehen. Dieses System implementiert mobile Agenten auf der Basis von Java und ist in Binärform frei verfügbar. Da diese Gruppe auch zu den Autoren des MAF Proposals gehören, verwundert es nicht, daß es wie eine Verfeinerung des obigen Modells wirkt.

Das Modell stellt Mechanismen bereit, die:

- den Transfer und die Kommunikation eines Agenten schützen
- die Zugriffkontrolle und das Auditing beim Ausführen eines Agenten erlauben
- es verhindern, daß Agenten sich gegenseitig angreifen
- es verhindern, das Agenten das Interpretersystem angreifen

Dazu stellt das Modell Formalismen bereit, die es

- dem Programmierer eines Agenten
- dem Besitzer eines Agenten
- dem Verwalter der Ausführungsumgebung
- dem Verwalter der Domain

erlauben, Sicherheitspolitiken zu definieren, die festlegen

- unter welchen Bedingungen Agenten auf andere Objekte zugreifen dürfen
- welche Authentifizierung von Benutzern und anderen Parteien verlangt wird
- welche Aktionen eine authentifizierte Einheit durchführen kann

- ob Einheiten ihre Rechte weitergeben können
- welche Kommunikationssicherheit zwischen Agenten und ihren Interpretern gefordert ist
- welcher Grad an Abrechnung für die sicherheitsrelevanten Aktivitäten gefordert ist

Die Sicherheitspolitiken der obigen Parteien werden dabei immer von denen der übergeordneten Parteien "überschrieben". Domains sind dabei Bereiche mit der gleichen Sicherheit, wobei Interpreter innerhalb dieser Domains von anderen Institutionen betrieben werden dürfen, als der, die die Domain betreibt.

Die Authorisierungssprache, die beschreibt, welche Agenten unter welchen Umständen auf welche Ressourcen zugreifen dürfen, erlaubt, positive und negative Regeln aufzustellen.

Das Aglets Sicherheitsmodell stellt einige der von einer Sicherheitskomponente benötigten Funktionen bereit und realisiert damit viele der Funktionen, die sich mit existierender Technik implementieren lassen. Allerdings macht es keine Aussagen zur Schlüsselverteilung und betrachtet das Problem böswilliger Interpreter als unlösbar.

6.3 Sicherheit: Zusammenfassung

Zusammenfassend läßt sich sagen: Sicherheit ist für das Gebiet der mobilen Agenten ein wichtiges Thema, weil es zum einen Einfluß auf die Akkzeptanz der Technik hat und zum anderen einige Anwendungsbereiche wie Eletronic Commerce erst ermöglicht. Durch den Aspekt der Mobilität von Agenten kommen nicht in allen Teilgebieten neue Aspekte hinzu; vieles kann unter Verwendung bestehender Techniken gelöst werden. Der Aspekt des Schutzes des Interpreters vor böswilligen Agenten hat über das Gebiet der mobilen Agenten hinaus Einfluß auf jeden Einsatz unbekannter Programme auf einem Rechner.

Ein Teilgebiet, das des Schutzes von mobilen Agenten vor böswilligen Interpretern ist jedoch zum einen wichtig, zum anderen stellt es ein neues Problem dar, das anspruchsvoll ist, und bisher noch nicht technisch gelöst wurde. Wichtig deshalb, weil Agenten auf der einen Seite geldwerte Daten transportieren können, und weil sie u.U. auf der anderen Seite Geheimnisse oder persönliche Daten des Eigentümers transportieren können, die dieser nicht preisgeben will. Bisherige Ansätze halten dieses Problem für technisch kaum lösbar und gehen andere Wege, die allerdings nur Teillösungen darstellen oder Restriktionen aufweisen. Die Sicherheit des Agenten ist daher die große Herausforderung auf dem Gebiet der Sicherheit bei mobilen Agenten; sie soll durch den Ausbau des Ansatzes der "Blackbox Security" angegangen werden.

7 Entwurf von Protokollen zur Waisenerkennung und zur Terminierung in Mobile-Agenten-Systemen [Bau97]

Waisenerkennung und Terminierung ist im Bereich der verteilten Systeme ein gut erforschtes Gebiet, innerhalb dessen viele Problemlösungen existieren. Diese Lösungen nutzen die wohldefinierten Beziehungen zwischen Eltern- und Kind-Entitäten (verteilte Prozesse oder Objekte) aus. Diese Lösungen sind jedoch im Bereich der Mobile-Agenten-System nicht anwendbar, da keine entsprechenden, aus der Aufrufstruktur automatisch ableitbaren Beziehungen zwischen Agenten existieren. Deshalb müssen in diesem Bereich neue Protokolle definiert werden.

Innerhalb des Berichtszeitraumes wurde ein Konzept entwickelt, daß sowohl Waisenerkennung als auch Terminierung erlaubt. Dieses Konzept baut auf zwei anderen Konzepten auf, dem sogenannten Energiekonzept, und dem Pfadkonzept. Das Energiekonzept bietet Waisenerkennung, das Pfadkonzept bietet die Möglichkeit, Agenten zu finden und damit zu terminieren.

Beide Konzepte haben Nachteile. Durch geschickte Kombination erhalten wir ein Protokoll, daß die Vorteile der Konzepte vereinigt, und zur gleichen Zeit die Nachteile minimiert. Dieses Protokoll ist das sogenannte “Schatten”-Protokoll. Es nutzt die Idee eines Platzhalters (Schatten) des Benutzers, der vom System jedem Agenten des Benutzers zugeordnet wird. Dieser Schatten speichert Informationen über alle abhängigen Agenten. Wenn dieser Schatten entfernt wird, sind alle abhängigen Agenten Waisen und können terminiert werden.

Wir werden jetzt das Energiekonzept und das Pfadkonzept präsentieren, um darauf aufbauend das Schattenprotokoll diskutieren zu können.

7.1 Das Energiekonzept

Ein Agent benötigt im Rahmen seiner Ausführung Ressourcen wie CPU-Zeit, Speicher, I/O, und benutzt vom System angebotene Dienste. Wir nehmen an, daß jeder in Anspruch genommene Dienst und jede benutzte Ressource eine bestimmte Menge Energie des Agenten benötigen. Zu Beginn seines Lebens bekommt der Agent eine bestimmte Menge Energie, und sobald diese vollständig verbraucht ist, wird der Agent zur Waisen erklärt und kann terminiert werden.

Dies bietet eine simple Methode um Waisen erkennen zu können, mit dem Vorteil, daß die Aktivität eines Agenten (also die Nutzung der Ressourcen) seine Lebensspanne bestimmt. Der Nachteil dieses Konzeptes ist, daß, sollte die Lösung eines Problems mehr Energie benötigt werden als von der Applikation (die die Anfangsenergie ausgibt) angenommen, der Agent terminiert wird, ohne die Aufgabe beenden zu können. In solchen Fällen wäre es sehr wertvoll, wenn der Agent zusätzliche Energie von der Applikation anfordern könnte. Ein Agent, der bemerkt, daß sein Energiestand gefährlich niedrig wird, könnte eine Nachricht an die Applikation senden, sich schlafen legen (und damit nur wenig Energie verbrauchen), und darauf warten, daß die Applikation mit zusätzlicher Energie antwortet. Dabei spielt es keine Rolle, ob die Applikation sofort reagiert, solange gewährleistet ist, daß die zusätzliche Energie innerhalb der verbleibenden Lebenszeit des Agenten ankommt. Dies bedeutet, daß das Konzept gegen kurzzeitige Netzwerkfehler oder Abstürze von beteiligten Rechensystemen unempfindlich ist. Die maximale Zeit, die ein solcher Fehler unbehoben existieren darf, ist schlicht die verbleibende Lebenszeit des Agenten minus der Nachrichtenlaufzeit.

Wenn die Applikation entscheidet, daß der Agent keine zusätzliche Energie bekommen soll (z.B. weil ein anderer Agent die Aufgabe bereits gelöst hat), antwortet sie einfach nicht. Das System garantiert die Terminierung nach Verbrauch der verbleibenden Energie.

Es muß allerdings sichergestellt werden daß dieser Mechanismus nicht überlistet werden kann. Wenn ein Agent einen anderen erzeugt, dann darf dieser nicht die gleiche Energie bekommen, die der erzeugende Agent ursprünglich bekam. Wenn dies der Fall wäre, könnte ein bösartiger Agent einfach einen Kindagenten erzeugen, der schläft, bis der Originalagent stirbt, dann seinerseits ein Kind erzeugt, und damit durch die Kette von Kindern unbegrenzt viel Energie bekommt. Die offensichtliche Lösung für dieses Problem ist, die Energie des Kindes dem Elternagenten in Rechnung zu stellen. Auch muß die maximale Menge an Energie, die ein Agent bekommen kann, begrenzt sein, um nicht, eine weitere potentielle Umgehungsmöglichkeit des Mechanismus zuzulassen.

7.2 Das Pfadkonzept

Wenn jeder Agent, der sich innerhalb des Agentensystems bewegt, eine Spur hinterläßt, dann kann man jeden Agenten dadurch finden, daß man dieser Spur vom Ort der Erzeugung des

Agenten bis zu seinem momentanen Aufenthaltsort folgt. Nachdem der Agent gefunden worden ist, kann er problemlos terminiert werden. Diese Idee ist im Bereich der verteilten System nicht neu, wurde aber im Bereich der mobilen Agenten bisher nicht verwendet.

7.3 Das Schattenkonzept

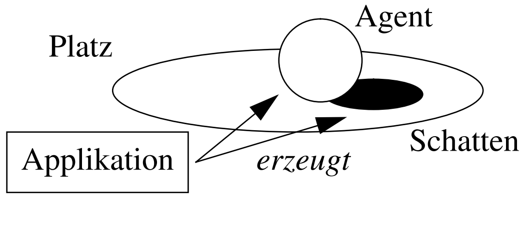


Schaubild 5: Die Erzeugung eines Schatten

Beim Schattenkonzept erzeugt die Applikation einen oder mehrere Schatten, eine Struktur auf einem Platz, der möglichst gut erreichbar ist für die zu erzeugenden abhängigen Agenten. Dieser Platz muß nicht notwendigerweise der gleiche sein, auf dem die erzeugende Applikation läuft. Jeder Agent der von der Applikation erzeugt wird, hängt von einem solchen Schatten ab. Sobald der Schatten von der Applikation entfernt wird, kann jeder abhängige Agent vom System terminiert werden. Solange der Schatten existiert, ist kein Kontakt zwischen Agenten und der Applikation oder dem System, auf dem die Applikation läuft, notwendig. Der Agent hängt nur noch vom Schatten ab, nicht mehr direkt von der Applikation. In regelmäßigen Abständen (*time to live*) kontaktiert das Agentensystem den Platz, auf dem der Schatten instantiiert wurde, um zu überprüfen, daß er nicht in der Zwischenzeit entfernt wurde. Wenn der Schatten zwischenzeitlich entfernt wurde, wird der abhängige Agent zur Waisen erklärt und terminiert.

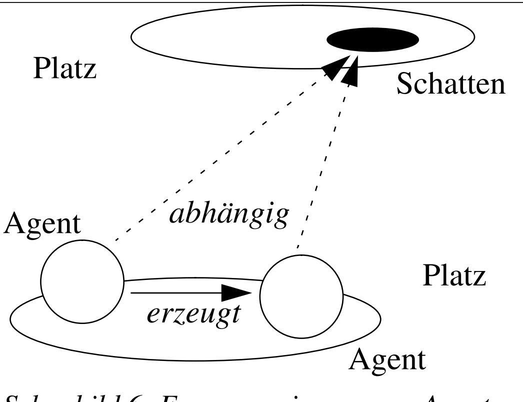


Schaubild 6: Erzeugen eines neuen Agenten

Wenn ein Agent einen neuen Agent erzeugt, dann bekommt dieser neue Agent den gleichen Schatten wie der erzeugende Agent zugeordnet, und die gleiche verbleibende Zeit bis zur nächsten Überprüfung. Es ist notwendig, die verbleibende Zeitspanne bis zur nächsten Überprüfung auf die verbleibende Zeit des erzeugenden Agenten zu beschränken, um zu verhindern, daß entsprechend der beim Energiekonzept skizzierten Möglichkeit bösartige Agenten eine unbeschränkte Lebensspanne bekommen können.

Wenn ein Platz, auf dem sich ein Schatten befindet, vom überprüfenden System nicht erreicht werden kann, wird ein *Timer* gestartet. Wenn nach Ablauf dieser Zeit das System wieder nicht erreicht werden kann, wird der *Timer* erneut gestartet und ein Zähler herabgezählt. Wenn der Zähler 0 erreicht, wird der Platz als unerreichbar angesehen, der Schatten als nicht mehr existent angesehen, und die abhängigen Agenten werden terminiert. Durch die Wahl von *Timeout* und Zähler können verschiedene Reaktionen auf Kommunikationsprobleme erreicht werden.

Diese einfache Version des Protokolls implementiert ein Konzept, das dem Energiekonzept vergleichbar ist, nur wurde der Energiebegriff ersetzt durch eine *time to live*. Der Nachteil dieses Ansatzes ist, daß unabhängig von der tatsächlichen Tätigkeit des Agenten in regelmäßigen Abständen Kommunikation stattfinden muß, um die Existenz des Schattens zu überprüfen. Der Vorteil ist, daß für die Zeit bis zur Terminierung nach der Entfernung eine obere Grenze angegeben werden kann.

In dieser einfach Form unterstützt das Protokoll nur passive Terminierung. Durch die Entfernung des Schatten werden alle abhängigen Agenten zu Waisen erklärt, und nach Ablauf der der *time to live* ist garantiert, daß alle Agenten terminiert wurden. Durch Hinzufügen des Pfadkonzeptes erlauben wir auch aktive Terminierung. Dies wird durch die sogenannten Agenten-Proxies erreicht.

7.3.1 Agenten-Proxies

Agenten-Proxies sind Datenstrukturen, die bei jedem Platz existieren, und die die Bewegungen aller Agenten verfolgen, die zu einem Schatten gehören. Sie implementieren die Pfade aus dem letzten Abschnitt, die es mittels der Erweiterung der Shattenfunktionalität ermöglichen, Agenten aktiv zu terminieren. Wir können den Beginn eines jeden Pfades dadurch ermitteln, daß wir den Platz speichern, an dem der Agent das letzte Mal überprüft wurde.

Wenn ein Agent an einem Platz ankommt, an dem noch kein Proxy existiert, wird automatisch eine solcher erzeugt (Schaubild 7a). Sobald der Agent auf einen anderen Platz migriert, wird der Zielplatz als Teil der "Reiseroute" zusammen mit der "time to live" abgespeichert (Schaubild 7b).

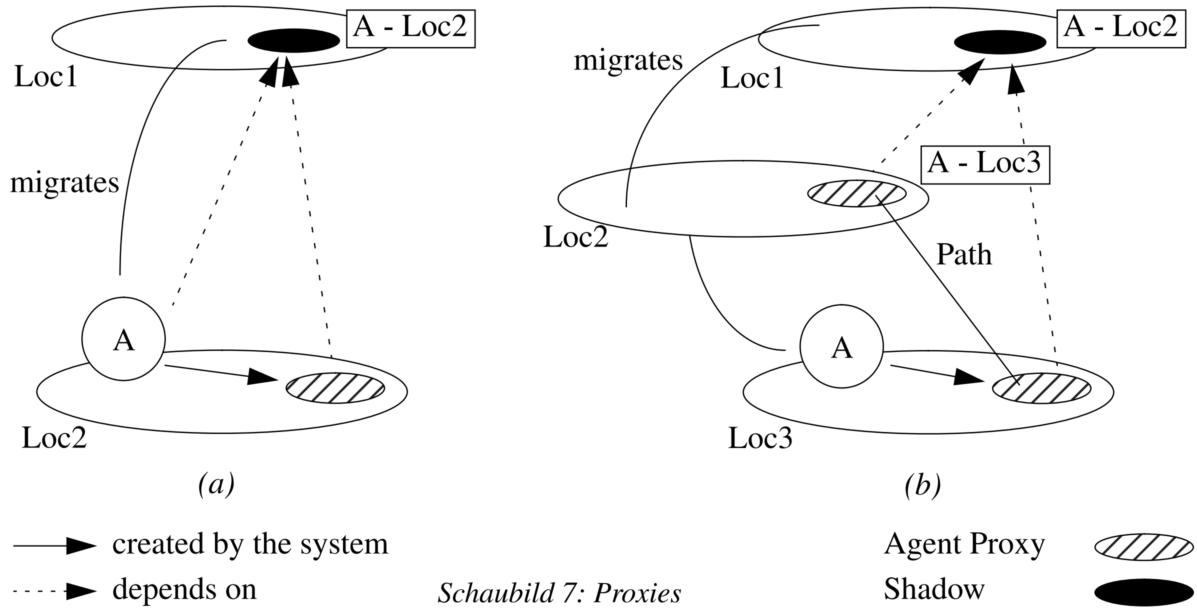


Schaubild 7: Proxies

Wenn das Ende des "time to live" erreicht ist, wird der Schatten des Agenten um eine Verlängerung des Lebenszeit des Agenten gebeten, daher wird ihm der neue Platz des Agenten bekannt gemacht (Schaubild 8a). Der Pfad, der bei den verschiedenen entlang des Migrationsweges des Agenten gespeichert ist, ist nun überflüssig und kann mithilfe des "time to live" gelöscht werden

(Schaubild 8b). Ein Eintrag kann ebenfalls gelöscht werden, wenn der Agent zu diesem Platz zurückmigriert (in diesem Fall ist das einfach eine Abkürzung des nun zirkulären Pfades.

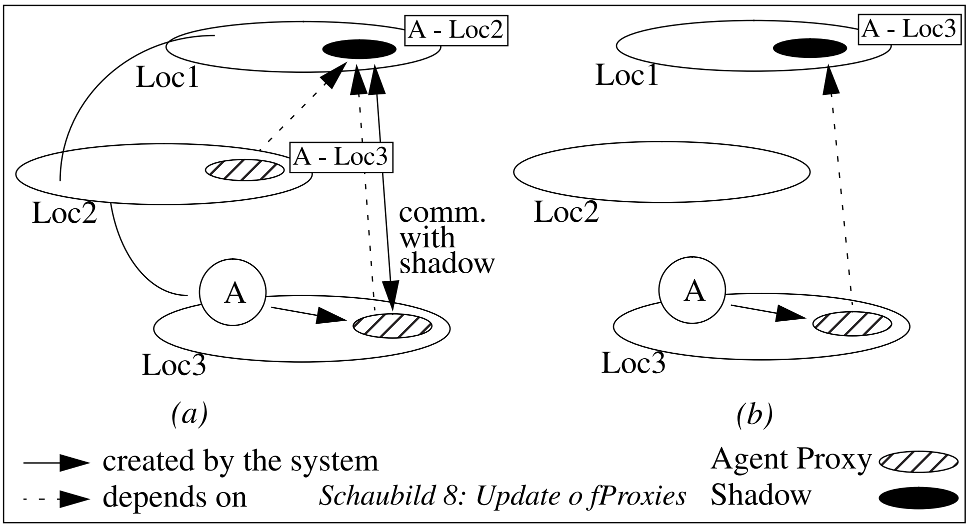


Schaubild 8: Update o fProxies

Ein Proxy existiert genau so lange, wie er Einträge enthält. Ist dies nicht mehr der Fall, kann er gelöscht werden. Das ist besonders dann hilfreich, wenn die Agenten aktiv terminiert werden. In diesem Fall werden alle Einträge aus dem Proxy gelöscht, was es dem System erlaubt, auch den Proxy zu löschen.

Nähere Informationen zu diesem Kapitel finden sich in [Bau97].

8 Anwendungen

Aus verschiedenen Anlässen wurden auf der Basis von Mole prototypische Anwendungen als Studien- oder Diplomarbeiten implementiert.

8.1 Reiseroutenplanung [Pin96]

Um die Eignung von Mobile-Agenten-Systemen in Informationssystem-Anwendungen zu eruieren wurde in Zusammenarbeit mit der Daimer-Benz AG wurde eine prototypische Anwendung erstellt, die in der Lage ist, basierend auf Mole, ausgehend von einem Startort, Zielort sowie einer Startzeit, eine Reiseroute zu bestimmen, die aus mehreren Verkehrsmitteln bestehen kann. Dazu fragt ein Routenplanungsagent mehrere Informationsquellenagenten, die Informationen von im Internet zugänglichen Informationsquellen wie z.B. einer über eine WWW-Seite abfragbare Datenbank holen können. Die Vorteile von mobilen Agenten liegen hier einerseits in der einheitlichen Dienstearchitektur, die es Programmen erlaubt, Daten anderer Programme über Agenten abzufragen, darin, daß in Agentensystemen nicht nur Dienstinanspruchnahmen, sondern auch Dienste selbst dynamisch angeboten werden können sowie in der Fähigkeit von Agenten, von konkreten Implementierungen auf der Ebene des Informationsaustausches abstrahieren zu können. Der Hauptnachteil des gewählten Ansatzes lag dabei nicht auf der Seite der mobilen Agenten, sondern in der Tatsache, daß die häufigen Änderungen im Zugriff auf die WWW-basierten Informationen, die einem menschlichen Benutzer nicht stören, zu der Notwendigkeit einer ebenso häufigen Änderung der Informationsquellenagenten führte, und die Anwendung damit chronisch "hinterherhinkte".

8.2 Active Mail [Nit97]

Active Mails sind elektronische Briefe, die nicht nur statische Daten, sondern auch Programme transportieren und zu bestimmten Zeitpunkten ausführen können. Active Mail ist eine Plattform für viele mögliche Anwendungsgebiete, z.B. CSCW. Während alle anderen Active-Mail-Ansätze eigene Laufzeitumgebungen benutzen, sollte diese Arbeit auf Mole aufbauen, da Active Mails und mobile Agenten viele Eigenschaften teilen, und weil so Active Mails als Agenten die Systemmechanismen von Mole (z.B. in Bezug auf Sicherheit oder Terminierung) nutzen kann. Die Arbeit umfasste das Erstellen eines Werkzeugs zum Erzeugen und Ansehen von Active Mails und einer Beispielanwendung, die auf Active Mail basiert. Diese Beispielanwendung bestand in einer Terminvereinbarungsapplikation. Es zeigte sich, daß ein solcher Ansatz sehr sinnvoll ist, weil er zum einen die unmittelbare Nützlichkeit eines Agentensystems erhöht und damit zu einer größeren Akkzeptanz beiträgt, und zum anderen dem Anwendungsgebiet durch die Systemmechanismen mehr Möglichkeiten angeboten werden können.

8.3 Suchen von Usenet-News [Wie97]

Um verschiedene Mobile-Agenten-Systeme untereinander besser vergleichen zu können wurde auf dem 2. Deutschen Mobile-Agenten-Treffen (DeMAT) in Waldfischbach beschlossen, daß alle vertretenen Gruppen eine bestimmte Anwendung für ihr System implementieren, um so anhand der gleichen Aufgabenstellungen z.B. den Aufwand bestimmen zu können, mit dem Probleme in dem jeweiligen System gelöst werden. Diese "Referenzapplikation" war ein Programm, mit dem es möglich sein sollte, in den Artikeln der Usenet-News nach Stichworten zu suchen, ähnlich wie die Suchmaschinen für WWW anbieten. Usenet-News ist ein elektronisches Informationssystem, mit dem weltweit Menschen über ein breites Spektrum an Themen diskutieren und Informationen austauschen. Spezialisierte Rechner, sog. Newsserver, verteilen diese Informationen weltweit. Die daraufhin durchgeführte Studienarbeit implementierte daher genau diese Referenzapplikation für Mole und verglich deren Möglichkeiten mit existierenden News-Suchmaschinen wie DejaNews. Dabei zeigte sich, daß dieser Ansatz gegenüber existierenden Index-Suchmaschinen nur dann einen Vorteil bietet, wenn aktuellere Ergebnisse nötig sind, oder wenn sich Index-Suchmaschinen aufgrund eines zu hohen Datenvolumens nicht einsetzen lassen.

9 Das Mole-System

Um die Praktikabilität der Forschungsergebnisse demonstrieren zu können und um eine Plattform für zukünftige Forschungen zu haben, wurde der als Vorleistung in AIDA I eingebrachte Systemprototyp neu implementiert und als Mole Version 1.0 auf dem Internet frei zugänglich gemacht. Das Mole-System war eines der ersten frei verfügbaren Mobile-Agenten-Systeme, eines der ersten, das Java verwendete und eines der wenigen, die im Quellcode verfügbar waren. Dadurch wurde das Mole-Projekt auch international nach außen sichtbar und weithin bekannt. Das Mole-System wird von einer Reihe von Anwendern im akademischen und im indutriellen Bereich benutzt, wie z.B. von der Universität Genf, der Universität Zürich, Siemens, Sony usw.

Die erste Version von Mole wurde im Mai 1996 freigegeben und wurde seitdem etwa 500 mal von mehr als 300 verschiedenen Rechnern geholt.

Die zweite Version von Mole wurde im Juli 1997 freigegeben und von Mitte Juni bis Mitte August ca 180 mal von 130 Rechnern aus 23 Ländern geholt.

Das Mole-System, das zur Zeit einen Umfang von etwa einem Megabyte Code hat, aus etwa 120 Klassen und 600 Methoden besteht, ist komplett in Java implementiert. Für das System existieren drei grafische Demos, die es erlauben, die Mobile-Agenten-Idee und das Mole-System anschaulich vorzuführen, darunter eine Mole-basierte Implementierung des Spiels "Scotland Yard" [Ahe97a], das am IPVR bereits einmal für immobile Agenten auf einem gemeinsamen Rechner implementiert wurde. Ein Teil von Mole ist ein graphischer Monitor [Bec96], mit dem sich pro Platz die Agenten und angebotenen Dienste anzeigen lassen kann sowie die Kommunikationsinteraktionen untersuchen kann.

Im Gegensatz zum Modell führt die Implementierung unterhalb von Plätzen eine weitere Komponente, die *Engine*, ein. Engines sind Mengen von Plätzen, die gemeinsam von einem Javainterpreter und damit einem Betriebssystemprozeß ausgeführt werden. Damit sind Plätze sehr "billige" Komponenten und können z.B. zur Strukturierung oder zum lokalen Testen oder Betreiben von Konfigurationen aus mehreren Plätzen eingesetzt werden.

10 Verwandte Arbeiten

Das Jahr 1996 und vor allem 1997 erlebte einen Boom an neuen Agentensystemen (siehe Tabelle 3). In diesem Abschnitt sollen sie kurz im Bezug auf die Forschung bei AIDA I charakterisiert werden.

Ara [Pei97] ist ein Mehrsprachensystem, das nach Tcl und C jetzt auch Java als Sprache anbietet. Da hier immer eine starke Migration vorhanden sein soll, wurde diese auch für Java realisiert, wobei dieses ohne große Eingriffe in die Laufzeitumgebung von Java dadurch vorgenommen werden konnte, daß einfach der komplette Zustand eines Javainterpreters verschickt wird. Da in Ara ein Agent immer durch einen Interpreter ausgeführt wird, war dieses Vorgehen möglich und effizient. Als Mehrsprachensystem leidet ARA im Bereich der Kommunikation daran, daß als Datentyp vor allem Strings, also der kleinste gemeinsame Nenner der Sprachen, verwendet wird, was eine Spezifikation von Dienstinanspruchnahmen auf der Ebene von Bytes notwendig macht.

ffMAIN [LDD95] ist ein Mehrsprachensystem, das zur Kommunikation einen lokalen Tupelspace benutzt und HTTP als Transfer- und Kommunikationsprotokoll verwendet. Auch hier tritt das Problem auf, daß die Ebene der Kommunikation umformatierte Strings sind.

Tacoma [JRS95] ist ein Mehrsprachensystem, das eher auf der Ebene der Remote Evaluation als auf der Agentenebene anzusiedeln ist, da im wesentlichen Code verschickt wird, sowie ein Datenpaket, daß die ausgeführte Einheit selbst zusammen- und entpacken muß.

AgentTcl [Gra96] ist ein fortgeschrittenes Agentensystem, das inzwischen auch einen Authentifikationsmechanismus implementiert hat.

Aglets [IBM96] ist ein Mobile-Agenten-System auf der Basis von Java, das von IBM Japan entwickelt und gepflegt wird. Es werden einige Anstrengungen von Seiten von IBM unternommen, um dieses System zu einer Basis für kommerzielle Anwendungen zu machen. Besonders hervorzuheben sind die Aktivitäten dieser Gruppe im Bereich Sicherheit. Ein kürzlich erschienener Text deutet daraufhin, daß das Aglets-System bald ein Rahmenwerk beinhalten wird, das die "konventionelleren" Funktionen wie Authentifizierung und Ressourcenkontrolle bereitstellt.

Concordia [WPW97] ist ein System von Mitsubishi, USA, das ebenfalls auf Java aufsetzt. Es soll ebenfalls eine Basis für agentenbasierte, robuste Anwendungen bieten. Es legt zur Zeit einen besonderen Schwerpunkt auf die Integration von Datenbanken in Agentenanwendungen.

Eine Besonderheit dieses Systems sind die Itinaries oder "Reisepläne", die im Voraus erstellt werden und zu jeden zukünftigen Ankunftsort die Funktionen nennen, die dort ausgeführt werden sollen. Itinaries können dynamisch modifiziert werden.

CyberAgents [FTP97] war ein Produkt von FTP Software Inc., das vor allem zum Management von PC-Netzen eingesetzt werden sollte. Da der Anwendungsbereich geschlossen war, war das Agentensystem auch nur sehr einfach gestaltet. Im Wesentlichen konnte man auf eine sehr einfache, graphische Weise existierende Agenten zu einer festen Liste von Rechnern schicken und sie dort beliebige Programme ausführen lassen. Die Agenten authentifizierten sich gegenüber den Rechnern mit einem Passwort, die Agenten selbst hatten alle Rechte des Benutzers unter dessen Kennung sie liefen. FTP nahm dieses Produkt jedoch 1997 vom Markt und integrierte es in andere Produkte.

Java-2-go [LM96] ist ein frühes System, das auf Java basiert, und nicht mehr weiterentwickelt zu werden scheint.

Kafka [Nis97] ist ein System, das ebenfalls auf Java aufbaut und von Fujitsu, Japan stammt. Die besonderen Eigenschaften dieses Systems sind:
- Agenten können ihren Programmcode zur Laufzeit ändern
- Möglichkeit zur Remote Evaluation von Code statt der Migration von Agenten
- Ein verteiler Verzeichnisdienst für Agentennamen

Messengers [Tsch95] sind eigentlich keine mobilen Agenten, sondern eine Art Vorstufe dazu. Zwar bestehen auch messengers aus Code (in M0 - einer Postscript-ähnlichen Sprache), das Laufzeitsystem bietet aber z.B. keine Maßnahmen an, um die Mächtigkeit einzelner Messengers zu beschränken.

MOA [Mil97] ist das java-basierte Agentensystem der "Open Group". Es ist zur Zeit nicht frei erhältlich, enthält aber einige interessante Konzepte:
- Kommunikation findet mittels Kanälen zwischen Agenten statt
- Plätze sind keine Agententreffpunkte, sondern "Agentengaragen", die für Agenten an Knoten angelegt werden und in denen Agenten Daten hinterlassen können. Plätze können verschachtelt sein
- Agentennamen bestimmten den Ursprungsort, der die Administration von Agenten übernimmt

MonJa [Mon97] ist ein Mobile-Agenten-System von Mitsubishi, Japan, über das zur Zeit nur japanische Informationen verfügbar sind.

Odyssey [GM97] ist das Nachfolgeprodukt zu Telescript. Es stellt ein Mobile-Agenten-System auf der Basis von Java und den Konzepten von Telescript dar, wird aber auch weiterentwickelt.

Telescript [GM96] ist als Produkt mit Sicherheit immer noch das am meisten entwickelte Agentensystem. Es verwendet eine eigene, Smalltalk-ähnliche Programmiersprache und hat ein sehr ausgearbeitetes Verarbeitungsmodell. Da diesem System mit dem Aufkommen von WWW und Java kein Markterfolg beschieden war, entwickelt jetzt auch die Firma, die dieses System erstellt hat, General Magic, ein Java-basiertes System. Odyssey. Sie hält außerdem ein Patent über ein Verfahren, das sehr stark an starke Migration erinnert.

Voyager [Obj97] ist ein kommerzielles Produkt für Java, das seine Systemdienste um einen ORB herum entwickelt und das einige auf kommerzielle Anwendungen ausgerichtet ist.

Tabelle 3: Agentensysteme

Systemname	Sprachen	Institution	Verfügbar?
ARA	Tcl, C, Java	Universität Kaiserslautern	ja (frei)
ffMAIN	Tcl, Perl, Java	Universität Frankfurt	nein
Tacoma	Tcl, C, Python, Scheme, Perl	Cormell (USA), Tromso (Norwegen)	ja (frei)
AgentTcl	Tcl	Dartmouth College, USA	ja (frei)
Aglets	Java	IBM, Japan	ja (nur binär)
Concordia	Java	Mitsubishi, USA	ja (nur binär)
CyberAgents	Java	FTP Software, Inc., USA	nicht mehr
Java-2-go	Java	University of California at Berkeley	ja
Kafka	Java	Fujitsu, Japan	ja (nur binär)
Messengers	M0	Universität Zürich, Schweiz	ja (frei)
MOA	Java	The Open Group, USA	nein
Mole	Java	Universität Stuttgart	ja
MonJa	Java	Mitsubishi, Japan	ja (nur binär)
Odyssey	Java	General Magic, USA	ja (nur binär)
Telescript	Telescript	General Magic, USA	ja (nur binär)
Voyager	Java	ObjectSpace, Inc. , USA	ja (nur binär)

11 Probleme von Mobilen-Agenten-Systemen

Obwohl mobile Agenten eine ganze Reihe von möglichen Vorteilen gegenüber herkömmlichen Client-Server-Architekturen haben, werden Mobile-Agenten-Systeme noch nicht in größerem Umfang eingesetzt. Die Gründe dafür sind nicht vollständig zu eruieren, aber u.a. folgende Probleme könnten zu diesem Umstand beitragen:

Der Sicherheitsaspekt ist noch nicht gelöst
Offene Fragen wie beim Problem der Sicherheit gegenüber böswilligen Interpretern schrecken noch viele potentielle Anwender ab, da dieser Bereich immer wichtiger wird, und die Grundlage für den Einsatz von immer mehr Anwendungen wird.

Es fehlt eine "Killerapplikation"
Wie man im Fall WWW gesehen hat, benötigen einige Technologien, auch wenn sie nützlich sind, manchmal Anwendungen, die erstens so vorher noch nicht bestanden, die zweitens einen starken Bedarf verzeichnen und die drittens auf der Technologie basieren, die sie in der Folge populär machen. Eine solche Anwendung, obwohl zur Zeit nicht in Sicht, könnte auch mobile Agenten zur einer kritischen Masse verhelfen.

Die Mobile-Agenten-Technik ist noch nicht auf dem Stand, der bei Client-Server-Systemen Standard ist, und der im kommerziellen Umfeld gebraucht wird
Client-Server-Systeme existieren im Vergleich zu Mobilen-Agenten-Systemen sehr viel länger, und die Forschung auf diesem Gebiet hat bereits viele Aspekte gelöst, die für robuste und effiziente kommerzielle Anwendungen notwendig sind. Aufgrund des fehlenden Alters von mobilen Agenten müssen diese Aspekte hier erst noch gelöst werden, sofern sie nicht aufgrund der größeren Mächtigkeit des Agentenansatzes übernommen werden können.

Die Verbreitung der Technik ist noch zu gering, als daß sie nützlich sein könnte
Ein Mobile-Agenten-System ist umso nützlicher in einem offenen Umfeld, je mehr Dienste angeboten werden, und umso mehr bestehende Datenquellen lokal per Agent erreichbar sind. Solange es aber keine Benutzer gibt, die diese Technologie einsetzen, gibt es auch keine Dienstanbieter, die ein solches System auf ihren Servern einsetzen. Um einem Teil dieses Problems entgegenzuwirken wurde Mole in der letzten Zeit so modifiziert, daß es als Applet auf Java-fähigen Browsern ablaufen kann (siehe [Ahe97b]). Ein Platz, der auf einem Browser läuft, hat nicht die technischen Möglichkeiten eines Platzes, der als normale Anwendung gestartet wird. Daher benötigen Browserplätze eine "Relay"-Komponente auf einem Platz im "normalen", festen Agentensystem. Mithilfe einer solchen Architektur ist es dann möglich, daß Benutzer Agenten lokal auf ihrem Browser starten und ablaufen lassen können und diese dann weitermigrieren lassen. Damit vergrößert sich die Basis an Benutzern dramatisch, die ohne weitere Maßnahmen wie der Installation eines Agentensystems Mole-Applikationen starten und benutzen können.

12 Zusammenfassung

Zusammenfassend läßt sich sagen, daß die gesetzten Ziele der ersten Projektphase erreicht wurden.

Es wurde ein Modell für die agentenbasierte Verarbeitung erarbeitet, das es erlaubt, Anwendungen mithilfe von mobilen Benutzer- und immobilen Systemagenten zu modellieren, es wurden Systemmechanismen für ein Mobile-Agenten-System erarbeitet, die es erlauben, Agenten interagieren und migrieren zu lassen. Weiter wurde der Bereich Sicherheit in einer ersten Analyse angegangen und Vorstellungen darüber entwickelt, wie ein Rahmenwerk für ein Agentensystem in diesem Bereich auszusehen habe. Darüberhinaus wurden Protokolle zur Waisenerkennung und zur Terminierung entworfen, sowie einige Beispielanwendungen auf der Basis des implementierten Agentensystems erstellt.

Ergebnisse des Projektes wurden auf dem Joint W3C/OMG Workshop on Distributed Objects and Mobile Code, 1996 in Boston, auf dem zweiten ECOOP Workshop über Mobile Object Systems, 1996 in Linz, auf dem dritten ECOOP Workshop über Operating System support for Mobile Object Systems, 1997 in Jyväskylä, Finnland, auf dem ersten internationalen Workshop über Mobile Agents, MA'97, 1997 in Berlin, auf der International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'97), 1997 in Las Vegas, sowie auf der International Working Conference on Distributed Applications and Interoperable Systems (DAIS'97), 1997 in Cottbus vorgestellt sowie in drei Fakultätsberichten veröffentlicht. Das Projekt wurde auf der CeBIT'97 und bei verschiedenen Gelegenheiten anderen Gruppen und in der Industrie präsentiert. Es wird angestrebt, weitere Ergebnisse auf weiteren Workshops und Konferenzen vorzustellen.

In der noch verbleibenden Zeit soll das Mole-System an Java 1.1 angepasst werden, das Performanzmodell soll erweitert werden, und es soll begonnen werden, die Komponenten des Sicherheitsbereich zu konzipieren, die mit existierenden Verfahren gelöst werden können. Zuletzt soll ein EventManager konzipiert werden, der in der Lage ist, mobile Teilnehmer zu bedienen, und der auch für eine große Zahl von Teilnehmern und Events performant genug ist.

13 Bibliographie

[Ahe97a] Aheimer, Holger: Mister X: Ein Spiel als Beispiel für die Koordinierung von Mobilen Agenten. Studienarbeit Nr. 1595, Universität Stuttgart, Fakultät Informatik, 1997.

[Ahe97b] Aheimer, Holger: Erschließung von WWW-Browsern und -Servern für die Ausführung von mobilen Agenten. Diplomarbeit, Universität Stuttgart, Fakultät Informatik, 1997.

[BHR97] Baumann, Joachim; Hohl, Fritz; Radouniklis, Nikolaos; Rothermel, Kurt and Straßer, Markus: Communication Concepts for Mobile Agent Systems, in: Proceedings of the First International Workshop on Mobile Agents, MA'97 (eds. K. Rothermel and R. Popescu-Zeletin), LNCS 1219, Springer, 1997.

[BR97] Baumann, Joachim and Radouniklis, Nikolaos: Agent Groups in Mobile Agent Systems, in: Proceedings of the International Working Conference on Distributed Applications and Interoperable Systems (DAIS'97). To appear.

[Bau97] Baumann, Joachim: A Protocol for Orphan Detection and Termination in Mobile Agent Systems. Universität Stuttgart, Fakultät Informatik, Fakultätsbericht Nr. 1997/09.

[Bec96] Beck, Bernhard: Konzeption und Implementierung eines graphischen Monitors für ein Mobile-Agenten-System. Studienarbeit Nr. 1523, Universität Stuttgart, Fakultät Informatik, 1996.

[Bec97] Beck, Bernhard: Terminierung und Waisenerkennung in einem System mobiler Software-Agenten. Diplomarbeit Nr. 1472, Universität Stuttgart, Fakultät Informatik, 1997.

[FG96] Franklin, Stan.; Graesser, Art: Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents, in: Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages, Springer-Verlag, 1996.

[FGS96a] Farmer, William; Guttmann, Joshua; Swarup, Vipin: Security for Mobile Agents: Issues and Requirements, in: Proceedings of the National Information Systems Security Conference (NISSC 96), 1996

[FGS96b] Farmer, William; Guttmann, Joshua; Swarup, Vipin: Security for Mobile Agents: Authentication and State Appraisal, in: Proceedings of the European Symposium on Research in Computer Security (ESORICS), 1996

[FTP97] FTP Software, Inc.: FTP Software Agent Technology.

http://www.ftp.com/product/whitepapers/4agent.htm

[GM96] General Magic (1996) The Telescript Reference Manual. http://www.genmagic.com/Telescript/Documentation/TRM/

[GM97] General Magic (1997) Odyssey. http://www.genmagic.com/agents/odyssey.html

[Gra96] Gray, Robert: Agent Tcl: A flexible and secure mobile-agent system, in: Proceedings of the Fourth Annual Tcl/Tk Workshop (TCL 96), Monterey, California, July 1996

[HKB97] Hohl, Fritz; Klar, Peter and Baumann, Joachim: Efficient Code Migration for Modular Mobile Agents, in: Proceedings of the Third ECOOP Workshop on Mobile Object Systems: Operating System support for Mobile Object Systems. To appear.

[Hogg91] Hogg, J.: Islands: Aliasing Protection in Object-Oriented Languages, in: Proc. OOPSLA '91.

[Hoh97] Hohl, Fritz: An approach to solve the problem of malicious hosts. Bericht Nr. 1997/03, Fakultät Informatik, Universität Stuttgart.

[IBM96] IBM Tokyo Research Labs (1996) Aglets Workbench: Programming Mobile Agents in Java. http://www.trl.ibm.co.jp/aglets

[JRS95] Johansen, D., van Renesse, R. and Schneider, F. (1995) An Introduction to the TACOMA Distributed System - Version 1.0. Technical Report 95-23, University of Tromso.

[Joc97] Jochum, Eric: Konzeption und Implementierung eines Energiekonzeptes für ein Mobile-Agenten-System. Studienarbeit, Universität Stuttgart, Fakultät Informatik, 1997.

[KLO97] Karjoth, Günter; Lange, Danny; Oshima, Mitsuru :A Security Model for Aglets, in: IEEE Internet Computing, Vol. 1, No. 4, July - August 1997. http://computer.org/internet/ic1997/w4068abs.htm

[Kir96] Kirschner, Frank: Konzeption und Implementierung eines Mehrbenutzerspiels auf der Basis von Mobilen Agenten. Studienarbeit Nr. 1549, Universität Stuttgart, Fakultät Informatik, 1996.

[Kla96] Klar, Peter: Persistenz als Basis der Migration in einem Mobile-Agenten-System : Design und Implementierung. Studienarbeit Nr. 1522, Universität Stuttgart, Fakultät Informatik, 1996.

[Kla97] Klar, Peter: Ein verteiltes Serversystem für die Codemigration mobiler Agenten. Diplomarbeit Nr. 1470, Universität Stuttgart, Fakultät Informatik, 1997.

[Kub97] Kubach, Uwe: Redesign/Reimplementation der Kommunikationsmechanismen in Mole. Studienarbeit, Universität Stuttgart, Fakultät Informatik, 1997.

[Kuh97] Kuhn, Wilfried: Ressourcenkontrolle in einem Mobile-Agenten-System. Diplomarbeit Nr. 1468, Universität Stuttgart, Fakultät Informatik, 1997.

[LDD95] Lingnau, Anselm; Drobnik, Otto; Dömel, Peter: An HTTP-based Infrastructure for Mobile Agents, Fourth WWW Conference 1995, http://www.w3.org/pub/Conferences/WWW4/Papers/150/

[LM96] Li, Weiyi; Messerschmitt, David: WWW page of the Java-2-go Project. http://ptolemy.eecs.berkeley.edu/dgm/javatools/java-to-go/

[MAF97] Crystaliz Inc., General Magic Inc., GMD Fokus and IBM Corp. Joint Submission for the OMG Mobile Agent Facility Specification, 1997

[Mil97] Milojicic, Dejan: The MOA Project. Talk at the University of Stuttgart, 11.7.1997

[Mon97] WWW page of the MonJa Project. http://www.melco.co.jp/rd_home/java/monja/

[NS78] Needham, R,; Schroeder, M.: Using Encryption for Authentication in Large Networks of Computers, in: Communications of the ACM, Vol. 21, Nr. 12, Dezember 1978

[Nis97] Nishigaya, Takashi: Design of Multi-Agent Programming Libraries for Java. White Paper. Fujitsu Laboratories Ltd., Kawasaki, Japan. http://www.fujitsu.co.jp/hypertext/free/kafka/paper/

[Nit97] Nitsche, Jan-Gregor: Konzeption und Implementierung eines Active-Mail-Systems auf der Basis Mobiler Agenten. Diplomarbeit Nr. 1511, Universität Stuttgart, Fakultät Informatik, 1997.

[OMG97] Object Management Group: CORBAservices - Security Service Part I, formal/97-02-20, ftp://ftp.omg.org/pub/docs/formal/97-02-20.ps

[Obj97] ObjectSpace, Inc.: WWW page of Voyager. http://www.objectspace.com/voyager/

[Ord96] Ordille, Joann J.: When agents roam, who can you trust?, in: Proc. of the First Conference on Emerging Technologies and Applications in Communications, Portland, May 1996

[Pal94] Palmer, E: An Introduction to Citadel - a secure crypto coprocessor for workstations, in: Proceedings of the IFIP SEC'94 Conference, 1994

[Pei97] Peine, H. (1997) Ara: Agents for Remote Action, in Mobile Agents: Explanations and Examples (eds. W. Cockayne and M. Zyda) Manning Publishing.

[Pin96] Pindonis, Ioannis: Kooperative Informationsbeschaffung mittels Mobiler Agenten am Beispiel Reiseroutenplanung. Diplomarbeit Nr. 1368, Universität Stuttgart, Fakultät Informatik, 1996.

[RHR97] Rothermel, Kurt; Hohl, Fritz and Radouniklis, Nikolaos: Mobile Agent Systems: What is Missing, in: Proceedings of the International Working Conference on Distributed Applications and Interoperable Systems (DAIS'97). To appear.

[Rie96] Rieg, Berthold: Robuste Warteschlangen für Nachrichten als Kommunikationsmechanismus in einemMobile-Agenten-System: Konzeption und Implementierung. Studienarbeit Nr. 1528, Universität Stuttgart, Fakultät Informatik, 1996.

[Röh96] Röhrle, Klaus: Finden von mobilen Agenten in einem weitverteilten System. Studienarbeit Nr. 1539, Universität Stuttgart, Fakultät Informatik, 1996.

[Röh97] Röhrle, Klaus: Konzeption, Implementierung und Analyse von Verwürfelungsmechanismen für Quellcode. Diplomarbeit, Universität Stuttgart, Fakultät Informatik, 1997.

[SBH97] Straßer, Markus; Baumann, Joachim and Hohl, Fritz: Mole: A Java based mobile agent system, in: Proceedings of the 2nd ECOOP Workshop on Mobile Object Systems (eds. J. Baumann, C. Tschudin and J. Vitek), dpunkt.

[SL95] Sandholm, T.; Lesser, V.: Issues in Automated Negotiation and Electronic Commerce: Extending the Contract Net Framework, in: Proceedings of the First International Conference on Multiagent Systems (ICMAS-95), 1995

[SS97] Straßer, Markus and Schwehm, Markus: A Performance Model for Mobile Agent Systems, in: Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'97), Volume II (ed. H. R. Arabnia). CSREA Press, 1997.

[Sch96] Schulz, Stefan: Konzeption und Implementierung einer graphischen Benutzeroberfläche für ein Mehrbenutzerspiel auf der Basis von Mobilen Agenten. Diplomarbeit Nr. 1413, Universität Stuttgart, Fakultät Informatik, 1996.

[Sza97] Szasz, Viktor: Protokolle zur Synchronisation mobiler Software-Agenten. Studienarbeit Nr. 1629, Universität Stuttgart, Fakultät Informatik, 1997.

[Tsch95] C. F. Tschudin. Protokollimplementierung mit Kommunikationsboten. Proceedings, Kommunikation in Verteilten Systemen '95, Springer Verlag 1995.

[Vig97] Vigna, G. (1997) Protecting Mobile Agents through Tracing. Accepted Submission for the Third ECOOP Workshop on Mobile Object Systems: Operating System support for Mobile Object Systems.

[Voi96] Voigt, Thiemo: Entwicklung und Implementation eines Modells zur quantitativen Beurteilung derImplementation und der Anwendung von Remote-Execution-Mechanismen. Diplomarbeit Nr. 1436, Universität Stuttgart, Fakultät Informatik, 1996.

[WPW97] Wong, D., Paciorek, N., Walsh, T. et al. (1997) Concordia: An Infrastructure for Collaborating Mobile Agents, in Proceedings of the First International Workshop on Mobile Agents, MA'97 (eds. K. Rothermel and R. Popescu-Zeletin), Springer.

[Wie97] Wieger, Thomas: Konzeption und Implementierung einer Suchmaschine für Usenet-News unter Verwendung von mobilen Agenten. Studienarbeit Nr. 1603, Universität Stuttgart, Fakultät Informatik, 1997.

[Zep96] Zepf, Matthias: Konzeption und Implementierung eines einfachen Orphan- Detection-Mechanismus für ein Mobile-Agenten-System. Studienarbeit Nr. 1552, Universität Stuttgart, Fakultät Informatik, 1996.