Ludwig–Maximilians–Universität München

Institut für Informatik

Mobile Agent System Architecture

—

Eine Plattform für flexibles IT–Management

Technischer Bericht 9902

Boris Gruschke, Stephen Heilbronner, Helmut Reiser

MNM
TEAM
Münchner Netzmanagement Team

Mobile Agent System Architecture

—

Eine Plattform für flexibles IT–Management

Boris Gruschke, Stephen Heilbronner, Helmut Reiser

MNM TEAM

Münchner Netzmanagement Team

Institut für Informatik
Ludwig–Maximilians–Universität München
Oettingenstr. 67, D-80538 München
masa@informatik.uni-muenchen.de
http://wwwmnmteam.informatik.uni-muenchen.de

August 1999

Inhaltsverzeichnis

1 Einleitung **1**

1.1 Historie der Mobile Agent System Architecture 3

1.2 Übersicht 4

2 Einsatzszenarien für MASA **7**

2.1 Policy Management 7

2.2 Nomadic Computing 9

2.3 Eventkorrelation mit Abhängigkeitsgraphen 10

2.4 Resultierende Anforderungen an MASA 12

3 Agenten und Management **15**

3.1 Klassische Managementagenten 15

3.2 Management by Delegation 16

3.3 Plattformen für Mobile Agenten 17

3.4 Common Object Request Broker Architecture 21

3.5 Mobile Agent System Interoperability Facility 23

4 Die MASA–Architektur **27**

4.1 Übersicht 27

4.2 Agentensystem 30

4.3 MASA–Agenten 31

4.3.1 Lebenszyklus eines Agenten 32

4.3.2 Semantik der Migration 34

4.4 MASA–Plattformdienste 34

4.4.1 Naming Service 35

4.4.2 Event und Notification Service 36

4.5 MASA–Systemerweiterungen 37

4.5.1 WWW–Agent 37

4.5.2 Graphische Benutzerschnittstelle von MASA 37

4.5.3 Voyager Gateway 38

5 Implementierung 41

5.1 Verwaltung des Quellcodes mit CVS 41

5.2 Installation 43

5.2.1 Verzeichnisstruktur 43

5.2.2 Makefile–Schema 44

5.2.3 Installationsvorgang 45

5.3 Implementierte MASA–Agenten 45

5.3.1 Abhängigkeitsgraphen — DGAgents 46

5.3.2 IPRouting Agent 46

5.3.3 Policy–Managementagenten 48

5.3.4 DHCP–Server 48

5.3.5 Managementagent für einen Linux–basierten Switch . . 49

5.3.6 Corba Topology Service 49

5.3.7 Email Management Agent 50

6 Ausblick 53

6.1 Security Integration 53

6.2 Weitere Corba Dienste 56

6.3 Agenten als Beans 56

6.4 Domänen 57

A Empfohlene Literatur 61

A.1 Einführende Bücher 61

A.2 Standards und Spezifikationen 61

A.3 WWW 62

Abbildungsverzeichnis **64**

Literaturverzeichnis **66**

Kapitel 1

Einleitung

Das Management verteilter IT–Infrastrukturen ist einer der am schnellsten wachsenden Bereiche innerhalb des IT–Betriebs. Die Gründe hierfür liegen insbesondere in der manuell kaum mehr beherrschbaren Komplexität der Netze, Systeme und Anwendungen. Schlecht beherrschbare Systeme, die zu kritischen Elementen in den Wertschöpfungsprozessen der Anwender geworden sind, verursachen im Fehlerfall hohe Kosten und Aufwand. Es ist daher nötig, durch geeignete (Management–)Systeme die Überwachung und Steuerung der IT–Infrastruktur zu gewährleisten. Diese Systeme stellen selbst komplexe, große, verteilte Anwendungen dar.

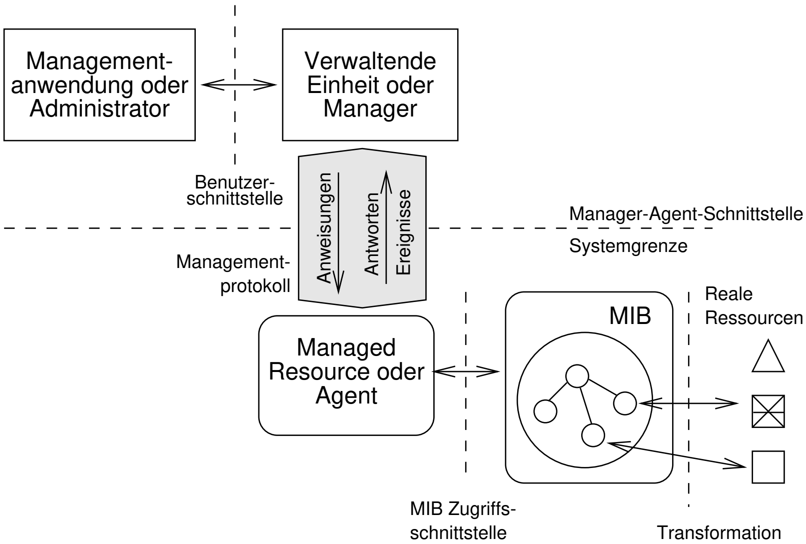

Abbildung 1.1: Aufbau eines Managementsystems (nach [HAN 99])

Klassische Managementsysteme

Traditionelle Lösungen für Managementaufgaben bestehen im Aufbau eines *Managementsystems*, das einen *Manager* (meist in Form einer sog. *Managementplattform*) und (darauf aufbauend) eine Vielzahl dedizierter *Managementanwendungen* umfaßt. Die Plattform vermittelt die Kommunikation von den Anwendungen zu den *Managementagenten* auf den verwalteten Ressourcen (*Managed Resources*). Bei diesen kann es sich um Hardware, z.B. Endsysteme wie PC–Clients, Terminals, Server, Netzinfrastrukturkomponenten oder Vermittlungsanlagen, oder um darauf ablaufende Software, z.B. Betriebssysteme, Middleware, Datenbanken und unternehmenskritische Anwendungen wie SAP R/3, handeln (vgl. Abbildung 1.1).

An dieser Stelle kann nicht auf die gesamte Problematik des IT–Managements eingegangen werden. Eine umfassende Einführung findet sich in [HAN 99] und [HAN 99a].

Modifizierbare Funktionalität

Allen klassischen und in der Praxis verwirklichten Ansätzen ist gemeinsam, daß die Management–*Funktionalität*, die durch die Agenten implementiert ist, zur Entwicklungszeit fest definiert wird. Sie ist damit weder zur Installations– noch zur Laufzeit veränderbar.

Heutige Daten– und Telekommunikationsnetze und insbesondere die darauf implementierten, verteilten Anwendungen unterliegen jedoch einem immer rascheren Wandel, der sich auch im Bedarf für ein flexibles und neuen Anforderungen angepaßtes Management äußert. Die Agenten für das Management der Ressourcen müssen also zunehmend auch nach der Inbetriebnahme in ihrer Funktionalität modifiziert werden können.

Offene Schnittstellen

Es wird zunehmend akzeptiert, daß auch Agenten nicht nur von einem Hersteller implementiert werden können, sondern auch in ihrem Inneren offene Schnittstellen aufweisen müssen, um Module unterschiedlicher Hersteller aufnehmen zu können. Die Offenlegung und Standardisierung von Schnittstellen zwischen allen Teilen des Management*systems* ist daher eine weitere Notwendigkeit zur Erhöhung der Flexibilität.

Transfer neuer Funktionalität

Die Verwirklichung einer Schnittstelle zum Laden neuer Funktionalität vom Manager auf den Agenten scheiterte jedoch immer wieder aus zwei Gründen:

1. Es konnte keine allgemein akzeptierte Implementierungssprache gefunden werden, die allen Bedürfnissen wie Performanz, Größe des ausführbaren Codes, Flexibilität, verifizierbare Implementierung und kontrollierbare Systemschnittstellen gerecht wurde.

2. Es gab keine geeigneten abstrakten Beschreibungsmittel, die es erlaubt hätten, das Verhalten von Managed Resources und daraus abgeleitet deren Managementbedarf zu beschreiben.

Corba und Java

Das Aufkommen von **Java** seit Mitte der Neunziger Jahre sowie die Etablierung von **Corba** als industrieweit[1] anerkannten Standard für Middleware ermöglichte jedoch, die Geschwindigkeit und die Art und Weise der Etablierung neuer Managementsysteme zu verbessern. Die Verwendung einer **Basisplattform** für häufig benötigte (Management–)dienste und einer einzigen

[1] DCOM von Microsoft ist im Managementumfeld kaum von Bedeutung.

Programmiersprache mit einer großen Anzahl standardisierter, allgemein anerkannter Schnittstellen reduziert den Aufwand, der für die Implementierung eines Managementsystems in einer ansonsten heterogenen Umgebung erforderlich ist.

Die Eignung von Corba als Basis für die Integration verteilter Managementsysteme läßt sich nicht nur aus der (immer noch wachsenden) Präsenz auf einschlägigen Konferenzen ersehen, sondern aus den Ankündigungen und Vorstellungen der Produkte führender Hersteller von Managementsoftware (IBM Tivoli TME, CA UniCenter, HP OpenView etc.). Das starke Interesse dieser Hersteller zeigt sich in der großen Aktivität in der entsprechend ausgerichteten Arbeitsgruppe *Telecom Domain Task Force (DTF)* innerhalb der *Object Management Group (OMG)* und in der Technology Integration Map [TMF 909] des TeleManagement Forums.

Die Abstützung auf die allgemein akzeptierte Middleware Corba vereinfacht zunehmend auch die Integration von Managementsystemen über Betreibergrenzen hinweg – ein Ziel, das zunächst nur in den Standards für das *Telecommunications Management Network (TMN)* und im Rahmen des TINA–Consortiums verfolgt wurde, weil es wegen der Heterogenität und Anzahl der Betreiber in der Datenkommunikation früher als nicht erreichbar erschien. In den letzten Jahren ergeben sich jedoch auch hier große Fortschritte, wie sich an der Vielzahl ins Leben gerufener Projekte zum Thema *Customer Network Management/Customer Service Management* zeigt [LLN 99]. Der Einsatz von mobilen Managementagenten, die über Betreibergrenzen hinweg migrieren und am Ziel z.B. zur QoS–Überwachung und Fehleranalyse eingesetzt werden können, ist ein hierbei besonders beachtetes Szenario.

1.1 Historie der Mobile Agent System Architecture

Die theoretische Fundierung der *Mobile Agent System Architecture (MASA)* geht auf [Moun 97] zurück, wo erstmalig die Theorie der intelligenten Agenten auf das IT–Management angewandt wurde. Dazu wurde in [Holl 97] und [Wenn 97] ein Agentensystem entwickelt.

Flexible Management Agenten (FMA)

Diese erste Version nannte sich *Flexible Management Agent (FMA)* gemäß der Terminologie aus [Moun 97], nach der der Begriff des flexiblen Agenten eingeführt wurde, weil die Definition eines intelligenten Agenten zu vage und uneinheitlich erschien. Dieses erste Agentensystem basierte auf dem Java Agent Template (JAT[2]) von Rob Frost und war bereits in Java implementiert. Es verwendete als Kommunikationsmodell die Knowledge Query

[2] Der Nachfolger ist JATLite: `http://java.stanford.edu/`

Manipulation Language (KQML, [FFMM 93]).

In [Kemp 97] und [Kots 97] wurden Managementagenten als FMAs entwickelt und das Agentensystem verbessert (Version 0.2). Dabei erwies sich die Middleware von FMA zunehmend als Hemmnis für die weitere Entwicklung. Zum einen gab es keine Werkzeugunterstützung für die Kommunikation über KQML, zum anderen waren in JAT nur wenige, prototypische Basisdienste implementiert; genutzt wurden nur ein Agent Naming Service (ANS) und ein Agent Event Service (AES). Außerdem wurde in KQML kein explizites Informationsmodell definiert. Da absehbar war, daß seitens der *Agent Society*[3] keine schnelle Verbesserung dieser Situation zu erwarten war, wurde in [Kemp 98] auf eine Middleware migriert, die weit ausgereifter, mächtiger, sowie besser werkzeugunterstützt war und im Management eine zunehmende Bedeutung erlangt: nämlich die *Common Object Request Broker Architecture (Corba)* der OMG [OMG 98-02-01].

MASA

Für diese Version wird in Anlehnung an die Terminologie der OMG (genauer: MASIF, vgl. Abschnitt 3.5) die Bezeichung *Mobile Agent System Architecture (MASA)* verwendet. Die Implementierung wurde vollständig neu entwickelt, basierte also nicht mehr auf JAT. Im Zuge der Migration wurde KQML durch das Corba–Kommunikationsprotokoll IIOP ersetzt und die Agenten–Basisdienste durch die entsprechenden Corba–Dienste ersetzt. Ferner wird der OMG–Standard MASIF [OMG 98-03-09] unterstützt. Implementierungssprache ist weiterhin Java.

Neben zahlreichen darauf aufbauenden Entwicklungen in Form von MASA–Agenten wurde in [Bran 99] und [Gerb 99] das MASA–Framework selbst weiterentwickelt. Dabei wurde die Migration auf bzw. die Interoperabilität zu anderen ORBs untersucht und eine GUI erstellt.

Momentane Arbeiten

Derzeit wird vor allem an einer Sicherheitsarchitektur für MASA gearbeitet und damit verbunden an einer Migration auf Java 2. Außerdem wird das System auf alternative ORBs portiert und die Konfigurierbarkeit von MASA verbessert. Hinsichtlich konkreter Anwendungen entsteht ein Managementsystem zur Unterstützung von Nomadic Computing (vgl. Abschnitte 2.2 und 6).

1.2 Übersicht

In den folgenden Abschnitten wird nun MASA genauer beschrieben: Einsatzszenarien für agenten–basiertes Management werden in Kapitel 2 beleuchtet und die Anforderungen für MASA abgeleitet. Der State of the Art auf den Gebieten Mobile Agenten, Management und der Corba–Architektur ist in

[3] www.agent.org

Kapitel 3 zusammengefaßt. MASA wird in Kapitel 4 in ihren Einzelheiten beschrieben. Die Implementierung einzelner, ausgewählter MASA–Agenten sowie Installationshinweise für die Software befinden sich in Kapitel 5. Der Ausblick auf zukünftige Entwicklungen schließt den Bericht.

Zum Verständnis dieses Berichts sind Grundkenntnisse auf folgenden Gebieten erforderlich:

Vorkenntnisse: Corba, IT–Management

- Integriertes Management von Systemen, Netzen und Anwendungen (IT–Management)
- Corba

Kenntnisse auf folgenden Gebieten sind hilfreich:

- Rechnernetze und Kommunikationsprotokolle
- Java
- Theorie der intelligenten Agenten

In Anhang A wird einführende Literatur zu den genannten Themen empfohlen.

Kapitel 2

Einsatzszenarien für MASA

In der Einleitung wurde bereits auf die Motivation für die Entwicklung einer neuartigen Architektur für Agentensysteme hingewiesen. Um diese Motivation zu verdeutlichen und um die praktische Anwendbarkeit von MASA nachweisen zu können, werden zunächst einige Managementaufgaben vorgestellt, die ohne MASA nur ineffizient lösbar wären.

2.1 Policy Management

Die Umsetzung konkreter Managementziele, d.h. die Ermittlung der Managementaktionen zur Durchführung einer Managementaufgabe, ist oft nicht in eindeutiger Weise entscheidbar und unterliegt zudem verschiedenen Einflüssen bzw. Rahmenbedingungen, die oft im Widerspruch zueinander stehen können.

> **Beispiel:** Zur Überwachung von Koppelelementen (Switches, Router, Proxies) auf sicherheitsrelevante Ereignisse hin besteht häufig die Anforderung, bei Auftreten eines bestimmten Ereignisses eine Managementaktion auf der gleichen Komponente anzustoßen und das Managementsystem zu benachrichtigen. Je nach Tageszeit und Verursacher des Ereignisses können Aktion und Benachrichtigung unterschiedlich sein.

Diese Managementaktion ist häufig betreiberspezifisch und kann daher grundsätzlich nicht vom Hersteller des Koppelelements in vordefinierter Form bei Auslieferung der Komponente bereitgestellt werden. Derartige Überwachungsregeln müssen auch leicht dokumentiert, sowie leicht geändert werden können.

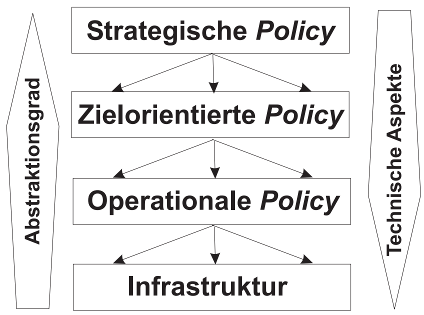

Abbildung 2.1: Policy Hierarchie

Im IT–Management hat sich für derartige Regeln der Begriff *Policy* entwickelt. Er umfaßt eine Vielzahl möglicher Interpretationen ([SlTw 94], [Wies 95]), von der jedoch hier die aus [KKK 96] abgeleitete Einteilung in [Heil 98a] verwendet werden soll: Die Rahmen–Betriebsziele und die mit dem Betrieb der IT–Infrastruktur verfolgten Ziele werden durch *strategische Policies* angeben. In textueller Form bilden sie die ursächliche Begründung für die aus ihnen abgeleiteten *zielorientierten Policies*, die in Form von Ge– und Verboten bezüglich Klassen von zugreifenden Entitäten den Zugriff auf Elemente der Infrastruktur regeln. Aus den zielorientierten Policies können nun mehr oder minder automatisiert die *operationalen Policies* abgeleitet werden, die tatsächlich in Form von Operationen auf den Managementschnittstellen der Infrastrukturelemente durchgesetzt werden.

Automatisierung der Policy–Transformation

Um die Flexibilität für Infrastrukturänderungen zu gewährleisten und eine Automatisierbarkeit und Dokumentation des Managements zu ermöglichen, sollte eine operationale Policy, also das Ergebnis eines formalen Transformationsprozesses von einer oder mehreren zielorientierten Policies, sein.

Lokale Überwachung

Damit das Gesamtsystem skaliert, d.h. daß eine Erweiterung nur einen zu ihr proportionalen Aufwand verursacht, sowie aus Performanzgesichtspunkten, sollte die Transformation sowie die Implementierung der Policy möglichst nahe an der Managed Resource erfolgen. Hingegen sollte die Speicherung und Verwaltung der strategischen und zielorientierten Policies zur Dokumentation und Integritätswahrung zentral erfolgen.

Die Kommunikation zwischen dem Agenten zur Implementierung der Policy und dem Managementsystem muß somit die Übertragung von Policy-Information und Policy–relevanten Ereignissen in effizienter und damit skalierbarer Weise unterstützen. Nur so ist eine feingranulare Implementierung von an zentraler Stelle festgelegten Betriebspolicies möglich.

Zusammengefaßt ergeben sich also folgende Anforderungen an ein mittels flexibler, mobiler Agenten implementiertes Policy–Managementsystem:

- Trennung der Speicherung und Implementierung der Policies
- Ressourcennahe Implementierung der Policies
- Berücksichtigung der (asynchronen) Ereignismeldungen anderer Agenten, um das ansonsten nötige *Polling* zu vermeiden.

Die für effizientes Policy–Management notwendige Verteilung der Management–Infrastruktur sowie die Nutzung von Basisdiensten eines Agentensystems läßt den Aufwand, den die Verwirklichung eines flexiblen

Managementsystem erfordert, angebracht erscheinen. Dies zeigt sich in der in Abschnitt 5.3.3 beschriebenen Implementierung genauer.

2.2 Nomadic Computing

Effizientes *Nomadic Computing*, d.h. die Möglichkeit, überall und jederzeit die momentan umgebende IT–Infrastruktur transparent nutzen zu können, scheitert derzeit an Restriktionen des Managements in der besuchten Umgebung. Diese sind auf Sicherheits– und Abrechnungspolicies zurückzuführen, die die Nutzung lokaler Ressourcen durch fremde Systeme nicht erlauben. Sie können aber auch in einer fehlenden Automatisierung des Konfigurationsmanagements begründet sein, womit eine manuelle Konfiguration für den meist nur kurzfristigen Aufenthalt des *nomadischen Benutzers* bzw. seines Systems zu aufwendig ist.

Eine weiteres Problem bildet der Austausch von Managementinformation zwischen *Heimat*– und *besuchter Domäne*, z.B. für Konfiguration und Aushandlung von Abrechnungsparametern der in der besuchten Umgebung benutzten Dienste.

Auch Management von Konfigurationsservern und Koppelelementen unterer (OSI–)Schichten (Switches, Router) ist wichtig, um für den nomadischen Benutzer eine möglichst umfassende und zugleich abgeschirmte Kommunikationsumgebung zu errichten.

Zur Unterstützung des Managements der IT–Infrastruktur für Nomadic Computing sind unter anderem folgende Komponenten notwendig:

- Bezüglich des Managements integrierte **Konfigurationsserver** auf der Basis von DHCP (vgl. Abschnitt 5.3.4) und anderer standardisierter Konfigurationsprotokolle

- Integriert managebare **Koppelelemente** der Verbindungsschicht (Schicht 2), sog. *Switches*, um drahtgebundene und drahtlose Anschlußmöglichkeiten für nomadische Systeme überwachen und steuern zu können (vgl. Abschnitt 5.3.5)

- Managebare **Trader**, um die Dienstvermittlung an nomadische Systeme auf der Basis von verbreiteten Standards (z.B. *Service Location Protocol (SLP)* [VGPK 97] oder JINI [JINI]) verwalten zu können [Demm 99].

- Mobile **Abrechnungsagenten**, um die Modalitäten der Abrechnung von in der besuchten Domäne erbrachten Dienste zwischen den beteiligten Managementsystemen in skalierbarer Weise aushandeln und das „Settlement“ der Abrechnung durchführen zu können.

Alle diese Agenten lassen sich in idealer Weise auf der Basis von MASA realisieren, da eine enge Integration untereinander und mit den Agenten für andere Managementaufgaben (z.B. Eventkorrelation) offensichtliche Vorteile für das integrierte Management der gesamten Infrastruktur bietet.

2.3 Eventkorrelation mit Abhängigkeitsgraphen

Ein wichtiges Problem beim Betrieb heutiger IT–Infrastrukturen ist die unangemessene Behandlung von Managementereignissen. Tritt ein Fehler im zu managenden System auf, werden die Administratoren häufig mit großen Mengen von Ereignissen konfrontiert, die jeweils ein Symptom desselben Fehlers beschreiben. Die Identifikation kausal zusammenhängender Ereignisse und ihre Verdichtung zu einem einzigen Ereignis ist Aufgabe eines Eventkorrelators.

Abhängigkeitsgraphen: Funktionale Abhängigkeiten zwischen MOs

In [Grus 99] wird ein solcher Eventkorrelator entworfen. Dazu wird zur Repräsentation des Korrelationswissens ein *Abhängigkeitsgraph* eingeführt. Ein Abhängigkeitsgraph ist ein gerichteter Graph, der die Managementobjekte des IT–Systems und funktionale Abhängigkeiten zwischen ihnen beschreibt. Ein Eventkorrelator, der auf Abhängigkeitsgraphen aufbaut, bildet die Eingabe–Ereignisse in den Abhängigkeitsgraphen ab und durchsucht ihn nach Knoten, von denen alle bzw. viele Initial–Knoten abhängen. Die Menge dieser Knoten wird als verdichtetes Ereignis weitergeleitet und bildet die möglichen, gefundenen Erklärungen für die empfangenen Symptommeldungen.

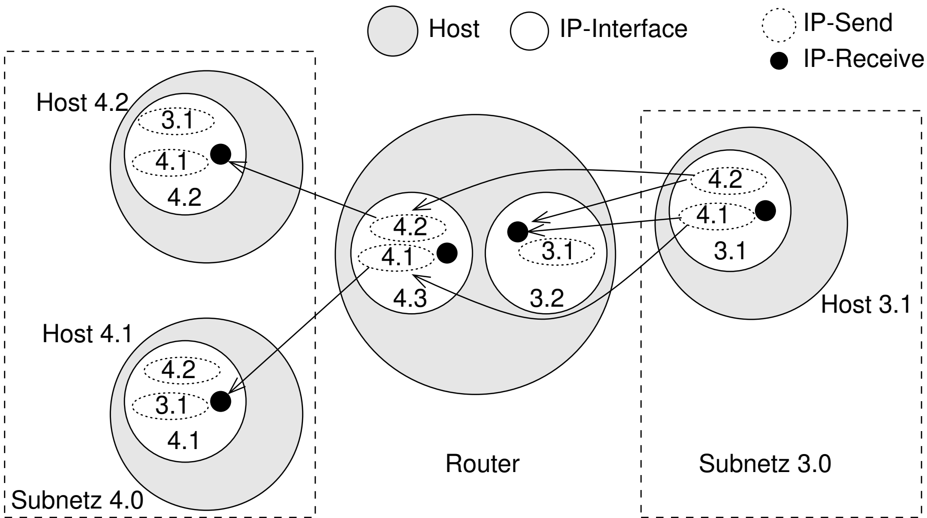

Abbildung 2.2: Abhängigkeitsgraph für ein IP–Netz

Abbildung 2.2 zeigt ein Beispiel für einen Abhängigkeitsgraphen. Modelliert wurde ein IP–Netz bestehend aus Rechnern (Hosts), IP–Protokollinstanzen (Interfaces) und dem IP–Dienst. Er wurde in einen Dienst „IP–Pakete empfangen“, dargestellt durch ausgefüllte Kreise, und Dienste „IP–Pakete an den Rechner X schicken“, dargestellt durch gepunktete Ellipsen, unterteilt, wobei der Zielrechner X notiert wird. Abhängigkeiten werden durch Pfeile dargestellt und — zur besseren Übersicht — dadurch, daß innere Objekte implizit von äußeren Objekten abhängen.

Aus dem modellierten Abhängigkeitsgraphen kann abgelesen werden, wovon das Verschicken eines IP–Paketes von Rechner 3.1 zu Rechner 4.1 funktional abhängt, d.h. was funktionieren muß, damit ein IP–Paket von 3.1 nach 4.1 übermittelt werden kann:

1. Das IP–Interface auf dem sendenden Rechner 3.1 muß funktionieren und damit auch Host 3.1 selbst.
2. Der Router muß auf dem IP–Interface 3.2 empfangen und dem IP–Interface 4.3 an 4.1 senden können. Damit muß mittelbar auch der Router selbst — sein Backplane, seine Stromversorgung, etc. — funktionieren.
3. Schließlich muß auch der Zielrechner 4.1 IP–Pakete empfangen können.

Modulare, verteilte Realisierung notwendig

Die Aufstellung solcher Abhängigkeitsgraphen für komplexe IT–Systeme — im Beispiel bestand die Modellwelt nur aus drei Rechnern und einem Router — ist aufwendig. Neben der Netztopologie müssen Informationen über die Endsysteme, die Anwendungen und die von ihnen erbrachten Dienste einfließen. Außerdem muß das Betriebskonzept eines IT–Betreibers berücksichtigt werden, da es bestimmt, wie feingranular bestimmte Managementobjekte im Abhängigkeitsgraphen repräsentiert werden sollen. Beispielsweise müssen Server detaillierter modelliert werden als Arbeitsplatzrechner, auch wenn sie vielleicht dieselbe Technik verwenden. Ziel muß es sein, möglichst viel dieser Information aus vorhandenen Managementwerkzeugen automatisch auszulesen bzw. zu generieren.

Kommunikation mit anderen Modulen und der Außenwelt

Das Problem einer integrierenden Komponente wie einem Abhängigkeitsgraphen ist, daß die Anbindung an z.B. zehn Managementwerkzeuge mit jeweils einem Versionssprung pro Jahr zu zehn Versionssprüngen in der Implementierung des Abhängigkeitsgraphen führen würde. Das Konzept der flexiblen Agenten dient dazu, die Implementierung eines Abhängigkeitgraphen unter dieser Randbedingung wartbar zu halten. Der Abhängigkeitsgraph wird dazu in verschiedene Module aufgeteilt, die jeweils die Managementinformation repräsentieren, die von einem einzelnen Managementwerkzeug verwaltet wird.

Mehrere administrative Zuständigkeiten

Ein weiteres Problem ist, daß ein komplexer Abhängigkeitsgraph nicht von einer einzelnen Person gewartet werden kann. Niemand besitzt genügend Wissen über alle Komponenten eines zu managenden Systems. Die Anforderung lautet also, daß das Managementsystem mit mehreren Administratoren, die jeweils für einen bestimmten Teil des Abhängigkeitsgraphen verantwortlich sind und diesen relativ unabhängig voneinander weiterentwickeln wollen, umgehen kann.

Flexible Konfiguration, auch zur Laufzeit

Abhängigkeitsgraphen sind dynamisch, weil sie ein dynamisches System modellieren. Die Anforderung daraus lautet, daß ein Höchstmaß an Flexibilität erforderlich ist: Neue Module müssen einfach hinzugefügt, auf einen neuen Stand gebracht oder entfernt werden können. Die Module, die den Abhängigkeitsgraphen repräsentieren, müssen miteinander und mit der Außenwelt, also den Managementwerkzeugen, von denen sie Daten beziehen, kommunizieren können. Die Kommunikation mit der Außenwelt muß mit einem der Protokolle möglich sein, die vom Kommunikationspartner verwendet wird — in Managementsystemen werden in der Regel mehrere, z.T. proprietäre Protokolle eingesetzt.

Die Module der Implementierung des Abhängigkeitsgraphen sollten durch Softwarekomponenten implementiert werden, die miteinander und mit der Außenwelt kooperieren, dynamisch und flexibel konfiguriert, einfach installiert und von verschiedenen Personen administriert werden können.

2.4 Resultierende Anforderungen an MASA

Anforderungen mit MASA umsetzbar

Die in den vorhergehenden Abschnitten abgeleiteten Anforderungen sind mit klassischen Managementsystemen nicht oder nur schlecht umsetzbar. Sie erfordern vielmehr eine möglichst flexible Infrastruktur, wie sie das Agenten–basierte Managementsystem MASA bietet.

Neben den Anforderungen aus den beschriebenen Szenarien sollen bei der Entwicklung von MASA folgende Aspekte berücksichtigt werden:

- Die Integration verschiedener MASA–Komponenten und damit realisierter Anwendungen soll möglichst unter Verwendung moderner Software-Engineering-Konzepte wie Objektorientierung, Komponentenbildung und Wiederverwendung erfolgen. Mächtige Middleware–Implementierungen wie sie im Corba–Umfeld insbesondere auch zu Managementzwecken entstehen, sollen genutzt werden können.
- Da das „Rad für Managementsysteme nicht neu erfunden werden soll", muß die MASA–Infrastruktur und damit realisierte Agenten mit existierenden Managementsystemen integriert werden können und deren Dienste nutzen können. Für diese Öffnung ist die Implementierung der Kommunikation der MASA–Komponenten untereinander auf der Basis von Corba besonders geeignet, denn Corba entwickelt sich zum Kommunikationsstandard der Wahl für Managementsysteme.

- Die Kommunikation innerhalb der Plattform zur Implementierung von Managementanwendungen sowie zur Migration der Agenten und dem Instanziieren von Agentenimplementierungen soll in (kryptologisch) gesicherter Weise erfolgen, um Angriffe auf die Managementinfrastruktur zu verhindern.

- Um die Skalierbarkeit der Anwendungen auch in größeren Managementumgebungen zu gewährleisten, ist die verteilte, kooperative, flexible Administration einer MASA–Domäne sicherzustellen. Managementdomänen sollen gebildet und zu größeren Einheiten hierarchisch zusammengefaßt werden können.

- Die Implementierung der Benutzerschnittstelle soll in möglichst verteilter Form erfolgen (z.B. auf der Basis von WWW und Java), um die Entwicklung dedizierter Zugriffssoftware vermeiden zu können und dennoch eine auf die spezielle Managementaufgabe zugeschnittene Administration zu realisieren. Die Implementierung einer Benutzerschnittstelle soll mit den Agenten in offener Weise (möglichst unter Verwendung einer Middleware) kommunizieren, um eine leichte Erweiterbarkeit zu gewährleisten.

- Wenn für Teile der Managementfunktionalität eine statische, feste Interaktion zwischen Agent und verwalteter Ressource erforderlich ist, dann muß es möglich sein, diese Teile als *Stationäre Agenten* zu isolieren. Eine Migration derartiger Agenten ist folglich nicht möglich, sie können dennoch die sonstige Funktionalität der Agentenplattform nutzen.

- Die universitäre Basis unserer Forschung zu verteilten Managementsystemen macht es wünschenswert, MASA–Software als eine flexible Lehrplattform zu entwickeln, die sich einerseits durch studentische Arbeiten leicht erweitern läßt und andererseits den Studenten Konzepte moderner SW–Entwicklung (wie z.B. Modularisierung, Middleware) und heutiger Programmiersprachen (wie z.B. Java) näherbringt.

Kapitel 3

Agenten und Management — State of the Art

In diesem Kapitel werden Technologien diskutiert, die durch MASA teilweise ersetzt werden sollen (Klassische Managementagenten, Abschnitt 3.1), Alternativen zu MASA (Abschnitte 3.2 und 3.3) und Technologien, die zur Realisierung von MASA verwendet werden (Abschnitte 3.4 und 3.5).

3.1 Klassische Managementagenten

In Managementsystemen und den ihnen zugrundeliegenden Managementarchitekturen wird seit jeher der Begriff des Agenten verwendet, der sich aber stark von dem unterscheidet, was mit flexiblen, intelligenten oder Mobilen Agenten gemeint ist.

Organisationsmodell

In den Organisationsmodellen der klassischen Managementarchitekturen, dem OSI–Management und dem Internet– oder SNMP–Management, werden zwei Rollen definiert:

- *(OSI– oder SNMP–) Agenten* stellen Managementinformation bereit (Management Information Base, MIB) und melden managementrelevante Beobachtungen (Managementereignisse). Agenten werden auf / für alle zu managenden Ressourcen, also Netzkomponenten, Endsysteme und Anwendungen, installiert.
- *Manager* fragen die Managementinformation von den Agenten ab und verarbeiten sie. Sie empfangen Managementereignisse und sollen auf sie reagieren. Managementplattformen wie HP OpenView, Spectrum, Tivoli TME oder CA Unicenter agieren in der Managerrolle. Sie stellen eine Plattform für Managementanwendungen und die Managementstationen, also die Arbeitsplätze der Administratoren bereit, die einzelne

Mangementaufgaben, z.B. Inventarisierung oder Software–Verteilung, realisieren.

Informationsmodell

Im Internet–Management wird die Schnittstelle zwischen Agent und Manager durch SMI (Structure of Management Information, [SCM+ 96, MPS 99]) definiert, ein datentyp–orientiertes Informationsmodell. Der Manager fragt also die Werte einfacher Variablen oder Tabellen ab. Im OSI–Management wird im Prinzip auch so gearbeitet, jedoch wird ein objektorientiertes Informationsmodell, definiert durch GDMO (Guidelines for the Definition of Managed Objects, [ISO 10165-4]), verwendet. In beiden Fällen sind die Agenten also *Datenserver*.

OSI und SNMP: starre Funktionalität der Agenten

Das Problem dieser Managementarchitekturen ist, daß es sich als schwierig und langwierig herausgestellt hat, MIBs zu definieren, die

- von allen Herstellern zu managender Ressourcen implementiert werden und
- alle Managementanforderungen aller IT–Betreiber erfüllen.

Zusätzliche MIBs zum dynamischen Nachladen

Zum einen werden ständig neue, unterschiedliche Technologien eingeführt, zum anderen haben die IT–Betreiber sehr unterschiedliche und sich ändernde Anforderungen an ihr IT–Management. Man ist daher dazu übergegangen, Möglichkeiten zu schaffen, um die Funktionalität der Agenten zur Laufzeit zu erweitern, also Programmcode („Skripten“) auf den Agenten zu laden [ISO 10164-21, LeSc 99].

MASA: gesamte Funktionalität dynamisch laden

Demgegenüber wird in MASA der Ansatz verfolgt, zunächst eine Plattform zu realisieren, mit der dynamisch Funktionalität auf die zu managenden Ressourcen gebracht werden kann, um mit Hilfe dieser Plattform die gesamte Managementfunktionalität dynamisch zu laden. Die Agenten, die die eigentliche Managementfunktionalität implementieren, werden dann von Administratoren oder anderen Agenten mit Hilfe von MASA instanziiert, anstatt sie manuell bei der Inbetriebnahme der Ressource zu installieren. Vorteil ist, daß die Managementfunktionalität relativ leicht neuen oder veränderten Anforderungen angepaßt werden kann.

Wenn ein MASA–Agent die Aufgaben eines klassischen SNMP– oder OSI–Agenten übernehmen soll, muß er mit einer zu managenden Ressource interagieren. Schließlich soll er eine managementrelevante Abstraktion dieser Ressourcen implementieren.

3.2 Management by Delegation

Klassische Managementsysteme sind als zentralisierte plattformbasierte Architekturen realisiert, d.h. von einer zentralen Managementplattform aus wird eine große Zahl von Agenten verwaltet. Die Agenten besitzen nur einfache Zugriffsschnittstellen und sind gewöhnlich nicht in der Lage, eine Vorverarbeitung der Managementdaten, die sie sammeln, durchzuführen. Dieser Ansatz wird als statisch bezeichnet, da das Vorhandensein vordefinierter Managementfunktionen vorausgesetzt wird und eine Änderung dieser Funktionalität zur Laufzeit nicht möglich ist, d.h. bei einer Änderung der Managementanforderungen oder der Managementinformation (MIB) ist eine Neukompilierung des Agenten oder die Installation eines neuen Agenten notwendig.

Aus dem zentralisierten Managementansatz ergeben sich gravierende Nachteile.

- Die zentrale Managementplattform stellt einen „single–point–of–failure“ dar.
- Durch die große Anzahl von Agenten und die fehlende Verdichtung der Managementdaten durch die Agenten kommt es zu sehr hoher Netzlast.
- Der zentrale Manager wird häufig überlastet.
- Die Anzahl der Applikationen, die auf einer zentralen Managementstation zur Ausführung gebracht werden können, ist begrenzt.
- Die Architektur ist unflexibel.

MbD: dynamisch ladbare Funktionalität

Um diese Defizite zu beseitigen wird eine Architektur benötigt, die eine leichte Anpaßbarkeit an Änderungen in der zu managenden Infrastuktur und eine dynamische Verlagerung von Funktionalität auf die Agenten ermöglicht. Unter dem Management by Delegation (MbD) Paradigma [YGY 91, GoYe 95] werden Ansätze zusammengefaßt, die es ermöglichen, Managementfunktion an die Agenten zu delegieren.

flexible Agenten: MbD + autonom und kooperativ

Eine Erweiterung des MbD–Paradigmas stellen die in [Moun 97] eingeführten flexiblen Agenten dar. Sie zeichnen sich dadurch aus, daß sie bis zu einem gewissen Grad autonom handeln und zur Erfüllung ihrer Aufgabe mit anderen Agenten kooperieren. Innerhalb eines Managementsystems können flexible Agenten zu Gruppen zusammengefaßt werden, die kooperativ eine bestimmte Aufgabe erledigen. Der Übergang von zentralen zu flexiblen, dynamischen Managementarchitketuren ist in Abb. 3.1 dargestellt.

3.3 Plattformen für Mobile Agenten

Der Begriff *Mobiler Agent (MA)* [RoPo 97, RoHo 98, PhKa98, BPW 98] erweitert das Konzept des flexiblen Agenten um den Aspekt der Mobilität. Ein Mobiler Agent ist ein Programm, das sich innerhalb eines Netzes bewegen kann und im Auftrag eines Nutzers oder einer Organisation bestimmte Aufgaben erledigt. Der MA kann „von außen“ zur Migration veranlaßt werden

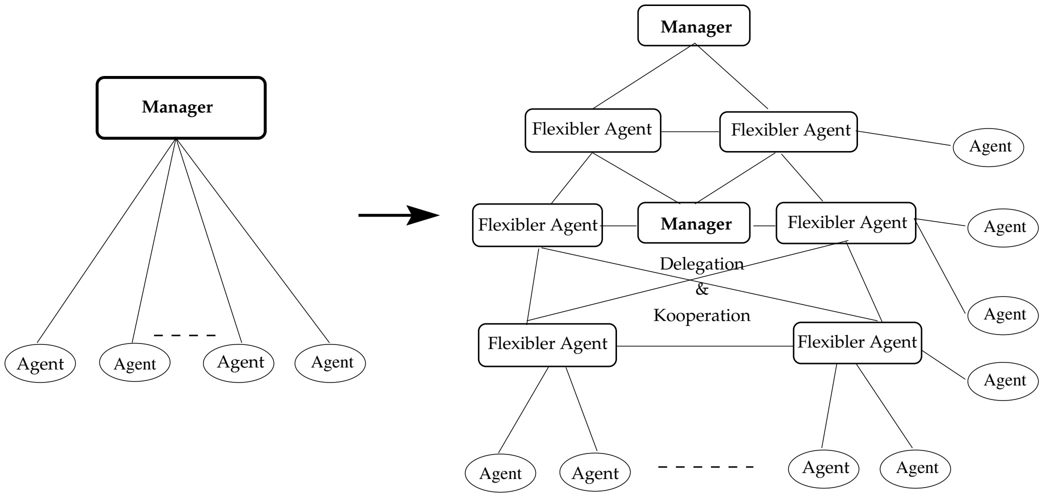

Abbildung 3.1: Übergang von zentralisiertem zum MbD Paradigma (nach [Moun 97])

oder er kann sich selbst auf eine andere Ablaufumgebung migrieren. Ein MA kann selbständig eine Reihe von Endsystemen und Netzkomponenten besuchen und dort Information über Managed Resources sammeln oder Konfigurationen verändern, wenn er dazu berechtigt ist. Er liefert nur die Ergebnisse an einen Manager oder an einen anderen Agenten, der ihn beauftragt hat.

Motivation für Mobile Agenten

Zu den Vorteilen, die der Einsatz Mobiler Agenten verspricht, gehören:

- die Reduzierung der Netzlast,
- die asynchrone, autonome und endsystemnahe Interaktion mit den zu managenden Ressourcen,
- eine verbesserte Fehlertoleranz,
- die Abstraktion bzw. Verschattung von heterogenen Systemplattformen,
- die dynamische Erweiterung der Funktionalität des Managementsystems,
- der effiziente Zugriff auf lokale Daten.

Beim traditionellen Manager–Agent Ansatz, holt sich der Manager u.U. riesige Datenmengen von den Agenten, um diese dann lokal zu verarbeiten. Für die Überwachung entfernter Netzkomponenten wurde dieser Ansatz, beim Internet Management durch die RMON–MIB [Stal 96, Wald 95, Wald 97] erweitert. Damit besteht zwar die Möglichkeit, Netzkomponenten dezentral zu überwachen, allerdings ist die RMON–MIB statisch definiert und kann nicht dynamisch erweitert werden.

Im Gegensatz zum traditionellen Manager–Agent Ansatz oder RMON, wird beim MA–Ansatz die Anwendungslogik zu den Daten geschickt. Die Verarbeitung erfolgt endsystem– und datennah; der MA liefert nur das Ergebnis zurück. Da sich der MA auf der zu verwaltenden Ressource befindet, können einerseits auch dann Managementdaten gesammelt werden, wenn die Netzverbindung zu der entsprechenden Ressource unterbrochen ist.

Endsystem– und datennahe Verarbeitung

Andererseits können mit Hilfe von MAs auch Tests, z.B. zur Überwachung eines Quality of Service, Überwachung von Service Level Agreements (SLA) u.ä., durch einen Provider auf Seite des Kunden durchgeführt werden [HaRe 99]. Der Kunde stellt dazu eine Ablaufumgebung für Mobile Agenten zur Verfügung. Der Provider braucht dann nur noch seinen Agenten zum Kunden zu schicken. Sollten sich SLA oder QoS Parameter ändern, erweitert er die Funktionalität des bestehenden MA oder schickt einen neuen MA zum Kunden. Die Möglichkeit, daß Agenten von einem (Kunden–) Endsystem aus flexibel Testoperationen durchführen, ist bei traditionellen Managementsystemen überhaupt nicht oder nur sehr schwer realisierbar.

Bestehende MA–Plattformen

Generell gibt es zwei Möglichkeiten für die Realisierung einer MA–Plattform, die Implementierung als spezieller Betriebssystemdienst, wie z.B. TACOMA[1] [JRS 95, Joha 98], oder die Realisierung als Anwendungssoftware. Der zweiten Alternative wird in der Mehrzahl der existierenden MA–Plattformen der Vorzug gegeben, da die Implementierung als Teil des Betriebssystems die gewünschte Plattformunabhängigkeit behindert.

Die zweite Gruppe läßt sich weiter unterteilen in MA–Plattformen, die Java als Implementierungssprache verwenden, und solche, die Skriptsprachen nutzen. D'Agents[2] (früher als Agent Tcl bezeichnet) [Gray 95] bspw. verwenden einen veränderten Tcl–Dialekt und damit auch einen erweiterten Tcl–Interpreter. Zukünftig sollen auch Python [Python], Scheme [Scheme] und Java unterstützt werden.

Zu den Java–basierten MA–Plattformen zählen u.a. Aglets[3] [LaOs 98, OKO 98], Concordia[4] [Mits 98], Grasshopper[5] [BBCM 98], Gypsy[6], Jumping Beans[7], Mole [8] [BHR 97] Odyssey[9] und Voyager[10].

Oft ist es schwierig, die Produkte in bestehende Managementsysteme zu integrieren. Sehr viele dieser MA–Plattformen verwenden proprietäre Middleware oder Kommunikationsprotokolle. Bei MASA wurde von Beginn an Corba als

[1] `http://www.tacoma.cs.uit.no`
[2] `http://agent.cs.dartmouth.edu/`
[3] `http://www.trl.ibm.co.jp/aglets`
[4] `http://www.meitca.com/HSL/Projects/Concordia/`
[5] `http://www.ikv.de/products/grasshopper`
[6] `http://www.infosys.tuwien.ac.at/Gypsy`
[7] `http://www.JumpingBeans.com/`
[8] `http://www.informatik.uni-stuttgart.de/ipvr/vs/projekte/mole.html`
[9] `http://www.genmagic.com/technology/odyssey.html`
[10] `http://www.objectspace.com/voyager/`

Middleware eingesetzt. Andere Plattformen, die Corba heute unterstützen, verwendeten zum Zeitpunkt der Entwicklung von MASA proprietäre Konzepte. Corba wurde erst nachträglich in ihre Architektur integriert.

Lediglich Grasshopper und Aglets unterstützen den MASIF–Standard (vgl. Abschnitt 3.5), bzw. haben angekündigt, MASIF unterstützen zu wollen. Aglets bspw. benutzen ein eigenes proprietäres Agent Transfer Protocol (ATP). Die in MASIF definierten Schnittstellen werden zwar von Aglets unterstützt, allerdings erfolgt die Realisierung nicht durch Corba Objekte und IIOP sondern durch „normale“ Java Klassen und ATP.

Grasshopper: Heterogene Kommunikationsarchitektur

Grasshopper, das den MASIF–Standard unterstützt, verwendet allerdings nicht nur IIOP/Corba als Kommunikationsmechanismus und Middleware sondern auch diverse andere Protokolle und Konzepte wie z.B. RMI, Sockets, u.a. Für den Entwickler von Mobilen Agenten ist es zwar angenehm, verschiedene Protokolle verwenden zu können, vom Sicherheitsstandpunkt des Betreibers des Agentensystems (vgl. Abschnitt 3.5) aus betrachtet, erweist es sich jedoch als äußerst schwierig die Vielzahl der Kommunikationsmechanismen, die entweder keine oder sehr unterschiedliche Sicherheitsmechanismen unterstützen unter einer einheitlichen, integrierten und umfassenden Sicherheitsarchitektur zusammenzufassen.

Die Nicht–Festlegung einer Kommunikationsarchitektur in Grasshopper birgt noch ein weiteres Problem. Das Agentensystem stellt hohe Anforderungen an das unterliegende Transportsystem (TCP/IP, Java RMI, Corba u.a.), die derzeit nur von „Mainstream“–Rechnern erfüllt werden. Da der Entwickler der Agenten frei wählen kann, welches der unterstützten Transportprotokolle er nutzen will, muß das Agentensystem alle unterstützen oder auf die Ausführbarkeit eines Teils der Agenten verzichten. Von einem Managementsystem fordert man, daß es universell einsetzbar sein soll, also das Management alle bei einem IT–Betreiber auftretenden Ressourcen integrieren kann. Das kann bedeuten, daß auch Netzkomponenten wie Hubs und LAN–Switches, Embedded Systems, Mobiltelefone etc. zu managen sind. Die Beschränkung auf ein einziges Transportsystem — Corba in MASA oder SNMP/UDP/IP im Internetmanagement — erleichtert die Verbreitung des Managementsystems erheblich.

Für Corba sind Abbildungen auf die unterliegenden Transportsysteme TCP/IP (IIOP) und auf DCE/RPC standardisiert. Die Standardisierung von GIOP über ATM (RFP [OMG 99-03-05]) hat begonnen. Minimum Corba (RFP [OMG 97-06-14], Revised Submission [OMG 98-08-04]) legt eine minimale Implementierung eines Corba–ORB fest.

3.4 Common Object Request Broker Architecture (Corba)

Bessere Integrationsfähigkeit durch Corba

MASA stützt sich in seiner Kommunikationsarchitektur und für die Dienste, die von der MASA–Plattform bereitgestellt werden, auf Corba [OrHa 98] ab. Neben der Vereinfachung der Implementierung an einigen Stellen, z.B. durch die Generierung von Stubs und Skeletons, werden damit vor allem zwei Ziele verfolgt:

- Ein großes Defizit existierender Managementsysteme ist die mangelnde Integration der einzelnen Anwendungen untereinander. Eine Ursache dafür ist, daß Managementsysteme aus relativ unabhängig voneinander entwickelten Bausteinen zusammengesetzt werden. Das führt zu Inkonsistenzen zwischen redundant verwalteten Datenbeständen und zu mehrfacher Arbeit. Eine notwendige Voraussetzung für eine bessere Integration der Anwendungen untereinander ist eine mächtige Middleware wie Corba. Die Hoffnung ist, daß verschiedene, unabhängig voneinander entwickelte Agenten mit Corba leichter integriert werden können als ohne.

- Ein neues Werkzeug muß mit bestehenden Managementsystemen zusammenarbeiten, in die große Investitionen getätigt wurden. Zwar haben viele Managementplattformen und –anwendungen heute nur proprietäre, d.h. herstellerspezifische, Schnittstellen, jedoch läßt sich ein Trend erkennen, auch Corba–Schnittstellen bereitzustellen. Nachdem die Vielzahl proprietärer Schnittstellen von MASA nicht unterstützt werden kann und soll, sollte zumindest eine Integration auf Basis des offenen Standards Corba unterstützt werden.

Corba ist eine von der OMG spezifizierte Plattform für verteilte Anwendungen. Der Corba–Standard in der aktuellen Version 2.2 ist in „The Common Object Request Broker: Architecture and Specification“ [CORBA 2.2] festgelegt (und kann vom Webserver der OMG[11] bezogen werden). Corba bietet eine Plattform, auf der *Objekte*, die auf verschiedene Rechner verteilt sein können, ortstransparent interagieren. Dazu werden die Schnittstellen der Objekte in einer einheitlichen, programmiersprachen– und systemarchitekturunabhängigen Sprache, der *Interface Definition Language (IDL)*, beschrieben. Die Syntax ähnelt C++/Java–Funktionsdeklarationen. Wichtigster konzeptioneller Unterschied zu C++ ist, daß keine direkten Speicheradressen (Pointer) verwendet werden können, weil das in verteilten Umgebungen keinen Sinn machen würde. Es können auch keine Klassen als Übergabe– bzw. Rückgabeparameter in IDL Schnittstellen verwendet werden, hier sind nur elementare Datentypen und Strukturen möglich.

[11] `www.omg.org`, lokal am Lehrstuhl unter /proj/Literatur/OMG

Corba = IDL, IIOP, Stubs, Skeletons, ORBs, `idl2java` usw.

Aus IDL–Beschreibungen können mit Hilfe von Compilern Schnittstellen in vielen Programmiersprachen generiert werden. Für MASA ist z.Zt. nur die Generierung von Java–Schnittstellen (*Java Mapping*) relevant, für die es eine Spezifikation der OMG gibt [OMG 99-07-53]. Server–Objekte müssen diese Schnittstellen implementieren; Client–Objekte können sie verwenden.

Neben den eigentlichen Schnittstellen werden von einem `idl2java`–Compiler weitere Java–Klassen generiert, die die Kommunikation zwischen Client und Server über das Corba–eigene Protkoll IIOP (*Internet Inter–ORB–Protocol*) abwickeln. Der Client erhält ein sog. Proxy–Objekt, das die Anfragen für den Client transparent via IIOP an das eigentliche Server–Objekt weiterleitet und die Antworten entgegennimmt. Für die Klasse, die das Proxy–Objekt implementiert, wird der Begriff *Stub* verwendet. Server–seitig werden die Anfragen zunächst wiederum von einer generierten Klasse, dem *Skeleton*, entgegengenommen, so daß sie für das eigentliche Server–Objekt wie lokale Aufrufe erscheinen. Ein Object Request Broker (ORB) verwaltet die verschiedenen Requests und Responses und die IIOP–Verbindungen. Werden Corba–Objekte wie in MASA in Java implementiert bzw. von Java–Maschinen aus genutzt, dann muß ein ORB Bestandteil jeder Java–VM sein, da er die Kommunikation zwischen den VMs abwickelt.

Dynamic Invocation Interface

Mit Hilfe des *Dynamic Invocation Interfaces (DII)* ist es auch möglich, daß der Client erst zur Laufzeit Informationen über die Schnittstellen des Servers erlangt und diese dann nutzt. In diesem Fall müssen zum Zeitpunkt der Übersetzung der Client–Objekte die Schnittstellen der Server–Objekte noch nicht bekannt sein.

Corba Services: Naming und Event

Zu den für MASA wichtigen Bestandteilen von Corba gehören einige der Corba–Dienste, also Objekte mit standardisierten IDL–Schnittstellen und festgelegter Semantik:

- Der *Naming Service* ist ein Verzeichnisdienst für Corba. Objektreferenzen können unter einem Namen (String) dort abgelegt werden (`bind`), damit andere Objekte diese Objektreferenz unter diesem Namen erfragen können (`lookup`).

- Der *Event Management Service* stellt *Event Channels* bereit, an den *Supplier* Nachrichten schicken können, die vom Event Service an die *Consumer* dieses Event Channels weitergeleitet werden.

  Eine beliebige Anzahl von Suppliern kann mit einer beliebigen Anzahl von Consumern über einen Event Channel verbunden werden. Die Ereignisse werden von den Suppliern an die Consumer geschickt. Supplier und Consumer können sich an den Event Channel nach dem Push–Modell (Erzeuger–initiierte Kommunikation, `PushSupplier` bzw. `PushConsumer`) oder dem Pull–Modell (Verbraucher–initiierte Kommunikation, `PullSupplier` bzw. `PullConsumer`) binden. Am selben Event

Channel können einige Entitäten nach dem Push–Modell und andere nach dem Pull–Modell kommunizieren. Ein Ereignis ist eine beliebige IDL–Datenstruktur, d.h. vom Typ `Any`.

Künftig werden auch der Property–, der Notification– und der Topology Service sowie der Security Service verwendet werden.

Es existieren eine Reihe von ORBs für Java (incl. idl2java–Compiler), von denen in MASA bisher vor allem VisiBroker 3.0 von Inprise[12] (früher Borland) verwendet wurde. Künftig soll vornehmlich ORBacus von OOC[13] bzw. der Java 2 ORB verwendet werden.

ORBs: VisiBroker, ORBacus, Java 2

Naming, Event und Property Service werden häufig als eigene Server–Prozesse implementiert, was von der Corba–Spezifikation jedoch nicht vorgeschrieben ist. Die Kommunikation mit den Objekten erfolgt dann über IIOP, so daß beliebige standardkonforme Implementierungen (in beliebigen Programmiersprachen) und von verschiedenen Herstellern verwendet werden können. Gegenwärtig wird der Naming und Event Service des ORBacus oder des VisiBroker in MASA verwendet.

Zu den fortgeschrittenen Corba–Techniken, die in MASA verwendet werden, gehört die *Tie Delegation* [OrHa 98]. Dieses Konzept ist manchmal notwendig, um die IDL–Vererbungshierarchie auf Java abzubilden. Da Java nur Einfachvererbung unterstützt, kann ein Konflikt auftreten. Der Konflikt entsteht, weil — in der Variante ohne Tie Delegation — eine Javaklasse, die ein IDL–Objekt implementiert von einer mittels idl2java generierten Klasse erben muß. In MASA muß ein Agent aber unabhängig davon bereits von einer Javaklasse — der Oberklasse aller MASA–Agenten — erben. Dieser Konflikt kann mit der Technik der Tie Delegation aufgelöst werden, auf die in [Kemp 98], [Roel 99b] und [Bran 99] näher eingegangen wird.

3.5 Mobile Agent System Interoperability Facility (MASIF)

Bestehende Plattformen für Mobile Agenten (MA) unterscheiden sich in den Konzepten, in ihrer Architektur und in der Implementierung erheblich voneinander. Diese Unterschiede erschweren oder verhindern eine Interoperabilität zwischen verschiedenen MA–Plattformen und damit auch eine rasche Verbreitung der MA–Technologie.

Aus diesen Gründen hat sich die OMG entschlossen, einen Standard für Mobile Agenten Plattformen zu entwickeln. Die *Mobile Agent System Interoperability Facility (MASIF)* [MASIF] soll die Interoperabilität zwischen

[12] `http://www.inprise.com`
[13] `http://www.ooc.com`

MA–Plattformen verschiedener Hersteller ermöglichen und eine einheitliche Begriffsbildung etablieren. Dazu wurde ein Basismodell und die grundlegenden Begriffe für eine verteilte MA–Plattform spezifiziert.

Agent

Mobiler Agent

stationärer Agent

Ein *MASIF–Agent* ist ein Programm, das autonom im Auftrag eine Nutzers oder einer Organisation (*Agent Authority*) handelt. Es werden mobile und stationäre Agenten unterschieden. Ein *Mobiler Agent* ist nicht an das System gebunden, auf dem er gestartet wurde. Er kann auf eigene Veranlassung, bzw. auch von außen initiiert, auf ein anderes System migrieren. Im Gegensatz dazu kann ein *stationärer Agent* nicht migrieren.

Agentensystem

Place

Ein *Agentensystem (Agent System, AS)* ist die Laufzeitumgebung für mobile und stationäre Agenten, innerhalb derer Agenten erzeugt, interpretiert, ausgeführt, suspendiert, reaktiviert, transferiert oder terminiert werden können. Auch das AS ist einer verantwortlichen AS–Authority zugewiesen. Als *Place* wird ein bestimmter Kontext innerhalb eines AS verstanden, in dem Agenten ausgeführt werden.

Region

Agentensysteme werden über eine Kommunikationsinfrastruktur miteinander verbunden. Die Agentensysteme, die derselben Authority unterstehen, werden zu einer sogenannten *Region* zusammengefaßt. Das Basismodell von MASIF ist in Abb. 3.2 dargestellt.

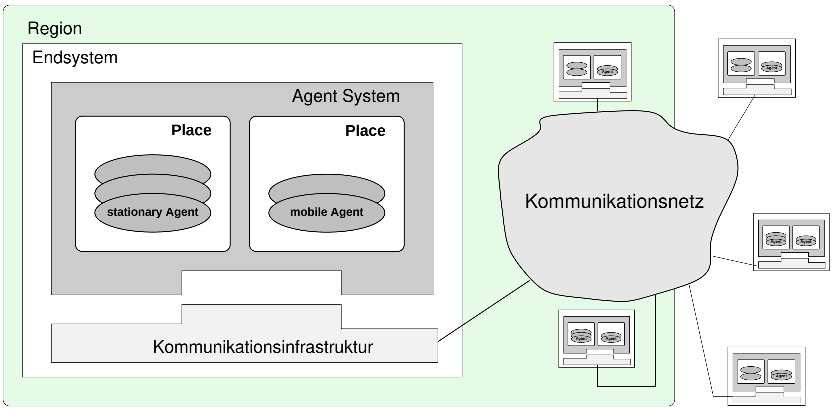

Abbildung 3.2: MASIF–Basismodell

Neben diesen grundlegenden Begriffen wurden in MASIF die folgenden Bereiche standardisiert:

Agent Management

1. Beim Agent Management werden einheitliche Zugriffsschnittstellen definiert, um Agenten erzeugen, suspendieren bzw. reaktivieren oder terminieren zu können. Ziel dabei ist, daß ein Administrator auf die verschiedensten AS mit denselben Operationen zugreifen kann.

Agent Transfer

2. Agent Transfer standardisiert die Art der Übertragung des MA, d.h. die Aufrufschnittstelle (`migrate_agent`) beim AS. Es wird ein Life–Cycle

für Agenten spezifiziert und festgelegt, daß es eine Möglichkeit der Serialisierung für Daten und Code des Agenten geben muß. Die konkrete Realisierung dieser Verfahren ist nicht Teil der Standardisierung, sondern bleibt der Implementierung überlassen.

3. Agent Naming definiert ein Namensschema und die Semantik der Namen sowohl für Agenten als auch für Agentensysteme. Agent Naming

4. Agent Tracking spezifiziert, wie ein Agent lokalisiert werden kann. Agent Tracking

Sicherheitsüberlegungen für Multi–Hop–Agenten, Übersetzung von Agenten in andere Implementierungssprachen und Bridges zwischen verschiedenen AS–Implementierungen, die eine derartige Umsetzung ermöglichen würden, werden explizit als Aspekte genannt, die im Moment nicht standardisiert werden können und sollen.

Kapitel 4

Die MASA–Architektur

Dieses Kapitel gibt einen Überblick über die Bestandteile der *Mobile Agent System Architecture (MASA)*. Für das Verständnis dieses Kapitels sind grundlegende Kenntnisse über Corba (Abschnitt 3.4) und über die Terminologie von MASIF (Abschnitt 3.5) notwendig (vgl. auch Anhang A).

4.1 Übersicht

Agentensysteme und Agenten

Abbildung 4.1 zeigt eine *MASA–Region* bestehend aus *MASA–Agentensystemen*. Ein Agentensystem läuft entweder auf einem zu managenden System (Managed System) oder auf einem Rechner, der zum Managementsystem gehört. Es ist ein Java–Programm in einer Java Virtual Machine (JVM) und stellt die Ablaufumgebung für *MASA–Agenten* bereit, die die eigentlichen Managementaufgaben implementieren.

Ein Agent entspricht einem Prozeß in einem Betriebssystem. Agenten sind die Verwaltungseinheit eines Agentensystems. Sie können auf die zu managenden Systeme über deren vorhandene Managementschnittstellen zugreifen, in der Abbildung am Beispiel eines SNMP–Agenten dargestellt. MASA–Agenten können mit anderen MASA–Agenten kooperieren, vorzugsweise über IIOP mittels Schnittstellen, die in Corba–IDL definiert werden; diese Corba Mechanismen werden auch von MASA selbst verwendet. Außerdem können sie, auch über heterogene Systemplattformen hinweg, von einem Agentensystem zum anderen migrieren.

Web–Oberfläche

Die meisten MASA–Agenten haben eine eigene WWW–basierte Oberfläche, die von einem beliebigen Webbrowser aus aufgerufen werden kann. In der Regel ist in der HTML–Seite ein Java Applet eingebettet, das einem Benutzer die Interaktion mit dem Agenten ermöglicht. Die Kommunikation zwischen Applet und Agent über Corba wird von MASA besonders unterstützt. Auch

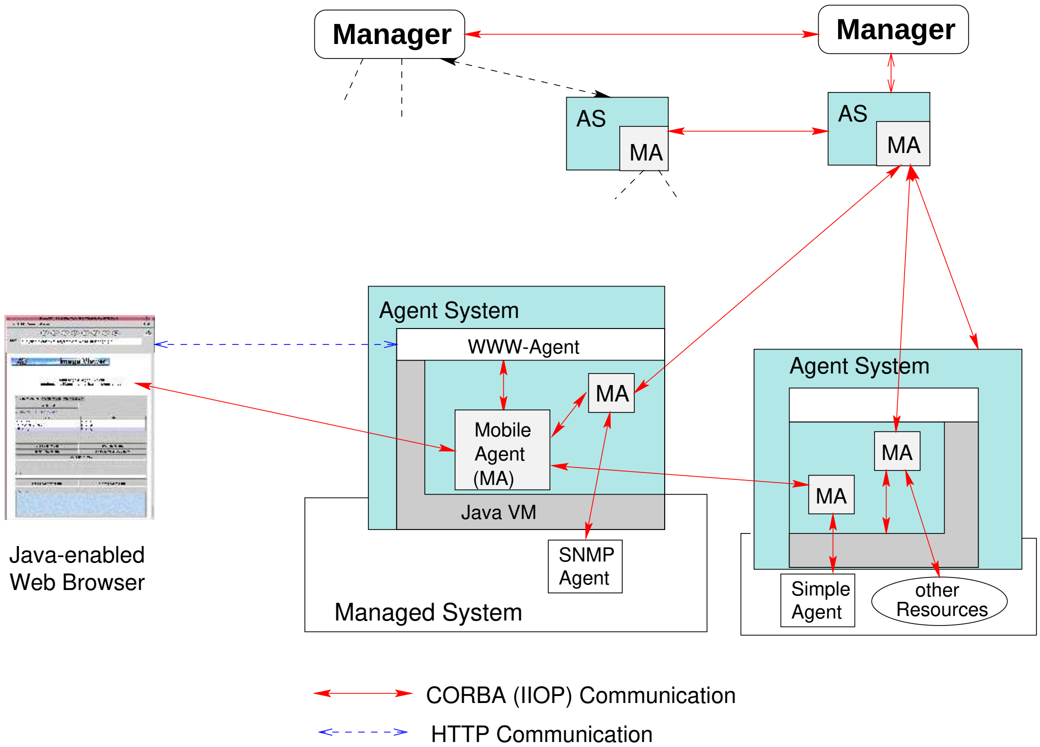

Abbildung 4.1: Die MASA–Architektur

für das Agentensystem selbst existiert eine solche Web–Oberfläche (vgl. Abschnitt 4.5.2) mit einem Java Applet, das die Interaktion mit dem Agentensystem erlaubt und im wesentlichen den in MASIF spezifizierten Funktionsumfang eines Agentensystems abdeckt.

Schichten von MASA

Die MASA–Architektur besteht aus mehreren Schichten, die in Abbildung 4.2 dargestellt sind. Das vernetzte System selbst, dargestellt durch Betriebs– und Transportsystem, ist gegeben. Darüber muß zunächst eine einheitliche Umgebung für die verteilte Anwendung MASA geschaffen werden, was durch Java als Ausführungsplattform und Corba als Transportsystem und Lieferant essentieller Basisdienste erreicht wird.

Diese Plattform für verteilte Anwendungen wird von MASA genutzt, um eine Plattform für verteiltes, kooperatives, Agenten–basiertes Management anzubieten. Dazu wird die Funktionalität der Java–VM um eine Laufzeitumgebung für Agenten erweitert, und die Corba Dienste angewandt.

Die eigentlichen Managementaufgaben werden von MASA–Agenten implementiert, die die oberste Schicht von MASA bilden. Sie realisieren zum einen Funktionalität, die in klassischen Managementsystemen von Managementplattformen erbracht wird, in der Abbildung als Management–Middleware bezeichnet, und Funktionalität, die Managementanwendungen zuzuordnen ist. Die Systemerweiterungen von MASA sind Agenten, die man bei Be-

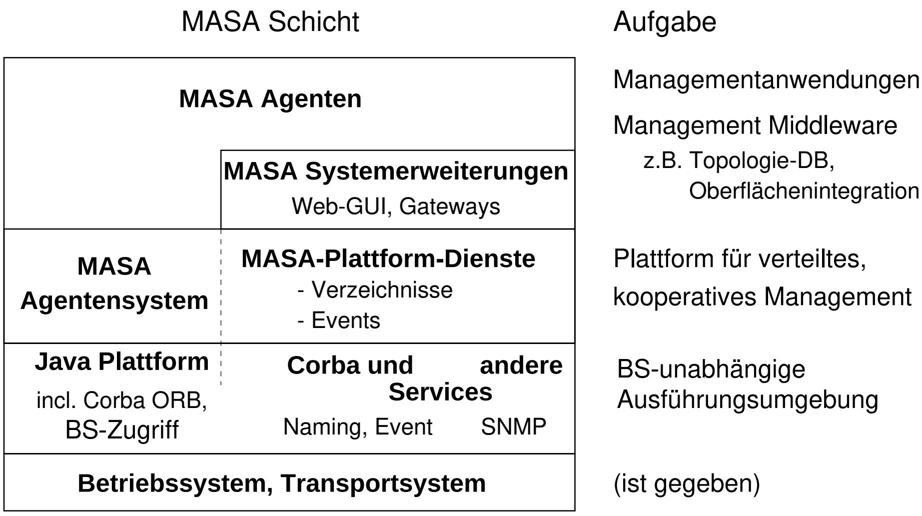

Abbildung 4.2: Schichtenarchitektur von MASA

triebssystemen als „Systemprozesse" bezeichnen würde. Sie stellen optionale MASA–Plattformdienste zur Verfügung (vgl. Abschnitt 4.4).

Komplexitäts–Problem mit Java, Corba und Agenten–Ansatz lösen

Diese Architektur wurde vor dem Hintergrund der Anforderungen an Managementsysteme gewählt. Ein Managementsystem ist eine komplexe, verteilt ablaufende und verteilt entwickelte Anwendung für heterogene IT–Umgebungen, d.h. Komponenten, die auf unterschiedlichen Systemarchitekturen ablaufen, müssen interagieren können und Komponenten müssen möglichst einfach auf unterschiedliche Systemarchitekturen portiert werden können.

Die Komplexität eines Managementsystems und die Notwendigkeit, daß Komponenten auf unterschiedlichen Systemarchitekturen interoperieren müssen, verlangen eine skalierbare, plattformunabhängige Middleware, wie Corba. Durch Java wird der Portierungsaufwand zwar nicht vollständig vermieden, aber gegenüber den anderen verbreiteten Programmierumgebungen erheblich reduziert. Das MASA–Agentensystem und viele MASA–Agenten sind tatsächlich auf vielen Betriebssystemen ohne Änderung ablauffähig. Der Agenten–basierte Ansatz zur Strukturierung des Managementsystems bestimmt die Verteilung der Anwendung zur Laufzeit und soll die verteilte Entwicklung von Managementsystemen erleichtern.

Umbrella–Management

In den letzten Jahren hat die Zahl der Managementarchitekturen stetig zugenommen [HAN 99a]. Um in einem großen Unternehmen oder bei einem Provider die verschiedenen gewachsenen Managementsysteme, die i.d.R. unterschiedlichste Managementprotokolle verwenden, integrieren zu können, ist ein *Umbrella–Management* [Kell 98] nötig, das die verschiedenen Informations–, Kommunikations– und Funktionsmodelle unterstützt. Bestehende Anwendungen oder Agenten müssen einfach in dieses Umbrella–Managementsystem integriert werden können.

MASA wurde so entwickelt, daß es einfach in verschiedene Managementsysteme integriert werden kann, oder auch selbst andere Managementsysteme integrieren bzw. verschiedene Managementparadigmen unterstützen kann. Das gemeinsame Motto ist hierbei „Java, Corba and the Web“. Damit können auch traditionelle Managementsysteme die Vorteile Mobiler Agenten nutzen.

Oberflächenintegration

Web–based Management

Als erster Schritt zu einer vollständigen Integration aller bestehenden Systeme unter einem einheitlichen Umbrella–Management wird zumeist eine *Oberflächenintegration* angestrebt, bei der alle Systeme von einer einheitlichen Oberfläche aus zugänglich sind. Der Manager soll dabei für den Zugang zum Managementsystem nicht auf eine dezidierte Management–Console angewiesen sein, sondern einen WWW–Browser benutzen können (*Web–based Management*).

4.2 Agentensystem

Eine grundlegende Prämisse beim Design von MASA war die Plattformunabhängigkeit. Ein MA sollte auf den verschiedensten (Betriebssystem–) Plattformen ablauffähig sein. Aus diesem Grund bietet es sich an, für die Agenten eine Interpretersprache zu verwenden und den Interpreter in jedes Agentensystem zu integrieren. In MASA wurde für diesen Zweck Java verwendet. Aber nicht nur die MAs selbst, sondern auch das AS und alle anderen Komponenten wurden in Java realisiert.

Das Agentensystem beinhaltet eine Java Virtual Machine für die jeweilige Zielplattform und stellt die Laufzeitumgebung für die MAs dar. Es verwaltet alle MAs, die sich auf den entsprechenden Endsystemen befinden, d.h. pro System und Authority darf maximal ein AS existieren.

API des Agentensystems

Das AS wird durch das Interface `MAFAgentSystem` aus MASIF definiert. Es bietet u.a. folgende Methoden zum Management von Agenten:

- `create_agent` zum Initialisieren und Starten eines Agenten.
- `receive_agent` zum Empfang eines Agenten von einem anderen AS.
- `terminate_agent`, um einen Agenten zu beenden.
- `suspend_agent`, um die Ausführung eines Agenten anzuhalten.
- `resume_agent`, um einen suspendierten Agenten wieder fortzusetzen.
- `migrate_agent`, um einen Agenten zu einem anderen AS zu migrieren. Diese Methode kann vom Agenten selbst, vom AS oder von außen aufgerufen werden.

Außerdem können über die folgenden Methoden Informationen über das AS und die darauf ablaufenden Agenten gewonnen werden:

- `get_agent_system_info` liefert Informationen über das AS, wie unterstützte Sprachen, Namensräume, u.ä.
- `list_all_agents` liefert eine Liste aller Agenten des AS.
- `get_agent_status` liefert Informationen über den Zustand eines Agenten.
- `get_authinfo` liefert Authentisierungsinformation.
- `list_all_agents_of_authority` liefert eine Liste aller Agenten einer bestimmten Authority, die sich auf dem AS befinden.

Die Methode `terminate_agent_system` wird verwendet, um das AS selbst zu beenden.

Um die Kommunikation zwischen MA sowie eine einfache Form der Persistenz zu realisieren, wird die Schnittstelle `AgentSystemService`, die von `MAFAgentSystem` erbt, verwendet. Sie bietet die folgenden Methoden an:

- `connectToAgent`, um mit einem bestimmten Agenten kommunizieren zu können, wird dessen Objektreferenz zurückgeliefert.
- `writeAgentToFile`, um einen Agent serialisiert in einer Datei abspeichern zu können.
- `readAgentFromFile`, um einen Agenten aus einer Datei einzulesen.

Die Aufgabe des `AgentSecurityManager` ist es, die Sicherheitsanforderungen auf dem AS durchzusetzen. Er soll das Sicherheitskonzept von MASA realisieren, also die MAs authentisieren und ihren Zugriff kontrollieren. Der `AgentSecurityManager` identifiziert ankommende MA und gewährt oder verwehrt ihnen, in Abhängigkeit ihres Status und einer Zugriffskontroll–Policy, den Zugriff auf Methoden des AS selbst, auf Ressourcen oder auf andere MAs. Die derzeit prototypische Implementierung wird in [Roel 99a, Zeil 99, Schi 99] zu einem umfassenden Sicherheitskonzept erweitert (vgl. auch Abschnitt 6.1).

4.3 MASA–Agenten

Die zweite Basiskomponente von MASA sind die MASA–Agenten, die ein „Programmiermodell“ bzw. einen Rahmen für den Entwickler darstellen. Dieser Rahmen muß nur mit spezifischer Funktionalität gefüllt werden, um damit alle Vorteile, die Mobile Agenten bieten (vgl. Abschnitt 3.3), nutzen zu können. Das AS und die Agenten zusammen bilden den Kern von MASA.

Es werden Mobile Agenten (MA) und stationäre Agenten (SA) unterschieden. Stationäre Agenten können nicht migriert werden. Sie werden verwendet, um

Methoden zu implementieren, die nur auf dem System ausgeführt werden sollen oder dürfen, auf dem der SA gestartet wurde.

Agenten–API

Beide Arten von Agenten sind von der Klasse `Agent` abgeleitet, die folgende Methoden implementiert:

- `init` Initialisierung und Start des Agenten.
- `initTransient` zur Initialisierung transienter, d.h. nicht serialisierbarer Attribute des Agenten.
- `writeObject` zur Serialisierung von Objekten.
- `readObject` zum Einlesen und Instantiieren serialisierter Objekte.
- `checkSerialization` wird von AS aufgerufen, wenn ein Agent serialisiert werden soll. Der Agent kann sich mit Hilfe dieser Methode in einen Zustand bringen, in dem er serialisiert werden kann oder er kann die Serialisierung ablehnen.
- `cleanUp` gibt vom Agenten belegte Ressourcen frei.

API des Mobilen Agenten

Der Mobile Agent unterscheidet sich vom stationären dadurch, daß er die Schnittstelle `Migration` implementiert, die zur synchronisierten Kommunikation zwischen AS und Mobilen Agenten dient. Zu einem bestimmten Zeitpunkt, kann maximal ein MA pro AS migriert werden. Die Synchronisation und die Zuteilung der „Migrationsressourcen“ wird durch diese Schnittstelle realisiert. Der Mobile Agent implementiert zusätzlich zu den oben angegebenen Methoden, noch folgende:

- `initMigrate` zur Initialisierung der Migration.
- `migrateTo` bzw. `migrateToAgentSystem` zur Migration des MA.

Neben der Unterscheidung von MA und SA werden multiple und exklusive Agenten unterschieden. Von multiplen Agenten darf es mehrere Instanzen innerhalb einer Region (vgl. Abschnitt 3.5) geben; bei den exklusiven Agenten existiert höchstens eine Instanz pro Region (vgl. auch 4.4.1).

4.3.1 Lebenszyklus eines Agenten

Erzeugung

Ein Agent wird durch den Aufruf der Methode `create_agent` des AS gestartet. Diese Methode kann von einem beliebigen Client (z.B. Managementplattform, Browser), einem anderen Agenten oder vom AS selbst aufgerufen werden. Als Parameter der Methode werden der Name und die *Code Base* des Agenten übergeben. Das AS überprüft, ob der Agent gestartet werden darf, lädt den Code, der den Agenten implementiert (lokal oder entfernt), und instantiiert den Agenten. Dann wird vom AS die `init` Methode des Agenten aufgerufen, durch die alle nötigen Attribute des Agenten gesetzt und dieser initialisiert und gestartet wird. Danach befindet sich der Agent im Zustand *running* (vgl. Abbildung 4.3).

Zustand *running*

Um einen MA auf ein anderes AS zu transferieren gibt es mehrere Möglichkeiten. Ein beliebiger Client, ein anderer Agent oder das AS rufen die Methode `migrate_agent` des Quell–AS auf. Der MA kann sich auch selbst migrieren, indem er eine seiner Methoden `migrateTo` bzw. `migrateToAgentSystem` aufruft. Diese Methoden werden auf einen Aufruf der Methode `mobileAgentWantToMigrate` des AS abgebildet. Nach dem Aufruf einer dieser Methoden, befindet sich der MA im Zustand *migrating*. Das AS überprüft, ob der Aufruf zulässig ist, und ruft dann `checkSerialization` beim MA auf, um diesem die Möglichkeit zu geben, sich auf die Serialisierung „vorzubereiten", oder die Migration, durch die Exception `CouldNotMigrate`, zu verhindern. Falls der MA die Migration nicht ablehnt, wird er vom AS durch den Aufruf von `suspend_agent` angehalten. Der MA befindet sich damit im Zustand *suspended*. Anschließend wird er serialisiert und durch Aufruf der Methode `receive_agent` des Ziel–AS auf dieses transferiert. Danach befindet er sich im Zustand *migrated*. Nach erfolgreicher Übertragung wird der MA auf dem Quell–AS durch den Aufruf von `terminate_agent` terminiert und gelöscht. Auf dem Ziel–AS wird der MA durch den Aufruf der Methode `initTransient` initialisiert und gestartet.

Zustand *migrating*

Zustand *suspended*

Zustand *migrated*

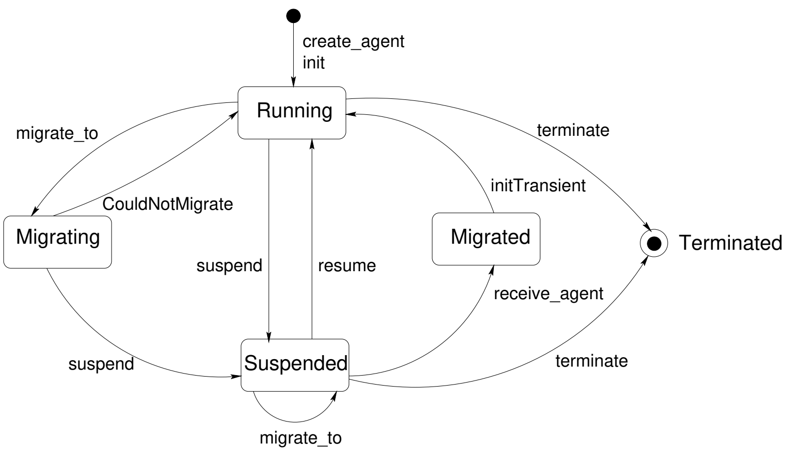

Abbildung 4.3: Lebenszyklus eines Agenten

Die Ausführung jedes Agenten kann durch `suspend_agent` unterbrochen und der Agent damit in den Zustand *suspended* versetzt werden. Um den Agenten wieder zu reaktivieren und in den Zustand running zu versetzen, muß `resume_agent` verwendet werden. Auch ein Agent, der sich im Zustand suspended befindet, kann, durch einen Aufruf von außen, migriert werden.

Anhalten und Fortsetzen *suspended*

Endgültig beendet und gelöscht wird ein Agent durch den Aufruf von `terminate_agent`. Auch diese Methode des AS kann vom AS selbst, von einem beliebigen Client, von einem anderen Agenten oder vom Agenten selbst aufgerufen werden.

Terminierung

4.3.2 Semantik der Migration

Bei der Migration eines Agenten A müssen einige grundsätzliche Überlegungen über die Semantik der Migration angestellt werden. Fragestellungen die dabei u.a. auftauchen sind:

- Muß zu jedem Zeitpunkt t mindestens ein Agent A laufen?
- Ist es erlaubt, daß zu einem Zeitpunkt t ein Agent A und dessen (Migrations–) Kopie A' auf verschiedenen AS laufen?
- Wie werden Fehlerfälle bei der Migration behandelt (Verzögerung, Kopie, Verlust, Beschädigung, ...)?

Um diese Fragen zu lösen, gibt es drei grundsätzliche Ansätze.

at least once Semantik

1. Der zu migrierende Agent läuft auf dem Quell–AS weiter oder wird dort suspendiert, aber nicht gelöscht. Nachdem der Agent auf das Ziel–AS übertragen wurde und dort gestartet werden konnte, wird er im Quell–AS terminiert und gelöscht.
2. Der zu migrierende Agent wird im Quellsystem terminiert, auf das Zielsystem übertragen und sofort gelöscht, ohne eine Bestätigung des Zielsystems abzuwarten.
3. Es wird ein stenges Transaktionskonzept für die Migration von Agenten definiert.

Bei MASA wird im Moment nur der erste Ansatz unterstützt, d.h. die Semantik der Migration ist at least once.

4.4 MASA–Plattformdienste

Prinzipiell reichen für den Aufbau einer lauffähigen MASA–Installation das AS und die Agenten, wie in den vorhergehenden Abschnitten beschrieben, aus. Allerdings können damit nur sehr kleine verteilte Systeme realisiert werden. Damit die Architektur auch skaliert, sind zusätzliche Dienste notwendig und wünschenswert.

Ein wichtiger Dienst, um AS und Agenten benennen und wiederfinden zu können, wird durch den *Naming Service* realisiert. Der *Event–* bzw. *Notification Service* als zweiter Basisdienst stellt einen dezentralen, asynchronen Kommunikationsmechanismus zur Verfügung, der insbesondere dann benötigt wird, wenn mit MASA IT–Management betrieben werden soll.

Im folgenden werden die beiden Basisdienste — die eine Verfeinerung der entsprechenden Corba Dienste darstellen — erläutert; weitere zusätzliche (optionale) Services werden in Abschnitt 4.5 vorgestellt.

4.4.1 Naming Service

Der MASA Naming Service realisiert einen Verzeichnis– und Lokalisationsdienst. Er tritt an die Stelle des in [MASIF] definierten, weniger mächtigen MAFfinders. Der MAFfinder stellt einen speziellen Verzeichnisdienst für Agenten dar. Da aber jedes MASA–Objekt (AS, Agent) mit einem global eindeutigen Namen im Naming Service eingetragen wird, kann die Funktionalität des MAFfinders auch durch den Naming Service erbracht werden. Mit jedem Namen wird auch eine global eindeutige Objektreferenz (Interoperable Object Reference, IOR) assoziiert. Mit der Objektreferenz kann das Objekt lokalisiert werden, d.h. für eine Anwendung ist der Zugriff mittels IOR ortstransparent.

Verzeichnis– und Lokationsdienst anstelle des MAFfinders

In MASA werden strukturierte Namen verwendet. Sie setzen sich aus dem *Agent System Type* der *Authority* und der *Identity* zusammen. Diese drei Teile bilden den Compound Name des AS bzw. des Agenten. Der Agent System Type beschreibt, welches Agentensystem zugrunde liegt. Für einige Agentensystemimplementierungen wurden von der OMG [MASIF] bereits ein Agent System Type definiert.
Die Authority definiert die für das AS verantwortliche Person oder Organisation. Alle AS, die unter der gleichen Authority betrieben werden, bilden eine MASA *Region*. Die Identity bezeichnet den Namen des AS bzw. des Agenten. Sowohl Authority als auch Identity können selbst wieder strukturierte Namen sein.

Namensschema

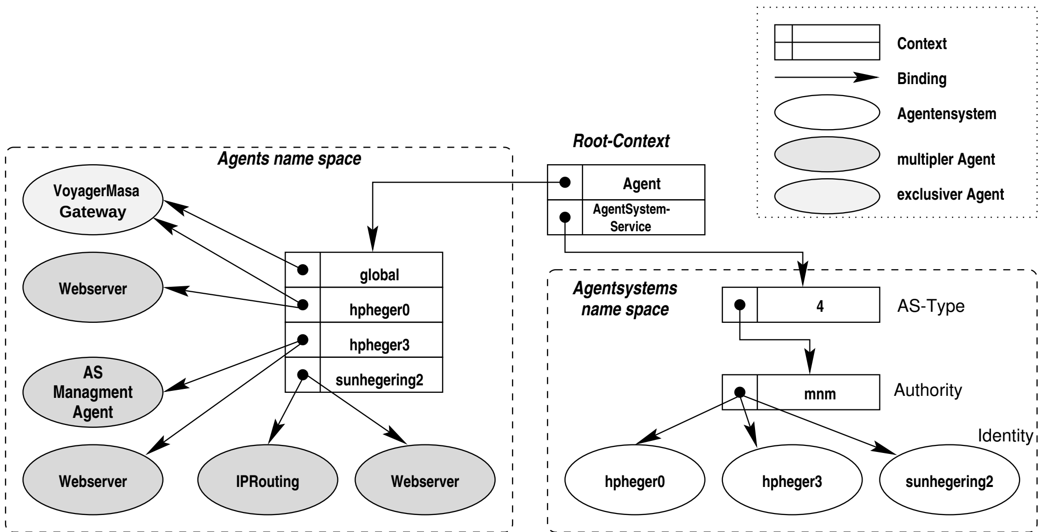

Abbildung 4.4: MASA Naming Service (Bsp. nach [Kemp 98])

Die elementare Einheit im Naming Service ist ein Naming Context, der durch Binding mit anderen Naming Contexten verbunden werden kann. Damit lassen sich hierarchisch strukturierte Verzeichnisse aufbauen (vgl. Abb. 4.4). An den Blättern des entstehenden Baumes befinden sich die Objekte bzw. deren

Struktur des Naming Service

IORs. Der Compound Name eines Objektes kann durch Konkatenation der Naming Contexte von der Wurzel bis zum Blatt des entsprechenden Objektes gebildet werden.

Im Root Context des MASA Naming Service werden Namensräume für Agenten und für Agentensysteme eingetragen. Ein AS mit dem AS–Type „4“, der vom Münchner Netzmanagement Team (mnm) verwaltet wird und auf dem Rechner „hpheger3“ läuft, erhält bspw. den Namen „AgentSystemService/4/mnm/hpheger3“. Der Webserver–Agent, der auf diesem System läuft heißt entsprechend „Agent/hpheger3/Webserver“. Ein Mobiler Agent ändert also nach einer Migration seinen Namen.

Naming Context „global“ für exklusive Agenten

Im Namensraum der Agenten gibt es neben den Naming Contexts, die auf das darunterliegende AS verweisen, einen Naming Context „global“. Darin werden alle exklusiven Agenten eingetragen, von denen es nur eine Instanz pro Region geben darf, d.h. ein exklusiver Agent erscheint zweimal im Naming Service; einmal unter seinem normalen Compound Name daneben aber auch unter „Agent/global“. Um die Konsistenz des Naming Service und die von exklusiven Agenten zu gewährleisten dürfen nur AS Einträge im Naming Service vornehmen.

4.4.2 Event und Notification Service

dezentraler, asynchroner Kommunikationskanal

Mit dem Event Service wird ein dezentraler, asynchroner Kommunikationskanal (*Event Channel*) realisiert, über den Nachrichten, sog. *Events*, verschickt werden können.
Objekte, die Events erzeugen, werden als *Supplier*, die die Events empfangen und verarbeiten, als *Consumer* bezeichnet. Der *Event Channel*, an den sich beliebig viele Supplier und Consumer binden können, realisiert die Entkoppelung der Partner.

Push– oder Pull–Betrieb

Der Event Channel kann im *Push*– oder *Pull–Betrieb* arbeiten. Im Push–Betrieb sendet der Supplier die Events an den Channel, der diese wiederum nach bestimmten Heuristiken an die Consumer weiterleitet. Im Pull–Betrieb fragt der Consumer aktiv beim Channel nach neuen Events, der daraufhin bei allen angeschlossenen Suppliern die bereits erzeugten Events abholt. Es ist auch möglich, den Channel im Push–Pull Modus zu betreiben, d.h. bspw. mit einem Push–Supplier und einem Pull–Consumer.

Notification Service als Erweiterung

Der Notification Service [OMG 98-11-01] ist eine Erweiterung des Event Service um Filtermechanismen und Quality–of–Service–Parameter (QoS). Zu diesem Zweck definiert der Notification Service *Structured Events* und eine Constraint–Sprache um Filter zu definieren. Mit Hilfe von QoS–Parametern, wie z.B. Prioritäten, Timeouts u.ä., lassen sich Dienstgütevereinbarungen realisieren. Mit den Filtermechanismen lassen sich Daten gezielt verdichten;

es können bspw. ganze Klassen von Events, abhängig von sehr feingranular spezifizierbaren Bedingungen, ausgewählt oder auch verworfen werden.

4.5 MASA–Systemerweiterungen

Auf dem in den vorigen Abschnitten beschriebenen Basissystem wurden Systemerweiterungen implementiert, die für die Realisierung eines Web–basierten Managementsystems hilfreich und nützlich sind.

4.5.1 WWW–Agent

Mit Hilfe von MASA wurden Managementanwendungen realisiert, die einen Web–Browser als Management–Konsole verwenden. Das AS selbst und jeder Agent bieten eine „Web–Page“ bzw. ein Applet an, mit dessen Hilfe Zustände abgefragt oder Aktionen ausgelöst werden können. Damit auf diese Seiten zugegriffen werden kann, wurde ein eigener *WWW–Agent* implementiert. Dabei handelt es sich um einen multiplen, stationären Agenten, d.h. innerhalb einer Region kann es mehrere, pro AS jedoch nur einen, WWW–Agenten geben. Der WWW–Agent implementiert das HTTP–Protokoll und liefert die Web–Seiten der Agenten, die auf seinem AS laufen; aus diesem Grund ist der Agent auch stationär und nicht mobil. Auf den WWW–Agenten aufbauend, wurde eine GUI für MASA entwickelt, die einen Überblick über das gesamte verteilte System bietet.

4.5.2 Graphische Benutzerschnittstelle von MASA

Die MASA–GUI [Gerb 99] setzt sich aus dem AgentSystem Applet, dem ASManagement Agent sowie dem RegionManagement Agent und dem RegionManagement Applet zusammen. Der RegionManagement Agent liefert eine globale Sicht auf alle Agentensysteme einer Region, wohingegen der ASManagement Agent die Schnittstelle zu einem bestimmten Agentensystem realisiert.

RegionManagement Agent + Applet

Beim RegionManagement Agent handelt es sich um einen multiplen, Mobilen Agenten, der auf einem oder mehreren beliebigen AS innerhalb einer Region gestartet werden kann. Der RegionManagement Agent liefert Informationen über alle in einer Region aktiven AS und alle exklusiven Agenten. Über das Applet dieses Agenten kann jedes AS und jeder Agent innerhalb einer Region erreicht, abgefragt und manipuliert werden. Wird ein bestimmtes AS ausgewählt, zeigt das RegionManagement Applet alle Agenten auf diesem System. Es bietet auch die Möglichkeit einzelne Agenten zu migrieren.

ASManagement Agent + AS Applet

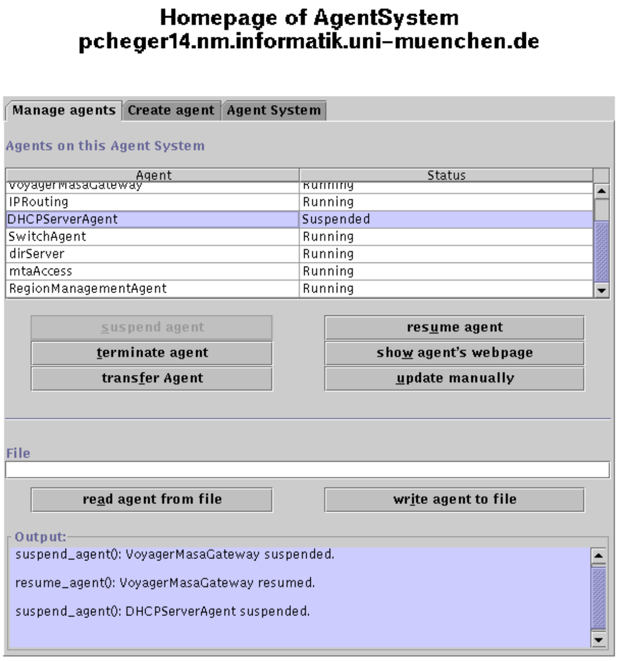

Abbildung 4.5: AgentSystem Applet [Gerb 99]

Über das RegionManagement Applet kann man die WWW–Seite eines bestimmten Agentensystems erreichen. In diesem Fall wird das AgentSystem Applet (vgl. Abb. 4.5) geladen, das die Schnittstelle zum ASManagement Agent darstellt, mit Hilfe dessen der Zugriff auf ein bestimmtes AS erfolgt. Im AgentSystem Applet wird eine Liste aller momentan auf dem AS befindlichen Agenten und deren Zustand angezeigt. Es können Agenten gestartet, suspendiert, reaktiviert oder terminiert werden. Es greift dabei auf die in Abschnitt 4.2 beschriebenen Methoden zu.

Ein bestimmter Agent kann selektiert und dessen Web–Page angezeigt werden. Daneben können über das AgentSystem Applet auch Informationen über das AS selbst gewonnen werden oder das AS kann terminiert werden. Da der ASManagement Agent für genau ein AS verantwortlich ist, wurde er, analog zum WWW–Agenten, als multipler, stationärer Agent implementiert.

Die vier Komponenten ASManagement Agent, AgentSystem Applet sowie RegionManagement Agent und –Applet bilden also die zentrale Management–Konsole zum Management mittels MASA aber auch zum Management von MASA selbst.

4.5.3 Voyager Gateway

Für eine Managementanwendung ist es wichtig und notwendig, bestehende Systeme relativ leicht integrieren zu können. Besonders wichtig ist dies für kommerzielle Agentensysteme und die Anwendungen, die dafür entwickelt wurden (z.B. [FKK 99]). Um zu zeigen, daß andere AS in MASA integriert werden können, wurde ein Voyager–MASA Gateway [Bran 99] entwickelt.

Ziel war dabei eine möglichst transparente Integration von auf Voyager (Version 2.0 [Voyager 2.0]) basierenden Systemen in MASA. Es sollten insbesondere Voyager Agenten aus MASA heraus überwacht werden können (Monitoring). Daneben sollte auch eine Steuerung der Voyager Agenten ermöglicht werden, d.h. Voyager Agenten sollten von MASA aus gestartet und gestoppt sowie Methoden der Agenten verwendet werden können. Sowohl Monitoring als auch Steuerung sollten über die GUI von MASA erfolgen (vgl. Abb. 4.6).

Obwohl auch Voyager Agenten in Java implementiert werden und eine Corba Schnittstelle bieten, sind die Basiskonzepte so unterschiedlich, daß Voyager Agenten nicht direkt in MASA integriert werden können. So benötigen Voyager Agenten bspw. ein als Voyager–Server bezeichnetes AS, um ablaufen zu können. Auch die Namensgebung, sowie die unterstützten Verzeichnisdienste unterscheiden sich grundlegend von denen von MASA.

VoyagerMasa-Gateway Agent

Die zentralen Komponenten des Gateways sind der VoyagerMasaGateway Agent, ein MASA–Agent, der Voyager Agent Manager, ein Teil des MASA–AS, sowie ein Naming Gateway. Der VoyagerMasaGateway Agent stellt die zentrale Schnittstelle zwischen Voyager und MASA dar und wurde deshalb als globaler, stationärer Agent implementiert, der auf einem beliebigen AS innerhalb einer Region ausgeführt werden kann. Der Voyager Agent Manager stellt eine Erweiterung des AS dar und verarbeitet alle Systemaufrufe, die Voyager Agenten betreffen.

Abbildung 4.6: VoyagerGateway Applet [Bran 99]

Voyager Agent Manager

Abbildung der unterschiedlichen Namensräume

Der Zugriff auf Voyager Agenten ist nur möglich, wenn eine geeignete Abbildung zwischen dem Voyager Naming Service und dem MASA Naming Service realisiert wird. Diese Abbildung wird durch das NamingGateway, unter Kontrolle des VoyagerMasaGateway Agenten, realisiert. Das Monitoring der Voyager Agenten erfolgt in folgenden Schritten (s. Abb. 4.7):

Monitoring

1. Ein Voyager Agent wird auf einem Voyager Server kreiert bzw. terminiert.

2. Das NamingGateway registriert die Veränderung im Voyager Namespace und schickt eine entsprechende Nachricht an den VoyagerMasa-Gateway Agenten in MASA.

3. Das VoyagerMasaGateway reicht diese Nachricht an den VoyagerAgent Manager weiter.

4. Der VoyagerAgentManager greift auf die MASA–Agentensystemschnittstelle zu, um einen VoyagerProxy (MASA–)Agenten zu erzeu-

gen.

5. Das MASA–AS erzeugt diesen VoyagerProxy Agenten, der sich für das AS nicht von einem mobilen MASA–Agenten unterscheidet.

6. Der Name des VoyagerProxyAgenten wird im MASA–Namensverzeichnis in einem ausgezeichneten Voyager-Verzeichnis eingetragen.

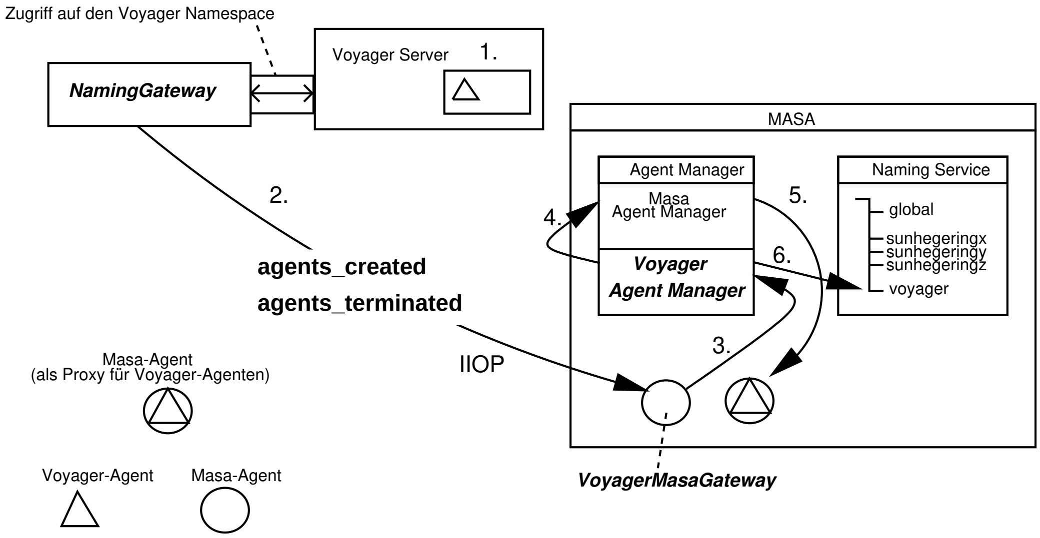

Abbildung 4.7: Monitoring von Voyager Agenten [Bran 99]

Auch für den VoyagerMasaGateway Agenten wurde ein Applet als Schnittstelle entwickelt. Über dieses Applet (vgl. Abb. 4.6) können Voyager Agenten erzeugt, migriert sowie terminiert werden. Es können Methoden des Voyager Agenten ausgeführt werden. Für alle Operationen, die an der Benutzerschnittstelle ausgelöst werden, werden vom Voyager Agent Manager entsprechende MASA–Aufrufe (z.B. zur Erzeugung des Proxies) sowie Aufrufe auf dem entsprechenden Voyager Server erzeugt. Der Voyager Agent Manager und das VoyagerMasaGateway konzentrieren also das „Voyager Wissen“ innerhalb von MASA, dadurch kann die API von MASA unverändert bleiben.

Kapitel 5

Implementierung

In diesem Kapitel werden grundlegende Informationen zur Übersetzung und Installation von MASA (Abschnitt 5.2) vermittelt. Dazu wird im nächsten Abschnitt das verwendete Versionsverwaltungssystem und die dabei zu beachtenden Policies vorgestellt. Im Abschnitt 5.3 werden einige der Agenten vorgestellt, die mit Hilfe von MASA implementiert wurden.

5.1 Verwaltung des Quellcodes mit CVS

Da in der Regel mehrere Entwickler gleichzeitig an MASA arbeiten, ist es notwendig, ein zentrales Code Repository zu besitzen und ein Versionsverwaltungssystem zu verwenden. Bei der Entwicklung von MASA wird das *Concurrent Versions System (CVS)* [CVS], verwendet. Die Grundidee dabei ist, daß jeder Enwickler sich den Quellcode aus dem zentralen Respository in sein lokales Home Verzeichnis „aus–checkt" (z.B. `cvs -d /proj/software/cvs checkout Masa`), auf der lokalen Kopie der Quellen arbeitet und danach seine Änderungen wieder ins zentrale Repository „eincheckt" (`cvs commit`)(siehe auch `http://wwwmnmteam.informatik.uni-muenchen.de/proj/software/htdocs` (nur am Lehrstuhl zugreifbar)). CVS stellt ein zentrales Repository zur Verfügung, ermöglicht eine Kontrolle über den Quelltext und zeichnet sich u.a. durch folgende Eigenschaften aus:

Concurrent Versions System (CVS)

- Änderungen anderer Entwickler können einfach durch ein Update (`cvs update`) in die lokale Kopie der Quellen übernommen werden.
- Entwickler können gleichzeitig an denselben Quellen arbeiten. Werden die Änderungen ins zentrale Repository eingecheckt, kann CVS die Änderungen der verschiedenen Entwickler automatisch zusammenführen. Dies ist jedoch nicht immer möglich. Wird von zwei Entwicklern bspw. dieselbe Zeile im Quelltext der jeweiligen lokalen Kopie

editiert, entsteht beim Einchecken ein Konflikt, den CVS nicht automatisch auflösen kann. Dieser Konflikt wird demjenigen, der als letzter eincheckt bekanntgegeben und muß von diesem manuell aufgelöst werden. Das Zusammenführen (merge) geschieht immer in der lokalen Kopie, d.h. CVS verlangt ggf. daß man ein `cvs update` macht, wobei dann der eventuell manuell zu korrigierende Konflikt in der lokalen Kopie entsteht, bevor ein `cvs commit` erlaubt wird.

- CVS verwaltet eine Historie über alle Änderungen im Quelltext. Damit ist es möglich für jede einzelne Quelltextdatei jede frühere Version wieder herzustellen.
- CVS ermöglicht Release–Stände zu definieren, um bspw. den Entwicklungsstand einer bestimmten Auslieferungsversion zu kennzeichnen und diesen, auch zu einem späteren Zeitpunkt, wieder im ursprünglichen Zustand herstellen zu können.
- CVS erlaubt „Seitenzweige“ in einem Repository. Damit lassen sich z.B. Bug–Fix Versionen oder Varianten mit speziellen Eigenschaften realisieren. Im Falle von MASA wurde dieser Mechanismus bspw. dazu genutzt, um MASA gleichzeitig für JDK Version 1.1 sowie für Java 2 zur Verfügung zu stellen. Im Idealfall können auch die Seitenzweige nach einer bestimmten Zeit (z.B. endgültiger Umstieg auf die Java 2 Plattform) wieder automatisch zusammengeführt werden (`cvs merge`).

Obwohl oder gerade weil CVS ein so mächtiges Werkzeug ist, müssen bestimmte Richtlinien für dessen Verwendung vorgegeben und von den Entwicklern auch eingehalten werden. Im Falle des MASA–Repository sind folgende Policies festgelegt worden:

- Es werden nur Quellen eingecheckt, die sich auch übersetzen lassen und deren Lauffähigkeit auch getestet wurde.
- Jede Datei, die generiert werden kann, wird nicht ins Repository mitaufgenommen. Es werden nur die primären Quelltexte (z.B. nicht generierte `.java`, `.idl`) und keine von Compilern generierten Dateien (z.B. `.class`) eingecheckt.
- Jedes Paket muß ein Makefile mit bestimmten Regeln enthalten (z.B. „all“ um alles zu erzeugen oder „clean“ um alle Dateien zu löschen, die nicht in cvs eingecheckt werden). Außerdem muß eine README Datei, die Funktionsweise, die Konfiguration und Informationen zu evtl. noch nicht behobenen Bugs enthalten. Die Installation des Pakets wird in einer INSTALL Datei beschrieben. Damit soll einem anderen Entwickler ermöglicht werden, fremde Pakete sehr schnell und einfach verwenden und in seine eigene Entwicklung integrieren zu können.

5.2 Installation

Für die Installation und die Übersetzung des MASA–Basissystems und der Agenten müssen die verwendeten Werkzeuge und Umgebungsvariablen möglichst zentral und einheitlich konfigurierbar sein. Es sollen verschiedene Betriebssystemplattformen sowohl für die Übersetzung als auch für die Installation unterstützt werden. Änderungen in der Konfiguration müssen einfach und konsistent durchführbar sein. Um dies zu erreichen wurde eine Verzeichnisstruktur und eine Struktur von Makefiles definiert, die im folgenden beschrieben wird.

5.2.1 Verzeichnisstruktur

Im (CVS–) Basisverzeichnis `Masa` liegen alle Komponenten des Agentensystems, Agenten sowie GUI–Komponenten. Unter diesem Verzeichnis liegt jeder Agent in einem eigenen Paket; das AS selbst liegt unter `system`, die graphische Oberfläche für das AS unter `system_gui`. Im folgenden wird die Struktur des `system` Pakets exemplarisch dargestellt (vgl. Abb. 5.1). Die Namen und Pfade für die schattiert dargestellten Verzeichnisse können durch Setzen geeigneter Makefile Variablen (vgl. Abschnitt 5.2.2) frei vergeben werden.

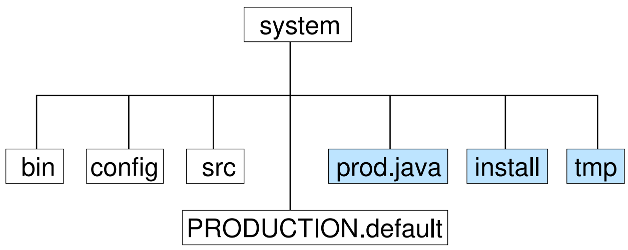

Abbildung 5.1: Verzeichnisstruktur des Agentensystems

- Das `bin` Verzeichnis enthält Hilfsprogramme, die zur Übersetzung des AS oder zur Erstellung spezieller Dateien (z.B. für die Konfiguration) benötigt werden.
- Im Verzeichnis `config` befinden sich Makefiles die in Abschnitt 5.2.2 näher beschrieben werden.
- `src` enthält in einer Reihe von Unterverzeichnissen, ausschließlich jene Quelltexte, die von Hand (also nicht durch Tools wie den IDL–to–Java Compiler) erstellt wurden.
- `PRODUCTION.default` enthält Dateien, die mittels CVS verwaltet werden sollen, aber noch nicht an ihrem endgültigen Bestimmungsort liegen

(z.B. HTML–Dateien die von Agenten benötigt werden, aber erst bei der Installation ins entsprechende Verzeichnis kopiert werden).

- `prod.java` enthält alle Quelltexte die durch einen Präprozessor oder andere Werkzeuge automatisch erstellt wurden, sowie die daraus übersetzten `.class`–Dateien.
- Das Verzeichnis `install` enthält alle zur Ausführung des AS notwendigen Komponenten mit Ausnahme der ORB und Swing Klassen. Informationen über den Aufbau dieses Verzeichnisses sind in [Roel 99a] zu finden.
- `tmp` dient für temporäre Dateien.

5.2.2 Makefile–Schema

Im Verzeichnis `system` befindet sich das Makefile, das alle Aktionen, die zur Übersetzung des Agentensystems notwendig sind, initiiert und koordiniert. In der Datei `Makefile.DEF` im selben Verzeichnis befinden sich Voreinstellungen, die Verzeichnisse betreffen. In dieser Datei sind bspw. die Variablen für die Verzeichnisse in die das Agentensystem installiert wird (vgl. Tabelle 5.1) zu setzen. Beide Dateien enthalten Einstellungen, die unabhängig von verwendeten Betriebssystemen oder Übersetzungswerkzeugen verwendet werden können.

Verzeichnisname	Makefile Variable	Bemerkung
`bin`	—	nicht veränderbar
`PRODUCTION.default`	—	nicht veränderbar
`src`	`BUILD_SOURCE_PATH`	sollte nur für Agenten und nicht für das AS verändert werden
`config`	`MASA_MAKE_CONFIG`	sollte nur für Agenten und nicht für das AS verändert werden
`prod.java`	`PROD_DIR`	
`install`	`MASA_INSTALL_PATH`	
`tmp`	`BUILD_TMP_PATH`	

Tabelle 5.1: Konfiguration von Verzeichnisnamen und Pfaden

Alle für Werkzeuge und Plattformen spezifischen Einstellungen sind in Makefiles im Unterverzeichnis `config` enthalten; sie werden automatisch vom Makefile im `system` Verzeichnis eingebunden. Im einzelnen sind dies:

- `Makefile.masaconf` enthält Basiseinstellungen für das gesamte MASA–System.

- `Makefile.buildtools` definiert die für die Übersetzung und den Start des Agentensystems notwendigen Tools.
- `Makefile.orbsetup` enthält Definitionen für Werkzeuge und Einstellungen für die Corba Entwicklungsumgebung.
- `Makefile.toolconfig` definiert zu verwendende Optionen für alle Werkzeuge die für die Übersetzung und den Start von Masa benötigt werden.

5.2.3 Installationsvorgang

Für ein Vollinstallation von MASA muß das Agentensystem, die graphische Benutzeroberfläche und der Webserver–Agent übersetzt und installiert werden. Dazu müssen die Quellen (`system, system_gui` und `Webserver`) aus CVS ausgecheckt werden (vgl. Abschnitt 5.1). Die im vorigen Abschnitt beschriebenen Makefiles müssen entsprechend angepaßt werden. Im jeweiligen Root Verzeichnis der entsprechenden Komponenten sind folgende Schitte auszuführen:

- `make proddir` erzeugt alle Dateien in `prod.java`. Der IDL–to–Java Compiler erzeugt dabei entsprechende Klassen. Danach werden die Quellen aus dem `src` Verzeichnis durch einen Präprozessor verarbeitet und dessen Ausgabe ebenfalls nach `prod.java` abgelegt.
- `make` startet den eigentlichen Übersetzungslauf.
- `make install` erzeugt Java Archive (`.jar` Dateien) und Startscripten und kopiert diese in die entsprechenden Verzeichnisse unterhalb des `install` Verzeichnisses.
- `make install_doc` erzeugt die Dokumentation mit Hilfe von `javadoc` und installiert diese unter `install/htdocs`.

Nachdem alle Komponenten übersetzt und installiert wurden, sollte in das `install` Verzeichnis gewechselt werden. Um das System zu starten muß dann noch
`./bin/masaScript start naming channel system`
eingegeben werden.

5.3 Implementierte MASA–Agenten

Neben dem Basissystem wurden mehrere MASA–Agenten implementiert.

5.3.1 Abhängigkeitsgraphen — DGAgents

In Abschnitt 2.3 wurde ein Ansatz für einen Eventkorrelator auf Basis von Abhängigkeitsgraphen vorgestellt. Die dargelegten Anforderungen an eine Realisierung des Abhängigkeitsgraphen können erfüllt werden, wenn die Module des Abhängigkeitsgraphen durch MASA–Agenten implementiert werden.

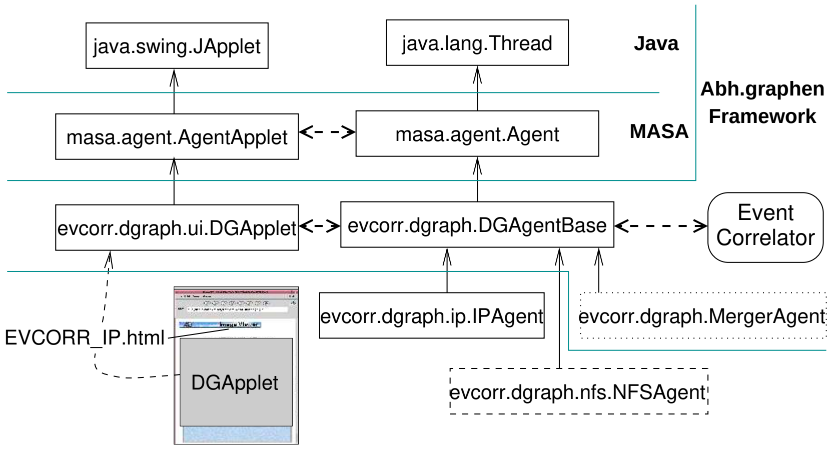

Abbildung 5.2: Softwarearchitektur des Prototypen

Abbildung 5.2 zeigt die Java–Klassenhierarchie von „Abhängigkeitsgraphen–Agenten“ (*DGAgents*) und ihrer Oberfläche. Die durchgezogenen Pfeile bezeichnen Subklassen–Beziehungen, die gestrichelten Pfeile Kommunikationsbeziehungen auf Basis von Corba/IIOP.

DGAgentBase stellt die IDL–Schnittstelle für den Zugriff auf einen Abhängigkeitsgraphen bzw. einen Teil davon zur Verfügung. Die Schnittstelle wird in den Subklassen von DGAgentBase implementiert. Abbildung 5.3 zeigt das zugehörige IDL–Modul. DGAgentBase implementiert ein DGAgent–Objekt.

Für Eventkorrelation sind nur zwei Operationen notwendig, die von DGAgents zu implementieren sind: getObject liefert einen Knoten des Abhängigkeitsgraphen. getAdjacent liefert zu einem gegebenen Knoten diejenigen Knoten, die von diesem Knoten abhängig sind, unter Umstände unter Zuhilfenahme anderer DGAgents. Die Knoten des Abhängigkeitsgraphen sind selbst auch als Corba–Objekte implementiert, was die Performance der Implementierung beeinträchtigt, aber den Vorteil einer leichteren Erweiterbarkeit mit sich bringt.

5.3.2 IPRouting Agent

Der DGAgent zur Darstellung von IP–Netzen stützt sich selbst auf einen MASA–Agenten ab, der die IP–Topologie für eine gegebene Menge an Rech-

```
module evcorr {

  module dgraph {

    exception NoSuchDGObjectException {};

    interface DGObject { string toString(); };

    typedef sequence<DGObject> DGObjectEnumeration;

    typedef string DGOIdentifier;

    interface DGAgent {

      DGObject getObject(in DGOIdentifier id)
           raises ( NoSuchDGObjectException );

      DGObjectEnumeration getAdjacent(in DGObject obj)
           raises ( NoSuchDGObjectException );
    };
  };
};
```

Abbildung 5.3: Abhängigkeitsgraphen API in Corba–IDL

nern ermittelt. Dieser IPRouting Agenten wurde in [Kemp 97] und [Kemp 98] entwickelt. Der IPRouting Agent selbst benutzt die Information der SNMP–MIB–II, um die IP–Topologie zu ermitteln, ähnlich wie das eine Netzmanagementplattform tun würde.

```
module evcorr {

  module iprouting {

    exception ResourceException {string reason; };
    ...Deklarationen der Datenstrukturen ...
    interface IPRouting : agent::Migration {
      HostList getHosts()
           raises ( ResourceException );

      HostInterfaceList getHostInterfaces(in string community)
           raises ( ResourceException );

      HostIPRoutingList getIPRouting(in string community)
           raises ( ResourceException );

      update();
    };
  };
};
```

Abbildung 5.4: API des IPRouting–Agenten in Corba–IDL

Abbildung 5.4 stellt die IDL–Schnittstelle des IPRouting Agenten dar. Sie besteht im wesentlichen aus Operationen zum Abfragen der IP–Topologie.

5.3.3 Policy–Managementagenten

Im Rahmen der Arbeiten zum Policy Management (vgl. Abschnitt 2.1 wurden folgende Agenten entwickelt:

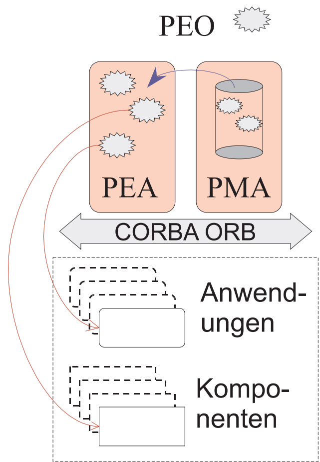

Abbildung 5.5: Policy Management Architektur

Der *Policy Managementagent (PMA)* dient zur Verwaltung der zielorientierten und der strategischen Policies sowie zur Bereitstellung der operationalen Policies beim Start eines *Policy Enforcement Agent (PEA)*. Dieser Agent wird zusammen mit den eigentlichen Managementagenten für die zu verwaltenden Ressourcen ausgeführt und implementiert die durchzusetzenden Policies. Technisch gesehen sind die Policies als Corba Objekte (*Policy Enforcement Objects (PEO)*) implementiert, die aufgrund vom Managementsystem definierter Ereignisse bestimmte Auswertungen vornehmen und Aktionen ausführen.

Der *Persistence Storage Agent (PSA)* implementiert eine allgemein nutzbare Schnittstelle, die andere Agenten zur persistenten Speicherung ihrer Managementobjekte nutzen können. Seine Funktionalität wurde zur Wiederverwendung in anderen Arbeiten als eigener Agent gekapselt.

Eine genauere Beschreibung der Implementierung einschließlich der Corba Schnittstellen sowie der Policy–Beschreibungssprache findet sich in [Radi 98] und [Goli 99].

5.3.4 DHCP–Server

Das *Dynamic Host Configuration Protocol (DHCP)* ist in Internet–Umgebungen das Protokoll zur dynamischen Konfiguration von Endystemen. Es bietet eine Reihe von Vorteilen, die in [Demm 98, May 98] genauer beschrieben sind. In [Demm 98] wurde nicht nur die Managementfunktionalität, sondern auch ein Teil der Serverfunktionalität der DHCP–Architektur selbst in Form eines MASA–Agenten implementiert. Damit ist in Kombination mit den in Abschnitt 2.1 beschriebenen Policy–Agenten eine größere Flexibilität bei der „Zuteilung“ von Konfigurationsinformation an Endsysteme möglich. Dies wird dann speziell zur Unterstützung unterschiedlicher zielorientierter Policies bezüglich des Anschlusses nomadischer Systemen verwendet.

5.3.5 Managementagent für einen Linux–basierten Switch

Zur Gewährleistung von Sicherheit in leitungsgebundenen Lokalen Netzen (LAN) ist es unabdingbar, die Zugänglichkeit in den unteren Schichten 1 und 2 (Physical und Data Link Layer) zu regeln. Da eine physische Sicherheit in öffentlich zugänglichen Bereichen nur unter hohen Kosten, und selbst dann nicht zuverlässig realisiert werden kann, muß eine Abschottung der Endsysteme auf Schicht 2 und 3 (Network Layer) erfolgen. Bereits zur Gewährleistung gewisser Leistungs–, Fehlerausbreitungs– und Sicherheitsanforderungen in weitgehend geschlossenen Umgebungen wurden hierzu das Konzept der *virtuellen LANs* kreiert.

Im Rahmen der Forschungsarbeiten des MNM–Teams wurde hierzu in [Allg 98] eine Switch–Implementierung auf Linux–Basis entwickelt, die den Managementzugriff auf die von ihr implementierten Funktionalität durch einen stationären MASA–Agenten verwirklicht.

5.3.6 Corba Topology Service

Der *Corba Topology Service* ist ein Standard der *Telecom Domain Task Force (DTF)* innerhalb der OMG. Im Rahmen von [Roel 99b] wurde er fast vollständig in Form von MASA–Agenten implementiert.

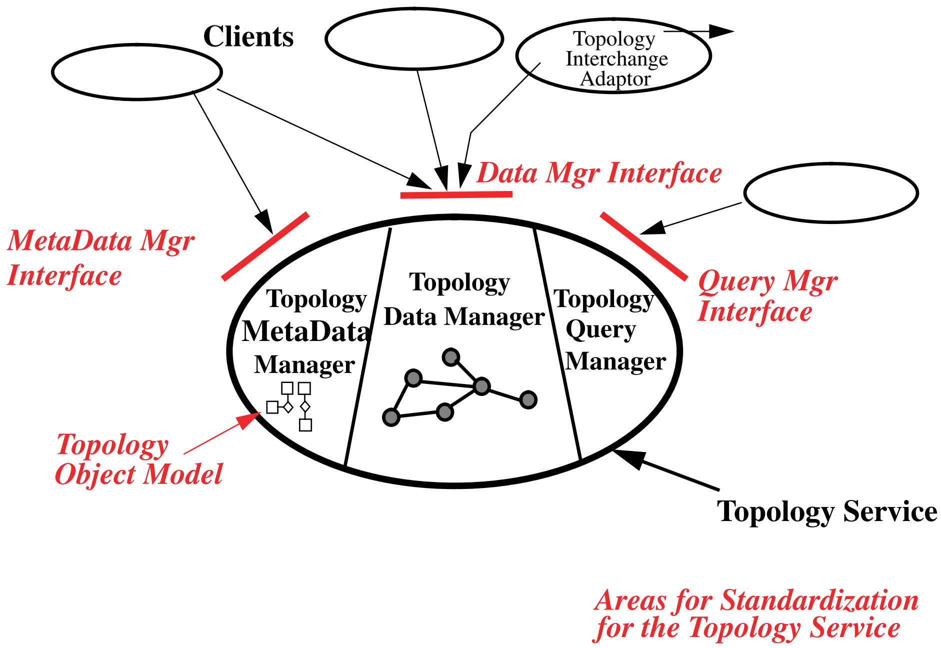

Abbildung 5.6: Schnittstellen des *Corba Topology Service* [OMG 97-10-02]

Der Corba Topology Service kennt drei Schnittstellen, um die durch ihn verwaltete (Topologie)–Information zu speichern (vgl. Abbildung 5.6):

- Über das *Meta Data Interface* werden Topologieregeln (*Rules*) verwaltet, die Restriktionen auf den Objekt(typ)en (im *Topology Object Model*) der gespeicherten Objekte ausdrücken können.
- Die eigentliche Topologieinformation ist über das *Data Manager Interface* manipulierbar, und
- mittels einer eigenen Abfragesprache kann am *Query Manager Interface* auf die gespeicherte Information zugegriffen werden.

Die Definition des Corba Topology Service selbst legt eine verteilte Implementierung nahe, wie sie im Rahmen von [Roel 99b] verwirklicht wurde und in Abbildung 5.7 dargestellt ist.

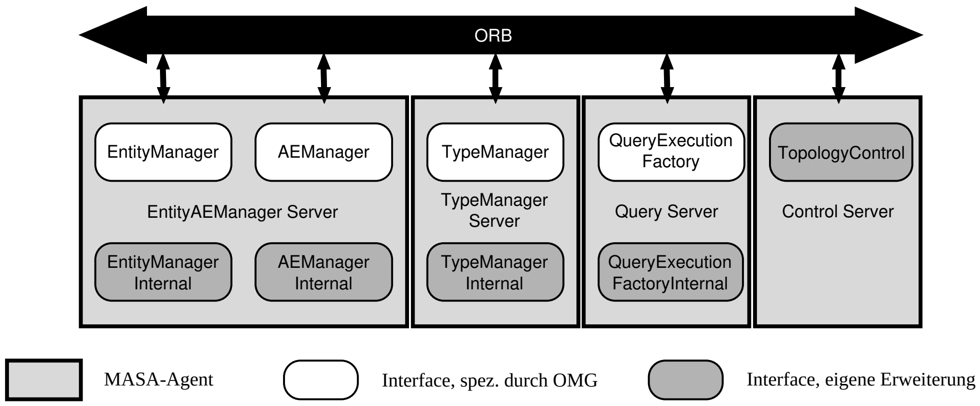

Abbildung 5.7: Implementierung *Corba Topology Service* als MASA–Agenten [Roel 99b]

5.3.7 Email Management Agent

Zum generischen *Message Tracking* in SMTP–basierten Email–Systemen wurde in [Coeh 98] ein objektorientiertes Modell derartiger Systeme auf der Basis von ODP Viewpoints entworfen. Eine prototypische Implementierung in Form eines MASA–Agenten zur Verwaltung der Liste erlaubter SMTP–Clients stellt folgende Funktionen zur Verfügung:

- `InitializeDatabase()`: Diese Methode erzeugt eine neue, leere Zugriffsliste.
- `GetKeyVal()`: Mit dieser Methode läßt sich gezielt nach Zugriffsrechten für einen oder mehrere Clients suchen.

- `RetrieveDatabase()`: Diese Methode liefert die gesamte Zugriffsliste zurück.
- `SetKeyVal()`: Der Aufruf dieser Methode setzt oder löscht Zugriffsrechte.
- `UpdateDatabase()`: Diese Methode entspricht einem Commit, d.h. mit SetKeyVal() vorgenommene Änderungen werden erst nach dem Aufruf dieser Methode übernommen.

Wie bei anderen MASA–Agenten wird der Zugriff auf diese Funktionalität auch in Form eines Applets geschaffen.

Kapitel 6

Ausblick

In den folgenden Abschnitten werden offene Fragestellungen und Themenschwerpunkte für zukünftige Entwicklungs– und Forschungsarbeiten vorgestellt. Sicherheitsapekte von Managementsystemen die auf Mobilen Agenten basieren oder diese verwenden werden im folgenden Abschnitt diskutiert. In Abschnitt 6.2 werden Corba Dienste vorgestellt, die für MASA sinnvoll und nützlich wären aber im Moment noch nicht implementiert bzw. integriert sind. Der Abschnitt 6.3 diskutiert inwieweit das JavaBeans Komponentenmodell Einfluß auf die weitere Entwicklung von MASA haben kann. Der letzte Abschnitt befaßt sich mit der Möglichkeit Domänen innerhalb von MASA–Regions zu etablieren.

6.1 Security Integration

MASA ist als Forschungsprototyp entstanden, mit dem das Konzept der Mobilen Agenten im IT–Management realisiert wurde; der Fokus lag deshalb nicht primär auf Sicherheitsüberlegungen.

Schutz vor feindlichen MAs

Bei der Verwendung von flexiblen, verteilten Managementarchitekturen spielen Sicherheitsaspekte im Hinblick auf die Akzeptanz und Anwendbarkeit der MA–Technologie eine kritische Rolle. IT–Management setzt die Möglichkeit der Kontrolle und des Zugriffs auf zu verwaltende Ressourcen voraus. Die verwendeten MAs müssen daher zum Teil erhebliche Rechte besitzen. Für gewisse Aufgaben ist es notwendig, einen MA mit Administratorrechten (root–Rechten) auszustatten. Eine Plattform für Mobile Agenten könnte daher z.B. für die Verteilung von Viren in Form von MAs mißbraucht werden.

Die Verfügbarkeit von strengen Sicherheitsmerkmalen erhöht die Akzeptanz eines Systems wie MASA nicht merklich; das Fehlen von Sicherheit führt jedoch dazu, daß solche Systeme, trotz aller Vorteile die sie bieten, im praktischen Betrieb nicht eingesetzt werden. Aus diesem Grund wird in derzeit durchgeführten und zukünftigen Arbeiten (z.B. [Roel 99a, Zeil 99, Schi 99]) eine Sicherheitsarchitektur für MASA entwickelt.

Mehrere administrative Domänen

Wegen der Größe vieler Unternehmen und ihrer IT–Systeme wird ein gemeinsames Managementsystem häufig von verschiedenen organisatorischen Einheiten betrieben. Ein ähnliches Problem taucht bei Customer–Provider Beziehungen auf; ein oder mehrere Provider stellen einem oder mehreren Kunden Infrastrukturen zur Verfügung, auf denen der Kunde eigene IT–Infrastrukturen aufbaut. Betrachtet man das Management, so wird klar, daß sowohl Kunde als auch Provider ihre Infrastruktur eigenverantwortlich verwalten wollen. Dies bedeutet, daß, zum Teil auf derselben Hardware, verschiedene Managementsysteme zum Einsatz kommen. MASA bietet nun die Möglichkeit, eine einheitliche Ausführungsplattform für die Managementagenten von unterschiedlichen organisatorischen Einheiten (Authorities) zur Verfügung zu stellen. Auf einem AS, das auf einem Endsystem von der Authority A betrieben wird, können MAs der Authority B und C ablaufen, die, mittelbar über das AS, auf Endsystemressourcen zugreifen und diese verändern können. Es ist klar, daß A sein Endsystem und sein AS vor mißbräuchlicher Benutzung und Beschädigung schützen möchte. Aber auch die MAs von B und C müssen voreinander geschützt werden können. Da ein AS einen MA bei dessen Ausführung vollständig unter Kontrolle hat, könnte ein feindliches AS sowohl dessen Code als auch seine Daten verändern. Auch dies sollte durch geeignete Mechanismen verhindert werden können.

Angriffe	
aktive	passive
Zerstörung Veränderung; Mißbrauch denial–of–service denial–of–execution Maskerade; man–in–the–middle Wiedereinspielung Logic Bomb	Abhören Enthüllung Verkehrsflußanalyse

Tabelle 6.1: Klassifikation von Angriffen

Um ein System abzusichern, muß eine Risikoanalyse durchgeführt werden, damit potentielle Angriffe (vgl. Abb. 6.1) auf ein System ermittelt werden. Die Quellen dieser Angriffe oder Bedrohungen werden auch als *Subjekte*, die Ziele als *Objekte* bezeichnet. Aus dem o.a. Szenario wird deutlich, daß Benutzer Subjekte des Systems sind und Agenten sowie Agentensysteme in einer Doppelrolle als Subjekt und als Objekt im System auftreten. Endsysteme und Endsystemressourcen sind Objekte.

Um nicht für jeden möglichen Angriff eines Subjektes auf ein bestimmtes Objekt eine spezielle Abwehrstrategie entwickeln und warten zu müssen, werden grundlegende Sicherheitseigenschaften und –dienste definiert, die möglichst generisch sind und auf ganze Problemklassen angewendet werden können. Folgende Sicherheitsdienste sollen zukünftig in MASA unterstützt bzw. realisiert werden:

- **Authentisierung:** Mittels Authentisierung kann die Identität eines Subjektes zweifelsfrei belegt werden. Authentisierung

- **Zugriffskontrolle und Delegation von Rechten:** Der Vorgang mit dessen Hilfe entschieden wird, ob eine Operation eines Subjektes auf einem Objekt zugelassen und ausgeführt werden darf, wird als Zugriffskontrolle bezeichnet. Da MAs zur Erledigung ihrer Aufgaben sowohl migrieren als auch andere Agenten beauftragen können, muß die Möglichkeit bestehen, Rechte zwischen Subjekten weitergeben zu können. Zugriffskontrolle, Rechtedelegation

- **Integrität:** Die Integrität von Nachrichten oder Objekten ist gewährleistet, wenn sie beim Empfänger unverändert und ohne Verzögerung ankommen, oder der Empfänger oder Benutzer jede Veränderung oder Wiedereinspielung erkennen kann. Integrität

- **Verläßlichkeit, Level–of–Trust:** Ein MA der auf ein AS migriert, muß sich darauf verlassen können, daß ihm vom AS eine geeignete Ausführungsumgebung zur Verfügung gestellt wird. Auf der anderen Seite muß sichergestellt werden können, daß der MA einem bestimmten „Programmiermodell“ entspricht. Auch der Delegator eines MA muß sicherstellen können, daß er den Daten, die sein MA liefert, vertrauen kann. Der Aufbau entsprechender Vertrauensbeziehungen wird unter dem Begriff Level–of–Trust zusammengefaßt. Verläßlichkeit, Level–of–Trust

- **Vertraulichkeit:** Vertraulichkeit ist gewährleistet, wenn ein unberechtigter Dritter Daten weder ausspähen, abhören noch auf eine andere Art verwenden kann. Vertraulichkeit

- **Verbindlichkeit:** Wenn ein Subjekt den Zugriff auf ein Objekt nicht leugnen und auch ein unbeteiligter Dritter diesen Zugriff „beweisen“ kann, spricht man von Verbindlichkeit. Verbindlichkeit

- **Auditing und Logging:** Um Daten über Dienstnutzung, Ressourcenverbrauch, sicherheitsrelevante Aktionen u.ä. zu erhalten, wird Auditing und Logging verwendet. Auditing, Logging

Obwohl für einige dieser Sicherheitsdienste auch Dienste auf Betriebssystemebene existieren, können diese in MASA nur bedingt verwendet werden. Aus Sicht des Betriebssystems ist das AS von MASA ein „normaler“ Benutzerprozeß und es ist nicht erkennbar, daß dieser Prozeß eine Ablaufumgebung für Agenten der unterschiedlichsten Benutzer (mit unterschiedlichsten Rechten) darstellt.

6.2 Weitere Corba Dienste

Notification Service: Filter, QoS, publish/subscribe

Eine Erweiterung des Event Management Service ist der *Notification Service* [OMG 98-11-01], der von der *Telecom DTF*, der für die Telekommunikationsindustrie zuständigen Arbeitsgruppe innerhalb der OMG, spezifiziert wurde. Der Notification Service führt strukturierte Events, die eine Liste von Properties, also (key,value)–Paaren enthalten, Filter und QoS–Parameter für die Auslieferung von Ereignissen ein. *Supplier* können erklären, welche Typen von Ereignissen sie liefern werden (publish), *Consumer*, an welchen Typen von Ereignissen sie interessiert sind (subscribe).

Event Type Repository

Optional kann ein *Event Type Repository* implementiert werden, in dem die Meta–Information zu den Ereignissen abgelegt wird, d.h. welche Typen von Ereignissen es gibt und wie deren Struktur aussieht. (Die vorhandenen Corba–Repositories Interface Repository und Implementation Repository können diese Information nicht enthalten, da das Format eines Ereignisses nicht Teil einer IDL–Objektschnittstelle ist.) Ein Produkt, das diesen Dienst implementiert, ist z.B. IONA/OrbixNotification [OrbixNotif].

Der Corba *Event Service* weist einige Unzulänglichkeiten auf, die ihn für den Einsatz in großen Anwendungen (wie z.B. Managementsystemen) nicht besonders geeignet erscheinen lassen:

- Der Empfänger erhält alle Ereignisse eines Channels. Eine feingranulare Unterscheidung bzgl. einzelner *Attribute* eines Ereignisse bei der Auslieferung ist nicht möglich.
- Der Event Service hat ein asynchrones Modell für die Auslieferung des Ereignisses. Eine verzögerte Auslieferung kann nicht spezifiziert werden, sondern ist implementierungsspezifisch.
- Persistenz von Ereignissen kann nicht explizit gewünscht werden, obwohl Implementierungen hier Erweiterungen machen könnten.
- Bei einer großen Anzahl Ereignisse sollten QoS–Parameter für die Auslieferung wichtiger Ereignisse spezifizierbar sein.

Diese Anforderungen erfüllt der Corba Notification Service.

6.3 Agenten als Beans

Einige Agentenplattformen stützen sich auf das Konzept der Beans[1] ab, beispielsweise das Java Dynamic Management Kit von Sun [JDMK 98] oder

[1] Wir verwenden den Begriff „Beans“ für Komponenten im Sinne Komponenten–orientierter Softwareentwicklung, da der Begriff „Komponente“ im IT–Management anderweitig belegt ist.

Java Mangement Extensions [Sun 99]. Das Konzept der Beans erweitert Java Schnittstellen um zwei Konzepte, Properties und Ereignisse, die durch Namenskonventionen für Java–Schnittstellen, sogenannten *Design Patterns*, realisiert sind. Für eine weitergehende Einführung in Java/Corba/Enterprise Beans siehe [OrHa 98].

Properties für die Konfiguration von Agenten zur Laufzeit

Java Bean Properties sind Einstellungen, die zur Laufzeit vorgenommen werden können. Ein Agent stellt dazu Methoden bereit, die der Namenskonventionen für Java Bean Properties gehorchen, nämlich eine Methode zum Lesen (get*PropertyName* bzw. is*PropertyName* bei Booleschen Werten) und zum Schreiben (set*PropertyName*). Durch Aufruf dieser Methoden kann der Agent zur Laufzeit konfiguriert werden bzw. seine Konfiguration kann ausgelesen werden.

Verbesserung der Ereigniskommunikation

Im Java Beans Konzept [JavaBeans 1.01] gibt es Ereignisse. Ereignisse sind Java Objekte, die von einer *Event Source* an einen oder mehrere *Event Listener* verschickt werden. Sowohl die Event Source als auch der Listener stellen Schnittstellen zur Verfügung, die bestimmten Konventionen gehorchen. Sie können z.B. von einem Beans–Entwicklungswerkzeug entdeckt und mittels Introspektion ausgelesen werden. Die Event Source bietet Methoden zum Registrieren und Deregistrieren von Event Listenern an (add*EventListenerType*, remove*EventListenerType*); der Event Listener implementiert eine Callback–Methode, den Listener (interface *EventListenerType*).

Die Signaturen dieser Methoden würden in einer Beans–basierten Managementarchitektur die Eventdeklarationen klassischer MIBs ersetzen, die auf Corba nicht ohne weiteres abbildbar sind. Ein Beans Ereignis würde zu einem Managementereignis. Für das Management ist die Kopplung von Event Source und Event Listenern zu eng: synchrone, lokale Aufrufe. Im Beans–Konzept ist dazu vorgesehen, daß zwischen Source und Listener ein *Event Adapter* geschaltet wird, der das Ereignis transformiert und die Ereigniskommunikation entkoppelt.

Corba Beans, EJB

Man gelangt zu Beans für verteilte Umgebungen, wenn man die Java Beans mit Corba–Schnittstellen versieht. Man erhält Corba Beans [OrHa 98], wie sie derzeit von der OMG standardisiert werden (RFP: [OMG 97-06-12], Final Submission: [OMG 99-02-05]). Ein weiteres Komponentenmodell sind Enterprise JavaBeans [EJB 1.0, EJB-CORBA 1.0], die Eigenschaften wie Transaktionsfähigkeit integrieren, welche auch für das Management interessant sind.

6.4 Domänen

Kopplung verschiedener Agentensysteme

Die Arbeiten zur Koppelung von MASA mit dem Voyager System (s. Kap. 4.5.3) haben gezeigt, daß der Namens– bzw. Objekthierarchie innerhalb des

Agentensystems große Beachtung geschenkt werden muß. Fragen, die dabei zu beantworten sind:

- Gibt es eine globale Donäne, in der alle Agentensysteme miteinander kommunizieren können?
- Sichtbarkeit: Welche Agenten auf einem oder anderen Agentensysteme sind sichtbar (lassen sich per Kommunikation über die MASA–Dienste erreichen)?
- Dürfen mehrere Agentensysteme in einer Managementumgebung zugleich existieren?
- Wie können Gateways zwischen verschieden Agentensystem–Domänen etabliert werden?

Management über organisatorische Grenzen hinweg

Neben diesen Fragen, die sich aus der Koppelung verschiedener Typen von Agentensystemen ergeben, müssen aber auch organisatorische– und Sicherheitsaspekte betrachtet werden. Eine Fragestellung die in letzter Zeit immer häufiger auftritt, ist das Management über Betreiber– und Organisationsgrenzen hinweg.

Ein Provider stellt einem Kunden Netzinfrastrukturen und gewisse (Basis–) Dienste sowie das Management dafür zur Verfügung. Der Kunde etabliert mittels dieser Infrastruktur Mehrwertdienste, die er entweder selber nutzt oder weiter verkauft. Das Management dieser Mehrwertdienste wird vom Kunden selbst durchgeführt. In diesen Szenarios zeigt sich sehr schnell, daß Managementaufgaben und –funktionen nicht mehr streng an Organisationsgrenzen gebunden werden können. Der Provider muß eine bestimmte Managementaufgabe erfüllen, kann dies aber nur, wenn er in der Lage ist, einen Managementagenten „innerhalb“ des Kundennetzes zu betreiben, d.h. auf einem System das nicht ihm sondern dem Kunden gehört [HaRe 99].

Ähnliche Problemstellungen können auch innerhalb eines Unternehmens auftreten, wenn bspw. verschiedene Abteilungen eigenverantwortlich ihre Infrastruktur verwalten, Teile davon jedoch gemeinsam nutzen oder anderen Abteilungen zur Verfügung stellen.

Einige Fragen, die dabei zu lösen sind:

- Wie können unterschiedliche Sicherheitsdomänen realisiert werden?
- Wie können organisatorische oder funktionale Gliederungen in einem Unternehmen mit Hilfe von MASA nachgebildet werden?
- Wie können MA über organisatorische Grenzen hinweg Managementaufgaben erledigen?

Als determinierende Parameter für die Entscheidung, wie die Domänenbildung in einer konkreten Implementierung eines MASA–basierten Agentensystems auszuführen ist, wurden bisher erkannt:

- Administrative Gegebenheiten (Organisationsstrukturen), sowie
- technische Rahmenbedingungen

Weitere Arbeiten auf diesem Gebiet sind jedoch noch erforderlich.

Anhang A

Empfohlene Literatur

A.1 Einführende Bücher

Programmieren in Java

- Arnold/Gosling *The Java Programming Language*, [ArGo 98]

oder

- Flanagan *Java in a Nutshell*, [Flan 97]

Verteilte Programmierung mit Java und Corba

- Orfali/Harkey *Client/Server Programming with Java and Corba*, [OrHa 98]

IT–Management

- Hegering/Abeck/Neumair *Integriertes Management vernetzter Systeme*, [HAN 99]

speziell SNMP

- Rose *The Simple Book*, [Rose 96]

Rechnernetze und Kommunikationsprotokolle, TCP/IP

- Comer „Computer Networks and Internets“, [Come 97]
- Feit „TCP/IP“ [Feit 97]
- Stallings/van Slyke „Business Data Communications“, [StSl 98]

A.2 Standards und Spezifikationen

Java

- Gosling/Joy/Steele *The Java Language Specification*, [JLS]

Corba

ORB, IIOP, Java Mapping

- *CORBA/IIOP 2.2* [OMG 98-02-01]

Naming Service, Event Service, Property Service, Security Service

- *Complete CORBAServices Book* [OMG 98-07-05]

Corba Beans

- *Corba Component Model* (aka „Corba Beans“) RFP [OMG 97-06-12]
- *Corba Component Model* Final Submission: [OMG 99-02-05]

Mobile Corba Agenten

- *Mobile Agent System Interoperability Facility (MASIF)* [OMG 98-03-09]

Ereigniskommunikation

- *Event Management Service* [OMG 97-02-09]
- *Notification Service* [OMG 98-11-01]

CVS

- CVS — Concurrent Versions System `http://www.delorie.com/gnu/docs/cvs/cvs_toc.html` oder `http://www.loria.fr/~molli/cvs/doc/cvs_toc.html`

A.3 WWW

Tutorials

Programmieren in Java, Java IDL, JavaBeans

- Java Tutorial `http://www.javasoft.com/docs/books/tutorial/index.html`

Corba

- Corba for Beginners `http://www.omg.org/corba/beginners.html`

CVS

- CVS Kurztutorial `http://www.ius.cs.cmu.edu/help/Archiving/cvs_tutorial.texinfo.html`

Intern

- CVS — Konventionen für MASA `http://wwwmnmteam.informatik.uni-muenchen.de/proj/software/htdocs/`

Gesammelte Dokumente zu vielen Themen

- Literatur des MNM–Teams `http://wwwmnmteam.informatik.uni-muenchen.de/proj/Literatur`

Installierte Software / lokale Anpassungen

- Übersicht `http://wwwmnmteam.informatik.uni-muenchen.de/internal/`
- Java `http://www.nm.informatik.uni-muenchen.de/proj/java/htdocs/`

MASA

- Referenzinstallation `http://wwwmnmteam.informatik.uni-muenchen.de/proj/fagent/masa-referenz`
- im CVS `http://wwwmnmteam.informatik.uni-muenchen.de/proj/software/htdocs/`
- Projektverzeichnis von MASA (aus historischen Gründen unter fagent) `http://wwwmnmteam.informatik.uni-muenchen.de/proj/fagent`
- Projektverzeichnis MobileAgents (enthält Doku und Installationen anderer MA–Plattformen) `http://wwwmnmteam.informatik.uni-muenchen.de/proj/MobileAgents`

Abbildungsverzeichnis

1.1 Aufbau eines Managementsystems (nach [HAN 99]) 1

2.1 Policy Hierarchie 8

2.2 Abhängigkeitsgraph für ein IP–Netz 10

3.1 Übergang von zentralisiertem zum MbD Paradigma (nach [Moun 97]) 18

3.2 MASIF–Basismodell 24

4.1 Die MASA–Architektur 28

4.2 Schichtenarchitektur von MASA 29

4.3 Lebenszyklus eines Agenten 33

4.4 MASA Naming Service (Bsp. nach [Kemp 98]) 35

4.5 AgentSystem Applet [Gerb 99] 38

4.6 VoyagerGateway Applet [Bran 99] 39

4.7 Monitoring von Voyager Agenten [Bran 99] 40

5.1 Verzeichnisstruktur des Agentensystems 43

5.2 Softwarearchitektur des Prototypen 46

5.3 Abhängigkeitsgraphen API in Corba–IDL 47

5.4 API des IPRouting–Agenten in Corba–IDL 47

5.5 Policy Management Architektur 48

5.6 Schnittstellen des *Corba Topology Service* [OMG 97-10-02] . . 49

5.7 Implementierung *Corba Topology Service* als MASA–Agenten [Roel 99b] 50

Literaturverzeichnis

[Aglets] *Aglets*, http://www.trl.ibm.co.jp/aglets.

[Allg 98] Allgeyer, P.: *Entwicklung und Implementierung einer Managementschnittstelle für einen PC-basierten Switch/Router.* Diploma Thesis, Technische Universität München, Betreuer: Heilbronner, Wienold, August 1998.

[ArGo 98] Arnold, K. und J. Gosling: *The Java Programming Language.* Addison Wesley, Zweite Auflage, 1998, http://cseng.aw.com/bookdetail.qry?ISBN=0-201-31006-6&ptype=0.

[BBCM 98] Breugst, M., I. Busse, S. Covaci und T. Magedanz: *Grasshopper - A Mobile Agent Platform for IN Based Service Environments.* In: *Proceedings of IEEE IN Workshop*, Seiten 279–290, Bordeaux, France, Mai 1998. , http://www.ikv.de/download/grasshopper.html.

[BHR 97] Baumann, Joachim, Fritz Hohl und Kurt Rothermel: *Mole - Concepts of a Mobile Agent System.* Fakultätsbericht 1997/15, Universität Stuttgart, 1997, http://www.informatik.uni-stuttgart.de/cgi-bin/ncstrl_rep_view.pl?/inf/ftp/pub/library/ncstrl.ustuttgart_fi/TR-1997-15/TR-1997-15.bib.

[BPW 98] Bieszczad, A., B. Pagurek und T. White: *Mobile Agents for Network Management.* IEEE Communication Surveys, 1(1), 1998, http://www.comsoc.org/pubs/surveys/4q98issue/bies.html.

[Bran 99] Brandt, R.: *Interoperabilität von CORBA ORBs in Mobilen Agentensystemen.* Systementwicklungsprojekt, Technische Universität München, Betreuer: Gruschke, Reiser, Mai 1999, http://wwwmnmteam.

`informatik.uni-muenchen.de/common/Literatur/MNMPub/Fopras/bran99/bran99.shtml`.

[Coeh 98] COEHN, C.: *Integriertes Management verteilter Email-Systeme auf der Basis flexibler Managementagenten.* Diploma Thesis, Ludwig-Maximilians-Universität München, Betreuer: HEILBRONNER, GRUSCHKE, August 1998.

[Come 97] COMER, D.: *Computer Networks and Internets.* Prentice Hall, 1997.

[Concordia] *Concordia*, `http://www.meitca.com/HSL/Projects/Concordia/`.

[CORBA 2.2] *The Common Object Request Broker: Architecture and Specification.* OMG Specification Revision 2.2, Object Management Group, Februar 1998.

[CVS] *CVS — Concurrent Versions System*, `http://www.delorie.com/gnu/docs/cvs/cvs_toc.html`.

[DAgents] *D'Agents*, `http://agent.cs.dartmouth.edu/~agent/`.

[Demm 98] DEMMEL, S.: *Implementierung eines portierbaren Java-DHCP-Servers mit einer CORBA-Managementschnittstelle.* Fortgeschrittenenpraktikum, Ludwig-Maximilians-Universität München, Betreuer: HEILBRONNER, September 1998.

[Demm 99] DEMMEL, S.: *Entwurf und Integration einer Anwendung für das Management nomadischer Systeme.* Diploma Thesis, Ludwig-Maximilians-Universität München, Betreuer: HEILBRONNER, Juli 1999.

[EJB 1.0] *Enterprise JavaBeans Specification Version 1.0.* Specification, Sun Microsystems, März 1998, `ftp://ftp.javasoft.com/docs/ejb/ejb.10.pdf`.

[EJB-CORBA 1.0] *Enterprise JavaBeans 1.0 CORBA Mapping.* Specification, Sun Microsystems, März 1998, `ftp://ftp.javasoft.com/docs/ejb/ejb-corba.10.pdf`.

[Feit 97] FEIT, S.: *TCP/IP.* McGraw Hill, 1997.

[FFMM 93] FININ, T., R. FRITZSON, D. MCKAY und R. MCENTIRE: *KQML: an Information and Knowledge Exchange Protocol.* Proceedings of International Conference on Building and Sharing of Very Large Scale Knowledge Bases, Dezember 1993.

[FKK 99] FERIDUN, M., W. KASTELEIJN und J. KRAUSE: *Distributed Management with Mobile Components.* In: SLOMAN, M., S. MAZUMDAR und E. LUPO (Herausgeber): *Integrated Network Management VI (IM'99)*, Seiten 856–870, Boston, MA, Mai 1999. IEEE Publishing.

[Flan 97] FLANAGAN, D.: *Java in a Nutshell.* O'Reilly, Zweite Auflage, Mai 1997, `http://www.oreilly.com/catalog/javanut2/`.

[Gerb 99] GERBER, S.: *Entwicklung einer Benutzerschnittstelle mit Java/CORBA für die Mobile Agent System Architecture (MASA).* Systementwicklungsprojekt, Technische Universität München, Betreuer: GRUSCHKE, REISER, April 1999, `http://wwwmnmteam.informatik.uni-muenchen.de/common/Literatur/MNMPub/Fopras/gerb99/gerb99.shtml`.

[Goli 99] GOLIAS, D.: *Entwicklung und Implementierung einer Benutzerschnittstelle für Policy-Managementagenten.* Fortgeschrittenenpraktikum, Ludwig-Maximilians-Universität München, Betreuer: HEILBRONNER, GRUSCHKE, 1999.

[GoYe 95] GOLDSZMIT, G. und Y. YEMINI: *Distributed Managment by Delegation.* In: *Proceedings of the 15th International Conference on Distributed Computing Systems*, Juni 1995.

[Grasshopper] *Grasshopper*, `http://www.ikv.de/products/grasshopper`.

[Gray 95] GRAY, R. S.: *Agent Tcl: Alpha Release 1.1*, 1995, `ftp://ftp.cs.dartmouth.edu/pub/agents/doc.1.1.ps.gz`.

[Grus 99] GRUSCHKE, B.: *Entwurf eines Eventkorrelators mit Abhängigkeitsgraphen (in Vorbereitung).* Dissertation, Ludwig-Maximilians-Universität München, Juli 1999.

[Gypsy] *Gypsy*, `http://www.infosys.tuwien.ac.at/Gypsy`.

[HAN 99] HEGERING, H.-G., S. ABECK und B. NEUMAIR: *Integrated Management of Networked Systems – Concepts, Architectures and their Operational Application.* Morgan Kaufmann Publishers, ISBN 1-55860-571-1, 1999.

[HAN 99a] HEGERING, H.-G., S. ABECK und B. NEUMAIR: *Integriertes Management vernetzter Systeme — Konzepte, Architekturen und deren betrieblicher Einsatz.* dpunkt-

Verlag, ISBN 3-932588-16-9, 1999, http://www.dpunkt.de/produkte/management.html.

[HaRe 99] HAUCK, R. und H. REISER: *Monitoring of Service Level Agreements with flexible and extensible Agents.* In: *Workshop of the OpenView University Association (OVUA'99)*, Bologna, Italy, Juni 1999. , http://www.hpovua.org/PUBLICATIONS/PROCEEDINGS/6_HPOVUAWS/Papers/hauck_reiser.pdf.

[Heil 98a] HEILBRONNER, S.: *Requirements for Policy-based Management of Nomadic Computing Systems.* In: SETHI, A. S. (Herausgeber): *Proceedings of the 9th IFIP/IEEE International Workshop on Distributed Systems: Operations & Management (DSOM 98)*, Newark, DE, USA, Oktober 1998. .

[Holl 97] HOLLERITH, A.: *Entwurf einer Architektur für flexible Agenten auf der Basis des Konzepts von Management by Delegation.* Diploma Thesis, Technische Universität München, Betreuer: MOUNTZIA, GRUSCHKE, Februar 1997.

[ISO 10164-21] *Information Technology – Open Systems Interconnection – Systems Management – Part 21: Command Sequencer.* DIS 10164-21, International Organization for Standardization and International Electrotechnical Committee, August 1996.

[ISO 10165-4] *Information Technology – Open Systems Interconnection – Structure of Management Information – Part 4: Guidelines for the Definition of Managed Objects.* IS 10165-4, International Organization for Standardization and International Electrotechnical Committee, 1992.

[JavaBeans 1.01] *JavaBeans.* Specification, Sun Microsystems, Juli 1997, http://java.sun.com/beans/docs/beans.101.pdf.

[JDMK 98] *Java Dynamic Management Kit: Technical Information.* White Paper, Sun Microsystems, 1998, http://www.sun.com/software/java-dynamic/tech-overview.html.

[JINI] *Jini connection technology.* WWW, http://www.sun.com/jini/. http://www.sun.com/jini/.

[JLS] GOSLING, J., B. JOY und G. STEELE: *The Java Language Specification.* Addison–Wesley, 1996, http://www.javasoft.com/docs/books/jls/.

[Joha 98] JOHANSEN, D.: *Mobile Agent Applicability.* In: ROTHERMEL, K. und F. HOHL [RoHo 98].

[JRS 95] JOHANSEN, D., R VAN RENESSE und F. B. SCHNEIDER: *Operating System Support for Mobile Agents.* In: *Proceedings of the 5th IEEE Workshop on Hot Topics in Operating Systems*, Orcas Island, Wa, USA, Mai 1995. , `http://cs-tr.cs.cornell.edu/TR/CORNELLCS:TR94-1468`.

[JumpingBeans] *JumpingBeans*, `http://www.JumpingBeans.com/`.

[Kell 98] KELLER, A.: *CORBA-basiertes Enterprise Management: Interoperabilität und Managementinstrumentierung verteilter kooperativer Managementsysteme in heterogener Umgebung.* Dissertation, Technische Universität München, Dezember 1998.

[Kemp 97] KEMPTER, B.: *Realisierung eines Managementwerkzeugs in Java für das Management von IP-Netzen.* Fortgeschrittenenpraktikum, Technische Universität München, Betreuer: GRUSCHKE, November 1997.

[Kemp 98] KEMPTER, B.: *Entwurf eines Java/CORBA-basierten Mobilen Agenten.* Diploma Thesis, Technische Universität München, Betreuer: GRUSCHKE, KELLER, August 1998.

[KKK 96] KOCH, T., C. KRELL und B KRÄMER: *Policy Definition Language for Automated Management of Distributed Systems.* In: *IEEE Workshop on Systems Management.* IEEE, 1996.

[Kots 97] KOTSELIDIS, T.: *Erweiterung eines Java–Agenten um Managementfunktionen.* Systementwicklungsprojekt, Technische Universität München, Betreuer: GRUSCHKE, MOUNTZIA, Dezember 1997.

[LaOs 98] LANGE, D. und M. OSHIMA: *Programming and Deploying Java[tm] Mobile Agents with Aglets[tm].* Addison Wesley Longman, 1998.

[LeSc 99] LEVI, D. und J. SCHOENWAELDER: *RFC 2592: Definitions of Managed Objects for the Delegation of Management Scripts.* RFC, IETF, Mai 1999, `http://sunsite.auc.dk/RFC/rfc/rfc2592.html`.

[LLN 99] LANGER, M., S. LOIDL und M. NERB: *Customer Service Management: Towards a Management Information*

Base for an IP Connectivity Service. In: *Proceedings of the 4th IEEE Symposium on Computers and Communications (ISCC'99)*, Sharm El Sheikh, Egypt, Juli 1999.
.

[MASIF] *Mobile Agent System Interoperability Facilities Specification.* OMG TC Document orbos/98-03-09, Object Management Group, März 1998.

[May 98] MAY, H.: *Erstellung eines Testprogramms für DHCP-Server.* Fortgeschrittenenpraktikum, Ludwig-Maximilians-Universität München, Betreuer: HEILBRONNER, Dezember 1998, `http://www.nm.informatik.uni-muenchen.de/common/Literatur/MNMPub/Fopras/may98/may98.shtml`.

[Mits 98] MITSUBISHI ELECTRIC ITA: *Mobile Agent Computing – A White Paper.* Technischer Bericht Mitsubishi Electric ITA, Januar 1998, `http://www.meitca.com/HSL/Projects/Concordia/documents.htm`.

[Mole] *Mole*, `http://www.informatik.uni-stuttgart.de/ipvr/vs/projekte/mole.html`.

[Moun 97] MOUNTZIA, M.-A.: *Flexible Agents in Integrated Network and Systems Management.* Dissertation, Technische Universität München, Dezember 1997.

[MPS 99] MCCLOGHRIE, K., D. PERKINS und J. SCHOENWAELDER: *RFC 2578: Structure of Management Information Version 2 (SMIv2).* RFC, IETF, April 1999, `http://sunsite.auc.dk/RFC/rfc/rfc2578.html`.

[Odyssey] *Odyssey*, `http://www.genmagic.com/technology/odyssey.html`.

[OKO 98] OSHIMA, M., G. KARJOTH und K. ONO: *Aglets Specification 1.1 Draft.* Technischer BerichtDraft 0.65, IBM Research Labs Tokio, September 1998, `http://www.trl.ibm.co.jp/aglets/spec11.html`.

[OMG 97-02-09] *CORBAservices - Event Management Service.* OMG Specification formal/97-02-09, Object Management Group, Februar 1997, `ftp://ftp.omg.org/pub/docs/formal/97-02-09.pdf`.

[OMG 97-06-12] *CORBA Component Model RFP, Final Version.* TC Document orbos/97-06-12, Object Management Group, Juni 1997.

[OMG 97-06-14] *Minimum CORBA RFP, Final Draft.* TC Document orbos/97-06-14, Object Management Group, Juni 1997.

[OMG 97-10-02] *Hewlett-Packard REvised Topology Service Submission.* TC Document telecom/97-10-02, Object Management Group, Oktober 1997, `ftp://ftp.omg.org/pub/docs/telecom/97-10-02.pdf`.

[OMG 98-02-01] *CORBA/IIOP 2.2 Specification - Full Version.* OMG Specification formal/98-02-01, Object Management Group, Februar 1998, `ftp://ftp.omg.org/pub/docs/formal/98-02-01.pdf`.

[OMG 98-03-09] *MASIF Revision.* TC Document orbos/98-03-09, Object Management Group, März 1998, `ftp://ftp.omg.org/pub/docs/orbos/98-03-09.pdf`.

[OMG 98-07-05] *Complete CORBAServices book.* OMG Specification formal/98-07-05, Object Management Group, Juli 1998, `ftp://ftp.omg.org/pub/docs/formal/98-07-05.pdf`.

[OMG 98-08-04] *Minimum CORBA (new) Revised Submission.* TC Document orbos/98-08-04, Object Management Group, August 1998, `ftp://ftp.omg.org/pub/docs/orbos/98-08-04.pdf`.

[OMG 98-11-01] *A revised version of the Notification Service Specification.* TC Document telecom/98-11-01, Object Management Group, November 1998, `ftp://ftp.omg.org/pub/docs/telecom/98-11-01.pdf`.

[OMG 99-02-05] *CORBA Components final submission.* TC Document orbos/99-02-05, Object Management Group, Februar 1999, `ftp://ftp.omg.org/pub/docs/orbos/99-02-05.pdf`.

[OMG 99-03-05] *GIOP over ATM draft RFP.* TC Document telecom/99-03-05, Object Management Group, März 1999, `ftp://ftp.omg.org/pub/docs/telecom/99-03-05.pdf`.

[OMG 99-07-53] *IDL_Java Language Mapping Full Specification.* OMG Specification formal/99-07-53, Object Management Group, Juli 1999, `ftp://ftp.omg.org/pub/docs/formal/99-07-53.pdf`.

[OrbixNotif] IONA Technologies, www.iona.com: *OrbixNotification Programmer's Guide and Reference*, Dezember 1998, `http://www.iona.com/info/products/docs/onotification-progguide.pdf`.

[OrHa 98] ORFALI, R. und D. HARKEY: *Client/Server Programming with Java and Corba.* John Wiley and Sons, 2nd Auflage, 1998.

[PhKa98] PHAM, A. und A. KARMOUCH: *Mobile Software Agents: An Overview.* IEEE Communications Magazine, 36(7):26–37, Juli 1998.

[Python] *Python - A interpreted, interactive, object-oriented programming language.* WWW, http://www.python.org/doc.

[Radi 98] RADISIC, I.: *Konzeption eines policy-basierten Konfigurationsmanagements für nomadische Systeme in Intranets.* Diploma Thesis, Ludwig-Maximilians-Universität München, Betreuer: HEILBRONNER, August 1998.

[Roel 99a] RÖLLE, H.: *Authentisierung und Autorisierung für das Java/CORBA–Agentensystem MASA.* Diploma Thesis, Technische Universität München, Betreuer: GRUSCHKE, REISER, August 1999.

[Roel 99b] RÖLLE, H.: *Prototypische Implementierung eines CORBA Topology Service.* Fortgeschrittenenpraktikum, Technische Universität München, Betreuer: HEILBRONNER, GRUSCHKE, Februar 1999.

[RoHo 98] ROTHERMEL, K. und F. HOHL (Herausgeber): *Mobile Agents (MA '98)*, Band 1477 der Reihe *LNCS*, Berlin; Heidelberg, 1998. Springer.

[RoPo 97] ROTHERMEL, K. und R. POPESCU-ZELETIN (Herausgeber): *Mobile Agents (MA '97)*, Band 1219 der Reihe *LNCS*, Berlin, April 1997. Springer.

[Rose 96] ROSE, MARSHALL T.: *The Simple Book — An Introduction to Internet Management.* Prentice-Hall, revised 2nd Auflage, 1996, http://www.prenhall.com/books/ptr_0134516591.html.

[Scheme] STEELE, G.-L. und G.-J. SUSSMAN: *Scheme - A LISP dialect*, http://www-swiss.ai.mit.edu/scheme-home.html.

[Schi 99] SCHILDBACH, A.: *Entwicklung einer Certification Authority (CA), mit Java/Corba, für die Mobile Agent Systems Architecture (MASA) (Arbeitstitel).* Systementwicklungsprojekt, Technische Universität München, Betreuer: REISER, Dezember 1999.

[SCM+ 96] SNMPv2 Working Group, J. Case, K. McCloghrie, M. Rose und S. Waldbusser: *RFC 1902: Structure of Management Information for Version 2 of the Simple Network Management Protocol (SNMPv2).* RFC, IETF, Januar 1996, `http://sunsite.auc.dk/RFC/rfc/rfc1902.html`.

[SlTw 94] Sloman, Morris S. und Kevin Twidle: *Domains: A Framework for Structuring Management Policy*, Kapitel 16. Addison Wesley, 1994.

[Stal 96] Stallings, William: *SNMP, SNMPv2, and RMON: Practical Network Management.* Corporate and Professional Publishing Group, Zweite Auflage, 1996.

[StSl 98] Stallings, W. und Richard van Slyke: *Business Data Communications.* Prentice Hall, 1998.

[Sun 99] Sun Microsystems, Inc.: *Java Mangement Extensions (JMX).* Preliminary Specification Draft 1.9, Sun Microsystems, Inc., Palo Alto, CA, Juni 1999.

[Tacoma] *Tacoma*, `http://www.tacoma.cs.uit.no`.

[TMF 909] *Technology Integration Map (GB 909).* Technischer Bericht TeleManagement Forum, 1999, `ftp://ftp.tmforum.org/specifications/techmap/TechMap.doc`.

[VGPK 97] Veizades, J., E. Guttman, C. Perkins und S. Kaplan: *RFC 2165: Service Location Protocol.* RFC, IETF, Juni 1997, `http://sunsite.auc.dk/RFC/rfc/rfc2165.html`.

[Voyager] *Voyager*, `http://www.objectspace.com/voyager/`.

[Voyager 2.0] Objectspace: *Voyager Core Technology 2.0 — Users Guide.* Technischer Bericht Objectspace, 1998.

[Wald 95] Waldbusser, S.: *RFC 1757: Remote Network Monitoring Management Information Base.* RFC, IETF, Februar 1995, `http://sunsite.auc.dk/RFC/rfc/rfc1757.html`.

[Wald 97] Waldbusser, S.: *RFC 2021: Remote Network Monitoring Management Information Base Version 2 using SMIv2.* RFC, IETF, Januar 1997, `http://sunsite.auc.dk/RFC/rfc/rfc2021.html`.

[Wenn 97] Wennrich, M.: *Erstellung eines Kommunikationsmodells für kooperierende Agenten für das Management von* *verteilten Systemen.* Diploma Thesis, Technische Universität München, Betreuer: MOUNTZIA, GRUSCHKE, Februar 1997.

[Wies 95] WIES, R.: *Policies in Integrated Network and Systems Management: Methodologies for the Definition, Transformation, and Application of Management Policies.* Dissertation, Ludwig-Maximilians-Universität München, Juni 1995.

[YGY 91] YEMINI, Y., G. GOLDSZMIDT und S. YEMINI: *Network Management by Delegation.* In: KRISHNAN, I. und W. ZIMMER (Herausgeber): *Proceedings of the 2nd International Symposium on Integrated Network Management, Washington.* IFIP, North-Holland, April 1991.

[Zeil 99] ZEILHOFER, R.: *Sicheres Laden und Ausführen Mobiler Agenten auf Basis der Java 2 Plattform (Arbeitstitel).* Fortgeschrittenenpraktikum, Ludwig-Maximilians-Universität München, Betreuer: GRUSCHKE, REISER, 1999.