Diplomarbeit

Anwendungs- und Infrastrukturmanagement im Agentensystem *AMETAS*

vorgelegt von:

Nadali Walizadeh

Geschrieben bei: Prof. Dr. Kurt Geihs

Betreuer: Michael Zapf

Johann Wolfgang Goethe-Universität

Frankfurt am Main

Eingereicht am:

12. Dezember 2001

Erklärung

Hiermit versichere ich, Nadali Walizadeh, dass ich die vorliegende Diplomarbeit

"Anwendungs- und Infrastrukturmanagement im Agentensystem *AMETAS"*

selbstständig und mit Hilfe der angeführten Literatur und der erwähnten Hilfsmittel angefertigt habe.

Die angegeben WWW-Adressen sind zuletzt am 11.12.2001 auf ihre Existenz hin überprüft worden.

Frankfurt am Main, den 12. Dezember 2001———————————————————

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

List of Algorithms

Abkürzungsliste

AMETAS	Asynchronous Message Transfer Agent System
AMAI	AMETAS Attachable Interface
AMSI	AMETAS Simple Interface
AMAWI	AMETAS Attachable WWW Interface
AWT	Abstract Window Tookit
CA	Certificate Authority
CI	Component Interface
CIM	Common Information Model
CIMOM	CIM Object Manager
CIOP	Common Inter-ORB Protocol
CMIP	Common Management Information Protocol
CMIS	Common Management Information Services
CMISE	Common Management Information Service Entity
CORBA	Common Object Request Broker Architecture
DCE	Distributed Computing Environment
DMI	Desktop Management Interface
DMTF	Distributed Management Task Force
DNS	Domain Name Service
ESIOP	Environment-Specific Inter-ORB Protocol
FCAPS	Fault Management, Configuration Management, Accounting Management, Performance Management, Security Management
FIPA	Foundation for Intelligent Physical Agents
GIOP	General Inter-ORB Protocol
HMMP	Hypermedia Management Protocol
HMMS	Hypermedia Management Scheme
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol

IAB	Internet Activities Board
IBM	International Business Machines Corporation
IDL	Interface Definition Language
IIOP	Internet Inter-ORB Protocol
IM	Informationsmodell
IP	Internet Protocol
ISO	International Standardization Organisation
IT	Information Technology
JDMK	Java Dynamic Management Kit
JMAPI	Java Management Application Interface
JMX	Java Management Extension
JSSE	Java Secure Socket Extension
JVM	Java Virtual Machine
KQML	Knowledge Query and Manipulation Language
KM	Kommunikationsmodell
LME	Layer Management Entity
MA	Managementarchitektur
MASIF	Mobile Agent System Interoperability Facility
MBean	Managed Bean
MI	Management Interface
MIF	Management Information Format
MIB	Management Information Base
MO	Managed Object
MOF	Managed Object Format
MoM	Manager-of-Manager
ODP	Open Distributed Processing
OID	Object Identifier
OM	Organisationsmodell
OMA	Object Management Architecture
OMG	Object Management Group
ORB	Object Request Broker
OSI	Open System Interconnection
PDA	Personal Digital Assistants
PDU	Protocol Data Unit
PNS	Place Name Service
RM	Reference Model
RPC	Remote Procedure Call
SCC	Signed Class Container

SMAE	System Management Application Entity
SMAP	System Management Application Protocol
SMFA	System Management Functional Area
SNIA	Storage Networking Industry Association
SNMP	Simple Network Management Protocol
SP	Service Provider
SPU	Signed Place User
SSL	Secure Socket Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TMN	Telecommunications Management Network
UDP	User Datagram Protocol
UML	Unified Modeling Language
UMTS	Universal Mobile Telecommunication System
WBEM	Web-based Enterprise Management Initiative
WWW	World Wide Web
XML	Extensible Markup Language

Kapitel 1

Motivation und Zielsetzung

1.1 Einleitung

Die vorliegende Arbeit beschäftigt sich mit dem Management eines Agentensystems am Beispiel ***A**synchronous **ME**ssage **T**ransfer **A**gent **S**ystem* (*AMETAS*) im allgemeinen und mit den Kontrollmechanismen für mobilen Agenten im Besonderen. Im Mittelpunkt dieser Arbeit stehen somit: Mobile Software-Agenten und deren Management.

1.1.1 Mobile Software-Agenten

Das Konzept der mobilen Software-Agenten hat in den vergangenen Jahren im Umfeld der verteilten und heterogenen Systeme zunehmend Interesse gefunden. Weltweit beschäftigen sich viele Forschungsgruppen an Hochschulen und wissenschaftlich-wirtschaftliche Organisationen mit den verschiedenen Aspekten dieser Thematik. Hintergrund des gewachsenen Interesses ist zweifellos das explosionsartige Wachstum des Internets in den letzten Jahren, das durch die rasante Entwicklung der Hard- und Software ermöglicht wurde, und das als Lösung für alle möglichen Probleme gesehen wird.

Die Dienste, die im Internet und vor allem im *World Wide Web* (WWW) angeboten werden, sind verblüffend vielfältig, und umfassen ein weites Spektrum. Online Banking, Einkaufen und Verkaufen, Steuererklärung, Suche nach medizinischen Informationen, Stimmenabgabe, virtuelle Erledigung von Behördengängen, usw. sind keine Fremdwörter im weltweiten Web. Die Menschen sind davon begeistert und fördern ihrerseits durch ihre Nachfrage weiterhin die Entwicklungen in diesem Bereich.

Die Folgen dieser Entwicklung sind die stark gestiegene Zahl der in diesem Bereich eingesetzten Hard- und Software, ihre Heterogenität und ihre Komplexität. Daraus ergeben sich viele Probleme, die geeignete und effiziente Lösungen verlangen. Deshalb werden seit einiger Zeit andere Lösungen auf der Basis unterschiedlicher Technologien erforscht. Eine dieser Technologien sind mobile Agenten, die effiziente und elegante Lösungen für die Probleme an-

bieten, die in verteilten und heterogenen Umgebungen auf RPC-Basis nicht mehr zufriedenstellend gelöst werden können. Einige dieser Problembereiche sind: Management, Suche im Internet, E-Commerce, Mobile Computers, drahtlose Kommunikation, die z.B. in den folgenden Quellen vor dem Hintergrund der Agenten-Technologie ausführlich behandelt wurden: [HE98], [OUKA99], [LISA99-1] und [LISA99-2].

1.1.2 Management

Trotz der steigenden Anzahl, der Heterogenität und der Komplexität der Hard- und Software müssen diese weiterhin überwacht, geschützt und gesteuert werden. Die Bandbreite des Internets ist nicht unbegrenzt, und die Zuverlässigkeit der Kommunikation mit einer vereinbarten Dienstqualität ist weiterhin ein zu lösendes und zu erforschendes Problem. Unter solchen Umständen spielt das Management mehr als eine unterstützende und begleitende Rolle.

Im Laufe der Entwicklung der Managementsysteme sind neben vielen proprietären Ansätzen eine bunte Landschaft von bereits standardisierten Architekturen entstanden, die teilweise in Konkurrenz zueinander stehen. Dabei wird die Heterogenität der Managementarchitekturen zur Heterogenität der zu managenden Anwendungen und Infrastrukturen hinzuaddiert. Es liegt auf der Hand, dass die offenen und interoperablen Managementlösungen zur Reduzierung der Komplexität des Managements beitragen.

Im Zuge des Wegs von proprietären Lösungen hin zur Offenheit und Kooperation hat sich in den letzten Jahren eine Strategie bewährt: Man stützt die gesamte Kommunikationsinfrastruktur auf vorhandene Internet-Protokolle und verwendet die Web-Browser als universelle, einfach benutzbare, plattformübergreifend verfügbare, graphische Benutzerschnittstellen, um über HTTP [3WHTTP] auf die Anwendungen und Daten zugreifen zu können. Außerdem stellt der Web-Browser eine für das Management wichtige plattformunabhängige Ausführungsumgebung für Java-Applets bereit. Mit dieser Strategie hat man eine Basis geschaffen, um auf dem Weg zum *integrierten Management* neben der Entwicklung von zugeschnittenen Lösungen für bestimmte Problembereiche auch die unterschiedlichen Managementinfrastrukturen unter ein Dach zu bringen. In [HeAN99] wurde das integrierte Management ausführlich behandelt.

An dieser sogenannten Web-basierten Lösung arbeiten viele Hersteller von Managementsystemen, und es gibt neben rein proprietären Lösungen zwei Ansätze, die jeweils auf eine offene Architektur für Web-basiertes Management abzielen: Die *Java Management API Architecture* (JMAPI), die von der Sun-Tochterfirma JavaSoft entwickelt und später in *Java Management Extension* (JMX) [JMX] umbenannt wurde, und die *Web-based Enterprise Management Initiative* (WBEM), deren Entwicklung an die *Desktop Management Task Force* (DMTF) [DMTF] übertragen wurde.

Ein anderer Gesichtpunkt bei der Betrachtung der immer wichtiger werdenden Managementanwendungen ist die Tatsache, dass sie auf dem sogenannten zentralistischen Ansatz basieren,

der eine hohe Netz- und Rechenlast, Verschwendung der Bandbreite, schlechte Skalierbarkeit und geringere Ausfallsicherheit (vgl. [HE98]), unzureichende Flexibilität sowie Interoperabilität (vgl. [KuWLuBa]) bedeuten. Deshalb werden seit einiger Zeit dezentrale Lösungen auf der Basis verschiedener Technologien erforscht.

Eine dieser Technologien sind mobile Agenten, die autonom durch ein Netzwerk wandern können, um diverse Managementaufgaben zu erledigen. Dabei übernehmen Sie im Rahmen der ihnen zur Verfügung stehenden Intelligenz eigenständig die Verantwortung für bestimmte Entscheidungen oder informieren hierarchisch höherstehende Instanzen (wie z.B. einen menschlichen Administrator), wenn vordefinierte Ereignisse eintreten. Durch die Verteilung und Dezentralisierung des Managements wird das Netzwerk entlastet. Es werden nicht mehr große Mengen von Rohdaten zur zentralen Auswertung übertragen, sondern die Software zur Auswertung dieser Daten gelangt in Form mobiler Agenten zu den Daten. Dies erhöht die Skalierbarkeit, die Robustheit und die Flexibilität solcher Managementanwendungen und ermöglicht die Einbettung von Mechanismen aus dem Bereich der *Künstlichen Intelligenz*.

1.2 Problemstellung

Obwohl die Entwicklung der Infrastrukturen für mobile Software-Agenten in vieler Hinsicht vorangetrieben wurde, wurden die Kontrollmechanismen der mobilen Agenten nicht genügend beleuchtet. Von irgendeinem Rechner im Netzwerk werden mobile Software-Agenten gestartet, die dann autonom quer durchs Netz wandern. Es gibt aber Gründe dafür, dass man wissen will, wo sich ein bestimmter Agent gerade aufhält. Z.B. erfordern viele Agentensysteme für eine (kostengünstige) Kommunikation mit den mobilen Agenten deren Lokalisierung. Außerdem hat man den Wunsch, einen Agenten terminieren zu können, wenn seine Aktivität nicht mehr benötigt wird. Die vorliegende Arbeit beschäftigt sich in erster Linie mit dieser Problematik. Das Hauptziel ist es dabei, geeignete Kontrollmechanismen für das Agentensystem AMETAS zu finden.

Befasst man sich mit den verschiedenen Aspekten von Kontrollmechanismen für mobilen Agenten, so stellt man fest, dass der Entwurf eines allgemeinen Managementrahmens für die Agenteninfrastruktur unumgänglich ist. Die Betrachtung des Sachverhalts von dieser Perspektive aus erleichtert die Wahrnehmung vieler wichtiger Aspekte dieser Problematik.

Dabei rückt zunächst das Zusammenspiel der Managementfunktionalität mit ihrer Infrastruktur in den Vordergrund. Die manuelle Überwachung und Steuerung einer verteilten Infrastruktur und ihrer Anwendungen ist wegen ihrer wachsenden Komplexität und Anzahl uneffektiv und oft unmöglich. Daher ist es notwendig, die Funktionalität in die Infrastrukturen zu integrieren. Dabei müssen geeignete Konstruktionsmethoden angewendet werden, die die charakteristischen Eigenschaften der zu managenden Objekte in Betracht ziehen und eine systematische Integration der unterschiedlichen Aspekte zulassen.

Ein anderer Gesichtpunkt ist, dass die Vielfältigkeit der managementrelevanten Aspekte sich aus verschiedenen Einflussgrößen ergeben, die nur szenario-spezifisch ermittelt werden können. Dabei sind die bei einer Agenteninfrastruktur gestellten Managementaufgaben unterschiedlicher als die bei einer herkömmlichen (verteilten) Anwendung. Es liegt also nahe, dass es nicht eine einzige Managementlösung geben kann, die überall einsetzbar ist. Außerdem müssen die Lösungen selbst einem Anpassungsprozess (Customizing) unterzogen werden. Dabei ist es sowohl bei der Wahl einer Managementlösung als auch im Anpassungsprozess eine große Herausforderung, das Management offen und herstellerübergreifend zu gestalten.

Ein erster Schritt für die Offenheit und Interoperabilität des Managements ist eine systematische Vorgehensweise bei der Klassifikation der Managementaufgaben. Dafür ist es sinnvoll, den Gesamtkomplex ”Management” in verschiedene Funktionsbereiche zu klassifizieren und dann bereichspezifisch typische Managementfunktionen zu beschreiben. Das OSI-Management bietet für diesen Zusammenhang ein universelles Modell an, das die Managementfunktionen in fünf Funktionsbereiche aufteilt. Sie sind im Funktionsmodell der OSI-Managementarchitektur festgelegt und bestehen aus dem Fehler-, Konfigurations-, Abrechnungs-, Leistungs- und Sicherheitsmanagement.

Weiterhin stellt das OSI-Management eine Art Referenzarchitektur dar. Das liegt an seinen mächtigen Modellierungsmöglichkeiten für Managementobjekte. Alle Teilmodelle einer Managementarchitektur werden hierdurch ausprägt. Diese Teilmodelle bieten eine Abstraktionsebene, in der die informations-, organisations-, kommunikations- und funktionsbezogenen Aspekte einer Managementarchitektur herstellerübergreifend, programmiersprachen- und architekturunabhängig modelliert werden können.

Ein weiterer Aspekt betrifft die Bestimmung der Managementarchitektur für das AMETAS. Um eine Wahl treffen zu können, muss man zunächst die gängigen Managementarchitekturen genau analysieren und bewerten. Dazu müssen gemäß der Besonderheiten eines Agentensystems Kriterien herauskristallisiert werden. Erst vor diesem Hintergrund wird die systematische Bewertung der existierenden Managementstandards aussagekräftig genug sein.

Baut man den Managementrahmen der gegebenen Infrastruktur im beschriebenen Sinne auf, so wird die Umsetzung der verschiedenen Strategien zur Kontrolle der Agenten deutlich vereinfacht. Außerdem erhält man eine günstige Ausgangbasis für jegliche Managementaktivitäten, die insbesondere mit Blick auf das Agentensystem viel an Komplexität gewonnen haben.

1.3 Gliederung der Arbeit

Um sich eine Übersicht über die mobilen Software-Agenten zu verschaffen, beschäftigt sich Kapitel 2 mit den Eigenschaften der mobilen Agenten. Gleichzeitig werden diejenigen Besonderheiten des Agentensystems AMETAS vorgestellt, die eine wesentliche Basis für die vorliegende Arbeit darstellen.

Kapitel 3 befasst sich mit der Rahmenbildung für das Anwendungs- und Infrastrukturmanagement. Dabei werden die Modellierungsmöglichkeiten einer Managementarchitektur näher betrachtet und die OSI-Funktionsbereiche erläutert. Diese spielen im Rahmen dieser Arbeit für die Klassifizierung der Managementfunktionen eine entscheidende Rolle und dienen als Basis für die Anforderungsanalyse des AMETAS-Managements. Die Anforderungsanalyse bildet den Gegenstand des Kapitels 4.

Im Kapitel 5 wird ein bewertender Überblick über einige gängige Managementarchitekturen geboten, die als Managementinfrastruktur für das AMETAS in Frage kommen könnten. Anschließend wird im Kapitel 6 die JMX-Managementarchitektur vorgestellt, die die Basis für das AMETAS-Management bildet. Das AMETAS-Management, seine Bestandteile und seine Funktionsweise werden ausführlich im Kapitel 7 behandelt.

Die Kapitels 8 und 9 sind der Lokalisierung von Agenten gewidmet. Dabei werden im Kapitel 8 die Notwendigkeit der Kontrollmechanismen für mobilen Agenten, mögliche Strategien und ihre Klassifizierung untersucht. Im Kapitel 9 werden die möglichen Lösungen für die Lokalisierung von Agenten im AMETAS auf die Gegebenheiten des Systems hin untersucht, und Lösungsvorschläge des Autors der vorliegenden Arbeit vorgestellt.

Kapitel 10 fasst die Arbeit so weit zusammen, dass sich ein zusammenhängendes Bild von allen Kapiteln und damit von der Gesamtaufgabe der vorliegenden Arbeit ergibt. Im Anhang finden sich Beispiele und Bilder.

Kapitel 2

Mobile Software-Agenten

Die Eigenschaften der mobilen Agenten prägen die Agentensysteme. Deshalb ist es aus der Managementsicht entscheidend, festzustellen, welche Eigenschaften der Software-Agenten im Vordergrund stehen. Die Wahl der richtigen Strategie im Umgang mit der Fehlerbehandlung, dem Erzeugen und Terminieren sowie mit der Sicherheit und den anderen Aspekten der Agenten hängt in erster Linie von den Eigenschaften der Agenten ab. Es gibt bisher keine einheitlichen Design-Richtlinien für ein Agentensystem. Deshalb werden in jedem Agentensystem je nach Zielsetzung einige Eigenschaften betont und demgemäss eigene Systemrichtlinien.

Um eine Übersicht über die Charakteristika des Agentensystems AMETAS zu gewinnen und dadurch seine Sicht der Agenten besser zu erklären, werden zunächst die Eigenschaften der mobilen Software-Agenten im Allgemeinen näher untersucht. Daraufhin werden die Infrastruktur, die AMETAS-Sicht der Agenten und der Kommunikationsmechanismus innerhalb vom AMETAS behandelt. Durch diese Vorgehensweise erhält man eine Vergleichsbasis, die die Besonderheiten vom AMETAS präziser in den Vordergrund treten lässt.

2.1 Eigenschaften mobiler Software-Agenten

Über die Definition von Software-Agenten und ihre Unterschiede gegenüber "normalen" Programmen gibt es keine Einigkeit. Je nach dem Ziel, das die Entwickler eines Agentensystems verfolgt haben, wurden die Agenten anhand einer oder mehrerer Eigenschaften definiert.

Im Allgemeinen charakterisiert man die Agenten mit Hilfe einer Menge der folgenden Eigenschaften:

- **Autonomie**: die Agenten bearbeiten ihre Aufgaben ohne Intervention von Menschen oder anderen Objekten, sie kontrollieren ihre Aktionen und ihren Zustand anhand der ihnen zugeteilten inneren Logik.
- **Reaktivität:** die Agenten verändern ihre Umgebung, indem sie ihre Umgebung beobachten und sich ein Bild von ihr machen gemäß ihrer Aufgabe(n), und das gemachte Bild in

Einklang mit ihren Zielen bringen.

- **Proaktivität**: die Agenten führen ihre Aktivitäten nicht in einer Aktions-Reaktionskette mit der Umgebung aus. Sie sammeln ihre Daten von der Umgebung, werten sie aus, und wenn es notwendig ist (das ist bestimmt durch die Schlussfolgerung aus den ausgewerteten Daten und deren Verhältnis zu den Zielen der Agenten), lösen sie selbständig und ohne weiteren äußeren Einfluss Aktionen aus.

- **Soziale Fähigkeiten**: die Agenten verfügen über die Fähigkeit, mit den anderen Agenten und gegebenenfalls auch mit menschlichen Anwendern zu kommunizieren. Dabei muss eine Sprache für die Kommunikation der Agenten festgelegt werden.

- **Adaptabilität**: dieses Verhalten bringt die Fähigkeit der Agenten zum Ausdruck, von den ausgelösten Aktionen und dem Einfluss dieser Aktionen auf ihre Umgebung zu lernen. Das schafft eine Wissensbasis, die den Agenten in der Zukunft hilft, ihre Aktionen zu optimieren, und ihr Anpassungsvermögen zu verbessern.

- **Mobilität**: mobile Agenten können eigenständig ihre Ausführungsumgebung ändern und durch das Netzwerk wandern, um die ihnen zugeteilten Aufgaben im Rahmen der ihnen zugeteilten Intelligenz zu erledigen.

- **Charakter**: Agenten können wie Menschen eine vertrauenswürdige Persönlichkeit und zielorientierte Verhaltensweisen nachweisen, einen emotionalen Zustand erreichen oder Vertrauen, Glauben und Wünsche haben.

Die ersten drei Eigenschaften werden meistens als Grundeigenschaften der Agenten bezeichnet, wobei, wie das folgende Beispiel aber auch das Beispiel der AMETAS-Agenten zeigt, die Autonomie die führende Rolle besitzt. Deshalb werden die Agenten manchmal auch *autonome* Agenten genannt. In [FG96] findet man folgende formale Definition von Agenten, die die Unterscheidung zwischen einem Agenten und einem beliebigen Programm erlaubt:

> *An **autonomous agent** is a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future.*

Diese Definition hebt die Autonomie, Reaktivität und Proaktivität als entscheidende Eigenschaften der Agenten hervor. Software-Agenten sind demnach Unterklassen der Agenten, deren Lebensraum Computer, Betriebssysteme, Netzwerke, usw. sind, und mobile Software-Agenten sind Software-Agenten, die um die Eigenschaft der Mobilität erweitert wurden. Die FIPA[1] definiert einen Agenten anders und stellt vor allem seine sozialen Fähigkeiten in den Vordergrund:

[1] FIPA (*Foundation for Intelligent Physical Agents*) ist eine nicht-kommerzielle Assoziation von Firmen und Universitäten, die im Agentenbereich tätig sind. Das Hauptziel der FIPA ist es, rechtzeitig Spezifikationen und Standards zur Erhöhung der Interoperabilität zwischen den agenten-basierten Anwendungen, Diensten und Geräten zur Verfügung zu stellen (http://www.fipa.org/).

> *An Agent is the fundamental actor in a domain. It combines one or more service capabilities into an unified and integrated execution model which can include access to external software, human users and communication facilities. ([FIPA-MA98] Page 3)*

Und ein mobiler Agent ist:

> *An agent that is not reliant upon the agent platform where it began executing and can subsequently transport itself between agent platforms. ([FIPA-MA98] Page 5)*

Nach der Definition der FIPA ist ein Agent eine Entität, die ihren eigenen Zustand, Steuerungs-Thread und ihre eigene Verhaltensweise hat. Er besitzt die Fähigkeit zur Interaktion und Kommunikation mit den anderen Entitäten wie z.B. menschlichen Anwendern oder anderen Agenten. Die Zusammenarbeit ist im Agenten-Paradigma anders als im Klient-Server-Paradigma. Um ihr Ziel zu erreichen, können Agenten mit anderen Agenten in einer gleichberechtigten Ebene vermittelnd, kollaborativ oder kooperativ zusammenarbeiten.

FIPA-Agenten erbringen Dienste, die entweder aus Informationen über diverse Software, von ihnen angebotenen Diensten und ihren Schnittstellen bestehen, oder selbst Schnittstellen zur Nutzung der integrierten Software anbieten. In diesem Sinne ist nach FIPA die Autonomie der Agenten davon abhängig, wie stark die Agenten in einer Server-Rolle gebunden sind. In dieser Rolle müssen die Agenten auf äußere Anregungen der Klienten reagieren, und haben damit wenig Spielraum, um ihre Autonomie im Sinne der oben eingeführten Beschreibung zu wahren.

2.2 Das Agentensystem AMETAS

AMETAS ist eine Abkürzung für **A**synchronous **ME**ssage **T**ransfer **A**gent **S**ystem. Es ist eine auf der Programmiersprache Java basierte Plattform für verteilte Anwendungen, die auf mobilen Agenten basieren, und über asynchrone Nachrichten kommunizieren. Der Entwurf des Agentensystem ist von der Tatsache geprägt, dass es im Rahmen eines Forschungsprojekts für die Typisierbarkeit von Agenten entstanden ist. Im AMETAS genießt die Autonomie der Agenten besonders hohe Relevanz[2], und für die Unterstützung dieser Eigenschaft wird ein geeigneter Kommunikationsmechanismus auf der Basis des asynchronen Nachrichtenaustauschs entwickelt.

Die Infrastruktur der AMETAS-Anwendungen ist die sogenannte *Stelle*. Sie ist eine Ausführungsumgebung für die AMETAS-Agenten und anderen *Stellennutzer* und stellt den Stellennutzern Mechanismen zur Migration, Kommunikation und Dienstnutzung zur Verfügung. Auf der

[2] In [HE98] wird die Unterstützung der Autonomie von *AMETAS*-Agenten als oberste Entwurfsziel von *AMETAS* bezeichnet.

Basis der AMETAS-Kommunikationsinfrastruktur wurde eine Einteilung der Stellennutzer in die eigentlichen Agenten, *Dienste* und *Benutzeradapter* vollzogen. Dienste sind Komponenten des Systems, die die Anfragen der Stellennutzer bedienen, und Benutzeradapter sind Stellennutzer, die die Interaktion der menschlichen Benutzer mit dem Agentensystem unterstützen.

Ein wichtiges Ziel des AMETAS-Managements ist es, das Agentensystem anhand der vom AMETAS bereitgestellten Werkzeuge zu managen. Um das bewerkstelligen zu können, ist ein genaue Kenntnis der wichtigen Systemeigenschaften notwendig. In diesem Zusammenhang muss die AMETAS-Sicht der Agenten und demzufolge die Autonomie der Agenten sowie der Kommunikationsmechanismus genauer betrachtet werden. Außerdem sind die Managementwerkzeuge selbst Stellennutzer, die zur Kommunikation mit den anderen Stellennutzern den vom AMETAS bereitgestellten Kommunikationsmechanismus verwenden.

2.2.1 AMETAS-Infrastruktur

Das zentrale Objekt im AMETAS ist die Stelle. Eine Stelle ist die Basiseinheit des Agentensystems, durch die die Rechner zu Stationen werden und wo sich die Agenten aufhalten können.

Auf jedem Rechner können eine oder mehrere Stellen installiert sein. Die Stellen (genau genommen die Dienste in den Stellen) dürfen unter der Einhaltung der Sicherheitsrichtlinien mit Hilfe von Sockets mit anderen Rechnern im Netz Verbindung(en) aufnehmen, um die Dienste und andere Ressourcen auf diesen Rechnern nutzen zu können. Somit können Rechner ohne installierte Stellen ins Agentensystem integriert werden. Weiterhin können die Dienste, die im Netz vorhanden sind und eine Standard- oder gut dokumentierte Schnittstelle zur Nutzung anbieten, vom Agentensystem genutzt werden. So erweitert sich der Operationsbereich des Agentensystems, ohne dass alle Rechner die Agenteninfrastruktur enthalten müssen. Das Agentensystem kann mit Hilfe der Sockets Standardschnittstellen nach Außen anbieten, so wie es in der vorliegenden Arbeit der Fall ist. Hier wird eine HTTP-basierte Schnittstelle angeboten, durch die sich der Manager über den Standardbrowser mit den Stellen verbinden und Managementaktionen ausführen kann.

Die Abbildung 2.1 zeigt den schematischen Aufbau einer AMETAS-Stelle. Jede Stelle bietet den Stellennutzern und damit auch den Agenten eine Ausführungsumgebung und stellt ihnen ihre Basisfunktionen und Dienstleistungen zur Verfügung. Diese Möglichkeiten umfassen folgende Basisdienste:

- **Erzeugung von Threads** zur Ausführung der Agenten und jedes Stellennutzers. Zwischen den Stellennutzern und der Stelle befindet sich der jeweilige Stellennutzertreiber, der auf die speziellen Bedürfnisse der Stellennutzer zugeschnitten ist. Der Treiber eines jeden Stellennutzers wird bei der Ankunft des Stellennutzers von der Stelle generiert und als ein eigenständiges Thread gestartet, der seinerseits den Stellennutzer startet. Der Treiber ist somit der einzige Bezugspunkt, den jeder Stellennutzer hat. Für die Agenten be-

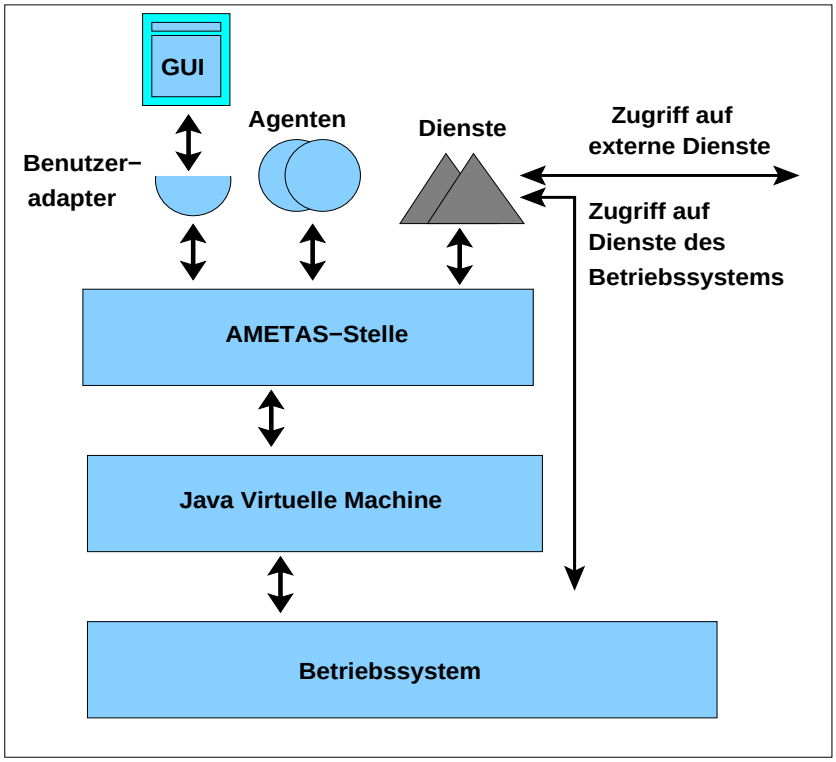


Abbildung 2.1: Schematischer Aufbau einer Stelle

deutet das, dass sie neben dieser Schnittstelle keine weitere Abhängigkeit haben. Welche weitgehenden Folgen das Fehlen jeder anderen Art von Abhängigkeit für die Lokalisierung der AMETAS-Agenten hat, wird in den Kapiteln 8 und 9 ausführlich behandelt.

- Die AMETAS-Stelle stellt den **Kommunikationsmechanismus** zum asynchronen Nachrichtenaustausch zwischen den Stellennutzern bereit. Dazu bietet die Stelle den Postamt-Dienst an. Dieser Dienst stellt einen Mechanismus zur Verfügung, der wie ein Postfach funktioniert. Die Stellennutzer können eine Nachricht an einen anderen in ein Postfach ablegen, so dass der Empfänger sie später abholen und auswerten kann.

- Die AMETAS-Stellen bieten einen **Ereignis-Mechanismus** an, der zur Verbreitung einiger in den Stellen abgespielten Ereignisse zu den interessierten Stellennutzern verwendet wird. Dabei können sich die Stellennutzer bei dem `AMETASEventManager` für bestimmte Ereignisse registrieren. Der Ereignismanager gibt dann alle Ereignisobjekte an die Stellennutzer weiter, die sich dafür registriert haben.

- Bereitstellung von **Migrationmechanismen** für die Agenten. Dazu wird das Absenden von Agenten durch den Aufruf einer Methode des Agenten-Treibers vollzogen. Dahingegen wird das Empfangen von Agenten von einem Dienst erledigt. Wenn ein Agent aus seiner inneren Logik heraus zu einer anderen Stelle auswandern möchte, ruft er seine `go`-Methode auf, die von seinem Treiber ausgeführt wird. Der Treiber nimmt eine Ver-

bindung zu der Stelle auf, die den Agenten empfangen soll. Es findet eine auf das sichere *Needhelm Schroeder Authentication*-Protokoll basierte Authentifizierung statt, nach deren Abschluss die Migration durchgeführt wird.

- **Dienstvermittlung** in den Stellen. Dabei hat jeder Stellennutzer die Möglichkeit, sich über seinen Treiber nach einem bestimmten Dienst zu erkundigen, der instantiiert oder aber als *SignedPlaceUser* (SPU)-Container registriert ist. Im zweiten Fall kann dieser dann zur Instantiierung aufgefordert werden.

- **Stellennamendienst**. Dieser Dienst wird als *PNS* (*Place Name Service*) bezeichnet und verwaltet die Stellennamen der AMETAS-Stellen. Dabei handelt es sich um die Abbildung der symbolischen Namen einer jeden Stelle auf der IP-Adresse des Rechners, auf dem die Stelle ausgeführt wird. Der PNS-Dienst ist ein eigenständiges Java-Programm, das die Anfragen der Klienten entweder selbst beantwortet oder die Antwort von den anderen ihm bekannten PNS einholt und vermittelt. Die Struktur des PNS-Dienstes ist DNS-ähnlich und hierarchisch aufgebaut. Das Format der symbolischen Stellennamen ist zwar DNS-ähnlich, die Domänen des PNS sind aber von den DNS-Rechnernamen unabhängig.

- Kommunikation mit den Anwendern über die Benutzerschnittstellen. **Benutzeradapter** integrieren die menschlichen Anwender in das Agentensystem. Das ist dadurch möglich, dass die Benutzeradapter in der gleichen Ebene wie die Agenten und Dienste liegen und die gleiche Kommunikationsinfrastruktur wie diese verwenden. Sie können Benutzeroberflächen darstellen, die die menschlichen Aktivitäten in Nachrichten übersetzen und entsprechend auf die Antworten der Agenten warten.

Auf der Basis des AMETAS-Kommunikationsmechanismus' werden die Stellennutzer in die Agenten, die Dienste und die Benutzeradapter eingeteilt. Damit wird die Dienstfunktionalität von den Entitäten mit Agentencharakter abgetrennt. Theoretisch dient diese Trennung zum Verdeutlichen und Hervorheben der Autonomie der Agenten. Dienste besitzen keine Autonomie, sie müssen nur die Klienten bedienen, und sonst abwarten. In diesem Sinne sind also die Dienste im AMETAS keine Agenten. Von dem praktischen Standpunkt aus gesehen hilft die Einteilung der Stellennutzer in Agenten, Dienste und Benutzeradapter der Entwicklung eines mächtigen und flexiblen Sicherheitssystems, das die möglichen Risiken, die im Agentensystem eintreten können, verringert bzw. verhindert. Zugleich besitzt es die nötige Flexibilität, um nützliche, mächtige und flexible Anwendungen zu unterstützen.

2.2.2 AMETAS-Sicht der Agenten

Das AMETAS unterstützt die Mobilität der Agenten und legt großen Wert auf die Autonomie der Agenten, so dass diese als die entscheidende Eigenschaft für die Bezeichnung der Objekte

als Agenten hervorzuheben ist (siehe [AMoA]). Die Agenten sind somit mobile und autonome Komponenten vom AMETAS, die einen beliebigen Auftrag seines Programmierers in seinem Namen durchführen können, ohne mit ihm in weiterem Kontakt zu stehen. Sie können dazu auch von einer Stelle zur nächsten migrieren und damit auch von einem Rechner zum anderen. Im [HE98] (S. 37) wurde folgendes notiert:

> AMETAS-Agenten sind mobile, autonome Klienten, die in sich abgeschlossen sind und keine Referenzen auf andere Agenten halten. Sie kommunizieren über einen asynchronen Nachrichtenaustausch-Mechanismus, um die Agentenautonomie zu wahren und die Kontrolle der Kommunikationsaktivitäten zum Zwecke ihrer Sicherheit und Typisierung zu gewährleisten. Ein direkter Zugriff auf das unterliegende System (Dateisystem, Graphikausgabe usw.) ist *AMETAS*-Agenten, ebenfalls aus Gründen der Sicherheit und Autonomie nicht gestattet.

Um die Autonomie der Agenten nicht einzuschränken, werden folgende Maßnahmen getroffen:

1. Die AMETAS-Agenten haben keine Schnittstelle, um direkt, z.B. durch Methodenaufruf, von den anderen Objekten im Agenten-Umfeld angesprochen zu werden. Aus AMETAS-Sicht wird durch das Versehen der Agenten mit RPC-ähnlichen Kommunikationsmechanismen die Autonomie eingeschränkt. Dadurch werden Methoden der Agenten lokal oder entfernt aufgerufen, was zur Folge hat, dass ihr Zustand durch andere Entitäten im Agentenumfeld überwacht, geändert und gesteuert werden kann (vgl. [AMoC]). Deshalb unterstützt AMETAS keine RPC-ähnlichen Kommunikationen zwischen den Agenten.

2. Eine direkte Modifikation des Systems durch die AMETAS-Agenten ist nicht erlaubt. Sie haben keinerlei Zugriff auf das Dateisystem, auf die Hardware oder auf andere Komponenten des Betriebssystems. Um diese Art von Aufgaben zu erledigen, müssen sie die AMETAS-Dienste in Anspruch nehmen. Dazu wird der asynchrone Nachrichtenaustausch verwendet. Die Idee hinter dieser Einschränkung ist die, dass ein direkter Zugriff auf externe Komponenten einen Agenten in eine Situation bringen kann, in der er seinem Auftrag nicht mehr nachkommen kann. Dieser Fall kann z.B. bei der Ein- und Ausgabe von Daten auf externe Komponenten eintreten.

3. Die direkte Interaktion mit einem Benutzer ist den AMETAS-Agenten nicht gestattet. Sie können weder eine grafische Oberfläche anbieten noch Reaktionen des Benutzers, etwa Mausklicks, Tastatureingaben oder ähnliches entgegennehmen. Würde ein Agent in eine direkte Interaktion verwickelt werden, so bestünde die Gefahr, dass dadurch der Agent auf irgendein Ereignis, das von Außen eintreten kann, wie z.B. Benutzereingaben, warten müsste. Da die Agenten aber **von sich aus** die Aktionen veranlassen, ist der Ausschluss mit Blick auf die Autonomie der Agenten unumgänglich (vgl. [AMTU]) . Gleichzeitig ist

diese Einschränkung sinnvoll, da die Agenten im Netz zu einer Stelle wandern können, in der die Darstellung seiner Schnittstelle nicht möglich ist.

4. Die Agenten sind nicht verpflichtet, die Nachrichten sofort zu verarbeiten. Sie können die Verarbeitung der Nachrichten verschieben oder ignorieren. Das bedeutet, dass auch der asynchrone Nachrichtenaustausch die Agenten nicht zu einer Aktion zwingen und damit ihre Autonomie einschränken darf.

5. Die Modifikation des internen Zustands eines Agenten beim direkten oder indirekten Setzen der internen Variablen ist nicht möglich. Diese geschieht gegebenenfalls durch den Agenten selbst, gesteuert durch seine inneren Logik.

6. Die AMETAS-Agenten entscheiden allein und ohne irgendeinen Einfluss von außen, wann und wohin sie migrieren wollen. Jede Art der Intervention in die Pfadänderung oder Verzögerung der Migration wird als eine Einschränkung ihrer Autonomie angesehen.

7. AMETAS-Agenten haben ihre eigenen Threads, die von der Laufzeitumgebung bereitgestellt werden. Genau genommen befindet sich zwischen dem Agenten und der Stelle der Agententreiber. Dieser leitet auf der einen Seite die Aufrufe der Stelle zu den betroffenen Agenten weiter und nimmt auf der anderen Seite die Aufrufe des Agenten entgegen, er bearbeitet sie und leitet sie an die Stelle weiter. Der Treiber ist somit die einzige Möglichkeit, über die der Agent einige Dienste der Stelle wie z.B. den Nachrichtendienst in Anspruch nehmen kann. Er hat keinen direkten Bezug auf die Stelle. Nur den Diensten ist es erlaubt, eine Stelle zu referenzieren.

Die AMETAS-Agenten verlassen ihre Klientenrolle in ihrem gesamten Lebenszyklus nicht und können durch die anderen Agenten nicht zur Erbringung eines Dienstes gezwungen werden.

Aus Managementsicht ist die uneingeschränkte Autonomie der Agenten aber gleichbedeutend mit der Verhärtung des Managements der Agenten. Wie schon angedeutet, schließen sich die Autonomie der Agenten und das Vorhandensein irgendwelcher Mechanismen zur Kontrolle und Steuerung der Agenten gegenseitig aus. Je strenger die Mechanismen zum Schutz der Agentenautonomie in das System integriert sind, desto schwerer wird es sein, die Agenten im System zu überwachen und gegebenenfalls zu terminieren.

Im Kapitel *8* wird der Zusammenhang der Agentenautonomie, der Agentenkommunikation und der Kontrollmechanismen für mobile Agneten ausführlich behandelt. Weiterhin werden in den Kapiteln 7 und 9 die Konzepte für das AMETAS-Management und die Kontrollmechanismen für AMETAS-Agenten näher erläutert. Es ist aber bereits an dieser Stelle festzuhalten, dass die Entwurfsprinzipien des jeweiligen Agentensystems die Rolle der Autonomie und damit auch die Rolle der Lokalisierung der mobilen Agenten bestimmen. Für eine RPC-ähnliche Kommunikation ist z.B. die Lokalisierung der Agenten unumgänglich, während im AMETAS

die Lokalisierung von Agenten einen anderen Stellenwert hat, der für die Funktionalität des Kernsystems nicht weiter von Bedeutung ist.

2.2.3 Kommunikation im AMETAS

Für die Unterstützung der Agentenautonomie und die bessere Überschaubarkeit der Kommunikationskanäle hinsichtlich ihrer Sicherheit verwendet das AMETAS für die Kommunikation zwischen den Agenten den asynchronen Nachrichtenaustausch. Es gibt zwar die Möglichkeit, über den Ereignis-Mechanismus bestimmte Ereignisse in einer AMETAS-Stelle zu empfangen, dieser Mechanismus ist aber die einzige Methode, mit der die Agenten miteinander kommunizieren können.

Jede Stelle bietet den Stellennutzern einen Postamt-Dienst an, der die Nachrichten der Sender entgegennimmt. Der Dienst speichert die Nachrichten im Postamt, bis der Empfänger sie abholt. Der Dienst stellt außerdem eine Funktion zum Löschen der Nachrichten zur Verfügung. Jede Stellennutzer kann über seinen Treiber, der über geeignete Schnittstellen Zugriff auf die Dienste des Postamts hat, diese in Anspruch nehmen.

Zur Adressierung der Nachrichten an jeden Stellennutzer wird ein Objekt der Klasse `AMETAS.data.AMETASPlaceUserIDMask` (*ID-Maske*) verwendet, das Felder für die Angabe der IP-Adresse des Herkunftsrechners, den Zeitpunkt der Erzeugung, den Stellennutzernamen, die Stellennutzergruppe und den Starter des Stellennutzers besitzt. Bei diesem Objekt kann man die Felder auf `null` setzen. Betrachtet man nun die Postfächer im Postamt, die den ID-Masken zugeordnet sind, wird ersichtlich, dass man die Nachrichten an einzelne Stellennutzer, an eine Gruppe von Stellennutzern oder an alle Stellennutzer adressieren kann.

Alle Nachrichten sind Instanzen der Klasse `AMETAS.data.AMETASMessage`. Sie haben den Aufbau, der in der Abbildung 2.2 vorgestellt wurde. Auf die Felder *Absender, ID, Kategorie, aktive Berechtigungen* und *Absendestelle* haben die Absender keinen Einfluss, während sie die Felder *Empfängermaske* und *Inhalt* setzen müssen. Das Setzen der anderen Felder ist optional.

2.2.4 AMETAS-Ereignis-Mechanismus

Der Ereignis-Mechanismus ist neben dem asynchronen Nachrichtenaustausch eine weitere Kommunikationsmöglichkeit, die das AMETAS-Laufzeitsystem befähigt, die Stellennutzer über bestimmte Ereignisse zu informieren. Dazu müssen sich die Stellennutzer für diese Ereignisse beim Ereignismanager einer jeden AMETAS-Stelle registrieren. Der Ereignismanager ist eine Komponente, die die Registrierung der Ereignisinteressenten steuert, die eintretenden Ereignisse filtert und sie an registrierten Stellennutzer zustellt. Dabei verfügt jeder Stellennutzer über eine Warteschlage, die die gelieferten Ereignisse aufnimmt. Registriert sich ein Stellennutzer für bestimmte Ereignisse, wird in seinem Treiber eine weitere Komponente (*EventNotificator*)

Absender	ID des Absenders dieser Nachricht
Empfängermaske	ID−Maske des/der Empfänger(s) dieser Nachricht
ID	Eindeutige ID (= Identifikation)
Anwort auf ID	ID der Nachricht, die durch diese Nachricht beantwortet wird
Löschbarkeit	Information darüber, wer diese Nachricht löschen darf
Gültigkeit	Die Zeit, nach der diese Nachricht vom Postamt automatisch gelöscht werden kann
Kategorie	Die Kategorie dieser Nachricht
Subkategorie	Die Subkategorie dieser Nachricht
Aktive Berechtigungen	Berechtigungen, die der Stellennutzer zum Zeitpunkt des Absendens innehatte
Optionen	Optionen, die die Nachrichtenverarbeitung betreffen
Absendestelle	Name der Absendestelle, wenn es sich um eine entfernte Nachricht handelt
Inhalt	Nutzlast der Nachricht

Abbildung 2.2: Aufbau der Nachrichten im AMETAS

angelegt, die in ihrem eigenen Thread ausgeführt wird und die Ereignisse in der Warteschlange verarbeitet.

Es gibt drei verschiedene Ereignissorten, die nach ihrem Informationsgehalt kategorisiert werden:

- **Nachrichtenereignisse** werden bei den Aktionen generiert, die mit dem Postamt im Zusammenhang stehen. Diese Aktionen sind die Zustellungen an das Postamt und das Löschen von Nachrichten aus dem Postamt. Diese Ereignisse ermöglichen es den Stellennutzern, die ihnen zugestellten Nachrichten direkt zu erhalten.

- **Stellenereignisse** sind solche, die beim Auftreten wichtiger Ereignisse in einer Stelle generiert werden, wie z.B. das Herunterfahren der Stelle oder das Auftreten bestimmter Fehler in einer Stelle.

- **Stellennutzerereignisse** sind das Starten eines Stellennutzers, das Terminieren eines Stellennutzers, usw. Diese Ereignisse sind für das Beobachten der Stellennutzer und der Agenten sehr hilfreich. Im Rahmen dieser Arbeit werden diese Ereignisse für die Lokalisierung der Agenten im AMETAS von den entsprechenden Diensten aktiv verwendet.

Die genannten Kategorien werden als separate Klassen implementiert, die alle von einer Klasse abgeleitet sind. Jede dieser Klassen besitzt ein Feld für die *Event-ID*, die ein Ereignis genauer

beschreibt. Die Strukturierung und die Verwendung der Event-IDs in den Ereignisklassen sind mit dem bekannten Ereignis-Mechanismus vom *AWT* vergleichbar, der in Java verwendet wird.

Ein weiterer Aspekt bei der Verwendung von Ereignismeldungen zur Kommunikation ist, dass dieser Mechanismus die Autonomie der Agenten in gewisser Weise aufhebt. Ein typisches Beispiel hierfür sind die Nachrichtenereignisse. Ein Agent kann sich dazu entscheiden, seine Nachrichten direkt über die Ereignisnachrichten zustellen zu lassen. Bei einer direkten Zustellung der Nachrichten werden die eintreffenden Nachrichten, die der Registrierung entsprechen, vom Postamt direkt an den *Notificator* des Agenten weitergereicht. Sie liegen solange in dessen FIFO-Event-Schlange, bis sie vom Notificator mit der *notifyListener*-Methode des Agenten bearbeitet werden. Sie sind vom Moment der Zustellung an nur noch in dieser Schlange gespeichert. Es kann der Fall eintreten, dass ein Agent an der Migration gehindert wird, bis die Ereignisnachrichten in der Warteschlange bis zum Ende verarbeitet wurden. Diese Gefahr ist um so größer, wenn ein Agent viele Nachrichtenereignisse erwartet und gleichzeitig öfters migriert (vgl. [AMTU]).

Kapitel 3

Anwendungs- und Infrastrukturmanagement

3.1 Motivation

Das Anwendungs- und Infrastrukturmanagement ist im Kontext der vernetzten, kooperativen und verteilten IT-Infrastrukturen Teil eines Gesamtkomplexes, der in einem allgemeinen Sinne *Management* genannt wird. Um die Besonderheiten dieses Bereichs im wesentlichen charakterisieren zu können, ist es sinnvoll, ihn in Verbindung mit dem allgemeinen Kontext zu analysieren.

Das damit verknüpfte Problem ist allerdings, eine umfassende Definition für das Management zu finden. Definiert man das Management mit Hilfe seines primären Zieles und stellt man die Zufriedenheit des Endanwenders eines verteilten Systems in den Vordergrund, so können die Maßnahmen als Management bezeichnet werden, die für die Verfügbarkeit, Zuverlässigkeit, Flexibilität und Performanz der angebotenen Dienste und Anwendungen sorgen. Durch so eine Zielorientierung werden Maßstäbe an Dienstgüten gesetzt, die Anforderungen an die technischen Managementprozesse stellen.

Stellt man im Gegensatz dazu die Ausschöpfung der technischen Gegebenheiten und den wirtschaftlichen Erfolg des Unternehmens in den Vordergrund, so versteht man unter Management:

> *alle Maßnahmen für einen unternehmenszielorientierten effektiven und effizienten Betrieb eines verteilten Systems mit seinen Ressourcen. ([HeAN99], S. 6)*

Eine weitere Sichtweise besteht darin, das Management aus der Perspektive des Systemadministrators zu betrachten, der für die Bereitstellung, Aufrechthaltung, Verfügbarkeit und Sicherheit der Dienste und Anwendungen des verteilten Systems sorgt.

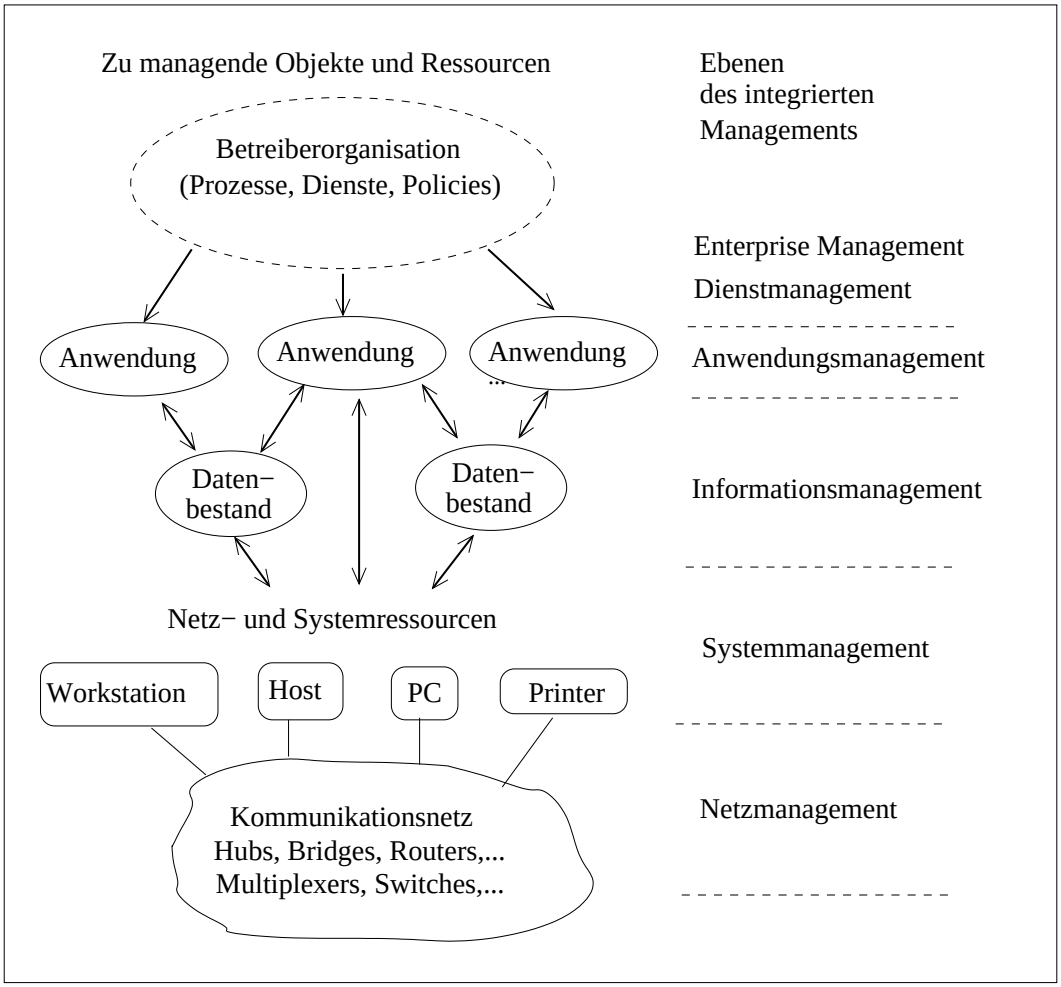


Abbildung 3.1: Ebenen des integrierten Managements

3.2 Managementbereiche

Das Management verteilter Systeme umfasst verschiedene Aspekte wie z.B. Verfahren, Programme sowie Werkzeuge und betrifft mehrere Ebenen der Betrachtung. Die verschiedenen Ebenen unterscheiden sich dadurch, dass jeweils das Management bestimmter Objekte im Mittelpunkt stehen. In *[HeAN99]* wurden z.B. die Managementebenen in einem Unternehmen wie in der Abbildung 3.1 dargestellt[1].

Dabei kann man die zum IT-Management gehörenden Bereiche folgendermaßen beschreiben:

- Das **Netzmanagement** beschäftigt sich mit dem Management von Kommunikationsdiensten und Netzkomponenten. Die Objekte des Netzmanagements sind z.B. Leitungen, Übertragungs- und Vermittelungseinrichtungen wie Hubs, Bridges, Routers sowie Protokollinstanzen.

[1]Eine andere Darstellungsweise wäre eine Pyramidenform, in der jede Ebene auf der Basis der darunter liegenden Ebene aufgebaut wird.

- Sind die Gegenstände des Managements Ressourcen von Endsystemen wie z.B. CPU's, Speicher, Platten, Peripheriegeräte, Server, Benutzer, Logs, Filesysteme oder Softwaremodule, so spricht man vom **Systemmanagement.** Die angebotenen Dienste des Systemmanagements sind z.B. Datenhaltung, Softwareverteilung, Lizenzkontrolle, Lastverteilung, Spooling und Abrechnung.

- Sind die zu managenden Ressourcen verteilte Anwendungen und verteilte Dienste, die den Kunden angeboten werden, dann ist das **Anwendungsmanagement** dafür zuständig. Beispiele für solche Dienste sind E-Mail, Verzeichnisdienste, Dokumentenverwaltung sowie Dienste für das Bestell- und Abrechnungswesen.

- Das **Informationsmanagement** hat mit dem Entwurf und der Pflege von Datenbeständen, ihrer Konsistenzhaltung und ihrer durchgängigen Verfügbarkeit zu tun.

In einem Unternehmen hat das **Enterprise-Management** die Aufgabe, verschiedene Managementbereiche unter unternehmensweiten Gesichtpunkten wie z.B. Geschäftprozesse zu koordinieren, und Zielvorgaben (Policies) für untere Bereiche festzulegen.

Bei einer näheren Betrachtung der Abbildung 3.1 kann man feststellen, dass die Übergänge zwischen den Ebenen unscharf sind, und dass eine noch feinere Einteilung der Ebenen möglich ist. Z.B. kann das **Infrastrukturmanagement**, das sich mit der Koordinierung der Prozesse eines Laufzeitsystems beschäftigt und den Systemanwendern die benötigten Basisdienste bereitstellt, zwischen dem System- und Anwendungsmanagement angesiedelt werden.

3.3 Orientierung

Die verschiedenen Managementlösungen stellen verteilte Anwendungen dar, die durch die Verteiltheit der zu managenden Objekte bedingt sind. Die Managementlösungen verursachen also eine weitere Komplexität, deren integrale Betrachtung dazu führt, dass bei einer Lösung solche Konzepte zu bevorzugen sind, die möglichst auf mehrere Managementschichten anwendbar sind.

Das Ziel bei der Entwicklung einer Managementlösung sollte so gesetzt werden, dass die Einbettung der Lösung in ein Gesamtkonzept möglich ist. Die Orientierung soll also ein *integriertes Management* sein, das eine integrale Betrachtung von verschiedenen Managementaspekten gestattet, Organisationsaspekte einbezieht (z.B. ein Domänenkonzept), heterogene Systeme unterstützt sowie einheitliche Programmierschnittstellen und Bedienoberflächen anbietet.[2]

[2]Eine detaillierte Beschreibung des integrierten Managements würde den Rahmen der vorliegenden Arbeit sprengen, daher wird an dieser Stelle auf die weiteren Ausführungen in [HeAN99] und [KELL98] verwiesen.

Die vorliegende Arbeit widmet sich im Kern Fragen des Managements vom AMETAS, insbesondere der Lokalisierung von Agenten. Notwendige Nebenbedingungen bzw. -ziele sind dabei:

1. Die verschiedenen weiterentwickelten Basisfunktionen für das Management sollen in die neue Managementlösung integriert werden können.
2. Die Managementlösung muss für die noch nicht gelösten Managementprobleme einen geeigneten Lösungsweg vorschlagen. An der Spitze dieser Probleme steht die Lokalisierung von Agenten im AMETAS.
3. Die Lösung soll auf einer offenen und universellen Managementarchitektur beruhen, die mit den anderen Managementstandards interoperabel ist, so dass ihre Kooperation gewährleistet ist.

Wenn man nach der Orientierung der beschriebenen strategischen Gedanken, der konkreten Schritte in diese Richtung, den vorgelegten Lösungen und der gewählte Managementarchitektur in einer zusammenhängenden Reihenfolge fragt, wird der Weg zum integrierten Management deutlich.

3.4 Managementarchitektur

Eine Managementarchitektur (MA) ist ein Rahmenwerk für managementrelevante Standards. Managementarchitekturen sind die Voraussetzungen für den Entwurf von Managementplattformen, die ihrerseits Trägersysteme für Managementanwendungen sind.

Eine MA kann hinsichtlich verschiedener Aspekte beschrieben werden. Diese Aspekte betreffen die Modellierung der zu managenden Ressourcen, die Managementaufgaben, die Verteilung der Managementkomponenten, die Managementfunktionalität, die Kommunikation der Komponenten, usw. In der Literatur wird eine MA mit Hilfe der Aspekte beschrieben, die in den folgenden Abschnitten erläutert werden.

3.4.1 Informationsmodell

Das Informationsmodell ist das Herzstück einer jeden MA. Es spezifiziert einen Beschreibungsrahmen für Managementobjekte und regelt die Methoden zur Modellierung und Beschreibung von Managementobjekten, die eine Abstraktion der Ressourcen darstellen, auf denen das Management operiert. Das Informationsmodell muss für jedes Managementobjekt (MO) die folgenden Punkte festlegen:

- die eindeutige Identifikation eines MO in einem lokalen oder globalen Managementkontext

- die managementrelevanten Attribute eines MO
- die Zugriffrechte auf einzelne Attribute eines MO
- die Methoden und Operationen zur Manipulation der Attribute
- die Methoden, Aktionen und Operationen zur Manipulation der ganzen MO
- die Schnittstellen zu den Managementprotokollen
- die Fähigkeit eines MO, ein eingetretenes Ereignis ohne Anforderung des Managers zu emittieren.

In der OSI-MA gibt es Spezifikationen zum Verhalten[3] und zur Beziehung des MO zu anderen Objekten betreffend die Position des MO im hierarchischen Baum.

Das Informationsmodell einer MA legt auch den Modellierungsansatz und eine eindeutige Syntax für die Beschreibung von Managementinformationen fest. Zum Beispiel wird in der OSI-MA, OMG, JMX und DMTF ein objektorientierter Ansatz gewählt, während im Internet-Management die MO durch den Datentypansatz beschrieben werden (vgl. [ISO 10165-4], [ISO 7498-4], [OMG], [JMXIAS], [DMTF] und [RO91]).

3.4.2 Organisationsmodell

Das Organisationsmodell einer MA legt die Akteure, ihre Rolle und die Grundprinzipien ihrer Kooperation fest. Vom topologischen und funktionalen Gesichtspunkt aus betrachtet, gibt es folgende Möglichkeiten:

- Die **Zentrale Architektur** besteht aus einem zentralen, einzelnen Manager, der für das Management des gesamten Netzes verantwortlich ist. Der Manager betreibt die Kommunikation mit den Managementagenten, die in den Netzkomponenten sitzen. Er trifft die Managemententscheidungen alleine und unterhält eine Datenbank.
- Die **Verteilte Architektur** verwendet mehr als einen Peer-Manager. Diese Architektur ist sehr gut für mehrstufige Domänen-Netze geeignet, denn für jede Domäne ist ein Manager zuständig. Falls Informationen von einer bestimmten Domäne benötigt werden, kommuniziert der Manager mit dem verantwortlichen Peer-Manager, um die benötigten Informationen zu erhalten. Skalierbarkeit ist der große Vorteil dieses Modells.
- Die **Hierarchische Architektur** benutzt ebenfalls eine Aufteilung in Domänen. Jeder Domänen-Manager ist verantwortlich für seiner Domäne, aber nicht für das restliche

[3] Engl. Behavior: Dies betrifft die Semantik und das Zusammenspiel von Attributen, Notifikationen und Operationen.

Netz. Weiterhin gibt es noch einen *Manager-of-Manager* (MoM). Der MoM arbeitet auf einer höheren Hierarchiestufe und fordert Informationen von den darunter liegenden Domänen-Managern an. Im Gegensatz zur vorherigen Architektur kommunizieren die Domänen-Manager aber nicht untereinander.

- Die **Netzwerk-Architektur** ist eine Kombination aus der verteilten und der hierarchischen Struktur, die Manager sind in einer vernetzten Struktur verbunden. Die Zuordnung von Ressourcen zu Managern ist nicht eindeutig, die Art der Zuordnung hängt vielmehr von verschiedenen Einflussgrößen ab. So können z.B. die gleichen Ressourcen mehreren Managern zugeordnet sein, etwa wenn der eine Manager für das Sicherheits- und der andere für das Leistungsmanagement zuständig ist.

Derzeit sind zwei unterschiedliche Kooperationsformen denkbar:

1. Der **Manager-Agenten-Ansatz** mit seiner asymmetrischen und hierarchischen Kooperationsform unterstellt ein Auftraggeber-Auftragnehmer-Verhältnis für eine Kooperationsbeziehung und ähnelt damit dem bekannten Klient-Server-Modell. Der Manager beauftragt den Managementagenten, eine bestimmte Operation auszuführen oder bestimmte Information bereitzustellen. Der Managementagent antwortet dann mit dem Ergebnis der Operation oder der gewünschten Information. Die Rollen der Akteure können sich dynamisch je nach Aufgabe verändern.

2. Der **Peer-to-Peer-Ansatz** geht von einer Kommunikation und Kooperation prinzipiell gleichberechtigter Objekte aus. Bei diesem Modell findet eine flexible, wechselseitige Auftragsbeziehung und ein Informationsaustausch in beide Richtungen statt.

3.4.3 Kommunikationsmodell

Das Kommunikationsmodell einer MA legt die Konzepte zum Austausch von Managementinformationen zwischen den Akteuren mit dem Ziel der Steuerung, Überwachung und Ereignismeldung fest. Das Kommunikationsmodell regelt folgende Aspekte:

- Festlegung der kommunizierenden Partner
- Festlegung des Kommunikationsmechanismus' für mögliche Kommunikationszwecke
- Festlegung von Syntax und Semantik der Protokoll-Datenstrukturen
- Einbettung von Managementprotokollen in die Protokollhierarchie der zugrundeliegenden Kommunikationsarchitektur. Dazu gehört die Festlegung der Schnittstellen zu den (Nichtmanagement-) Protokollinstanzen, Anwendungsprozessen und lokalen Betriebssystemprozessen.

Zum Austausch der Managementdaten gibt es zwei verschiedene Konzepte: Das *Pull*-Modell, das auf dem Abfragen von Managementagenten durch den Manager basiert. In diesem Modell liefern die Managementagenten Informationen auf explizite Aufforderungen des Managers hin. Wenn die Anfragen in zeitlichen Intervallen wiederholt werden, spricht man vom *Polling*. Das andere Modell ist das *Push*-Modell. In diesem senden die Managementagenten ohne vorherige Aufforderung durch den Manager selbsttätig die Managementdaten an den Manager. Dieses Modell wird insbesondere zur asynchronen Ereignismeldung verwendet. Man muss allerdings feststellen, dass in der Praxis meistens eine Mischform der beiden Modelle angewendet wird.

3.4.4 Funktionsmodell

Das Funktionsmodell einer Managementarchitektur zergliedert die Managementaufgaben in Funktionsbereiche und versucht, generische Funktionen (teilweise bereichspezifisch) festzulegen. Im Funktionsmodell sind für die einzelnen Funktionsbereiche die erwartete Funktionalität und die Dienste sowie die zur Erbringung der Funktionalität erforderlichen Managementobjekte zu definieren. Somit ist das Funktionsmodell die Basis für Bibliotheken von Management-Teillösungen. Für offene Plattformen ist es sehr wichtig, die Aufrufschnittstellen der Managementfunktionen sowie die manager-internen Schnittstellen und
managementagenten-internen Schnittstellen zu spezifizieren.

3.5 Funktionsbereiche der OSI-Architektur

Das Funktionsmodell der OSI-MA teilt Managementaufgaben in fünf Funktionsbereiche, die bei OSI *Systems Management Functional Areas (SMFAs)* heißen, und in der englischsprachigen Literatur oft als FCAPS bezeichnet werden. FCAPS ist ein Akronym und wird aus den Anfangsbuchstaben von **F**ault (Fehler), **C**onfiguration (Konfiguration), **A**ccounting (Abrechnung), **P**erformance (Leistung) und **S**ecurity (Sicherheit) Management gebildet.

Diese Funktionsbereiche, die weiter unten erläutert werden, gelten grundsätzlich für alle Objekttypen. D.h. diese Klassifizierung gilt für alle Managementebenen, die zum Teil in der Abbildung 3.1 dargestellt sind. Sie ist damit orthogonal zur Klassifizierung der Managementbereiche nach Ebenen. Man kann anhand des Funktionsmodells vom OSI-Management typische Managementaufgaben beschreiben. Dabei steht die Funktionalität des Managementsystems im Vordergrund. Daher orientiert sich diese Arbeit bei der Unterteilung der Managementaufgaben und der Managementfunktionen an FCAPS.

3.5.1 Konfigurationsmanagement

Das Konfigurationsmanagement beschäftigt sich mit:

- der Beschreibung der Ressourcen in einem verteilten System bzgl. der physischen und geographischen Anordnung von Ressourcen,
- dem Einrichten einer Systemumgebung für den Normalbetrieb der Ressourcen und der zu managenden Objekte,
- dem Sammeln von Daten zur Konfiguration der zu managenden Ressourcen,
- der Konfiguration der Ressourcen und Objekte,
- der Manipulation der Struktur der verteilten Systeme bzw. der Ressourcen mit Hilfe der gesammelten Daten wie z.B. das Setzen und Ändern von Parametern, die den Normalbetrieb eines Systems regeln, oder eine Anpassung der Konfiguration zur Behebung von Fehlerzuständen,
- der Speicherung und Wartung der Konfiguration und
- der Darstellung der Konfiguration.

In einem Konfigurationsvorgang werden die Daten gesammelt, wobei die Konfiguration modifiziert, gespeichert und eventuell in einer geeigneten Form dokumentiert wird.

Mit den Werkzeugen, die dem Konfigurationsmanagement zugeordnet werden, kann man aus der konkreten realen Systemumgebung eine Konfigurationsbeschreibung ableiten, Dokumentationssysteme für Konfigurationsbeschreibungen pflegen, Bestandsdatenbanken unterhalten, die Konfigurationsdaten visualisieren, Konfigurationsparameter setzen oder Systemzustände abfragen, etc.

3.5.2 Fehlermanagement

Das Entdecken, Eingrenzen und Beheben von Fehlern sind Aufgaben des Fehlermanagements. Als Fehler kann man die Abweichung von gesetzten Betriebszielen, Systemfunktionen oder Diensten bezeichnen. Als Fehlerquellen kommen Datenübertragungssysteme, Netzkomponenten, Endsysteme, Softwarefehler, Programmierfehler, etc. in Frage. Die Ressourcen und Dienste durch schnelle Entdeckung und Beseitigung von Fehlern verfügbar zu halten, ist eine schwierige Aufgabe. Wenn die offensichtlichen Symptome der Fehler sichtbar werden, ist meist der Normalbetrieb des Systems oder der Infrastruktur beeinträchtigt. Die Fehler müssen also erkannt werden, bevor sie auftreten. Insbesondere die Isolation von Fehlern ist eine große Herausforderung. Um die genannten Ziele zu erreichen, ist folgende Vorgehensweise empfehlenswert:

- Überwachung des Zustands der Ressourcen,
- Entgegennahme und Verarbeitung der Alarmmeldungen,

- Diagnostizieren von Fehlern auf ihre Ursachen hin und Feststellung der Fehlerfortpflanzungen,
- Einleitung der Fehlerbehebungsmaßnahmen, und wenn es möglich ist: Die Beseitigung der Fehler inklusive ihrer Ursachen.

Für die Durchführung der genannten Aufgaben muss das Fehlermanagement über geeignete Werkzeuge verfügen. Der Einsatz einfacher Tools kann das Auftreten des Fehlers implizieren, während die fortgeschrittenen Tools sie automatisch erkennen und den Administrator benachrichtigen können.

3.5.3 Abrechnungsmanagement

Das Abrechnungsmanagement führt Buch über die verursachten Kosten, die durch die Benutzung der Ressourcen entstehen und auf die Kostenverursacher umgelegt werden müssen. Die Aufteilung der Kosten erfolgt gemäß einer Abrechnungspolitik, die vorgeschrieben sein muss bzw. vom Administrator bestimmt werden kann. Das Abrechnungsmanagement hat also die Aufgabe:

- die Daten über die Nutzung der Ressourcen zu sammeln,
- gemäß der Abrechnungspolicies und Metriken die Kosten auszurechnen,
- die Kosten den Konten der Verbraucher zuzuordnen.

Der Prozess der Daten- und Statistiksammlung kann dem Administrator helfen, Überblick über die Verwaltung und Zuordnung der Ressourcen zu bewahren, so dass bei einer Ressourcenknappheit die notwendigen Maßnahmen getroffen werden können. Das führt wiederum zu der Aufrechthaltung und manchmal auch zur Verbesserung der angebotenen Leistung.

3.5.4 Leistungsmanagement

Es besteht ein Zusammenhang zwischen dem Leistungs- und Fehlermanagement, so dass das Leistungsmanagement als konsequente Weiterführung des Fehlermanagements angesehen werden kann. Während das Fehlermanagement dafür sorgt, dass das System und die Dienste aufrecht erhalten bleiben, trägt das Leistungsmanagement Sorge dafür, dass das System und die angebotene Dienste gemäß einer vereinbarten Dienstgüte arbeiten. Eine Dienstgüte ist eine Schnittstelleninformation zwischen dem Dienstanbieter und dem Kunden. Eine vollständige Definition der Dienstschnittstellen ist im allgemeinen sehr schwierig, sie wird unter anderem durch die Spezifikation des Dienstes und des Diensttyps, die Festlegung der relevanten Parameter der Dienstgüte, die Angabe des Messverfahren, der Messstellen und der Messgrößen beeinflusst.

Das Leistungsmanagement umfasst vor allem folgende Maßnahmen zur Sicherstellung von Dienstgüten gemäß der getroffenen Dienstgütevereinbarungen (engl. Service Level Agreements):

- Bestimmen von Dienstgüteparametern
- Überwachen aller Ressourcen auf Leistungsengpässe
- Durchführen von Messungen
- Auswerten von History Logs, d.h. von Aufzeichnungen über Systemaktivitäten, Fehlerdateien, usw.
- Aufbereiten von Messdaten und Verfassen von Leistungsberichten
- Durchführen von Leistungs- und Kapazitätsplanungen.

3.5.5 Sicherheitsmanagement

Das Sicherheitsmanagement beschäftigt sich mit der Sicherheit in einem verteilten System. Die Sicherheitsmaßnahmen richten sich sowohl gegen die unsachgemäße Verwendung von Ressourcen wie z.B. Fehlbedienung oder Fehlfunktion von Ressourcen, oder auch gegen die Bedrohung der Ressourcen in einem System. Die Bedrohungen sind Angriffe gegen die Sicherheit des Systems, die von passiver oder aktiver Natur sein können. Passive Angriffe sind z.B. das Abhören von Informationen, unzulässige Datenverkehranalysen, etc.. Die aktiven Angriffe sind noch bedrohlicher. Dazu zählen z.B. Identitätsvortäuschung (Maskerade), Umordnung der Nachrichtensequenzen, Modifizierung der Nachrichten, Überlasten der Ressourcen, Umkonfiguration der Ressourcen, Umprogrammierung, Viren und trojanische Pferde.

Folgende Aufgaben kann man dem Sicherheitsmanagement zuordnen:

- Analysieren von Bedrohungen
- Festlegung der Sicherheitsrichtlinien ausgehend von den Bedrohungsanalysen
- Überprüfung und Feststellung der Identitäten (dazu müssen sichere Authentifizierungsmethoden wie Signaturen, Zertifizierung oder sogar Notarisierung verwendet werden)
- Überprüfen der Autorisierungen und Durchführen von Zugriffskontrollen
- Überwachung des Systems und der Ressourcen auf Sicherheitsangriffe
- Verschlüsselung der ausgetauschten Daten.

Es gibt im Bereich des Sicherheitsmanagements eine Menge von frei verfügbarer Software, in der die stabilen und anerkannten Sicherheitsverfahren verwendet werden. Das Sicherheitsmanagement kann diese Software auf geeignete Weise in die Managementarchitektur einbetten und gemäß den Bestimmungen der Sicherheitspolitik durchgängig steuern.

3.5.6 Ergänzung

Es gibt weitere Klassifizierungsansätze, die die vom OSI-Management festgelegten Funktionsbereiche erweitern, wie z.B. das *Bestandsmanagement*, das die Funktionen für Bestandsführung, Archivierung, usw. einschließt. Oder das *Inventory Management*, worunter die Pflege der Dokumentationssysteme verstanden wird. Die genannten Funktionsbereiche können aber auch dem Konfigurationsmanagement zugeordnet werden.[4]

Eine weitere Thematik, die sowohl das Funktions- als auch das Organisationsmodell betrifft, ist die Verteilung der Managementfunktionalität. Darunter versteht man die Platzierung der zur Auswertung der gesammelten Managementdaten benötigten Intelligenz. Das herkömmliche Modell hierfür ist der zentralistische Ansatz. Dieser sieht vor, dass die Managementdaten von den Managementagenten zu den zentralen Managern geschickt und dort ausgewertet werden. Im Gegensatz dazu sieht der dezentrale Ansatz vor, dass die notwendige Intelligenz zu den Managementagenten geschickt wird.[5]

[4] Da eine umfassende Beschreibung der weiteren Managementfunktionen den Rahmen dieser Arbeit sprengen würden, sei an dieser Stelle auf [HeAN99] und die dort genannte Literatur verwiesen.

[5] Für eine ausführliche Beschreibung der Ansätze und ihrer Vor- und Nachteile wird z.B. [HE98] empfohlen.

Kapitel 4

Anforderungen an das AMETAS-Management

Im Mittelpunkt der Anforderungen an das AMETAS-Management steht die Entwicklung von Mechanismen, die die Lokalisierung von Agenten ermöglichen. In diesem Zusammenhang stellt sich die Frage, ob die dafür entwickelten Werkzeuge einen proprietären Charakter haben sollen oder auf der Basis einer offenen, interoperablen Managementplattform zur Verfügung gestellt werden können. Diese Arbeit orientiert sich an der zweitgenannten Vorgehensweise.

Eine Managementinfrastruktur für das Agentensystem AMETAS muss die Aspekte des Systems als ein Agentensystem und als eine Middelware-Infrastruktur für verteilte Anwendungen in Betracht ziehen. Hierfür sind zunächst die Managementaufgaben aus der Sicht vom AMETAS zu formulieren. Das hierdurch entstehende Szenario setzt Maßstäbe für ein Managementsystem. Im Folgenden werden vor dem Hintergrund von FCAPS die vorhandenen Mechanismen und die erwünschte Funktionalität analysiert.

4.1 Fehleranalyse

AMETAS wird in einer JVM ausgeführt, die ihrerseits auf einem Betriebssystem sitzt. Dementsprechend kann ein abnormales Verhalten vom AMETAS durch Fehler verursacht werden, die aus zwei unterschiedlichen Fehlerquellen stammen:

- Die Fehler, deren Wurzeln außerhalb vom AMETAS liegen: Sie beeinträchtigen entweder die Infrastruktur oder eine (Gruppe von) Systemkomponente(n). Zu diesen Fehler zählen z.B. Betriebssystem- und seine Ressourcenfehler oder Fehler, die in der JVM eintreten.
- Die Fehler, die im System liegen: In diesem Fall sind eine oder mehrere Komponenten fehlerhaft programmiert oder aber bestimmte Richtlinien z.B. bezüglich der Systemsicherheit nicht eingehalten worden. Zu dieser Gruppe gehören z.B. Fehler in AMETAS-Stellen, in PNS, in Diensten, in Benutzeradaptern und Agenten.

Der AMETAS-Kern verfügt über eingebaute interne Mechanismen, die alle Fehler in der ersten Kategorie und viele in der zweiten Kategorie feststellen und sichtbar machen können. Ziel des Fehlermanagements ist es, die erfassten Fehler aus den Log-Dateien des Systems herauszulesen, sie in Managementkonsolen geeignet darzustellen und gegebenenfalls geeignete Maßnahmen einzuleiten.

In erster Linie tragen die Programmierer der AMETAS-Komponenten dafür Sorge, dass die ins Leben gerufenen Komponenten die Systemrichtlinien einhalten und fehlerfrei funktionieren. Wenn die Dienste oder die Benutzeradapter trotzdem nicht wünschgemäß arbeiten, können sie mit den vorhandenen Systemmechanismen und den dafür entwickelten Werkzeugen entfernt werden, ohne dass die Funktionalität einer AMETAS-Stelle bzw. einer Gruppe der Stellen irgendwie beeinträchtigt wird. Kompliziert wird es aber, wenn die Agenten fehlerhaft sind. Sind die fehlerhaften Agenten stationär, ist die Vorgehensweise vergleichsweise weniger kompliziert. Die wichtige Frage ist aber: Was ist zu tun, wenn ein fehlerhafter Agent in einem Verbund von Stellen hin und her migriert? Dabei sind die folgenden Antworten von vornherein nicht adäquat:

- Die Beendigung eines (fehlerhaften) Agenten durch eine direkte Verbindungen mit dem Agenten. Laut AMETAS-Vorgaben können die Agenten nicht synchron angesprochen werden. Eine direkte Verbindung zu den Agenten hat nur die Stelle, die ihnen die Ausführungsumgebung zur Verfügung stellt.

- Finden der Stelle, wo sich der Agent aufhält, und Beendigung des Agenten. Die Stelle, wo sich der (fehlerhafte) Agent aufhält, ist unbestimmt, denn der Agent wandert ab und kündigt seine Zielstelle(n) nicht an, was es zusätzlich erschwert, den Agenten zu lokalisieren und zu terminieren.

- Die Beendigung der betroffenen AMETAS-Stellen ist keine gute Lösung, da noch weitere Anwendungen das System benötigen.

- Im Allgemeinen ist die Hinterlegung einer Beendigungsnachricht für den Agenten ebenfalls wirkungslos, denn nicht alle Agenten behandeln solche Nachrichten. Außerdem können Agenten solche Nachrichten ignorieren.

Diese Szenarien, eine mögliche Lösung des Problems und die Werkzeuge dazu werden ausführlich im Kapitel 9 besprochen. Es ist allerdings an dieser Stelle darauf hinzuweisen, dass die Bewältigung dieser Aufgaben im Verantwortungsbereich des Fehlermanagements liegen. Dazu müssen Mechanismen implementiert werden, die das Problem zufriedenstellend im Rahmen der Systemgegebenheiten lösen und die Lösungen den Managementkonsolen über Schnittstellen verfügbar machen.

4.2 Konfigurationsanalyse

Der aufwendigste Teil der Managementaufgaben im AMETAS betrifft die Konfiguration. Hier muss die Konfiguration aus organisatorischen, sicherheitsrelevanten und administrativen Gesichtpunkten betrachtet und auf geeignete Weise dokumentiert werden.

Auf der Basis der organisatorischen Sichtweise muss festgestellt werden, auf welchen Rechnern wie viele Stellen installiert werden müssen. Diese Feststellung ist von keiner dauerhaften Bedeutung, denn es können jederzeit neue Stellen installiert oder schon installierten Stellen wieder entfernt werden. Die Übersicht über die geographische Verteilung der Stellen liegt damit in der Verantwortung des Konfigurationsmanagements. In diesem Zusammenhang ist die organisatorische Verteilung der Dienste und Stellennutzer auf die Stellen ebenfalls zu berücksichtigen.

Berücksichtigt man die Sicherheitsaspekte, so muss man überlegen, welche Stellen zu einer Stellendomäne gehören, welche Stelle(n) gültige Zertifikate vergeben, welche Stelle der Domänenverwalter ist, wie die Migrationspolitiken festzulegen sind, wie die Privilegien und Berechtigungen zu setzen sind, welche Identitäten zu erzeugen sind, welche Berechtigungen und Privilegien sie erhalten, usw.

Aus administrativer Sicht ist festzulegen, wie die Stellen in einer administrativen Gruppe zusammengestellt werden können, wie ihre Daten administrativ zu verwalten , zu pflegen und auszuwerten sind, usw.

Der Vorgang der Konfiguration im AMETAS besteht aus den Aufgaben für die Erlangung eines lauffähigen Systems, aus den in AMETAS-Stellen notwendigen Aufgaben, sowie aus den Aufgaben, die für das Betreiben der Stellennutzer unerlässlich sind. Die Installation und Inbetriebnahme der AMETAS-Infrastruktur besteht aus den folgenden Teilaufgaben:

- Einrichten der für die AMETAS-Stellen notwendigen Verzeichnisse, Installieren der notwendigen Dateien und Setzen der System-Umgebungsvariablen
- Konfiguration und Starten eines PNS, der die symbolischen Namen der Stellen auf IP-Adressen abbildet
- Anlegen der Konfigurationsdateien der Stellen und der Diensten. Diese Datei beinhalten die einstellbaren Parameter einer Stelle und ihrer Dienste
- Erzeugen einer *Zertifizierungsautorität* (CA), die die Identitäten zertifiziert und ihre Vertrauenswürdigkeit bestätigt
- Erzeugen, zertifizieren und importieren der Stellen- und Benutzeridentitäten
- Definition der CA-Politik und Zuteilung der Privilegien.

Das Konfigurationsmanagement muss außerdem in der Lage sein, Änderungen an der Konfiguration vornehmen und die Konfigurationsdateien laden zu können. Die Stellennutzer müssen kompiliert und signiert werden, bevor sie in einer Stelle gestartet werden, wobei die signierten Stellennutzer und sämtliche von ihnen benötigten Dateien in einem SPU-Container verpackt werden. Zum Starten der Stellennutzer auf einer Stelle werden geeignete Werkzeuge benötigt, die bestimmte Befehle von externen Objekten entgegennehmen und geeignete Aktionen durchführen.

Zum Konfigurationsvorgang gehört weiterhin die Überwachung des PNS, der Stellen und Stellennutzer. In diesem Vorgang wird der Zustand des Systems hinsichtlich der Funktionalität der Systemkomponenten, ihrer Aufrechterhaltung, Systemmeldungen, usw. überwacht. Zur Überwachung gehört logischerweise eine geeignete Darstellung der zu überwachenden Ressourcen und (Log-)Dateien, was ein gute Beschreibung und Dokumentation der Konfiguration in jeder Hinsicht voraussetzt.

Für viele der aufgezählten Konfigurationsaufgaben im AMETAS gibt es bereit Funktionen und Werkzeuge, die den Umgang mit dieser Problematik erleichtern:

- Für die Aufgaben, die die Sicherheit des Agentensystems betreffen, wird das Hilfsprogramm ***SecAdmin*** das einen großen Funktionsumfang bietet. SecAdmin ist ein rein externes eigenständiges Java-Programm. Es ist skriptfähig, die Kommandos können unabhängig von ihren Vorgängern laufen, so dass die Konfigurationsaufgaben off-line durch sukzessive Aufrufe von SecAdmin durchgeführt werden können. Änderungen während der Laufzeit sind mit SecAdmin stets möglich und bisweilen auch notwendig. Es besteht weiterhin die Möglichkeit, SecAdmin als ein Java-Objekt aus einem Dienstprogramm heraus zu initiieren und die Konfigurationsaufgaben während der Laufzeit zu verrichten.[1]

- Mit der grafischen Schnittstelle *AMETAS Attachable Interface (***AMAI**) (siehe [AMTU]) kann ein Benutzer sich an einer Stelle seiner Wahl anmelden, wobei die Authentifizierung über eine Identität/Mantra-Kombination stattfindet. Man kann über AMAI die Stellennutzer starten, die Konfiguration der Stelle anpassen, ihre Aktionen mitverfolgen oder sie stoppen.

- Das *AMETAS Simple Interface* (**AMSI**) ist eine befehlszeilenorientierte Alternative zu AMAI. Da AMSI kein Fenstersystem benötigt, ist es unabhängig von den Systemgegebenheiten und verbraucht weniger Speicher.

- Das *AMETAS Attachable WWW Interface* (**AMAWI**) ist eine weitere AMAI-ähnliche grafische Schnittstelle, die im Rahmen dieser Arbeit entwickelt wurde. Sämtliche Funktionen von AMAI werden auch von AMAWI angeboten. Im Unterschied zum AMAI läuft

[1] In [AMAPI, AMTU] kann man sich einen ausführlichen Überblick über die vielfältige Funktionalität von SecAdmin verschaffen.

das AMAWI in einem applet-fähigen Web-Browser. Mit diesem kann durch einen sogenannten *WWWPort*, der in der Parameterdatei einer Stelle vorgegeben sein muss, die selbe angesprochen werden. Nach dem Austausch der digitalen Signatur, die von der Stelle an den Klienten geschickt wird, werden die Applets im Browser geladen und stehen dann zur Verfügung. Die Kommunikation erfolgt über HTTP und wird mit dem sogenannten *Transport Layer Security (TLS)*-Protokoll [RFC2246] gesichert, das der Nachfolger vom *Secure Socket Layer (SSL)*-Protokoll [SSLv3] ist[2]. Der große Vorteil vom AMAWI ist, dass es über einen applet-fähigen Browser funktioniert, der auf allen Plattformen vorhanden ist. Der Nachteil aber ist die Größe des Pakets, die die Leistung beeinträchtigt und schnellere Rechner mit großer Speicherkapazität erforderlich macht.[3]

- Der ***Agenten-Monitor*** ist ein Werkzeug zur Visualisierung und Überwachung von AMETAS-Anwendungen. Er dient insbesondere zur Beobachtung und zum Analysieren der Migrationstätigkeit mobiler AMETAS-Agenten.

Zieht man die genannten Werkzeuge in Betracht, so wird ersichtlich, dass die große Herausforderung im Konfigurationsbereich die Entwicklung von standardisierten und automatisierten Werkzeugen ist. Dabei sollen einerseits die bereits vorhandenen und unverzichtbaren Werkzeuge, wie z.B. SecAdmin, integrieren werden. Anderseits sollen Mechanismen zur Verfügung gestellt werden, die zusammen mit den möglichst offenen, plattformunabhängigen Schnittstellen die im Folgenden aufgeführten Punkte ermöglichen:

- Software-gesteuerte Erfassung der Konfigurationsdaten der Komponenten im AMETAS
- Speicherung und Pflege der Daten in einem standardisierten Format wie dem XML-Format
- Automatisierte Konfiguration der Komponenten mit Hilfe der gespeicherten Daten
- Weitere Änderungen in der Konfiguration der Komponenten auf persistente Daten abbilden.

Man muss davon ausgehen, dass das Konfigurationsmanagement in der Lage ist, die Stellen und Stellennutzer in der obengenannten Art und Weise zu konfigurieren, bei Bedarf z.B. bestimmte Dienste auf mehreren Stellen zu installieren, Identitäten auf gewünschte Stellen zu verteilen, usw.. Weiterhin müssen alle Funktionen offene Schnittstellen zu den Managementkonsolen anbieten.

Für die meisten der aufgezählten Konfigurationensaufgaben kann man die mobilen AMETAS-Agenten verwenden, die die Aufgaben effizient lösen können. Man kann überall dort, wo eine AMETAS-Infrastruktur vorhanden ist, von den Vorteilen des mobilen Agenten-Einsatzes Gebrauch machen.

[2] Auf SSL wird im Abschnitt 7.5.1 näher eingegangen.

[3] Die Abbildung A.1 veranschaulicht die AMAWI-Oberfläche.

4.3 Abrechnungsanalyse

Die Agenteninfrastrukturen und Dienste, die angeboten werden, erfordern einen Kostenausgleich. Bereits dann, wenn die Agenten eines Anwenders irgendwelche Stellen, die ihm nicht gehören, als Zwischenstationen nutzen und sich dabei auf die Sicherheit und weiteren Möglichkeiten der fremden Stelle verlassen, müssen sie dafür eine Gegenleistung erbringen.

Es ist die Aufgabe des Abrechnungsmanagements, ein geeignetes Abrechnungssystem zu entwickeln, das den Anforderungen des Agentensystems genügt. Es gibt in diesem Bereich im AMETAS offene Punkte, die eine umfassende Behandlung dieser Problematik erschweren. Insbesondere die folgenden Fragen sind im Rahmen der Abrechnungsanalyse noch zu klären:

- Für welche Ressourcen bzw. Leistungen entstehen welche Kosten? Vorstellbar sind z.B. Kosten für das Starten neuer Stellennutzer, Migrationskosten für die Agenten, Kosten für den Aufenthalt der Stellennutzer, insbesondere der Agenten, Kosten für die Nachrichtenübermittlung, usw.
- Wie werden Konten eröffnet?
- Wie wird die Kontenhöhe einer Agentenanwendung errechnet?
- Wie lässt sich ein Konto auffrischen?
- Was passiert mit den Anwendungen, deren Konto leer ist?

Die Liste dieser Fragen kann noch weiter ergänzt werden. AMETAS verfügt über ein integriertes Abrechnungssystem, das allerdings wegen der noch offenen Fragen deaktiviert wurde. Diese ergeben sich einerseits aus dem gegenwärtigen Forschungsstand, der weiter vorangetrieben werden muss. Andererseits aber gilt, dass das AMETAS bisher über keine kommerzielle Einsatzerfahrung verfügt, die die Beantwortung vieler der obengenannten Fragen erleichtern könnte.

4.4 Leistungsanalyse

Die Leistung im AMETAS bezieht sich auf die Leistung der Infrastruktur. Denn die Leistung der Agentenanwendungen ist zum einen Teil durch die Infrastruktur und zum anderen Teil durch die Programmierer bestimmt. Die Infrastruktur soll Mechanismen zur Verfügung stellen, die eine möglichst optimale Zuordnung der vorhandenen Ressourcen zu den Komponenten erlauben, so dass die Agentenanwendungen ihren Aufgaben nachkommen und dabei die erwartete Leistung erbringen können. In diesem Zusammenhang ist es beispielweise denkbar, eine Art Prioritätenkonzept einzuführen, durch das die nicht zeitkritischen Agentenanwendungen während der

Ausführung angehalten werden können, damit die zeitkritischen Anwendungen in einem nicht belasteten Zustand der Infrastruktur ausgeführt werden können.

Gleichzeitig muss man sich für den Fall einer Ressourcenknappheit in den Stellen über mögliche Auswege Gedanken machen, um eine (drastische) Leistungssenkung verhindern zu können. Die Auswegmaßnahmen können auch für eine Leistungssteigerung von Bedeutung sein. Um den Sachverhalt zu verdeutlichen, stellt man sich konkrete AMETAS-Stellen vor, die synchron von mehreren Stellennutzern als Ausführungsumgebung genutzt werden und dadurch deutlich belastet sind. Eine mögliche und durchaus interessante Vorgehensweise ist es, in einer solchen Situation eine Art Lastenausgleich durchzuführen, in dem man Stellen findet, die weniger belastet sind. Diesen überlässt man in der Belastungszeit einige Aufgaben. Oder man installiert neue Stellen und stellt sie der belasteten Stelle zur Verfügung, wodurch eine Leistungssenkung verhindert wird. Diese Technik bietet sich auch für gewisse Leistungssteigerungen im Rahmen der Infrastrukturgegebenheiten an.

Die leistungsbeeinflussenden Programmiertechniken sind Gegebenheiten der Programmiersprache. Das AMETAS und seine Anwendungen sind Java-basiert, weshalb die Leistungsanalyse von den Gegebenheiten der JVM abhängig ist. Die JVM nimmt die meisten Aufgaben in die Hand und lässt für mögliche Regulierungen wenig Raum. Abgesehen von den Maßnahmen, die die Leistung der JVM steigern und bereit in der JVM integriert wurden[4], gibt es einige Programmiertechniken, die in konkreten Anwendungsszenarien eingesetzt werden und die Leistung der Java-Programme beeinflussen. Einige dieser Techniken sind:[5]

- Einsetzen von Multithreading-Technik zur Verringerung des Zeitverbrauchs
- Verwendung der Thread-polling-Technik zur Verringerung des Ressourcen- und Zeitverbrauchs
- Verwendung nativer Threads zur Erhöhung der Schnelligkeit, wenn dieser unterstützt wird
- Vermeidung unnötiger Objekterzeugung
- Gemeinsame Benutzung der Konstanten von Klassen zur Reduzierung des Speicherverbrauchs
- Verwendung der Caching-Technik.

[4]Für Details Siehe: Java *Performance Enhancements* unter
http://java.sun.com/products/jdk/1.2/docs/guide/performance/index.html
http://java.sun.com/j2se/1.3/docs/guide/performance/index.html
http://java.sun.com/j2se/1.4/docs/guide/performance/index.html
Die Existenz der Seiten wurde zuletzt am 5.12.2001 nachvollzogen.

[5]Für detaillierte Information siehe u. a. [AcP200].

AMETAS hat bereits einige dieser Techniken integriert und in Form von Java-Paketen zur Verfügung gestellt, so dass die Agentenprogrammierer sie leicht verwenden können, ohne den genauen Hintergrund kennen zu müssen.

Als Beispiel kann man das Paket `AMETAS.util.threadpool` nennen. Ein *Threadpool* stellt eine begrenzte, möglicherweise variable Anzahl sogenannter *Worker-Threads* zur Verfügung, die eine große Anzahl von Aufträgen parallel bearbeiten können.

Nennenswert ist weiterhin das Paket `AMETAS.util.cache`. In einem Cache werden im Allgemeinen oft benötigte Daten auf einem schnellen Speichermedium bereit gehalten, um einen beschleunigten Zugriff auf sie zu ermöglichen. Das genannte Paket stellt dem Programmierer Klassen zur Verfügung, welche die Verwendung eines Caches in eigenen Stellennutzern oder außerhalb von AMETAS unterstützen.[6]

Neben der Entwicklung ähnlicher Techniken, die der Leistung im Agentensystem zugute kommen, ist es eine Herausforderung des Leistungsmanagements, das Konzept der Lastbalancierung zwischen den weniger belasteten und den überdurchschnittlich belasteten Stellen auf geeignete Art und Weise zu implementieren. Dabei ist das Problem durch folgende inhärenten Systemeigenschaften erschwert:

- Es ist ein Eingriff in die Agentenautonomie, einen Agenten gegen seinen Willen zu verlagern. Ohne Veränderung der Systemgegebenheiten wird es also nicht möglich sein, einen Agenten durch eigene Steuerung auf eine andere Stelle migrieren zu lassen.

- Das Agentensystem unterstützt nur eine sogenannte *schwache* Migration. Das bedeutet, es ist für einen Agenten nicht möglich, nachdem er seine Ausführung begonnen und gewisse Schritte in seinem Programmcode abgearbeitet hat, seine Ausführung abzubrechen und die restlichen Abläufe auf einer anderen Stelle weiterzumachen. Solch eine Vorgehensweise wäre im Rahmen einer sogenannten *starken* Migration möglich, die vom System aber nicht unterstützt wird.

Offene Fragen im Rahmen der Leistungsanalyse sind:

- Wie wird die Leistung im AMETAS gemessen, bzw. wo liegen die Leistungsgrenzen?

- Wie lassen sich die Ersatzstellen bestimmen?

- Sind redundante Stellen notwendig, um die Lastbalancierung zwischen den Stellen durchzuführen? Wo und wann werden sie installiert?

- Welche Komponenten lassen sich auf eine andere Stelle verlegen?

- Ist es möglich, eine vernünftige und effiziente Lösung für die Lastbalancierung zu finden, ohne den Systemkern modifizieren zu müssen?

[6] Weiterführende Informationen finden sich in [AMAPI, AMTU].

Diese Fragen sind im Rahmen des AMETAS-Management zu beantworten, zur Behandlung der zentralen Frage dieser Arbeit, nämlich der Lokalisierung von Agenten, können Sie an dieser Stelle zur Begrenzung des Umfangs der Arbeit zurückgestellt werden.

4.5 Sicherheitsanalyse

Die Gewährleistung von Sicherheit ist in einem Agentensystem wesentlich komplizierter als in einem konventionellen Softwaresystem, denn die autonomen und mobilen Agenten, die die menschlichen Anwender vertreten, wandern durch das Netz. Außerdem sind AMETAS-Komponenten Java-Programme, d.h. sie können alle Aktionen durchführen, die in einer JVM durchzuführen sind. Somit sind alle Komponenten des Agentensystems potentiellen Bedrohungen ausgesetzt, die in einer passiven oder aktiven Art versuchen, Informationen abzuhören oder auszuspionieren oder aber Schaden anzurichten. Im folgenden sind einige typische Sicherheitsbedrohungen im Agentensystem aufgelistet:

- Die Agenten einer nicht-autorisierten Personen können Zugang zum System erhalten und unerwünschte Aktionen durchführen
- Die Agenten können mit den vom Angreifer programmierten Agenten vertauscht werden, um dem Angreifer zu dienen, anstatt die vorgesehenen Ziele zu verfolgen
- Die Agenten können von Rechten Gebrauch machen, die ihnen nicht zu stehen
- Die Agenten der Angreifer können von den Stellen Rechte beantragen und bekommen, die ihnen nicht gewährt werden dürfen
- Die böswilligen Agenten können andere Agenten zu ihren Gunsten ausnutzen
- Die Angreifer können im Namen der autorisierten Personen ihre böswilligen Agenten losschicken
- Es ist möglich, dass die Angreifer eine unverdächtige Komponente installieren, die dann andere böswillige Komponenten und Dienste aufruft
- Die bösartigen Agenten können die Arbeit der anderen Agenten verhindern, so dass diese ihrer Aufgabe nicht mehr nachkommen können
- Die Agenten können attackiert werden, um ihre vertraulichen Daten und Informationen (wie Kreditkartennummern, Geheimnummern, usw.) zu lesen und eventuell zu missbrauchen
- Die Agenten aber auch die anderen Daten können beim Transportieren zwischen den Stellen oder sonst wo gelesen oder verändert werden

- Die Angreifer können ihre eigenen Stellen installieren, und die Agenten zu diese Stellen hinlocken. Dabei können sie die eingeladenen Agenten untersuchen und sie eventuell umprogrammieren.

Man kann diese Auflistung noch vervollständigen und weitere Sicherheitsaspekte hinzu addieren. Im Allgemeinen reichen die Sicherheitsvorkehrungen eines Betriebssystems nicht aus, um die bösartigen Prozesse von ihren Vorhaben abzuhalten. AMETAS verfügt bereits zur Eindämmung der oben genannten Risiken über ein Sicherheitssystem. Die wesentlichen Sicherheitsvorkehrungen betreffen die Authentifizierung und Autorisierung, die sichere Kommunikation zwischen den Stellen und mit den Stellen, und schließlich die Code-Integrität.[7]

Authentifizierung und Autorisierung

Die zentralen Punkte des Sicherheitskonzepts in diesem Bereich lassen sich folgendermaßen zusammenfassen:

1. Jedem Anwender und jeder Stelle wird eine **Identität** zugewiesen, die eine Instanz der Klasse `AMETASIdentity` ist. Jede Identität besitzt einen eindeutigen Identifikator der Klasse `AMETASIdentityID` (Identitäts-ID). Mittels der Identitäts-ID ist die Identität eindeutig gekennzeichnet (sie ist von `AMETASUniqueID` abgeleitet) und lässt sich in der Stellendatenbank auf einfache Weise wieder auffinden. Jede Identität besitzt genau eine solche ID, die automatisch erstellt wird und die sie für immer behält. Um die Vortäuschung einer Identität zu verhindern, werden die Identitäten von den Stellen zertifiziert , die als Zertifizierungsautoritäten (CA) fungieren.

2. Um die Autorisation zu sichern, werden die einzelnen **Berechtigungen** in sogenannten **Privilegien** zusammengefügt. Eine Berechtigung ist die kleinste Einheit in der Autorisierung einer Identität. Ein Privileg ist ein Container, der die Berechtigungen enthält, er kann aber auch alleine verwendet werden.

3. Da nur die Benutzer und nicht die Agenten die Privilegieninhaber sind, können nur die Benutzer diese von den Stellen erhalten. Dabei müssen die Benutzer in der Datenbank der Stelle vorhanden sein.

Für die Definition und Verwaltung diverser Sicherheitsmaßnahmen oder Richtlinien existieren eine Reihe von Politiken, die durch den `AMETASPolicyManager` verwaltet werden.

- **Identitätspolitiken** steuern die Authentifizierung und Autorisierung von Benutzern und Stellennutzern. Dabei entscheidet die Domänenzugriffspolitik (*DomainAccessPolicy*) anhand der Identität des Starters des Agenten, welche Privilegien dem Stellennutzer zuteil

[7] Detaillierte Informationen über das Sicherheitskonzept im AMETAS finden sich in [AMoA, AMoS, AMTU].

werden sollen, und die Authentifizierungspolitik (*AuthenticationPolicy*) befasst sich mit Fragen wie z.B.: Soll bei ankommenden Agenten eine Authentifizierung des Absenders und der Autoren durchgeführt werden? Oder: sollen Agenten nicht authentifizierter Absender bzw. Autoren zugelassen werden?

- **Stellennutzerpolitiken** betreffen Vorgänge, die mit Stellennutzern in Verbindung gebracht werden. Die Migrationspolitik (*MigrationPolicy*) legt fest, welche Sicherheitsmaßnahmen bei der Migration eines Agenten zu treffen sind. Die Delegationspolitik (*DelegationPolicy*) bestimmt, welche Teilmenge der Privilegien dem Stellennutzer tatsächlich zugeteilt werden sollen.

- **Stellenpolitiken** regeln die Rollen von Stellen im Agentensystem, wobei die Stellendomänenpolitik (*PlaceDomainPolicy*) bestimmt, aus welchen Stellen eine Domäne besteht. Eine der Stellen ist dabei der Domänenverwalter. Die Politik ist für die Migration von Agenten von Bedeutung. Gemeinsam mit der Migrationspolitik legt sie fest, ob ein Agent beispielsweise verschlüsselt oder im Klartext übertragen werden soll. Die Zertifizierungsautoritätspolitik (*CertificationAuthorityPolicy*) bestimmt, welche Stellen als Zertifizierungsautoritäten (CAs) akzeptiert werden.

Sichere Kommunikation

Die Kommunikation bezieht sich auf die externe Kommunikation, d.h. die Kommunikation zwischen den Stellen bzw. zwischen den Stellen und den externen Anwendungen wie z.B. AMAI. Um die Vertraulichkeit und auch die Integrität der Daten zu gewährleisten, werden die folgenden Verschlüsselungstechniken verwendet:

- die symmetrische Verschlüsselung, bei der Sender und Empfänger den selben Schlüssel verwenden

- die asymmetrische Verschlüsselungstechnik, bei der Sender und Empfänger ein zusammengehörendes Schlüsselpaar verwenden.

Für beide Techniken werden AMETAS-Implementierungen verwendet.

Codeintegrität

Die Codeintegrität beschäftigt sich mit der Erhaltung der Integrität des Codes, der ins Agentensystem geladen, ausgeführt und eventuell über das Netz zwischen den Stellen transportiert wird. Das im AMETAS verwendete Konzept ist das Signieren aller Stellennutzer. Dazu werden, wie schon angedeutet, die sogenannten SPU-Container verwendet. Prinzipiell sind die SPUs die einzigen Objekte, die als Stellennutzer geladen und gestartet werden können. Zwischen Stellen verschickte Pakete, die migrierende Agenten enthalten, sind ebenfalls SPUs.

Ein SPU-Container kann neben den Klassen, die zu einem Stellennutzer gehören, weitere Agenten beinhalten. Klassen, die nicht von einem Agenten mitgenommen werden, die aber in eine bestehende Anwendung dynamisch eingegliedert werden sollen, können in Form von *SignedClass*(SCC)-Containern angeboten werden. Sie müssen ebenfalls eine gültige Signatur aufweisen.

Fazit

Das AMETAS verfügt über ein Sicherheitskonzept, das trotz seiner Kompliziertheit sehr zufriedenstellende Lösungen für die meisten sicherheitsrelevanten Probleme bietet. Als Werkzeug für die Konfiguration des Sicherheitssystems dient das eigenständige Programm **SecAdmin**, das bereits bei der Behandlung des Konfigurationsmanagements vorgestellt wurde.

Ein Nachteil des implementierten Sicherheitssystems ist aber, dass alle entwickelten und implementierten Klassen und Werkzeuge nur in der AMETAS-Umgebung von Bedeutung sind. Ein AMETAS-Agent, der im AMETAS signiert wurde, kann sich in einem anderen Agentensystem, das den Provider ***AMETAS*** nicht kennt, nicht ausweisen, weshalb er in solch einer Umgebung bestenfalls ein unbekanntes Wesen ist. Die vom AMETAS ausgestellten Zertifikate sind nur im AMETAS gültig. Außerdem können die weit verbreiteten, standardisierten, digitalen Zertifikate im X.509-Format im AMETAS nicht verwendet werden.

Diese Nachteile lassen sich dadurch beheben, dass man die Implementierung der Konzepte um standardisierte Zertifikate erweitert. Eine derartige Standardisierung ermöglicht es, mit Werkzeugen wie z.B. `keytool` von Java Zertifikate zu erzeugen, sie bei einer weltbekannten Zertifikatsautorität signieren zu lassen. Diese können nicht nur im AMETAS zum Signieren der Stellennutzer sondern auch für andere Zweck wie zum Beispiel im WWW verwendet werden.

Sind die grundlegenden Funktionen für das Sicherheitsmanagement vorhanden, muss das AMETAS-Management geeignete Schnittstellen durch die Management-Konsole zur Verfügung stellen, über die die sicherheitsrelevanten Aufgaben effektiv und benutzerfreundlich erledigt werden können.

4.6 Anforderungen an die Managementplattform

Die Umsetzung der im vorherigen Abschnitt aufgezählten Managementaufgaben im AMETAS benötigt eine geeignete Managementarchitektur. Für die Formulierung allgemeiner Anforderungen orientiert man sich an einem Referenzmodell, dem sogenannten *Reference Model of Open Distributed Processing* (RM-ODP), das in [ISO 10746-x] festgelegt wurde. Es handelt sich dabei um einen Standard, der ein Rahmenwerk für allgemeine verteilte Anwendungen in heterogen Umgebungen darstellt. Mit Hilfe von RM-ODP wird der Einsatz von objektorientierten Technologien ermöglicht.

Hier werden einige aus der Sicht des Autors wichtige Eigenschaften erwähnt, die für die im AMETAS eingesetzte Managementplattform wesentlich sind:

- **Offenheit** bezeichnet das Vorhandensein von einer standardisierten Notation zur Definition von Komponentenschnittstellen. Dadurch wird die Kooperation der Managementkomponenten offengelegt.
- **Portabilität** verleiht der Plattform die Fähigkeit zur Ausführbarkeit auf unterschiedlichen Systemen ohne die Vornahme von Modifikationen
- **Integration** definiert die Fähigkeit, verschiedenartige Ressourcen unabhängig von ihrer architekturellen Herkunft oder ihrer Leistungsfähigkeit unter ein Managementdach zu bringen.
- **Flexibilität** bezeichnet die Fähigkeit der Anpassung an Veränderungen in der Außenbeziehung eines Managementsystems. Dies schließt die Existenz und den Betrieb bereits existierender älterer Systeme mit ein.
- **Modularität** beschreibt die strukturelle Aufteilung eines Systems in autonome Komponenten, die logisch zusammengehören. Sie ist eine Basis für die Erweiterbarkeit der Plattform.
- **Föderation** ist die Kombination von Systemen aus unterschiedlichen technischen oder administrativen Domänen zum Erreichen eines gemeinsamen Ziels. Dies schließt insbesondere die Kooperation bestehender Dienste mit dem Ziel der Erbringung eines neuen, hochwertigen Dienstes ein.
- **Sicherheit** bezeichnet die Gesamtheit aller Mechanismen zum Schutz von Systemressourcen und Daten vor unberechtigtem Zugriff: Diese umfassen unter anderem Zugangskontrolle, Protokollierung, Authentifizierung, Integrität, Schlüsselverwaltung.

Zusätzlich zu den allgemeinen Forderungen an verteilte Anwendungen in einer heterogenen Umgebung muss eine Managementplattform weitere Eigenschaften besitzen. Diese können folgendermaßen formuliert werden:

Unterstützung des dezentralistischen Managementansatzes

Es handelt sich dabei um einen Mechanismus, der die Verarbeitung der Managementdaten dezentral unterstützt. Viele der Managementdaten werden also nicht mehr über das Netz übertragen, sondern sie werden ausgewertet, wo sie gesammelt sind. Dafür geht die auswertende Software zu den Daten. Für die Dezentralisierung des Managements sind im Entwicklungsprozess der Managementstrategien einige Technologien entstanden. Nennenswert ist z.B. *Management*

by Delegation. In diesem Ansatz werden die Funktionen des Managers auf die Managementagenten verlagert. Dabei gibt verschiedene Arten der Delegation.

Ein weiterer Ansatz ist der Einsatz von mobilen Agenten zur Dezentralisierung. Dabei werden die Eigenschaften der mobilen Agenten ausgenutzt, um die Managementaufgaben dezentral zu verarbeiten, was eine geringere Netzlast und höhere Ausfallsicherheit mit sich bringt.

Im Rahmen dieser Arbeit werden mobile Agenten für Managementaufgaben eingesetzt, wobei sie in erster Linie die Agentenarchitektur für ihren Einsatz verwenden. Deshalb wird eine Managementarchitektur benötigt, die den Einsatz von intelligenten Managementagenten unterstützt, die selbstständig viele Managementaufgaben bearbeiten können.

Unterstützung des dynamischen Managements

Die Dynamik im Agentensystem erfordert eine Managementarchitektur, die diese Dynamik erfassen kann. Dazu muss die Managementarchitektur die Instrumentierung der dynamischen Komponenten ermöglichen und eine für ihr Management ausreichende Funktionalität zur Verfügung stellen.

Interoperabilität

Obwohl die oben genannte Offenheit impliziert, dass die Managementarchitektur für AMETAS standardisierte Schnittstellen besitzen soll, muss man ihre Interoperabilität mit den gängigen Managementstandards betonen. Man darf heutzutage nicht erwarten, dass in einer heterogen Umgebung bereits eine einheitliche Managementlösung existiert.

Bessere Integration im Agentensystem

AMETAS ist eine Java-basierte Plattform, d.h., sowohl die Infrastruktur als auch die Anwendungen sind Java-Programme. Eine Java-basierte Managementinfrastruktur kann daher ohne große Hindernisse in das System integriert werden. Dagegen können Managementlösungen, die in einer anderen Programmiersprache entwickelt wurden, viele Schwierigkeiten bereiten. Insbesondere wenn diese Lösungen plattformabhängig sind, ist ihr Einsatz im AMETAS unter Umständen mit einem großen Aufwand verbunden.

Kapitel 5

Managementstandards

Im folgenden Abschnitt werden einige der bereits standardisierten Managementarchitekturen so weit vorgestellt, dass eine Bewertung hinsichtlich ihrer Eignung für das Anwendungs- und Infrastrukturmanagement möglich ist. Dabei wird aber auf die Vorstellung des OSI- und Internet-Managements verzichtet. Die Gründe dafür sind:

Obwohl die OSI-Managementarchitektur alle Teilmodelle einer Managementarchitektur ausprägt und eine Art Referenzarchitektur darstellt, spielt sie wegen ihrer Kompliziertheit im Umfeld der Datenkommunikation keine große Rolle. Sie wird in erster Linie in Telekommunikationsnetzwerken eingesetzt und ist eine Basis für das *Telecommunication Management Network (TMN)*.[1]

Das Internet-Management ist ebenfalls netzorientiert, aber wegen der Nicht-Objektorientiertheit seines Informationsmodells ist die Instrumentierung der Objekte im Objektorientierungssinne sehr schwierig. Außerdem ist der Sicherheitsstandard seines Kommunikationsmodells (SNMP) relativ gering.[2]

5.1 Object Management Architecture (OMA)

Die *Object Management Architecture (OMA)* [OMAG] wurde 1992 von der *Object Management Group (OMG)* [OMG] veröffentlicht. Das primäre Ziel des OMG-Ansatzes ist die Förderung der globalen, orttransparenten Interoperabilität allgemeiner verteilter Anwendungen, wobei objektorientierte Techniken angewendet werden. Der OMG-Ansatz richtet sich an alle verteilten Anwendungen, und damit hat er für das effiziente und kostengünstige Management von Endsystemen und Anwendungen, die auf OMG-Technologie setzen, ebenfalls an Bedeutung gewonnen.

[1] Detaillierte Informationen über OSI findet man im Management Framework [ISO 7498-4] [RO90], [STA93] und [HeAN99].

[2] Weiterführende Informationsquellen für Internet-Management und seine Teilmodelle sind z.B. [RO91], [HeAN99], [STA93] und [KELL98].

Der Grund dafür ist, dass man eine einzige Architektur und Infrastruktur für die Entwicklung, den Einsatz und das Management einer verteilten Anwendung einsetzen kann. Weil sich die Entwicklung von verteilten Anwendungen und ihren Managementanwendungen in der Entwicklungsphase nicht unterscheidet, kann man die Nutz- und Managementdaten mit den gleichen Mitteln modellieren und teilweise die gleichen Werkzeuge einsetzen. Eine derart entwickelte verteilte Anwendung verfügt somit über zwei Schnittstellen, die in einer einheitlichen Syntax spezifiziert wurden:

- die Dienstschnittstelle stellt den anderen Modulen die eigentliche Dienstfunktionalität zur Verfügung
- die Managementschnittstelle macht die für das Management erforderlichen Dienste für die Managementanwendungen zugreifbar.

Der Übertragungsmechanismus ist für die Nutz- und Managementdaten in dieser Architektur ebenfalls gleich, was bedeutet, dass man ein Kommunikationssystem schaffen, installieren und pflegen muss.

Die OMA besteht aus mehreren Teilarchitekturen und Spezifikationen. Den Kern bildet die *Common Object Request Broker Architecture (CORBA)*, die bis auf das Funktionsmodell alle Teilmodelle einer Managementarchitektur festlegt. CORBA ist in der Fachwelt so weit verbreitet, dass man statt von einer OMA oft von CORBA-basiertem Management spricht.

Zum Referenzmodell gehört neben den Objektschnittstellen, die im Funktionsmodell der OMA behandelt werden, ein sogenannter *Object Request Broker (ORB)*. Dieser ist eine Infrastrukturkomponente, die die Interaktion der Klient/Server-Anwendungen ortstransparent ermöglicht. Dabei hat der Broker die Fähigkeit, denjenigen Server auszuwählen, der die Aufgabe des Klienten am besten erfüllen kann. Im Prinzip trennt er die vermittelbaren Schnittstellen von der Implementierung des Servers, wodurch Änderungen der Anzahl und die Implementierung der Server dem Klienten verborgen bleiben.

Die Abbildung 5.1 zeigt die OMA als Managementarchitektur.

5.1.1 Organisationsmodell

Das Organisationsmodell der OMA basiert auf einem symmetrischen *Peer to Peer*-Ansatz. Im Gegensatz zum hierarchischen Modell des Internet-Managements geht man von einer Kooperation zwischen prinzipiell gleichberechtigten Objekten aus. Dieser Ansatz ist so allgemein, dass nicht nur Manager/Agent- und Manager/Manager-Beziehungen möglich sind, sondern auch Agent/Agent-Beziehungen[3]. Man kann hiermit nicht nur die Funktionalität an hierarchisch gleichwertige Managementsysteme verteilen oder an hierarchisch niedrigere Systeme delegieren lassen, sondern auch autonome Kooperation zwischen Managementagenten zulassen. Die

[3]Diese Aussage gilt, sofern es sich hierbei um Management- und nicht um Software-Agenten handelt.

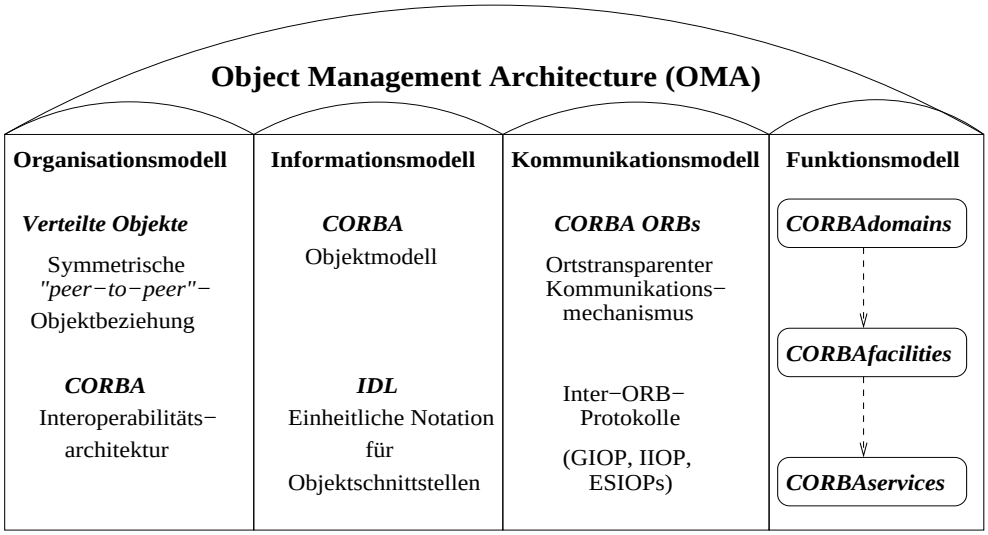


Abbildung 5.1: OMA als Managementarchitektur

Akteure sind somit Objekte, die je nach Anforderung die Klient- oder Server-Rolle übernehmen.

Dieses Konzept wurde von CORBA 2 um ein Domänenkonzept erweitert, um den Zusammenschluss unabhängig voneinander entwickelter ORBs auf der technischen Interoperabilitätsbasis zu ermöglichen. Die Mitglieder einer Domäne sind Objekte, die bestimmte gemeinsame Eigenschaften haben. Domänen können als Objekte modelliert werden und somit wiederum Mitglieder von Domänen sein.

5.1.2 Informationsmodell

Das Informationsmodell der OMA basiert auf einem objektorientierten Ansatz. Im Gegensatz zum OSI- oder Internet-Management werden aber nicht die Managementobjekte sondern die grundlegenden Eigenschaften allgemeiner Objekte definiert. Der Grund hierfür ist, dass OMA nicht speziell auf das Netz- oder Systemmanagement ausgerichtet ist, sondern prinzipiell alle verteilten Anwendungen unterstützen soll.

Das Kern-Objektmodell (KO) (engl. *Core Object Model*) der OMA stellt eine Grundlage dar, auf der ein portabler Entwurf verteilter Anwendungen möglich ist. Das KO definiert Objektbegriff, Objektschnittstellen und Vererbung. Es bildet die konzeptionelle Basis für die Erweiterung um sogenannte *Komponenten*.

Zur Beschreibung der Aufrufschnittstellen wird die Notation *Interface Definition Language* (*IDL*) eingeführt. IDL ist programmiersprachenunabhängig. Sie wird auf die meisten gängigen Programmiersprachen abgebildet.

5.1.3 Kommunikationsmodell

Objekte kommunizieren miteinander über einen ORB. Dabei können sie an der Schnittstelle zum ORB Operationen auf anderen Objekten aufrufen und Ergebnisse erhalten. Der ORB nimmt die Anforderungen des Klienten entgegen, leitet sie an einen geeigneten Server weiter und stellt dem Klienten das Ergebnis nach der Beendigung der Operation zu. CORBA spezifiziert auf der Klienten- und Serverseite unterstützende Komponenten und Festlegungen, um diese Kommunikationsform zu ermöglichen.

Bei einer Kommunikation zwischen ORBs verschiedener Hersteller kommen drei Protokolle zum Einsatz:

- Das *General Inter-ORB Protocol (GIOP)*, das die Syntax und Semantik der zwischen den ORBs auszutauschenden Nachrichten festlegt, wobei es auf ein beliebiges verbindungsorientiertes Transportprotokoll zugreift. GIOP legt außerdem ein Format fest, in dem zu einer Objektreferenz auch die dazugehörige ORB-Domäne und das Protokoll übermittelt werden können.

- Das *Internet Inter-ORB Protocol (IIOP)*, das das GIOP auf TCP abbildet und umgekehrt. Mit IIOP ist also eine uneingeschränkte Interoperabilität aller ORBs gegeben.

- Die *Environment-Specific Inter-ORB Protocols (ESIOPs)* werden für die Kommunikation der ORBs über andere Protokolle bzw. Protokollstapel verwendet. Als Beispiel ist das DCE-CIOP[4] zu nennen, dessen Ziel die Festlegung der Kommunikation zwischen den ORBs über den DCE-RPC war.

5.1.4 Funktionsmodell

Das Funktionsmodell der OMA wird aus drei Kategorien[5] von Objektschnittstellen gebildet, wobei die Dienste der oberen Kategorien sich mit Hilfe der objektorientierten Techniken auf die darunter liegenden Dienste abstützen, und die Grenzen zwischen den Kategorien teilweise fließen sind.

- Die Spezifikationen für *CORBAservices*[6] liefern die elementare Funktionalität und die Systemdienste, um das System und die Objekte in einer verteilten Umgebung effektiv nutzen zu können. Die Dienste dieser Kategorie erweitern den Funktionsumfang der ORBs um notwendige und nützliche Funktionen. Diese Dienste sind in keiner Weise managementspezifisch, sie sind aber für Managementanwendungen notwendig oder nutz-

[4] Englisch: Distributed Computing Environment-Common Inter-ORB Protocol

[5] Das Referenzmodell der OMA stellt eine weitere Objektschnittstelle zu den *Application Objects* zur Verfügung, die die eigentlichen Anwendungen sind, aber keiner Standardisierung unterliegen.

[6] In der [OMAG] werden diese Dienste *Object Services* genannt.

bringend einsetzbar. Dienste zur Instantiierung von Objekten, Namensgebung, zur dauerhaften Speicherung der Objekte, zum Empfang und Versand von Ereignismeldungen, Lizenzierung, usw. sind Beispiele für *CORBAservices.*

- Die *CORBAfacilities*[7] bilden die zweite Schicht des Funktionsmodells. Diese Spezifikation stellt universell verwendbar Dienste für alle Anwendungen zur Verfügung. Hierzu zählen z.B. Benutzerschnittstellen, Dienste zum Informationsmanagement, zum Systemmanagement etc. Die *Mobile Agent Facility Specification*, die eine Standardisierung der Interoperabilität von verschiedenen Agentensystemen hinsichtlich des Agentenmanagements, des Agententransfers, usw. zum Ziel hat, zählt ebenfalls zu dieser Kategorie.

- *Domain Interface* sind Dienste, die auf einen speziellen Anwendungsbereich wie z.B. E-Commerce, Finanzwelt, Gesundheitswesen, usw. zugeschnitten sind.

5.1.5 Bewertung

CORBA spielt eine führende Rolle als Middelware-Architekur und hat dort die DCE weitgehend abgelöst. Neben ihrem Hauptanwendungsgebiet als Architektur für verteilte objektorientierte Programmierung ist sie zum effizienten und kostengünstigen Management von Endsystemen und Anwendungen, die auf OMG-Technologie setzen, gut geeignet. Man kann, auf OMA basiert, verteilte Anwendungen entwickeln, deren Management ein integraler Bestandteil der Anwendung ist. Dieses Management basiert auf dem gleichen Kommunikationssystem wie die Anwendung selbst. Die Tatsache, dass CORBA auch für integriertes Management einsetzbar ist (siehe [HeAN99, KELL98]), ist ein weiterer Punkt, der für CORBA spricht.

Die Abbildung von IDL auf die Implementierungssprache ist einzigartig und hat keine Parallele in den anderen Managementarchitekturen. Insbesondere hat der Anschluss von IDL an Java und die CORBA-Unterstützung in der Version 1.2 von Java Effekte gezeigt.

Der größte Mangel von CORBA sind die vordefinierten Managementinformationen, die in großer Zahl bei OSI- oder Internet-MIBs vorliegen. Eine mögliche Lösung wäre es, die genannten MIBs mit Hilfe standardisierter Übersetzungsverfahren im CORBA-Umfeld nutzbar zu machen.

Die relativ komplexe Architektur von CORBA ist ein weiterer Nachteil. Insbesondere im Vergleich mit der Internet-Architektur ist OMA aufwendig zu realisieren, was sich hinderlich auf die Einführung preiswerter Komponenten auswirken kann.

Zwei weitere Schwächen vom CORBA sind im Rahmen dieser Arbeit besonders relevant: Der eine Punkt ist der Mangel an managementspezifischen Diensten. Obwohl einige managementspezifische Dienste in CORBA bereits spezifiziert und standardisiert sind, hat CORBA

[7] [OMAG] nennt diese Dienste *Common Facilities.*

als Managementarchitektur nicht im Entferntesten den Funktionsumfang, den z.B. die OSI-Managementarchitektur oder die JMX, die in 6 besprochenen wird, bieten.

Der andere und entscheidende Punkt ist der zugrunde liegende Kommunikationsmechanismus in CORBA. Die Interoperabilität der Objekte, die statisch in einem Computer in der verteilten Umgebung installiert sind, wird durch die RPC ermöglicht, wodurch die Verteiltheit verborgen bleibt. In diesem Modell müssen die Daten zur Auswertung im Netz hin- und hergeschickt werden, was einen erheblichen Netzverkehr verursacht. Außerdem ist eine beständige Netzverbindung mindestens im Laufe der Datenverarbeitung unverzichtbar. Diese Schwäche von RPC-basierter Middleware ist eine Motivation für die Entwicklung des mobile Agentenparadigmas, das einen neuen Ansatz zur Entwicklung verteilter Anwendungen darstellt. Ausgehend von der Feststellung dieser Mängel wurde in [LISA98] ein Ansatz vorgestellt, der die mobile Agenten-Technologie für das Management der CORBA-basierten Systeme anwendet.

5.2 Desktop Management Interface

Das *Desktop Management Interface (DMI)* [DMIS] ist eine Managementarchitektur, deren Ziel die Einbindung heterogener Arbeitsrechner mit all ihren Komponenten in ein integriertes Management ist. Mit anderen Worten schlägt DMI eine Brücke zwischen der betriebssystemspezifischen Hard- und Software in den Rechnern (PCs und Unix-Workstations) und den standardisierten Managementarchitekturen. DMI wurde 1994 von der *Desktop Management Task Force (DMTF)*[8] veröffentlicht, die 1992 als eine Vereinigung von Hard- und Software-Herstellern gegründet wurde. Die zweite und um den entfernten Zugriff auf die Managementinstanz und die Interoperabilität mit dem Internet-Management erweiterte Version wurde 1996 freigegeben.

Das zentrale Element einer DMI-konformen Architektur ist der sogenannte *Service Provider (SP)*, der ein Bindeglied zwischen den zu managenden Ressourcen eines Rechners und den Managementanwendungen ist. Diese Funktionalität hat er mit einem SNMP-Hauptagenten gemeinsam. Die Schnittstellen zu den Managementanwendungen werden als *Management Interface (MI)* bezeichnet, über die der SP die Anfragen und Aufträge von lokalen und entfernten Anwendungen entgegennimmt oder Ereignismeldungen der zu managenden Komponenten an ihre Empfänger ausliefert. Die Schnittstelle zu den Ressourcen bezeichnet man als *Component Interface (CI)*, deren Realisierung betriebssystemspezifisch ist. Der SP bearbeitet die Aufträge entweder selbst oder leitet sie über das CI an die Ressourcen weiter.

Das Funktionsmodell von DMI besteht nur aus den Grundfunktionen, die die standardisierten Schnittstellen auf der SP-Seite und dementsprechend auf der Komponenten- und Managementanwendungsseite implementieren. Nur das Ereignismanagement ist aufgrund seiner Filtermöglichkeiten und Registrierungsmöglichkeiten zum Empfang der Ereignismeldungen weit genug entwickelt.

[8] Der Name wurde inzwischen in *Distributed Management Task Force (DMTF)* geändert.

5.2.1 Informationsmodell

Das Informationsmodell des DMI heißt *Management Information Format (MIF)*, das nicht objektorientiert ist und dem Informationsmodell des Internet-Managements ähnlich ist. Im Unterschied zur *Structure of Management Information* im Internet-Management aber enthält die MIF neben den üblichen Definitionen der Eigenschaften von Komponenten auch Festlegungen zur technischen Realisierung der Kooperation der Komponenten mit dem SP. Es wird z.B. festgelegt, mit welchen Funktionen die Werte der Attribute auf einem Betriebssystem zur Laufzeit des SP gelesen oder modifiziert werden können.

Eine Komponente wird als eine Liste von Attributsgruppen gefolgt von einer Liste von Tabellen in einer MIF-Datei beschrieben und dem SP zur Registrierung übergeben. Die Attributsgruppen beinhalten ihrerseits die Definition der Attribute, und jede Tabelle enthält die Instanzen einer Attributsgruppe, wobei die Instanzen die Zeilen einer jeder Tabelle bilden. Die [DMIS] (Kapitel 3, S. 41) definiert drei Standardgruppen:

1. Die Attributsgruppe *ComponentID*, die eine minimale Menge von Informationen über die betroffene Komponente enthält. Diese Attributsgruppe muss in jeder MIF-Datei enthalten sein.

2. Die Gruppe *Event* ist ein Muster, das das Format für Daten der standardisierten Ereignisse beschreibt.

3. Die Attribute in der Gruppe *SP* müssen bei der Implementierung eines jeden DMI-SP berücksichtigt werden.

Zusätzlich werden zu den wichtigen Komponenten der Arbeitsplatzrechner und den an sie angeschlossenen Geräten wie z.B. Prozessoren, Festplatten, Bildschirme oder Softwarepakete konkrete Attributsgruppen von den Arbeitsgruppen des DMTF-Standards gebildet. Diese Attributsgruppen werden von den Herstellern der Komponenten verwendet, und ihre MIF-Dateien mit Werten vorbelegt.

5.2.2 Kommunikationsmodell

Die Kommunikation zwischen den Managementanwendungen und dem SP basiert auf RPC. Es gibt eine Abbildung der RPC auf das SNMP, die eine erhöhte Flexibilität bietet und die Integration in das SNMP-Management gestattet. Die Kommunikation mit dem CI basiert auf prozeduralen Definitionen, wobei die Komponenten Operationen wie Lesen und Schreiben der Attribute, Manipulation von Tabellen, usw. zur Verfügung stellen. Der SP implementiert Operationen zum Registrieren der Komponenten und Mechanismen zur Ereignismeldung seitens der Komponenten.

5.2.3 Bewertung

DMI ist speziell auf Arbeitsplatzrechner und PCs ausgerichtet, und ist in diesem Bereich ohne Konkurrenz. Die Situation stellt sich so dar, dass das Internet-Management und das DMI die Komponenten des LANs teilen: Die Netzkomponenten werden mit Internet-MIBs und SNMP und die Endsysteme mit DMTF-MIFs, SNMP und RPCs gemanagt.

Außerdem erlauben die vorgesehenen Mechanismen die Integration und Koordination mehrerer Management-Subagenten, so dass sie als Ressourcen eines Systeme registriert werden können. Das führt zu ausdifferenzierten hierarchischen Organisationsformen.

Der Nachteil dieses Informationsmodells ist, dass es nicht objektorientiert ist und die Vorzüge der Objektorientiertheit daher nicht genutzt werden können. Insbesondere erschwert die Verwendung von Tabellen im Informationsmodell die Pflege und Konsistenzhaltung der Daten.

Außer dem Ereignismanagement hat DMI keine weitere Managementfunktionalität definiert, was in anderen bekannten Managementarchitekturen wie z.B. im OSI-Management der Fall ist.

Das Kommunikationsmodell schreibt die RPC für die Kommunikation zwischen den Managementanwendungen und dem SP vor. Deshalb leidet das DMI unter allen Nachteilen, die für alle RPC-basierten verteilten Systeme charakteristisch sind. Die Tatsache, dass die Implementierung von CI optional ist und vom Hersteller geeignet und proprietär realisiert werden kann, spricht ebenfalls gegen die Zukunft von DMI. In diesem Zusammenhang ist die Haltung von Microsoft zur DMTF und zum DMI nicht klar. Z.B. stützt sich das von Microsoft implementierte CI auf die universelle Microsoft-Treiberschnittstelle und verhindert somit die Standardisierung des CI, was für die Zukunftschancen des DMI nachteilig ist.

Die DMTF fördert das WBEM, so dass das DMI in den Schatten gestellt wurde. Es zeichnet sich ab, dass das Informationsmodell vom WBEM, das der DMTF übertragen wurde, mittelfristig der Standard für ein Informationsmodell ist.

5.3 Web-based Enterprise Management (WBEM)

Die *Web-based Enterprise Management (WBEM) Initiative* wurde von einem Herstellerkonsortium bestehend aus *Microsoft*, *Intel*, *Cisco Systems*, *Compaq*, *BMC Software* und später auch *Sun Microsystems* sowie *Hewlett Packard* im Jahr 1996 gegründet.

Das Ziel vom WBEM lautet, eine offene, herstellerübergreifende Architektur zum Web-basierten Management der gesamten DV-Infrastruktur eines Unternehmens zu entwickeln, die zur Administration von Netzwerkkomponenten, Endsystemen und Anwendungen dient. Das zweite Ziel von WBEM ist die Integration von anderen Managementarchitekturen wie dem DMI, dem SNMP-Management und dem OSI-Management.

Zunächst wurde das Informationsmodell, das sogenannte *Common Information Model (CIM)*[9], und später das komplette WBEM an DMTF übertragen, was zu wesentlichen Änderungen in der Architektur insbesondere im Kommunikationsbereich führte.

Ein WBEM-konformes Produkte besteht aus einem *CIM Object Manager (CIMOM)*, der Schnittstellen zu den Managementanwendungen bietet, über die Anfragen der Klienten entgegengenommen werden, und über die wiederum die erzeugten Ereignisse der Ressourcen zu den Anwendungen weiterleitet werden können. Außerdem hat CIMOM die Aufgabe, die CIM-Daten in ein sogenanntes CIM-*Repository*, das ein weiteres Bestandteil eines WBEM-konformen Produkts ist, zu speichern und sie bei Bedarf ausfindig zu machen. Als Anwendungsschnittstelle zum CIMOM fungieren die Web-Browser, in denen die Managementanwendungen ablaufen. CIMOM bietet weiterhin Schnittstellen zu den *Providern*, die als Bindeglied zwischen dem CIMOM und den zu managenden Objekten fungieren, die in einem anderen Informationsmodell instrumentiert sind. Es gibt Provider für XML, SNMP, DMI und *Windows Registry*. Es zeigt sich, dass das Organisationsmodell der WBEM-Architektur dem Internet-Management ähnlich ist.

Das Funktionsmodell der WBEM-Architektur sieht zur Zeit nur Mechanismen des Ereignismanagements vor wie z.B. das Abonnieren und Filtern von Ergebnissen. Im Kommunikationsbereich ist man von dem ursprünglichen Kommunikationsprotokoll, dem sogenannte *Hypermedia Management Protocol (HMMP)*, abgekommen und favorisiert stattdessen das HTTP, mit dem in XML geschriebene Daten transportiert werden können. Im Windows-Betriebssystem setzt man auch plattformabhängige Technologien wie z.B. AktiveX ein.

5.3.1 Informationsmodell

Das Informationsmodell der WBEM-Architektur CIM ermöglicht einen uneingeschränkten, verlustfreien Austausch von Informationen. Dies ist eine zentrale Anforderung zur Integrationen und Kooperationen der Managementplattformen. Mit dem CIM ist es also möglich, die existierenden Modelle mit minimalem Informationsverlust aufeinander abzubilden. Andererseits stellt CIM eine sehr gute Alternative zu den datentyporientierten Informationsmodellen des SNMP-Managements und auch des DMI dar: Der Ansatz vom CIM basiert auf der Objektorientiertheit und Programmiersprachenunabhängigkeit, weshalb DMDF-MIF das Informationsmodell des DMI durch CIM ersetzt hat.

Drei Eigenschaften des CIM, die in [CIMS] definiert sind, machen es für die Präsentation der Managementinformationen interessant:

[9] CIM wurde früher als *Hypermedia Management Scheme (HMMS)* bezeichnet.

Das *Meta-Schema*

Die formale Definition des Modells wird mit Hilfe eines Meta-Schemas dargestellt, das die Ausdrucksformen des Modells, seine Verwendung und seine Semantik festlegt. Die Struktur des Meta-Schemas wird mit Hilfe der *Unified Modeling Language (UML)* definiert, die eine standardisierte Notation zur Spezifikation, Visualisierung, Konstruktion und Dokumentation der Softwaresysteme ist (siehe [**?**, UMLS]).

Das Meta-Modell enthält folgende Elemente, die für das CIM wesentlich sind:

- Eine **Klasse** definiert eine Menge von Objekten mit gemeinsamen Eigenschaften. Die Vererbung zwischen den Klassen ist einfach, und die Vererbungsbeziehungen sind nicht strikt geregelt, d.h. die ererbten Eigenschaften oder Methoden können überschrieben werden.

- Ein **Schema** ist eine Gruppe von zu administrativen Zwecken zusammengestellten Klassen, die einen Besitzer haben.

- Eine **Eigenschaft** (*property*) ist ein Wert zum Charakterisieren der Instanz einer Klasse. Im JMX-Sinne sind Eigenschaften die *Member*-Variablen einer Klasse, die mit Hilfe zweier Funktionen gelesen oder gesetzt werden können.

- Die **Methoden** definieren die Signaturen der Operationen, die auf Objekten einer Klasse ausgeführt werden können. Eine Signatur legt den Namen, den Rückgabewert und die Parameter einer Operation fest.

- Die **Qualifikatoren** legen zusätzliche Eigenschaften der Klassen sowie der Methoden der Klassen fest.

- Die **Assoziationen** sind Klassen, die Beziehungen zwischen zwei oder mehr Objekten repräsentieren.

- Die **Referenzen** stellen verweise auf Klassen dar und definieren die Rolle einer jeden Klasse in einer Assoziation. Nur Assoziationen enthalten Referenzen, und jede Referenz repräsentiert eine Klasse in einer Assoziation. D.h. jede Assoziation enthält mindestens zwei Referenzen.

- Ein **Trigger** ist die Bestätigung einer Zustandsänderung wie z.B. das Erzeugen, Löschen oder Aktualisieren (*update*) einer Klasseninstanz oder einer Eigenschaft.

- Die **Ereignisklassen** (*indications*) definieren Klassen für die Ereignismeldungen. Wenn ein Trigger durch entsprechende Trigger-Bedingungen entsteht, werden die entsprechenden Ereignisklassen instantiiert.

Das *Managed Object Format (MOF)*

Das *Managed Object Format (MOF)* legt die Syntax für die Definition der Managementobjekte fest, die auf dem CIM-Meta-Schema basieren. Das MOF ist eine IDL-ähnliche Sprache, mit deren Hilfe die CIM-Klassen und weitere Managementinformationen in einem Texteditor oder einer anderen geeigneten Entwicklungsumgebung beschrieben werden können (siehe [CIMS]).

Das Managementschema

CIM ist so strukturiert, dass die Managementumgebung als eine Sammlung von hierarchischen Systemen bezeichnet werden kann, die aus diskreten Elementen bestehen. In diese Strukturierung stehen die konkreten Managementobjektklassen in einer Beziehung zueinander, die von einer Vererbungs- und Spezialisierungshierarchie geprägt ist. Dazu spezifiziert CIM drei Hierarchieschichten:

1. Das *Kernmodell (Core Model)* ist eine kleine Menge von Klassen, Assoziationen und Eigenschaften, die eine Basis für das Beschreiben und Analysieren der zu managenden Systeme schafft. Das Kernmodell repräsentiert einen Startpunkt für das gemeinsame Modell und dient als Ausgangpunkt für weitere Verfeinerungen. Ein Beispiel hierfür ist die Klasse `ManagedSystemElement`, aus der die Klassen `PhysicalElement` und `LogicalElement` abgeleitet sind.

2. Das *gemeinsame Modell (Common Model)* ist eine Verfeinerung des Kernmodells und beinhaltet die Klassen, die einerseits technologieunabhängig sind, anderseits aber konkret genug sind, um als Basis für die Managementanwendungen zu dienen. Netze, Netzkomponenten, Endsysteme und Anwendungen werden von diesem Bereich abgedeckt.

3. Das *Erweiterungsschema (Extension Schema)* stellt eine technologieabhängige Erweiterung zum gemeinsamen Modell dar. Das gemeinsame Modell wird mit Bezug auf die Objekte entwickelt, die im Erweiterungsschema definiert sind.

5.3.2 Bewertung

Die Kopplung der Zukunft von WBEM an DMTF, durch die teilweise konkurrierende Interessen mit einander verknüpft wurden, macht es schwierig, eine realistische Abschätzung der weiteren Entwicklung des WBEM zu geben. Während Microsoft sein eigenes kommerzielles WBEM-konformes Produkt, die sogenannte *Microsoft Windows Management Instrumentration (Microsoft WMI)*, vertreibt, gründeten einige der Mitglieder der DMTF eine Initiative, die ein Projekt für die Koordinierung und Interoperabilität der verschiedenen *open source* WBEM Produkte betreut[10].

[10] Nähere Information über diese Produkte findet man unter: http://www.opengroup.org/wbemsource/.

Die meisten gängigen Managementarchitekturen bieten Schnittstellen zum WBEM. Das Informations- und das Kommunikationsmodell der Architektur sind plattformunabhängig. Wenn man außerdem die angebotenen Schnittstellen in CIMOM einbezieht, hat WBEM die Chance, eine Managementarchitektur für Managementarchitekturen zu werden, die in den Unternehmen eingesetzt werden kann. Dagegen ist WBEM wenig geeignet, um als Managementinfrastruktur in einer Anwendung eingesetzt zu werden. Denn bis jetzt ist der Ansatz ziemlich allgemein gehalten, so dass die konkreten Managementwerkzeuge nicht oder nicht weit genug entwickelt sind. Z.B hat Sun ein in Java geschriebenes Paket zum WBEM freigegeben. Dieses ist aber mit der JMX hinsichtlich der Managementfunktionalität und der Instrumentierung der Ressourcen mit Blick auf das erreichte Niveau noch nicht vergleichbar.

Die Etablierung von WBEM als Managementarchitektur der Managementarchitekturen ist ebenfalls noch offen. Dazu fehlt ein bestimmter Reifegrad, denn das Projekt ist noch ziemlich jung und hat erst seit zwei Jahren seine neue Orientierung. Nachteilig beeinflusst wird die Akzeptanz der Architektur aber auch dadurch, dass die Vermarktungsstrategien der beteiligten Unternehmen sich gegenseitig ausschließen.

Kapitel 6

Java Management Extension (JMX)

6.1 Überblick

JMX ist eine universelle und offene Managementtechnologie, die der Nachfolger von ***Java Management API*** *(JMAPI)* ist und laut Sun Microsystems ein Synonym für Java Management ist (vgl. [JMX]). JMAPI wurde von JavaSoft 1996 eingeführt, um die Verwendung der Programmiersprache Java für die Implementierung von Managementanwendungen zu unterstützen.

In der JMAPI-Spezifikation wurde eine neue Managementarchitektur definiert, die auf der Basis einer sogenannten *Proxy-Lösung* vor allem gute Möglichkeiten für das Web-basierte Management enthielt. Bei dieser Lösung wird eine Managementplattform mit ihrer Erweiterung um die HTTP-Server-Funktionalität über jeden Browser zugänglich. Die Kommunikation mit den Ressourcen kann weiterhin über die vorhandenen Protokolle, wie z.B. SNMP, abgewickelt werden, und HTTP wird nur zur Kommunikation mit den Benutzern verwendet (vgl. [HeAN99], S. 223).

Die wachsenden Marktanforderungen und die Tatsache, dass Sun Microsystems seine eigene Managementinfrastruktur zur Unterstützung seiner Produkte brauchte, haben Sun dazu bewegt, an einer JMX-Spezifikation zu arbeiten. In der *Java Specification Request* (JSR) [JSR3] Nummer 3, mit der die erste Anforderung zur Spezifikation der JMX vorgelegt wurde, wurden die ersten Schritte von *JCP*[1] eingeleitet. Man wollte die Vorzüge von JMAPI Version 1.0 beibehalten und auf der Basis von *Java Dynamic Management Kit* (JDMK) [JDMK] ein universelles Rahmenwerk entwerfen. Dieses dient inzwischen als Kern aller bisher entwickelten Java Management Werkzeuge. Im September 2000 wurde die erste Version der JMX-Spezifikationen von JCP frei gegeben. Diese bestand aus einer Einführung in die Architektur des Managements, die Klassenbibliotheken für die Anwendungsprogrammierung und die dazugehörigen in Java dokumentierten Schnittstellenbeschreibungen der Programmierungsobjekte.

[1] *Akronym für: Java Community Process*, http://jcp.org/

6.2 JMX-Architektur

Die JMX-Spezifikation teilt die Architektur von der JMX in drei Ebenen auf. Wie das Schaubild 6.1 zeigt, sind alle Schlüsselkomponenten von der JMX in diesen Ebenen eingesiedelt. Die unterste Schicht der Architektur bildet die Instrumentationsschicht (engl. *Instrumentation Level*). Diese Stufe der Spezifikation legt die Regel für die Implementierung der durch die JMX zu managenden Ressourcen, die sogenannten *Managed Beans (MBeans)*, fest. Die Instrumentierung der Ressourcen schafft Schnittstellen, durch die Ressourcen von der nächst höheren Stufe angesprochen werden können. Dazu brauchen sie aber nicht von der Existenz der nächsten Stufe zu wissen. Die Schnittstellen sind offen, und somit können sie nicht nur von der JMX sondern von jeder anderen Anwendung, die das Konzept unterstützt, angesprochen werden.

Die Agentenschicht (engl. *Agent Level*) stellt die Spezifikationen zur Implementierung von Agenten im Managementsinne zur Verfügung. Die JMX-Managemententität, die in einer JVM läuft, ist ein JMX-Managementagent, der die zu managenden Ressourcen bzw. ihre Repräsentanten und die Schnittstellen zu den Managementanwendungen umfasst. Er verbindet die zu managenden Ressourcen mit den Managementanwendungen. Sein Herzstück ist ein sogenannter *MBean-Server*, der eine Zentrale für die Registrierung von MBeans ist. Ein JMX-Managementagent enthält noch einige andere Dienste, die vom Managementagenten genutzt werden. Diese Dienste werden im Funktionsmodell der JMX-Architektur näher erläutert.

Die oberste Stufe ist die verteilte Dienstebene (engl. *Distributed Services Level*). Diese Schicht stellt die Schnittstellen zur Implementierung von JMX-Managern zur Verfügung. Mit den Schnittstellen und Komponenten, die hier definiert werden, können Managementagenten bzw. auf einer Hierarchie von Managementagenten angesprochen werden.

Neben den genannten Schichten der Architektur bietet Java noch Spezifikationen und Programmierschnittstellen zur Interoperabilität zwischen der JMX und anderen Managementstandards. Diese Erweiterungen spielen aber bei der Funktionalität der JMX keine Rolle und sind nicht Teil der Architektur der JMX.

6.2.1 Informationsmodell

Das Informationsmodell der Architektur wurde in der Ressourcenspezifikation festgelegt, die den Entwicklern Werkzeuge zum Instrumentieren der Ressourcen zur Verfügung stellt. Das Informationsmodell ist objektorientiert, wobei die Diagramme mit Hilfe von UML modelliert werden und die Implementierungssprache Java ist.

Alle Ressourcen können zum Management durch die JMX instrumentiert werden. Die Ressourcen können also Geräte, Dienste, Anwendungen, usw. sein. Die MBeans können ein kleines Teil einer Ressource oder das Ganze instrumentieren und repräsentieren. Somit haben die Entwickler freie Hand, ihre Ressourcen so präzise wie möglich als MBeans zu instrumentieren.

Die Spezifikation legt eine einfache und flexible Konzeption vor, mit der die Ressourcen

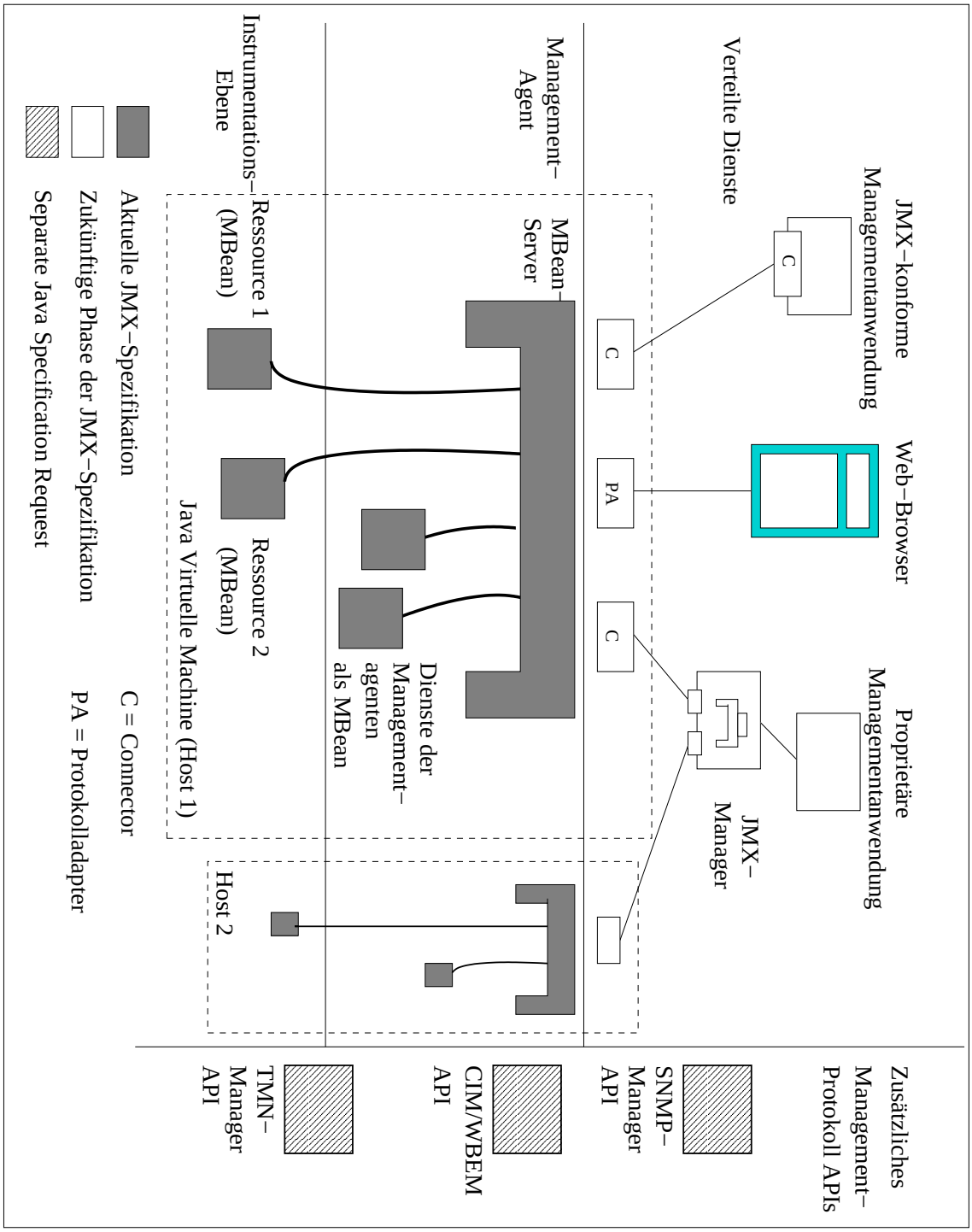


Abbildung 6.1: JMX-Architektur

```
public AttributeType getAttributeName ();
public void setAttributeName (AtrributeType value);
```

Tabelle 6.1: Entwurfschema zum Lesen und Ändern von Attributswerten

entweder in einer standardisierten Art ihre eigenen konformen Schnittstellen oder die vordefinierten dynamischen Schnittstellen implementieren, die während der Laufzeit mehr Flexibilität erlauben.

Die Hauptkomponenten der Instrumentationsebene und des Informationsmodells ist eine MBean, die im folgenden Unterabschnitt vorgestellt wird.

6.2.1.1 MBeans

MBeans sind konkrete Java-Objekte, die Managementschnittstellen implementieren. Eine MBean kann folgende managagementrelevanten Bestandteile haben:

- Konstruktoren, die die Java-Klassen instantiieren können. Die für das Management definierten Konstruktoren sind mit dem Attribut `public` zu bezeichnen, wovon eine MBean mindestens einen zur Verfügung stellen muss. Die Klasse kann dann bei Bedarf (d.h. wenn es durch eine Anwendung verlangt wird) vom MBean-Server instantiiert werden. Wenn in einer MBean kein Konstruktor definiert wird, gelten die Regeln vom Java Compiler. In diesem Fall stellt der Compiler einen *default*-Konstruktor ohne Argument zur Verfügung.

- Methoden zum Zugriff auf Werte der Attribute einer MBean. Attribute einer MBean sind diskrete Eigenschaften einer MBean, wobei jedes Attribut einen eindeutigen Namen hat. Die Attribute definieren die Erscheinung und das Verhalten einer MBean oder einer Ressource, die durch die MBean instrumentiert wird. Diese Attribute werden immer durch Methoden manipuliert, die dem Java-Objekt gehören, das sie besitzt. Es gibt für die Attribute, die im Managementsinne *read-only* oder *write-only* Zugriffrechte besitzen, die sogenannten *getter*- bzw. *setter*-Methoden. So werden z.B. für ein Attribut mit den *read-write* Zugriffrechten zwei Methoden definiert; eine getter- und eine setter-Methode. Die Tabelle 6.1 zeigt die Definition von getter- und setter-Methoden für ein Attribut an. Um eine Redundanz zu verhindern, wird der Wert eines Attributs vom Type `boolean` mit Hilfe nur einer Methode abgefragt, die entweder eine getter-Methode ist, oder das in der Tabelle 6.3 auf der nächsten Seite vorgestellte Entwurfsschema hat.

- Optionale Operationen, die zu Managementzwecken ausgeführt werden müssen

- Optionale Methoden, die die Notifikationen aussenden.

```
public void is AttributeName ();
```

Tabelle 6.3: Entwurfschema zum Lesen der Attribute vom Typ `boolean`

```
public interface MyClassMBean {
   public Integer getState();
   public void setState(Integer s);
   public void reset();
}
```

Abbildung 6.2: Definition der Standard-Mbean-Schnittstelle MyClassMBean

Eine Java-Klasse zum Management muss entweder als *Standard*-MBean als *dynamische* MBean entworfen werden.

Standard-MBean

Standard-MBean werden zu Instrumentierung der Ressourcen verwendet, bei denen die Struktur der zu managenden Daten im voraus bekannt ist und sich nicht dynamisch ändert.

Standard-MBean basieren auf einem Namensschema, das es den Monitordiensten bzw. Managementagenten erlaubt, die Klassen als MBean und die Rolle ihrer Methoden und Operationen zu identifizieren. Eine Standard-MBean implementiert ein `interface`, das nach ihr benannt und um eine `MBean` erweitert wurde. Die Schnittstelle listet dabei alle Methoden auf, die für das Management wichtig erscheinen, wobei ihre Implementierung durch die Absichten festgelegt wird, die das Management verfolgt. Alle Methoden einer Klasse, die nicht in der Schnittstellendefinition vorkommen, werden von Managementanwendungen nicht beachtet.

Dieses Entwurfsprinzip und seine Überprüfung werden in der Fachsprache *introspection*-Prozess genannt. In diesem Prozess überprüft ein JMX-Managementagent eine Klasse und ihre möglichen Superklassen, um festzustellen, ob sie eine MBean mit dem vorgegebenen Entwurfsprinzip repräsentieren. Nur im positiven Fall akzeptiert er den Namen der Attribute und der Operationen der MBean.

Die Abbildungen 6.2 und 6.3 auf der nächsten Seite stellen die Verwendung von Standard-MBean vor, wobei die Klasse `MyClass` in der Tabelle 6.3 auf der nächsten Seite die Schnittstelle `MyClassMBean` in der Tabelle 6.2 implementiert.

Aus den Tabellen geht hervor:

- Die Klasse `MyClass` hat zwei Membervariablen, von denen nur die Variable `state` für Managementanwendungen zum Lesen und Schreiben sichtbar ist.

```
public class MyClass implements MyClassMBean {
  private Integer state = null;
  private String hidden = null;
  public Integer getState() {
    return state;
  }
  public void setState(Integer s) {
     state = s;
  }
  public String getHidden() {
    return hidden;
  }
  public void setHidden(String h) {
     hidden = h;
  }
  public void reset() {
    state = null;
    hidden = null;
  }
}
```

Abbildung 6.3: Definition der Standard-MBean MyClass

- Die Membervariable `hidden` kann von Managementanwendungen nicht manipuliert werden, da in der Schnittstellendefinition keine Methode für diesen Zweck bekannt ist.

- Groß- und Klein-Schreibung sind bei der Definition und Bekanntgabe der Methoden in der Managementschnittstelle zu unterscheiden. Z.B. würden im genannten Beispiel `getstate` und `setState` zwei Attribute definieren; `state` und `State`, wobei die `state` nur das read-only Zugriffsrecht und die `State` nur das write-only Zugriffsrecht erlauben würde.

Eine Java-Klasse wird ebenfalls eine MBean, wenn eine von den möglichen Elternklassen Managementschnittstellen implementieren. Auch die Managementschnittstellen zeigen, wie jede andere in der Programmiersprache Java definierte Schnittstelle, ein Vererbungsverhalten, deren Beschreibung den Rahmen der vorliegenden Arbeit sprengen würde. Für eine detaillierte Beschreibung des Vererbungsverhaltens wird auf [JMXIAS] (S. 44ff.) verwiesen.

Dynamische MBean

Die dynamischen MBeans bieten im Gegensatz zu den Standard-MBeans den Entwicklern mehr Flexibilität. Sie ermöglichen es, die Ressourcen, deren Daten sich im Laufe der Zeit verändern,

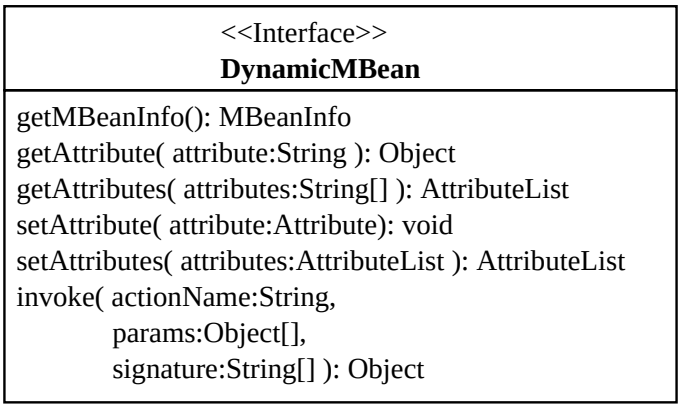


Abbildung 6.4: Die Schnittstelle `DynamicMBean`

zu instrumentieren. Konzeptionell hat eine dynamische MBean managementbezogene Attribute und Operationen. Es ist aber nicht vorgesehen, dass sie wie bei Standard-MBeans durch besondere Methodennamen bekannt gegeben werden. Stattdessen werden die Attribute, Operationen und ihre Signaturen während der Laufzeit bekannt gegeben. Das geschieht dadurch, dass die Ressourcen durch die in der Abbildung 6.4 vordefinierte `DynamicMBean`-Schnittstelle instrumentiert werden. Diese bietet die gleichen Möglichkeiten, die von einer Standard-MBean angeboten werden und stellt sie für das Management zur Verfügung. Somit können die JMX-konformen Managementanwendungen die dynamischen MBeans in der gleichen Weise wie die Standard-MBeans manipulieren.

Wie man von der im UML-Diagramm definierten Abbildung 6.4 ablesen kann, ist die zentrale Methode einer `DynamicMBean`-Schnittstelle `getMBeanInfo`, die ein Objekt vom Typ `MBeanInfo` (Siehe [JMXIAS], S. 53) zurückgibt. Diese Klasse beinhaltet eine Liste von diversen Java-Klassen, die die Attribute, Konstruktoren, Operationen und Ereignisse einer MBean beschreiben und zusammen mit `MBeanInfo` die sogenannten *Metadaten*-Klassen von MBean bilden. `MBeanInfo` liefert mit Hilfe der Objekte der Metadaten-Klassen sämtliche Informationen über die Managementschnittstelle einer MBean, unabhängig davon ob sie eine Standard- oder eine dynamische MBean ist.

Ein Problem verbunden mit den dynamischen MBeans ist es, dass der MBean-Server nicht wie bei Standard-MBeans die Selbstbeschreibung der Methoden, also die Introspection, überprüft. Der Entwickler muss diese Verantwortung selber übernehmen und die notwendigen Regeln einhalten.

Da die Managementschnittstellen einer dynamischen MBean während der Laufzeit bekannt werden, können sich die Managementschnittstellen einer Ressource dynamisch verändern, so dass die aufeinander folgenden Aufrufe der `getMBeanInfo`-Methode unterschiedliche Ergebnisse liefern können. Das wirft ein Problem auf, da die Anpassung der Managementanwendungen an die Dynamik der Managementschnittstellen erschwert wird.

Offene MBean

Der Einsatzbereich von offenen MBeans ist das Management von Objekten, die während der Laufzeit entdeckt werden. Mit der Instrumentierung dieser Objekte als offene MBeans können Managementanwendungen in die Lage versetzt werden, mit den zugeschnittenen Mechanismen die Managementdaten zu verwenden, ohne dass die Managementanwendungen neu angepasst, kompiliert oder dynamisch gelinkt werden müssen.

Die Spezifikation der offenen MBeans ist noch nicht vollständig in der JMX festgelegt und kann noch Änderungen erleben. Deshalb ist noch keine Dokumentation zu den Programmierschnittstellen freigegeben, und es kann kein Konformitätstest gemacht werden. Dieser ist im JMX-Managementagent intern durchzuführen.

ModelMBean

ModelMBeans sind weitere dynamische MBeans, die eine universelle Instrumentierung aller Sorten von Ressourcen in einer einfachen, generischen und leicht konfigurierbaren Weise erlauben. Ein weiterer Vorteil dieser MBeans ist, dass sowohl die statischen als auch die dynamischen Eigenschaften der Ressourcen damit instrumentiert werden können. Anders als bei den offenen MBeans ist der Konformitätstest hier bereits gegeben: Die JMX stellt hierfür eine Klasse bereit, die die `ModelMBean`-Schnittstelle implementiert, die eine abgeleitete Schnittstelle der `DynamicMBean`-Schnittstelle ist.

Die Ressourcen können in einem beliebigen Zeitabschnitt mit Hilfe der ModelMBeans erfasst werden. Dazu ist eine ModelMBean durch den JMX-Agenten zu initialisieren. Diese ist mit den gesammelten Daten zu konfigurieren, und ihr Verhalten zum Management festzulegen.

6.2.2 Organisationsmodell

Das Grundkonzept des Organisationsmodells der JMX-Architektur basiert auf einem JMX-Managementagenten, der in einer JVM-Umgebung ausgeführt wird und als eine Verbindungskomponente zwischen den Managementanwendungen und MBeans fungiert. Er besteht aus den MBeans, dem MBean-Server, einigen Diensten und Schnittstellen für die Managementanwendungen, also die sogenannten *Protokolladapter* und *Connectoren*.

Der MBean-Server ist eine zentrale Stelle, bei der MBeans registriert werden müssen. Jeder MBean-Server wird mit einem Domänennamen versehen, der den MBean-Server eindeutig identifiziert. Alle Ressourcen, die bei einem MBean-Server registriert sind, werden von diesem Server gemanagt. Ihre Identifizierung wird ebenfalls durch die Identifizierung der Domäne festgelegt. Die Verwaltung einer Domäne durch einen MBean-Server legt also die Organisation der Managementdomäne fest. Somit können die zu managenden Ressourcen, die irgendwie zusammengehören, in einer Domäne zusammengefasst und bei einem MBean-Server registriert werden. Die Zusammengehörigkeit der MBeans können verschiedene Aspekte der Ressourcen,

wie z.B. ihre Zugehörigkeit zu einer Organisation, Abteilung oder einem Softwarepaket widerspiegeln. Das Konzept ist flexibel genug, um die Ressourcen hinsichtlich der verschiedenen Aspekte instrumentieren und in einer Domäne managen zu können.

Jeder Managementagent bietet die volle Funktionalität einer Managemententität. Er kann als Server für die Managementanwendungen dienen. Er kann aber auch die Rolle des Klienten gegenüber anderen Managementagenten übernehmen. Das Konzept ist damit so flexibel, dass es alle in 3.4.2 genannten Organisationsmodelle unterstützt.

6.2.3 Funktionsmodell

Zum Funktionsmodell der JMX-Architektur gehören die teilweise bereits behandelten Funktionen des MBean-Servers, die Funktionen des Ereignismodells, die Dienste der Managementagenten und die verteilten Dienste zur Kommunikation mit den Managementanwendungen. Außerdem bietet die JMX die Möglichkeit, seine eigenen Dienste zu implementieren und ähnlich wie die von der JMX implementierten Dienste verwalten zu lassen.

6.2.3.1 Ereignismodell

JMX-Spezifikationen definieren ein Ereignismodell, das die MBeans im Umfeld eines JMX-Managementagenten befähigt, bestimmte Ereignisse, die *Notifications* genannt werden, an interessierte Objekte zu senden, die sich im Umfeld des selben Agenten befinden. Dieser Mechanismus ist den Aktionen, die durch Managementanwendungen an MBeans durchgeführt werden, entgegengesetzt. Die Anwendungen und andere Objekte, die Interesse an solchen Ereignissen haben, registrieren sich einmal als *Listner* bei den MBeans, die als *Broadcaster* fungieren, und können dann die sämtlichen eintretenden Ereignisse entgegennehmen. Das JMX-Ereignismodell besteht aus der `Notification`, dem `Notificationbroadcaster`, dem `NotificationListner` und dem `NotificationFilter`. (Siehe [JMXIAS], S. 47ff.).

6.2.3.2 Dienste der Managementagenten

Mit der Integration der Dienste in den Managementagenten verlagert man die Intelligenz in den selbigen, womit man mächtige Managementlösungen entwickeln kann, die der gegebenen Komplexität gerecht werden. In der JMX sind diese Dienste Objekte, die selber als MBeans im MBean-Server registriert werden und Operationen auf MBeans ausführen können. Die JMX definiert vier Dienste, welche verschiedene Aufgaben übernehmen:

Dynamisches Laden

Die Dienste zum dynamischen Laden (*dynamic loading services*) eignen sich zum Laden der Java-Klassen und nativen Bibliotheken von einem beliebigen Server im Netzwerk. Diese Aufgabe wird durch die sogenannten *Management Applets (m-let)* ausgeführt, indem der m-let-Dienst eine m-let-Textdatei lädt, die die Information über die zu ladenden MBeans enthält. Diese Information wird durch einen XML-ähnlichen Tag, den sogenannten `MLET`-Tag, dargestellt. Die Stelle der Textdatei wird als URL angegeben, und wenn eine m-let-Textdatei geladen wird, werden alle darin spezifizierten MBeans geladen, instantiiert und beim MBean-Server registriert.

Monitoring

Die *Monitoring*-Dienste ermöglichen die Beobachtung der Attributswerte von MBeans und deren Änderungen. Die Attribute werden in festgelegten Zeitabständen beobachtet. Wenn die Attribute einen festgelegten hohen und/oder niedrigen Wert ereicht haben, lösen die Monitore eine Notifikation aus. Eine Notifikation kann auch ausgesendet werden, wenn eine gewisse Art von Fehler beim Beobachten der Attribute auftritt.

Timer

Ein *Timer*-Dienst dient dazu, zu einem im voraus festgelegten Zeitpunkt oder in bestimmten vordefinierten Zeitintervallen eine Notifikation an die registrierten Interessenten zu senden. Dieser Dienst ist eine MBean, die als ein konfigurierbarer Scheduler gemanagt werden kann.

Relationsdienst

Ein Relationsdienst (*Relation service*) überwacht die Konsistenz der definierten Relationen zwischen den MBeans. Eine Relation ist im objektorientierten Sinne eine benutzerdefinierte n-äre Verknüpfung zwischen den MBeans. Dabei haben die MBeans jeweils eine festgelegte Rolle.[2]

Eine Relation in der JMX-Spezifikation besteht aus einer Liste der Rollen, die die MBeans in einer Verknüpfung annehmen können. Der Typ einer Relation wird mit Hilfe der Informationen über eine Menge von Rollen definiert, die an der Relation teilnehmen. Ein Relationstyp ist also ein Muster für die Relationsinstanzen, die die Rollen und somit die MBeans, zu denen die Rollen gehören, verknüpfen. (Siehe [JMXIAS], S. 149ff.).

6.2.3.3 Verteilte Dienste

Verteilte Dienste stellen Schnittstellen der Managementagenten für die Managementanwendungen und deren Komponenten zur Verfügung. Zu diesen Diensten zählen die *Protokolladapter*

[2] Eine ausführliche Behandlung der Thematik findet sich z.B. in [RUM91] und [UMLS].

und *Connectoren*, deren Spezifikation in der bisherigen Phase der Entwicklung der JMX noch fehlt. Ein MBean-Server nutzt diese Dienste, um die Managementagenten den Anwendungen außerhalb der JVM zugänglich zu machen.

Ein Protokolladapter erlaubt das Management aller registrierten MBeans im MBean-Server durch ein bestimmtes Protokoll. Ein Beispiel für einen Protokolladapter ist ein HTML-Adapter, der die Datenein- und -ausgabe über einen Browser ermöglicht.

Connectoren sind Schnittstellen, die die Kommunikation zwischen den Managementanwendungen und den Managementagenten zustande bringen. Die Connectoren müssen sowohl auf der Managerseite als auch auf der Agentenseite vorhanden sein, damit eine transparente Kommunikation über das Netzwerk möglich ist. Somit können die Anwendungen unabhängig vom Kommunikationsprotokoll transparente Verbindungen zu den Managementagenten aufnehmen und Managementaufgaben erledigen.

Da die Operationen des MBean-Servers auf registrierten MBeans durch die Managementanwendungen nur über einen Protokolladapter oder einen Connector ausgeführt werden können, muss jeder Managementagent mindestens einen Adapter oder Connector zur Verfügung stellen.

6.3 Weitere Java-Erweiterungen

Java bietet Schnittstellen zu den meisten Managementstandards. Diese sind nicht Teil der JMX und spielen bei der Funktionalität der JMX keine Rolle. Die Erweiterungen werden für die Interoperabilität von WBEM [JSR48], CORBA [JSR70] und TMN [JSR71] mit der JMX benötigt. Um die Entwicklung von Managementlösungen für J2EE beschleunigen zu können, beschäftigt sich [JSR77] mit der Spezifizierung eines standardisierten und universellen Managementmodells.

6.4 Bewertung der JMX

Durch die Anwendung der JMX für Managementlösungen ergeben sich folgende Vorteile:

- Einsatzfähigkeit der JMX im Netz-, System- und Anwendungsmanagement. Diese wird dadurch erreicht, dass die JMX die notwendigen Schnittstellen zum Managen der Ressourcen definiert. Dadurch gewinnt man Flexibilität, so dass die JMX überall zum Einsatz kommen kann.

- Java-Anwendungen können ohne großen Aufwand administriert werden. Das wird dadurch erleichtert, dass der Kern von JMX hauptsächlich aus dem MBean-Server besteht, der die zu managenden Objekte registriert und ihre Managementschnittstellen festhält. Gleichzeitig kann er auf jedem Java-fähigen Gerät ausgeführt werden. Somit kann er durch Managementanwendungen gesteuert werden.

- Die JMX-Spezifikation stellt nur minimale Basisdienste für den Managementagenten zur Verfügung. Die weiteren Dienste von JMX-Managementagenten, die zum Zweck der Konformität mit den anderen Managementlösungen entwickelt werden, sind unabhängige Module und können abhängig von der jeweiligen Anforderung an einen Managementagenten dynamisch in die Infrastruktur eingebunden werden. Dieser komponentenbasierte Ansatz stellt einen Rahmen zur Verfügung, durch den die JMX-Lösung sowohl auf kleinen als auch auf großen Geräten eingesetzt werden kann.

- Die JMX-Managementagenten können dazu befähigt werden, über einen HTML-Browser durch das HTTP oder über Managementanwendungen durch andere standardisierte Protokolle wie z.B. SNMP, WBEM, IIOP und TMN gemanagt zu werden.

- Wirksamer Einsatz von und in den existierenden Java-Technologien.

- Wirksamer Einsatz von zukünftigen und zukunftorientierten Managementkonzepten. Die Flexibilität und Dynamik, die durch die JMX-Lösung hervorgehoben werden, erlauben es, die neuen Technologien im Managementbereich einzusetzen. Z.B. kann eine zu der JMX-Spezifikation konforme Managementlösung *Jini*-Technologie unterstützen.[3]

Ein Nachteil der JMX ist, dass sie kein universelles Objektsystem für verteilte Anwendungen ist. Außerdem legt die JMX keine Managementinformationen in Form von MIBs oder ähnlichem fest. Es ist wünschenswert, dass die Managementschnittstellen der Komponenten etwas detaillierter definiert würden. Möchte man die Komponenten eines Systems instrumentieren, so hat man kein vordefiniertes Muster zur Hand.

Ein weiterer Nachteil der JMX ist, dass die Schnittstellen für die Managementanwendungen nicht spezifiziert wurden. Außerdem wäre es von Vorteil, wenn die spezifizierten JMX-Managementanwendungen überall die gleiche Funktionalität und auch Oberfläche anbieten würden.

[3]Die von Sun entwickelte Jini-Technologie ist ein Verbindungsprotokoll, das es den Klienten in einer heterogenen Umgebung ermöglicht, die benötigten und geeigneten Ressourcen und Dienste im Netz zu finden. In einer Demonstration, die durch Sun Microsystems entwickelt wurde, leistet Jini ein spontanes Finden von Ressourcen und Diensten, die durch die JMX gemanagt werden. Diese Kombination wird von Sun Microsystems *Spontaneous Management* genannt.

Kapitel 7

AMETAS-Management

7.1 Überblick

Im Rahmen dieser Arbeit wurde für das AMETAS eine Managementinfrastruktur entwickelt, die konform zur im Kapitel 6 vorgestellten JMX-Spezifikation ist. Alle Komponenten vom AMETAS können dabei mit den Werkzeugen des JMX-Informationsmodells instrumentiert werden. Es werden bisher nur die ModelMBeans zur Instrumentierung verwendet. Diese MBeans stellen generische Werkzeuge zur Verfügung, so dass die Managementinformationen aller Ressourcen des Agentensystems erfasst werden können. Es ist aber durchaus vorstellbar, dass bei einer Erweiterung des Managements noch weitere Sorten von MBeans zum Einsatz kommen.

Die Managementinfrastruktur stellt für jede AMETAS-Stelle einen Managementagenten zur Verfügung, der die Verantwortung für diverse Managementaufgaben in der Stelle übernimmt. Mit Hilfe der für die zu Managementzwecken entwickelten Dienste und Agenten kann der Managementagent bestimmte Ereignisse, die in einer AMETAS-Stelle passieren, beobachten. Aus seinen Beobachtungen zieht er bestimmte Schlüsse, evtl. agiert oder reagiert er auch selber.

Beim AMETAS-Management stellt man folgende Eigenschaften fest:

- Es ist JMX-konform. Die Konformität bezieht sich auf die Instrumentierung der Managementressourcen und auf den internen Aufbau der Managementagenten. Die Konformität wird durch den Managementagenten intern getestet.

- Die Managementinfrastruktur verwendet die Agenteninfrastruktur und wird ein Teil von ihr. Genau genommen wird die Managementinfrastruktur als ein AMETAS-Dienst aufgefasst und kann alle Instrumente und Dienste, die von einer AMETAS-Stelle zur Verfügung gestellt werden, in Anspruch nehmen. Sie unterliegt allen Einschränkungen, mit denen alle Stellennutzer, vor allem die AMETAS-Dienste, konfrontiert sind.

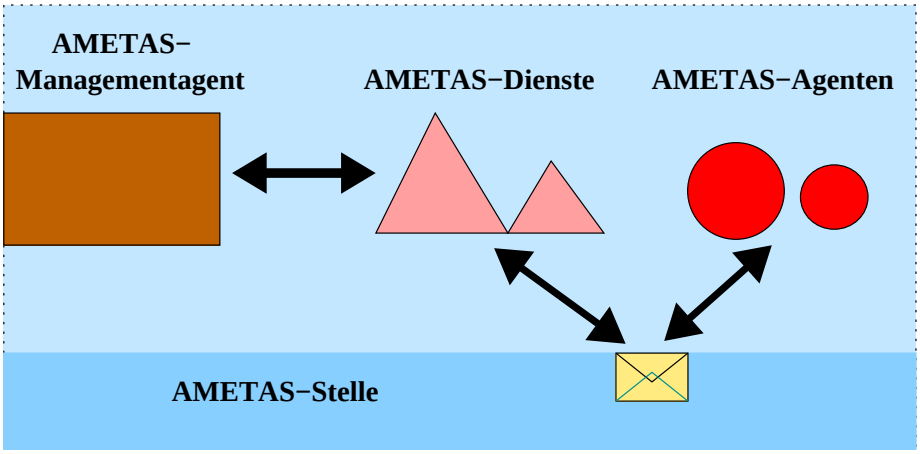


Abbildung 7.1: Darstellung des Managementagenten in einer Stelle

- Als Managementwerkzeuge werden die AMETAS-Dienste und AMETAS-Agenten genutzt.

- Das System managt sich selbst. Die Managementagenten sind mit genügend Intelligenz ausgestattet, um die systeminternen und managementbezogenen Aktivitäten erfassen zu können. Das liegt zum großen Teil an der AMETAS-Infrastruktur, die den Diensten und Agenten solche Möglichkeiten bietet. Z.B. spielt das Ereignissystem vom AMETAS eine Schlüsselrolle bei der Benachrichtigung der Managementdienste über diverse Ereignisse. Der Kommunikationsmechanismus der AMETAS-Infrastruktur wird auch für die Kommunikation zwischen den Managementdiensten und Agenten genutzt.

7.2 Infrastrukturaufbau

Jede AMETAS-Stelle wird mit einem Managementagenten ausgestattet, der Verantwortung für die Managementaufgaben trägt. Die Abbildung 7.1 veranschaulicht einen Managementagenten und seine Beziehung zu den weiteren beteiligten Komponenten in einer Stelle.

Jeder Managementagent wird vom einem Dienst gestartet und steht mit dem Dienst im weiteren Verlauf in Verbindung. Das bedeutet aber auch, dass seine Ausführungsumgebung eine AMETAS-Stelle ist. Wird eine Stelle aus irgendeinem Grund gestoppt, so werden sämtliche Managementkomponenten gestoppt. Die Schnittstellen des Managementagenten zur Dienstseite erlauben die Ausführung von folgenden Operationen:

- Operationen, die zur Konfiguration des Managementagenten notwendig sind. Die Konfigurationsdaten werden beim Start von einer Textdatei (*PU.prm*) und im vom AMETAS vorgegebenen Format eingelesen und mit Hilfe dieser Methoden an den Managementagenten weitergeleitet.

- Operationen zur Anforderung von Diensten, die der Managementdienst in Verbindung mit der Stelle und den weiteren Stellennutzern erbringen muss. Dazu zählen Methoden zum Starten der AMETAS-Agenten, sowie Anfragen an weitere AMETAS-Dienste, die z.B. die automatische Konfiguration oder Konfigurationsänderung auf der AMETAS-Stelle vornehmen können.

- Operationen, die die von Managementdiensten geforderten Anfragen bearbeiten oder an MBean-Server bzw. an Managementanwendungen weiterleiten. Hierzu zählen z.B. Methoden, die die managementrelevanten Ereignisse und die dadurch entstandenen möglichen Zustandsänderungen der Stelle und ihrer Komponenten auf die jeweiligen MBeans abbilden. Dadurch werden die Managementdaten aktualisiert, so dass sie bei der nächsten Abfrage zur Verfügung stehen. Zum derzeitigen Stand der Implementierung sind diese Schnittstellen für Aufgaben geeignet, die in erster Linie mit den AMETAS-Agenten in Verbindung stehen. Sie leiten die Ereignisse, die das Starten, Ankommen oder Terminieren eines Agenten betreffen, und ihre Daten an Managementagenten bzw. MBean-Server weiter. Diese Aufgaben werden im Kapitel 9 näher erläutert. Man muss hier aber festhalten, dass eine mögliche Erweiterung der Managementfunktionalität eine Erweiterung der adäquaten Methoden der Schnittstellen unumgänglich macht. Die Infrastruktur bietet einen Rahmen, in den Erweiterungen ohne wesentliche konzeptionelle Änderungen integriert werden können.

Andererseits stellen die Schnittstellen des Managementdienstes Mechanismen zur Kooperation mit dem Managementagenten zur Verfügung. Die in diesem Zusammenhang vorhandenen Methoden sind solche, die Anfragen des Managementagenten entgegennehmen und solche, die Anforderungen an den Managementagenten stellen.

Der Hauptdienst des Managements ist ein AMETAS-Dienst, der sogenannte *ManagerService*, der zusätzliche Schnittstellen besitzt. Dieser Dienst kann alle Vorzüge eines AMETAS-Dienstes genießen. Er verbindet entsprechend seiner Funktionalität den Managementagenten mit der Stelle und den Stellennutzern. Seine Kommunikation mit den anderen Stellennutzern ist asynchron und verläuft über das AMETAS-Postamt.

7.3 AMETAS-Managementagent

Das Schaubild 7.2 stellt den Aufbau des AMETAS-Managementagenten dar. Die AMETAS-Managemententität[1] besteht aus einem MBean-Server und einem Adapter, der eine sichere Kommunikation mit den Managementanwendungen und anderen Managementagenten unterstützt. Darüber hinaus verfügt der Managementagent, wie im vorangegangenen Abschnitt ange-

[1] Sie ist einer Instanz der Klasse `ManagerAgent`, die im Rahmen dieser Arbeit entwickelt wurde.

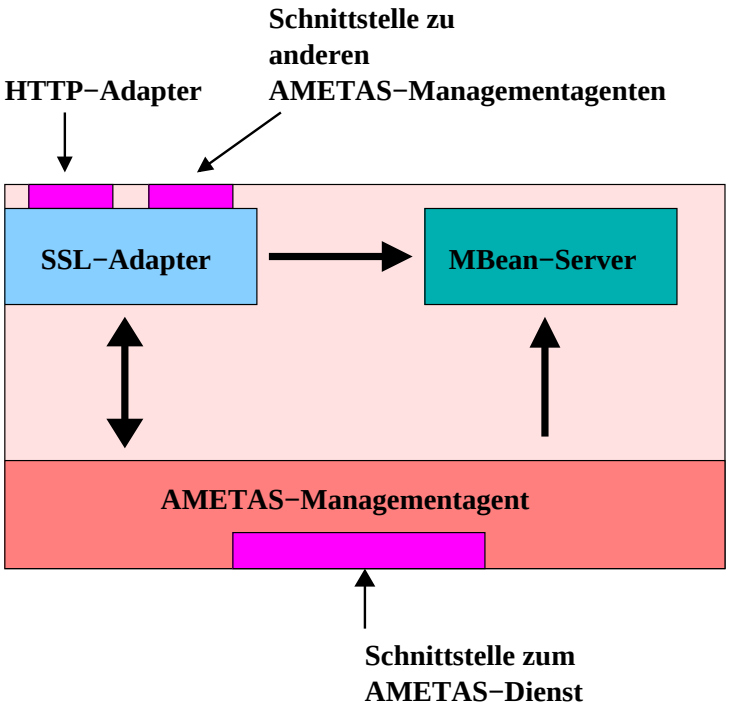


Abbildung 7.2: Aufbau des AMETAS-Managementagenten

sprochen, über Schnittstellen, die die Kommunikation der Managementanwendungen mit den AMETAS-Diensten und Agenten zustande bringen.

Die Hauptaufgabe des Managementagenten ist die Steuerung der Aktivitäten, die mit dem MBean-Server in Verbindung stehen. Diese Aktivitäten bestehen aus:

- Der Erzeugung und Initialisierung des MBean-Servers. Dazu wählt er den Namen der jeweiligen AMETAS-Stelle als Domänenname. Damit ist jede Stelle eine Managementdomäne, und die Organisation dieser Domänen spiegelt die Organisationsform der AMETAS-Stellen. Die Verteilung der AMETAS-Stellen legt fest, wo die Managementdomänen liegen. Bei jeder AMETAS-Stelle können somit die MBean erzeugt und registriert werden. Das schließt aber die Registrierung der entfernten Objekten beim MBean-Server nicht aus. Dazu müssen aber die Mechanismen für den Managementzugriffe vorhanden sein.

- Der Erzeugung von Managementobjekten. Das geschieht gemäß der Konfiguration, die beim Starten des Managementagenten vorliegt. Die managementrelevanten Daten werden entweder durch die von der AMETAS-Infrastruktur zur Verfügung gestellten Werkzeuge neu gesammelt und initialisiert, oder aber die Daten werden im Falle einer persistenten Speicherung der Konfiguration von den Dateien bzw. Datenbanken gelesen. Während der bisherigen Implementierung ist nur die Speicherung der Daten in den persistenten Dateien und nicht in den Datenbanken vorgesehen. Die MBeans können aber auch in der Laufzeit erzeugt und registriert werden.

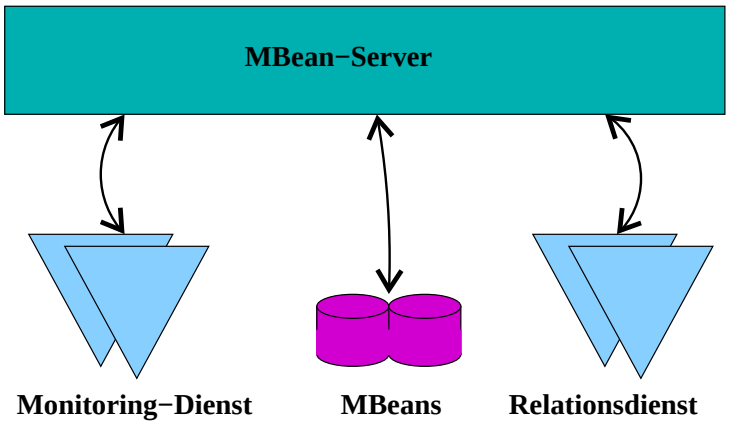


Abbildung 7.3: Aufbau des AMETAS-MBean-Servers

Eine weitere Funktionalität des AMETAS-Managementagenten besteht darin, dass er das Bindeglied zwischen den AMETAS-Diensten und den Managementanwendungen ist. Dazu muss er einerseits mit den Managementanwendungen über einen sogenannten *SSL-Adapter* kommunizieren, der im Abschnitt 7.5 auf Seite 73 näher vorgestellt wird. Andererseits muss er über die dazu entwickelten Schnittstellen mit den AMETAS-Diensten kooperieren.

Viele der Aktivitäten des Managementagenten müssen über einen (oder mehrere) Zugriff(e) auf den MBean-Server erfolgen. Möchte z.B. eine Managementanwendung einen bestimmten Stellennutzer ausfindig machen, wird diese Anfrage zuerst an den MBean-Server gestellt. Wird der gesuchte Stellennutzer im MBean-Server gefunden, wird die geforderte Operation auf ihm ausgeführt. Im negativen Fall[2] bzw. in einem Sonderfall[3] wird die Frage über den Managementagenten an einen Dienst, der die Anfrage bearbeiten kann, weitergeleitet.

Umgekehrt werden die AMETAS-Ereignisse, die für das Management relevant sind, über den Managementagenten an die Managementanwendungen weitergeleitet. Diese Vermittlungen laufen ebenfalls über den MBean-Server. Das Registrieren und Starten von Stellennutzern sind Beispiele für Aktivitäten, die nicht unmittelbar über den MBean-Server erfolgen. Erst die Auswirkung dieser Operationen macht den Zugriff auf den MBean-Server unentbehrlich.

7.4 AMETAS-MBean-Server

Das Schaubild 7.3 stellt den AMETAS-MBean-Server dar. Der AMETAS-MBean-Server ist eine Komponente, die Mechanismen und Werkzeuge zur Manipulation der MBeans zur Verfügung stellt. Seine primäre Aufgabe besteht darin, als Verantwortungsträger für die Registrie-

[2]Dieser Fall kann nur vorkommen, wenn ein Stellennutzer gerade gestartet wurde, oder ein Agent gerade angekommen ist, und die Managementdienste noch nicht auf aktuelle Ereignisse reagiert haben. Sonst sind alle Stellennutzer im MBean-Server registriert.

[3]Ein Sonderfall liegt vor, wenn ein Agent zu lokalisieren ist, der sich nicht in der aktuellen Stelle befindet.

rung aller möglichen MBeans zu agieren. Er implementiert die `MBeanServer`-Schnittstelle[4] (Siehe [JMXIAS], S. 115), die alle für Registrierungszwecke notwendigen Methoden zur Verfügung stellt. Die Registrierung kann durch die Managementanwendungen oder durch eine andere MBean geschehen. Zum Registrieren kann eine neue Instanz der MBean erzeugt werden. Es ist auch möglich, die existierenden Instanzen zu nutzen.

Die Eindeutigkeit der registrierten MBeans in einem MBean-Server wird durch die Instanzen der Klasse `ObjectName` gewährt. Jeder MBean wird beim Registrieren ein eindeutiger *Objektname* zugewiesen, der ein Objekt vom Typ `ObjectName` ist, und vom MBean-Server auf Eindeutigkeit geprüft wird. Die Managementanwendungen benutzen diesen Namen, um die MBeans auf einem MBean-Server zu identifizieren und Managementoperationen auf ihnen auszuführen. Der Objektname spielt also in der JMX die gleiche Rolle wie OID in der OSI-Managementarchitektur.

Ein Objektname besteht aus:

1. einem Domänen-Namen, der ein case-sensitiver String ist und sich auf einen Namensraum in einem JMX-Managementagenten bezieht. Dieser befindet sich in einem globalen Managementkontext.

2. einer Menge von aussagekräftigen Eigenschaften und ihren Werten, die die MBeans beschreiben. Diese Eigenschaften müssen nicht unmittelbare Charakteristiken einer MBean sein, sondern können bei ihrer Benennung mitwirken. Die Liste muss mindestens ein Eigenschafts-Werte-Paar (`property=value`) enthalten.

Die allgemeine Syntax für den Objektnamen ist:

```
[domainName]:property=value[, property=value] *
```

wobei, der Domänenname sich auf den Namen bezieht, der dem MBean-Server bei seiner Erzeugung zugewiesen wurde. Wird aber einem MBean-Server kein Name zugewiesen, so wird ihm in diesem Fall ein vordefinierter Wert zugeordnet, der als `DOMAIN` bezeichnet wird.

Wenn die Eigenschaften eines Objektes lexikalisch sortiert eingetragen werden, so spricht man vom *kanonischen Namen* des Objektes. Man kann die Repräsentation der Objekte durch ihren kanonischen Namen benutzen, um die Objekte im MBean-Server gezielt zu selektieren.[5]

Eine weitere Aufgabe des MBean-Servers besteht darin, folgende Managementoperationen, die von den Managementanwendungen und Administratoren verlangt werden, auf den registrierten Objekten auszuführen:

- Suchen einer bestimmten MBean mit ihrem gegeben Objektnamen

[4]Die API-Dokumentation der JMX gibt einen umfassenden Überblick über die Methoden dieser Schnittstelle. Sie ist verfügbar unter: http://java.sun.com/products/JavaManagement/.

[5]Eine ausführliche Behandlung des Themas findet sich in [JMXIAS], Seite 105ff.

- Suchen einer Menge von MBeans, wobei die Übereinstimmung ihrer kanonischen Namen mit dem gegebenen String-Muster geprüft wird
- Abfrage des Wertes eines oder mehrerer Attribute(s) einer MBean
- Abfrage der Managementschnittstellen einer MBean
- Ausführen einer Operation auf einer MBean
- Registrieren von Ereignisinteressenten bei einer registrierten MBean. Somit erhalten die Listner alle Ereignisse, die von einer bestimmten MBean ausgesendet werden.

Die Schnittstelle `MBeanServer` stellt für die genannten Aufgaben generische Methoden zur Verfügung, die einen Objektnamen als Parameter verwenden. Der MBean-Server muss das Objekt finden, das mit dem gegebenen Objektnamen referenziert ist. Danach muss er feststellen, ob die gewünschte Operation auf der MBean zulässig ist. Im positiven Fall muss er die Operation ausführen und das Ergebnis der Operation an den Aufrufer zurückliefern. Auf diese Weise ist es möglich, lokale und entfernte Operationen auf den MBeans auszuführen.

Eine weitere Aufgabe des MBean-Servers besteht darin, die Ereignisse wie z.B. das Registrieren oder Deregistrieren von MBeans zu emittieren. Diese Ereignisse werden aber nicht durch den MBean-Server selbst gebroadcastet. Diese Aufgabe wird von einem im MBean-Server registrierten Objekt der Klasse `MBeanServerDelegate` mit dem Objektnamen `JMImplemetation:type=MBeanServerDelegate`[6] übernommen. Dieses Objekt identifiziert und beschreibt den MBean-Server. Das registrierte Objekt hat Attribute, die Informationen ausgibt über die eindeutige Identität des MBean-Servers, den Namen und die Versionsnummer der Spezifikation sowie den Hersteller- und Implementationsnamen. Die Abbildung A.2 im Anhang A zeigt die oben genannten Attribute des AMETAS-MBean-Servers im Browser an.

7.4.1 Relationsdienst

Im MBean-Server vom AMETAS ist eine MBean des Relationsdienstes registriert. Der Relationsdienst überwacht nur die Verknüpfung zwischen den registrierten MBeans. Mit den Managementoperationen, die auf den MBeans ausgeführt werden müssen, hat er nichts zu tun. Die Erzeugung von ungültigen Relationstypen wird, z.B. im Falle ungültiger Rolleninformationen, ungültiger Relationen oder ungültiger Objektnamen, verhindert. Wenn eine erzeugte Relation in einem Relationsdienst gelöscht wird, so verlieren alle MBeans, die durch diese Relation verknüpft waren, ihre Beziehung zu einander. Wenn eine MBean deregistriert wird, so wird ihre Rolle von allen Relationen gelöscht, da der Relationsdienst alle Notifikationen des MBean-Servers registriert und dementsprechend reagiert.

[6] Der Domänenname `JMImplementation` ist bei der JMX reserviert. Die Deregistrierung von `MBeanServerDelegate` ist unzulässig.

Eine Einschränkung der Relationsdienstes ist es, dass er die Konsistenz der Beziehung zwischen den erzeugten Relationen nicht überwacht. Mit anderen Worten: Der Entwickler kann die Anzahl der Relationen nicht festlegen, in denen eine MBean als Rolle auftauchen soll. Es ist z.B. denkbar, in der JMX eine Relation «*Stelle läuft in*» mit Rollen «*Stelle*» und «*Rechner*» zu definieren und zwei Instanzen der Relationen für zwei Rechner zu erzeugen. Es ist theoretisch möglich, in der JMX dieselbe Stelle in beiden Relationen einzufügen. Es ist die Aufgabe der Managementanwendungen bzw. des Entwicklers, solche unsinnigen Ergebnisse zu verhindern.

7.4.2 Monitoring-Dienst

Der AMETAS-Monitoring-Dienst beobachtet die Attributswerte von MBeans und deren Änderungen. Dabei gibt es je nach dem Typ der Attributswerte drei Monitorarten:

- Ein `CounterMonitor` beobachtet die Attribute, deren Werte vom Java-Typ `Byte`, `Integer`, `Short` und `Long` sind und sich wie ein Zähler verhalten. Dieser hat einen Wert größer gleich Null, kann nur hochgezählt werden und eventuell überlaufen. Wenn der Wert ein festgelegtes Threshold ereicht, emittiert der Monitor eine Notifikation.

- Ein `GaugeMonitor` beobachtet die Attribute, deren Werte vom Typ `Byte`, `Integer`, `Short`, `Long, Float` und `Double` sind und sowohl hoch als auch runter gezählt werden können. Wenn der Wert eine zuvor festgelegte untere oder obere Grenze erreicht, meldet der Monitor dies in Form einer Notifikation.

- Ein `StringMonitor` beobachtet Attribute vom Typ `String` und emittiert eine Notifikation bei jeder Abweichung des Wertes von einem vorgegebenen String.

7.5 AMETAS-SSL-Adapter

Der AMETAS-SSL-Adapter ist eine Komponente, die Schnittstellen für die Managementanwendungen bereitstellt.[7] Sie fungiert als die einzige Schnittstelle zwischen den Managementagenten und den Administratoren. Diese Schnittstelle ist von der JMX-Spezifikation nicht festgelegt worden. Sie ist den verteilten Diensten zuzuordnen, die im Abschnitt 6.2.3.3 besprochen wurden.

Dieser Adapter ist ein Dienst, der in einem eigenen Thread ausgeführt wird. Der Prozess wird vom Managementagenten gestartet und steht in ständiger Bereitschaft, um die Anfragen der Managementanwendungen entgegenzunehmen und zu bearbeiten. Die Portnummer des

[7] Diese Komponente ist eine Instanz der Klasse SSL-Adapter, die im Rahmen dieser Arbeit entwickelt wurde.

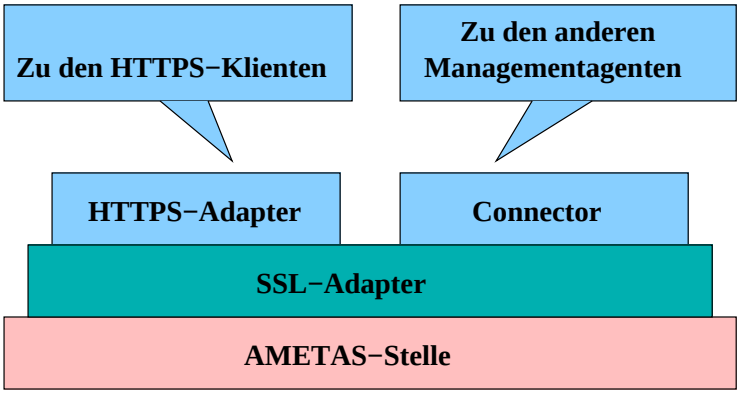


Abbildung 7.4: AMETAS-SSL-Adapter

Dienstes muss in der Parameterdatei der Stelle eingetragen und bei den Anwendungen bekannt gegeben werden.[8]

Wie im Schaubild 7.4 dargestellt, basiert der AMETAS-SSL-Adapter auf SSL, der die Mechanismen für die Sicherheit der zwischen den Managementanwendungen und dem Managementagenten ausgetauschten Daten zur Verfügung stellt. Darauf basierend kann der Adapter im Prinzip jede Art von Anfragen verarbeiten, vorausgesetzt, dass man den Dienst um die entsprechenden Mechanismen erweitert. Während der bisherigen Implementierung des AMETAS-Managements wurde eine Basis für zwei Arten von Schnittstellen geschaffen (siehe Abbildung 7.4) , wobei bisher tatsächlich nur der HTTP-Adapter genutzt wird.

7.5.1 Verschlüsselungstechnik

Secure Socket Layer (SSL) [SSLv3] ist das am weitesten verbreitete Protokoll für die verschlüsselte Datenübertragung im Bereich des WWW. SSL wird vor allem dazu verwendet, eine zuverlässige Authentifizierung durchzuführen und die ausgetauschten Daten vor den Angreifer zu schützen, die die Daten zu lesen oder auf irgendeine Weise zu manipulieren versuchen. Dazu verwendet SSL die sogenannte *asymmetrische* Kryptographie[9] für die Authentifizierung und die sogenannte *symmetrische* Kryptographie[10] für die Verschlüsselung der Nachrichten.

Die symmetrische Kryptographie ist sehr langsam, bietet aber Mechanismen, die eine sichere Identifizierung des gegenüberliegenden Partner erlaubt. Dabei wird das logistische Problem der Schlüsselverteilung dadurch gelöst, dass einer oder beide Partner ein Schlüsselpaar besitzen. Der öffentliche Schlüssel wird von einer zuverlässigen CA beglaubigt. Er enthält die für die sichere Identifizierung notwendigen Informationen. Solche beglaubigten öffentlichen Schlüssel (digitale Signaturen) können jedem an die Hand gegeben werden, der eine sichere,

[8] Die Bekanntgabe der Portnummer kann man vermeiden, indem man z.B. die Portnummer wie die Portnummer der Stelle in der Domänendatei des PNS-Servers (*cs.pns*) einträgt. Für die Anfragen muss der PNS-Server um die Fähigkeit zur Bearbeitung von einigen HTTP-Methoden erweitert werden.

[9] In der Literatur wird sie auch *Public Key*-Kryptographie genannt.

[10] Gelegentlich wird sie *Secret Key*-Kryptographie genannt.

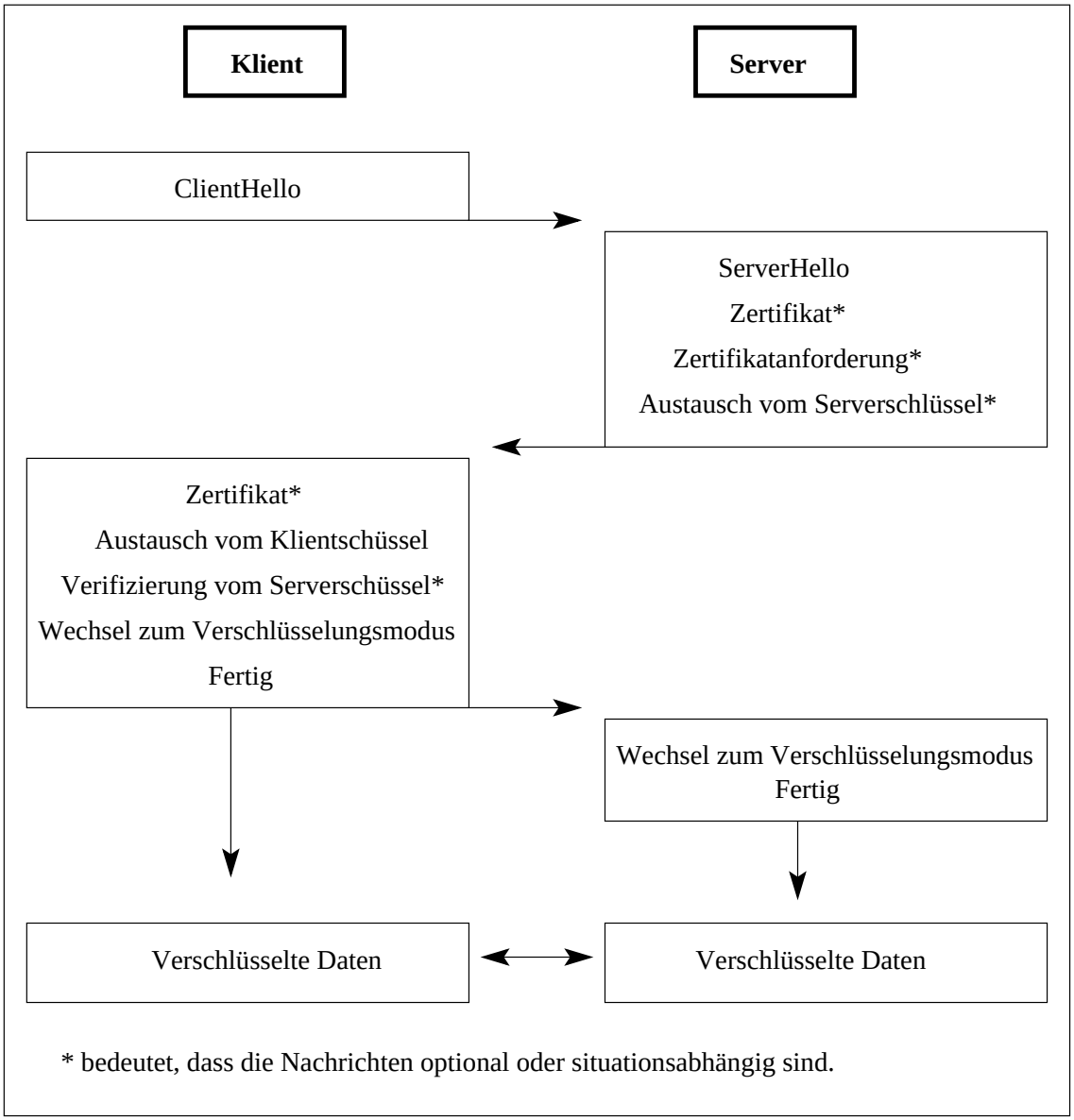


Abbildung 7.5: Der Ablauf des Handshake-Protokolls

verschlüsselte Kommunikation wünscht. Mit Hilfe der asymmetrischen Kryptographie und der kryptographischen *Hash*-Funktion kann ein Geheimschlüssel ausgetauscht werden, der einen Wechsel zur schnellen symmetrischen Kryptographie erlaubt. Anschließend können die auszutauschenden Nachrichten mit einem Schlüssel ver- und entschlüsselt werden.

Der Nachfolger von SSL und zugleich ein standardisiertes Protokoll ist *Transport Layer Security (TLS)*[11] [RFC2246]. Diese Protokoll bietet Mechanismen an, die auf der Basis von TCP/IP eine sichere Kommunikation der unabhängigen Partner erlauben. TLS unterstützt das sichere *Handshake*-Protokoll, das die Authentifizierung und den Austausch des geheimen Schlüssels durchführt. Die Abbildung 7.5 veranschaulicht den Ablauf des Handshake-Protokolls.

[11] Obwohl TLS v. 1.0 auf SSL v. 3.0 basiert, sind die beiden nicht interoperabel (siehe [RFC2246]).

Die im AMETAS-Management verwendete Implementation von SSL und TLS basiert auf der *Java Secure Socket Extension (JSSE)*, Version 1.0.2 [JSSE]. JSSL stellt eine Java-basierte Implementation von SSL v. 3.0 und TLS v. 1.0 zur Verfügung, welche TCP/IP-Basis-Funktionen zur Datenverschlüsselung, Serverauthentifizierung, Datenintegrität sowie zur optionalen Authentifizierung von Klienten zur Verfügung stellt.[12]

7.5.2 HTTP-Adapter

Die Aufgabe des HTTP-Adapters besteht darin, auf SSL-Basis mit den Managementanwendungen über HTTP[13] zu kommunizieren. Er kann alle Protokolldateneinheiten des HTTP-Protokolls entgegennehmen, zur Zeit aber nur eine Untermenge der HTTP-Methoden verarbeiten. Diese Methoden beschränken sich auf die `GET`- und `POST`-Methode. Die Basis ist dabei so gelegt, dass mögliche Erweiterungen ohne wesentliche Schwierigkeiten vorgenommen werden können.

Die Klienten müssen ihre Anfragen konform zu den beiden genannten Methoden formulieren und an den Adapter senden. Nach der Verarbeitung werden die Antworten wieder in Form von HTTP-Paketen zurückgesendet. Darüber hinaus enthalten die Antworten HTML-Elemente. Es wird deutlich, dass nur HTTP-basierte und HTML-fähige Anwendungen wie z.B. Web-Browser in der Lage sind, mit den AMETAS-Managementagenten sinnvoll zu kommunizieren.

Bei der Kontaktaufnahme eines Klienten sendet der SSL-Adapter sein öffentliches Zertifikat an den Klienten. Wenn die Handshake-Phase abgeschlossen ist, erscheint im Browser das in 7.6 dargestellte Bild. Für das Einloggen wird eine Zeichenkette verwendet, die die String-Darstellung der Klasse `AMETASUniqueID` (UID) ist. Die UID muss administrative Rechte (Privileg `ADMIN`) auf der AMETAS-Stelle besitzen.

7.5.3 Connector

Im Rahmen der verteilten Dienste für die JMX-Architektur dienen die Connectoren dazu, die entfernten Managementanwendungen zu befähigen, ortstransparente Managementoperationen auf den MO durchzuführen, die nicht lokal sind. Benötigt werden dazu zwei Connectoren, dabei gilt folgendes:

1. Ein Connector liegt auf der Klientenseite. Dieser nimmt die Anfragen der Managementanwendungen entgegen und leitet sie an die Serverseite weiter. Wenn es einen Rückgabewert von der Serverseite gibt, nimmt er ihn entgegen und leitet ihn an die Managementanwendung weiter.

[12] JSSL ist ein optionales Paket von Java Version 1.2 und 1.3. Sie ist ein Bestandteil von der Version 1.4, die sich zur Zeit in der Beta-Phase befindet.

[13] Aufgrund der Möglichkeiten der SSL-Basis, die den gegenüberliegenden Partner identifizieren lassen, wird zur Zeit das HTTP v. 1.0 verwendet. Wird aber eine höhere Version benötigt, kann die Vereinbarung ohne besondere Schwierigkeiten getroffen werden.

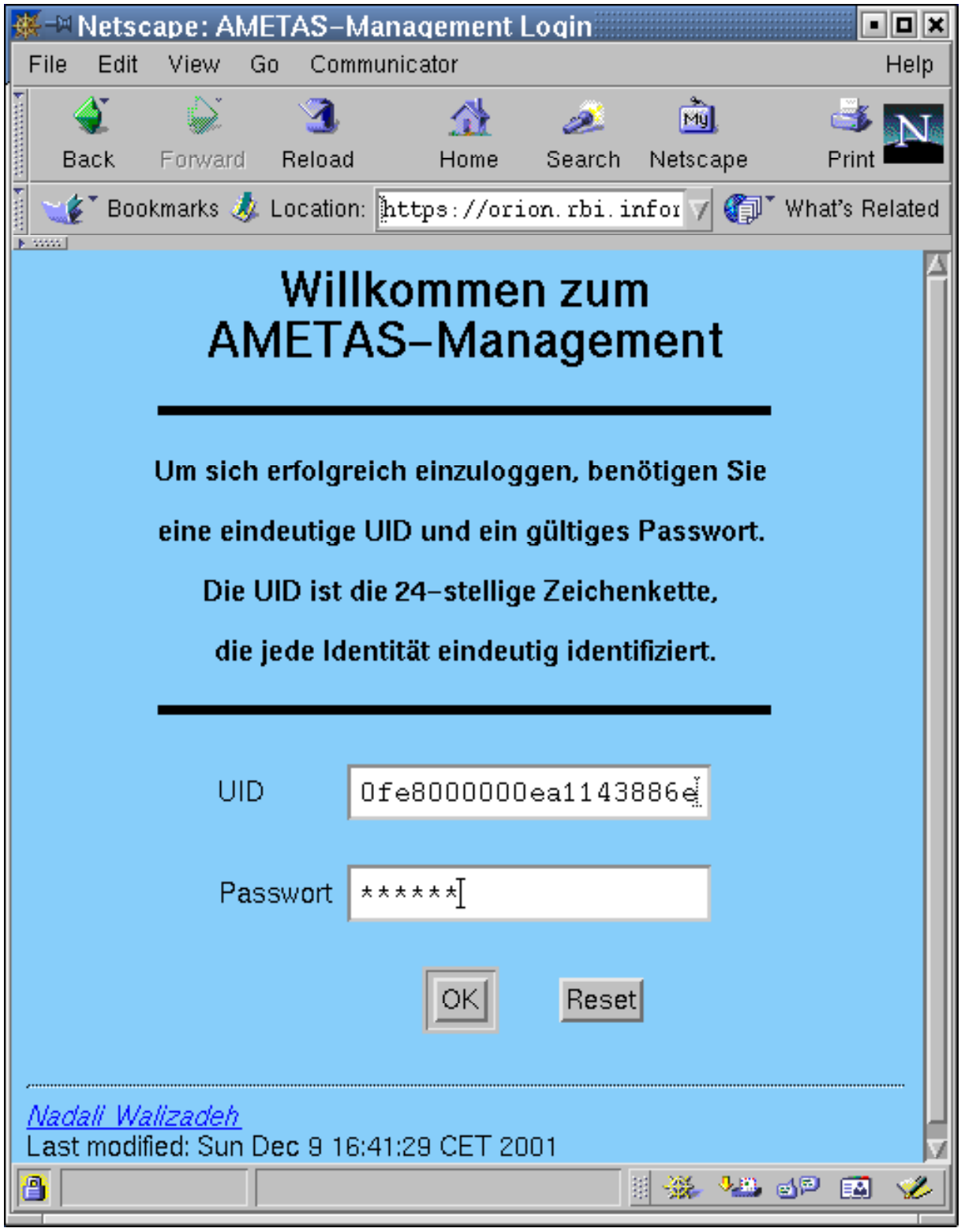


Abbildung 7.6: AMETAS-Management Login-Oberfläche

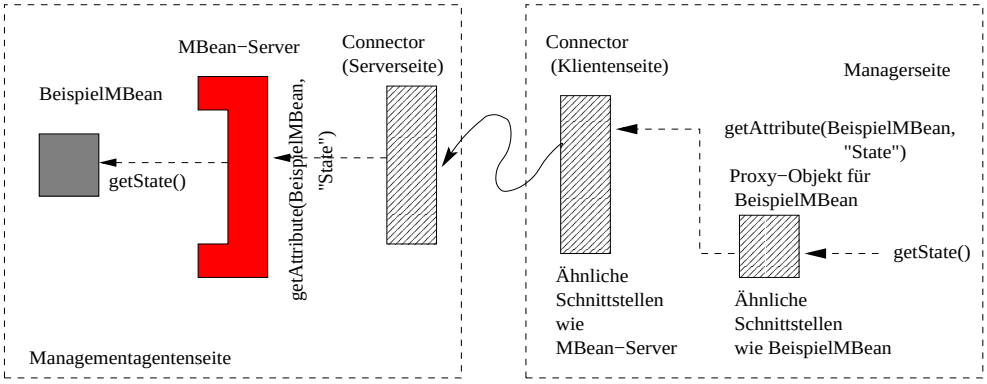


Abbildung 7.7: Beispiel einer entfernten Operation auf einer MBean

2. Der andere Connector liegt auf der Serverseite. Dieser ist für die Entgegennahme der Anfragen der Connectoren auf der Klientenseite zuständig. Er leitet diese Anfragen an den lokalen MBean-Server weiter. Wenn es eine Antwort auf eine Anfrage gibt, nimmt er sie vom MBean-Server an und leitet sie an die Klientenseite weiter.

Hierfür wird eine geeignete Kommunikationsinfrastruktur benötigt, die im AMETAS-Management vom SSL-Adapter zur Verfügung gestellt wird.

Die Funktionsweise einer entfernten Operation auf einer MBean ist in der Abbildung 7.7 veranschaulicht. Möchte man den Wert einer Zustandsvariablen von der Standard-MBean Namens `BeispielMBean` lesen, so kann dies auf zweierlei Weise erfolgen:

- Entweder ruft die Managementanwendung direkt die `getState`-Methode des *Proxy*-Objektes auf. In diesem Fall leitet das Proxy-Objekt den Aufruf durch die beiden Connectoren zum entfernten MBean-Server weiter. Den Rückgabewert erhält das Proxy-Objekt auf dem umgekehrten Weg und gibt ihn an die Managementanwendung zurück.

- Oder die Managementanwendung ruft eine generische Methode des Connectors auf der Klientenseite auf. In diesem Fall fungiert der Connector als Proxy-Objekt für den MBean-Server. Er leitet die Anfrage an die Serverseite weiter und ermittelt auf dem umgekehrten Weg den Rückgabewert.

Im AMETAS-Management könnte eine weitere Verwendung der Connectoren darin bestehen, die Managementagenten zu verbinden, um Managementaufgaben zu erledigen, die in verschiedenen Stellen vorhanden sind. Dazu wurde der SSL-Adapter so implementiert, dass der Managementagent auf der vorhandenen Basis sowohl als Server als auch als Klient agieren kann. Der vom Browser verwendete Mechanismus ist so, dass die Anwendung eine Anfrage initiiert, und der Server mit der Sendung seines öffentlichen Schlüssels reagiert. Der AMETAS-SSL-Adapter erlaubt es, dass ein AMETAS-Managementagent als Klient eine Anfrage an einen anderen Agenten auf SSL-Basis initiieren kann.

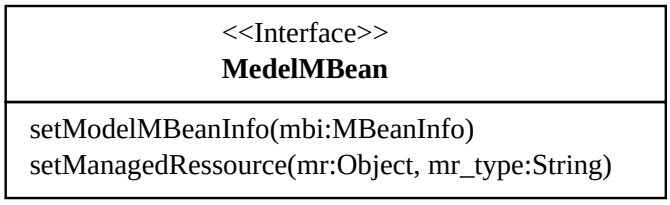


Abbildung 7.8: Die Schnittstelle `ModelMBean`

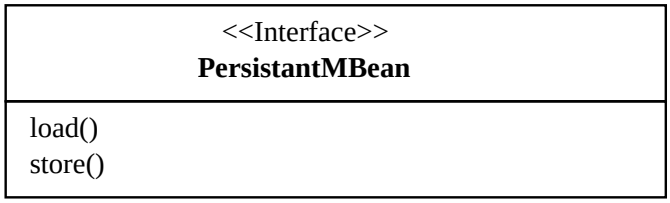


Abbildung 7.9: Die Schnittstelle `PersistantMBean`

Ob es aber tatsächlich zum Einsatz dieses Mechanismus' kommt, ist davon abhängig, wie vorteilhaft er im konkreten Fall ist. Denn es existiert zwischen den AMETAS-Stellen ein vom AMETAS-Kern zur Verfügung gestellter Kommunikationsmechanismus, der auch die Migration der AMETAS-Agenten erlaubt. Zur Zeit werden die Managementaufgaben, die auf einer anderen Stelle zu lösen sind, mit Hilfe mobiler Agenten vorgenommen, die sich in vieler Hinsicht als vorteilhaft erwiesen haben. Deshalb sind die Connectoren, die die Kommunikation der AMETAS-Stellen unterstützen, bisher nicht weiter entwickelt worden.

7.6 Instrumentierung, Registrierung und Darstellung

Für die Instrumentierung der AMETAS-Ressourcen werden ModelMBeans verwendet, die im Abschnitt 6.2.1.1 kurz eingeführt wurden. Mit Hilfe von ModelMBeans kann man sowohl die statischen als auch die dynamischen Eigenschaften der AMETAS-Ressourcen instrumentieren.

Die in der Abbildung 7.8 definierte Schnittstelle `ModelMBean` implementiert die Schnittstellen `DynamicMBean`, `PersistantMBean` und `MBeanNotifikationBroadcaster`[14], die in den Abbildungen 6.4, 7.9 und 7.10 in den jeweiligen UML-Diagrammen dargestellt sind. Eine ausführliche Beschreibung dieser Schnittstellen findet man in [JMXIAS] und in der JMX-API-Dokumentation. Hier sei nur erwähnt, dass die Schnittstelle `PersistantMBean` mit ihren Methoden die dauerhafte Speicherung der MBean-Daten und das Einlesen dieser Daten vom dauerhaften Speicher erlaubt.

Ähnlich wie bei den dynamischen MBeans benötigen die ModelMBeans ihre geeigneten Metadaten-Klassen. Die Metadaten-Klassen für ModelMBeans sind: `ModelMBeanInfo`, `ModelMBeanAttributeInfo`, `ModelMBeanConstructorInfo`, `ModelMBeanOperationInfo`

[14] Diese Schnittestelle implementiert die im Abschnitt 6.2.3.1 genannte Schnittstelle `NotificationBroadcaster`.

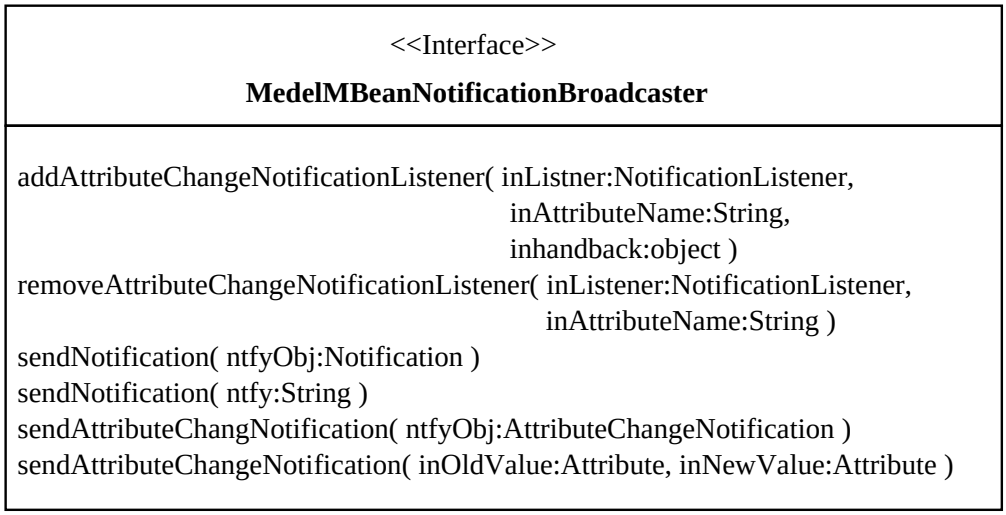


Abbildung 7.10: Die Schnittstelle `ModelMBeanNotificationBroadcaster`

und `ModelMBeanNotificationInfo` (siehe [JMXIAS], S. 73ff.).

Jedes Objekt der genannten Metadaten-Klassen wird mit einem Objekt initiiert, das die Schnittstelle `Descriptor` implementiert. Die *Deskriptoren* sind notwendige Komponenten der ModelMBeans. Sie beinhalten die Managementinformationen über die Komponenten einer MBean und legen somit die dynamischen, erweiterbaren und konfigurierbaren Eigenschaften der jeweiligen Komponenten fest. Außerdem beinhalten die Deskriptoren die Informationen über Verknüpfungen zwischen den Komponenten und den Methoden, die die Komponenten manipulieren.

Die Eigenschaften sind in den Deskriptoren in Feldern gespeichert, welche die Informationen in der Form: *Name=Wert* enthalten. Abgesehen von einigen Feldern, die ausgefüllt werden müssen, sind viele Felder optional. Man kann aber auch eigene Felder definieren und in den jeweiligen Deskriptoren einfügen. Die Tabelle 7.2 zeigt einige mögliche Felder eines Deskriptors, der ein Attribut beschreibt.

Die Bedeutung dieser Felder bringt einige interessante Aspekte des Managements zum Vorschein. Füllt man z.B. die Methodenfelder (*getMethod* und *setMethod*) nicht aus, hat dies zur Folge, dass jede `getAttribute`-Operation den Wert vom Deskriptor liest, und jede `setAttribute`-Operation den neuen Wert in den Deskriptor schreibt. D.h. die eigentlichen Ressourcen sind von diesem Vorgang nicht betroffen. Das kann z.B. für statische Werte sehr sinnvoll sein, denn dadurch werden die Unterbrechungen der zu managenden Ressourcen minimiert.

Mit Hilfe des Feldes für *ProtocolMap* kann man z.B. ModelMBeans auf die OID des SNMP-Managements abbilden. Möchte man z.B. die OID einer AMETAS-Stelle darstellen, kann man folgendes in den Deskriptor der jeweiligen Stelle eintragen:[15]

[15] Die Informationen über die OID der AMETAS-Komponenten wurden aus der Anwendung *AMETAS Place Monitor* entnommen. Dieses Werkzeug wurde von Klaus Herrmann, einem der Entwickler vom AMETAS, zur Verfügung gestellt. Mehr Information darüber findet man unter http://www.ametas.de/docs/PlaceUser/PMon/manual/index.html.

Name	**Wert**
`name`	Name des Attributes (case-sensitive)
`descriptorType`	Ein String, der immer den Wert `"attribute"` hat
`value`	der Wert des Attributes
`default`	Der Rückgabewert, wenn `value` und `getMethod` nicht definiert sind
`displayName`	Beschreibender Name des Attributes
`setMethod`	Name der Operation, die den Wert von `value` liest
`getMethod`	Name der Operation, die den Wert von `value` schreibt
`protocolMap`	Ein Objekt vom Typ `Descriptor`, das die Paare der Form Protokollname=Protokollwert enthält. Dieses Feld verknüpft ein Attribut mit dem standardisierten Identifizierer wie z.B. OID von SNMP-MIB oder CIM-Schema
`persistansPolicy`	`OnUpdate | OnTimer | NoMoreOftenThan | Always | Never`
`persistensPeriod`	in Sekunden. Dieses Feld wird genutzt wenn die `persistensPolicy` auf `OnTime` oder `NoMoreOftenThan` gesetzt ist.
`currencyTimeLimit`	Zeit in Sekunden, die angibt, wie lange der Wert von `value` gültig ist
`lastUpdateTimeStamp`	letzter Zeitpunkt, zu dem der Wert von `value` aktualisiert wurde
`iterable`	ein boolescher Wert, der angibt, ob `value` eine Enumeration ist
`visibility`	Ein Integer-Wert, der die Stufe festlegt, wo die gestufte Managementanwendungen das Attribut beobachten können. Dabei bedeutet Ziffer 1, dass alle Anwendungen das Attribut sehen können
`presentationString`	Ein String im XML-Format, der das Attribut beschreibt.

Tabelle 7.2: Die Felder eines Attributdeskriptors. Die optionalen Felder sind kursiv dargestellt.

```
Descriptor protocolmapDescriptor =
     new DescriptorSupport(new String[] {
        "SNMP=.iso.org.dod.internet.experimental.ametas.place.0"}); bzw.
        "SNMP= .1.3.6.1.3.1.0"});
Descriptor placeDescriptor = new DescriptorSupport();
placeDescriptor.setField("protocolMap", protocolmapDescriptor);
```

Ein weiterer und bedeutender Aspekt betrifft diejenigen Felder eines Deskriptors, die die Werte für den Ort und den Namen der dauerhaften Speicherung der MBean enthalten. Die Informationen solcher Felder werden auch zum Lesen (Schreiben) der Managementdaten z.B. durch *Java Data Base Connectivity (JDBC)* von den (bzw. in die) Datenbanken benötigt.

Im Anhang A wird ein Beispiel für die Instrumentierung einer AMETAS-Stelle vorgestellt.[16] Wie das Beispiel zeigt, sind für die Instrumentierung einer Komponente und ihre Registrierung bei einem MBean-Server folgende Schritte notwendig:

- Für eine AMETAS-Komponente, die durch eine ModelMBean repräsentiert wird, ist ein Deskriptor zu definieren. Der Typ dieses Deskriptors ist ”`mbean`”.
- Die Informationen über alle Eigenschaften einer Stelle wie ihr Name, Hostname, usw. müssen in einer `MBeanAttributInfo` gesammelt werden. Dazu muss jedes Attribut einen eigenen Deskriptor vom Typ ”`attribute`” haben.
- Die Informationen der Managementoperationen, die entweder auf Attributen operieren oder andere managementrelevante Effekte erzielen, müssen in der jeweiligen `MBeanOperationInfo` mit ihren eigenen Deskriptoren zusammengestellt werden. Diese Deskriptoren sind vom Typ ”`operation`”.
- Die Angaben über die `public`-Konstruktoren, die einen Repräsentanten der AMETAS-Komponente darstellen, sind in den jeweiligen `ModelMBeanConstructorInfo` zusammenzustellen. Dazu müssen die Felder der Deskriptoren mit geeigneten Werten konfiguriert werden. Auch diese Deskriptoren sind vom Typ ”`operation`”.
- Die Notifikationen müssen ebenfalls in einer `MBeanNotificationInfo` zusammengefasst werden. Die Deskriptoren hierfür sind vom Typ ”`Notification`”.
- Eine Instanz der `ModelMBeanInfo` ist mit den gesammelten `MBeanAttributInfo`, `MBeanOperationInfo`, `MBeanConstructorInfo`, `MBeanNotificationInfo` und dem Deskriptor vom Typ ”`mbean`” zu erzeugen.

[16] Dieses Beispiel spiegelt nicht alle Eigenschaften einer Stelle. Es dient nur der Demonstration der Instrumentierung der Attribute, Operationen, Konstruktoren und Notifikationen.

- Für jede Komponente ist ein aussagekräftiger Namen auszusuchen. Z.B. kann jede AMETAS-Stelle den Namen "`place`" haben. Mit diesem Namen und der eigentlichen Domäne, die eindeutig ist, muss man ein Objekt vom Typ `ObjectName` instantiieren.
- Assoziiert mit diesem Namen ist ein Objekt des Typs `ModelMBean` beim MBean-Server zu erzeugen.
- Durch den MBean-Server muss man weiterhin die ModelMBeanInfo und das Objekt, das die Komponente repräsentiert, dem Objekt zuordnen, das beim MBean-Server erzeugt wurde.

Wenn man den Managementdienst auf einer Stelle startet und sich erfolgreich über ihn bei dieser Stelle einloggt, stellen sich die Managementinformationen, die man wie im Anhang A konfiguriert hat, wie das Schaubild A.3 im Browser dar.

7.7 Ausblick

Das AMETAS-Management ist ein offenes Management, das mit anderen Standards kooperieren kann. In dieser Hinsicht besteht die Möglichkeit, geeignete Schnittstellen zu gegebenen Standards anzubieten. Sun Microsystems verfeinert ständig die Möglichkeiten der JMX. Sofern diese Entwicklung anhält, erhöhen sich die Möglichkeiten des AMETAS-Managements, mit den gängigen Managementstandards zu kooperieren.

Die Instrumentierung der AMETAS-Komponenten erfordert weitere Konfigurationsschritte. Besonders die managementrelevanten Komponenten und ihre Eigenschaften sind noch festzulegen. Weiterhin sind einige im Konfigurations- und Systemsicherheitsbereich existierende Werkzeuge ins Management zu integrieren. Dazu bietet die Managementinfrastruktur eine gute Grundlage.

In den folgenden Kapitel werden das Fehlermanagement und die Lokalisierung von Agenten behandelt. Es gibt aber weitere Problembereiche, die noch genauere Überlegungen verlangen. Insbesondere im Leistungs- und Abrechnungsbereich sind weitere Studien mit dem Ziel der Entwicklung zusätzlicher Werkzeuge erforderlich.

Kapitel 8

Lokalisierung von Agenten

Mobile Agenten sind aktive Entitäten, die sich autonom durch ein Netzwerk von Stelle zu Stelle bewegen, um ihre Aufgaben zu bearbeiten. Die Anwender erteilen ihren Agenten bestimmte Aufgaben und starten sie von irgendeinem Rechner im Netzwerk. Solange die Agenten noch nicht zurückgekehrt sind, hat der Anwender fast keine Informationen über seine Agenten. Er weiß nicht einmal, ob seine Agenten noch am Leben sind[1]. Insbesondere haben Anwender manchmal den Wunsch, mit ihren Agenten zu kommunizieren. Wenn die Informationen über den aktuellen Aufenthaltsort eines Agenten nicht vorhanden ist, wird die Kommunikation mit dem Agenten wenn nicht unbedingt unmöglich, so doch möglicherweise sehr teuer. Die gleiche Schwierigkeit stellt sich, wenn der Anwender einen Agenten terminieren möchte, weil die Aufgabe des Agenten nicht mehr relevant ist.

Während die Forschung im Bereich der Infrastruktur, Kommunikation und Sicherheit der mobilen Agentensysteme vorangegangen ist, wurde die Frage der *Kontrollmechanismen* für mobile Agenten vernachlässigt. Die meisten Forschungsgruppen haben dieses Problem nicht behandelt. Lösungen für diese Frage haben im Rahmen ihrer Agentensysteme bis jetzt nur *Mobile Agent System Interoperability Facilities (MASIF)*, *Aglets Workbench* [AGLETS] und *Mole* [MOLE] vorgelegt.

Bevor diese Beispiele näher vorgestellt werden, soll der Begriff Kontrollmechanismus näher beleuchtet werden, da er eine Schlüsselrolle in den nachfolgenden Abschnitten spielt:

1. **Lokalisieren von Agenten.** Es handelt sich um einen Mechanismus, der den aktuellen Aufenthaltsort eines Agenten bekannt gibt. Ansonsten gibt er die Feststellung bekannt, dass der gesuchte Agent im System nicht existiert. In vielen Fällen ist es notwendig, diesen Mechanismus anzuwenden, um sich zu vergewissern, ob ein Agent noch am Leben ist und in welcher Stelle er sich befindet. Wenn man mit einem Agenten kommunizieren will, ist es sinnvoll, den Agenten zunächst zu lokalisieren. Dadurch können die Kommunikationskosten reduziert werden. Es gibt allerdings eine Art der Kommunikation, die die

[1] Durch beobachten der Log-Datei der Stellen kann man im AMETAS Indizienbeweise darüber enthalten, ob ein Agent noch aktiv ist.

Lokalisierung von Agenten nicht benötigt. Dieser Art der Kommunikation ist eine anonyme Gruppenkommunikation, die in erster Linie von einem Ereignissystem unterstützt wird. Im Allgemeinen verursacht diese Art der Kommunikation mehr Kosten als die auf der Basis der Lokalisierung der Agenten.[2]

2. **Terminieren von Agenten**. Es handelt sich um einen Mechanismus, der die Funktionalität bereitstellt, einen Agenten zu stoppen und vom System zu entfernen. Dabei bleibt der aktuelle Aufenthaltsort des Agenten transparent. Dieser Mechanismus ist notwendig, wenn ein migrierender Agent Fehlfunktionen ausführt, oder die Resultate der Arbeit eines Agenten nicht mehr relevant sind. Eine mögliche Lösung dieses Problems ist es, einen Agenten zu lokalisieren, und ihm dann die Terminierungsnachricht zu senden. Es ist aber kostengünstiger, wenn man die Lokalisierung mit der Terminierung in einem Schritt erledigen kann.

In [BAUJ99] (S. 18ff.) wird außer den beiden genannten Mechanismen auch noch die *Erkennung von Waisen* zu den Kontrollmechanismen gezählt. Dieser Mechanismus wird im Unterabschnitt 8.3 erläutert.

8.1 MASIF

MASIF ist ein Standard für mobile Agentensysteme, die auf der OMG-Technologie basieren. Sie wurde von OMG vorgelegt.[3]

MASIF schlägt eine Liste von Techniken vor, die für die Lokalisierung von mobilen Agenten nützlich sein können. Dabei wird vorausgesetzt, dass der Pfad der Migration der Agenten im voraus nicht bekannt ist. Diese Techniken, die als Methoden in dem Interface *MAFFinder*[4] vorgelegt wurden, sind:

- **Brute Force**-Methode. Bei dieser Technik wird jedes Agentensystem in einer bestimmten *Region* durchsucht, um einen bestimmten Agenten zu lokalisieren. Dazu schickt man z.B. einen Agenten los, der alle Stellen des Systems durchsuchen muss.

- **Logging.** Wenn ein Agent eine Stelle verlässt, hinterlässt er Spuren, die darauf hinweisen, wohin der Agent migriert. Wenn man diese Spuren verfolgt, kann man den Agenten irgendwann finden. Diese Methode kann auch für die Spurenbeseitigung verwendet werden, wenn ein Agent stirbt.

[2] Siehe [BHRRS97] und [BAUJ99], S. 19.

[3] MASIF ist ein Standardisierungsvorschlag, der versucht, die verschiedenen Aspekte der (mobilen) Agentensysteme zu standardisieren. Seit März 1999 wird MASIF in enger Zusammenarbeit mit der FIPA vorangetrieben.

[4] Siehe [MASIFS], S. 3ff. und [MIL ET AL.98], S. 61f..

- **Registrieren von Agenten**. Jeder Agent registriert sich in einer Datenbank, die die aktuelle Information über die Stelle hat, in der sich der Agent befindet. Für diese Funktionalität muss der Agent mehr Programmzeilen enthalten als sonst. Allerdings kann eine zentrale Registrierungsdatenbank zu einem Flaschenhals werden.

- **Advertising** (*Agent Advertisement*). Man registriert die Stellen, in denen die Agenten ausgeführt werden. Die Agenten werden nur registriert, wenn sie die Registrierung verlangen. Für ihre Suche muss man, wenn sie nicht registriert sind, die erste Methode anwenden.

Die Rolle des MAFFinder ist es, den Namensdienst in CORBA um die Mechanismen zu erweitern, die die mobilen Agenten unterstützen können. Die Erweiterung muss insbesondere in der Lage sein, die notwendigen Daten der neuen Agentenstelle wie den Rechnernamen und die Portnummer in den Objektnamen des Agenten einzubinden.

Diese Methoden können in jedem Agentensystem eingesetzt werden. Die Brute Force-Methode ist sehr teuer, und die Suche erfordert Broadcasting, was insbesondere große Netze belasten kann. Das Problem beim Logging ist die Spurensicherung und die Verwaltung der Log-Dateien. Ob die Aufbewahrung der Log-Informationen in den Datenbanken geschieht oder aber in den Hauptspeichern, stellt ein weiteres Problem dar.

Ein kritischer Punkt beim Registrieren und auch beim Advertising ist die Vorgehensweise. Wenn man die Agenten dazu zwingt, bedeutet dies eine Einschränkung ihrer Autonomie. Eine andere Möglichkeit ist es, in ihrer Vertretung ihre Registrierung vorzunehmen, ohne ihre Autonomie zu beeinflussen. Weiterhin spielt die Verwaltung der registrierten Agenten eine Rolle. Alternativ stehen die zentralistische und verteilte Verwaltung der Daten zur Verfügung, die im konkreten Fall abgewogen werden müssen.

8.2 Aglets Workbench

Aglets Workbench wurde im IBM Tokyo Research Laboratory entwickelt und ist eine Kombination von Java-Applets, die in einem Agentenmodell um die Mobilität erweitert wurde. Dieses Agentensystem hat folgende Bestandteile:

1. Agentenserver, die eine dauerhafte Netzwerkverbindung haben. Ein Agentenserver hat eine Ausführungsumgebung (genannt *Kontext*), in der die Agenten erzeugt, gestartet und ausgeführt werden können. Er kann die Agenten migrieren lassen und sie empfangen.

2. Klienten, die eine Ausführungsumgebung für die Erzeugung der Agenten bereitstellen. Sie können die lokalen oder entfernten Agenten steuern.

Die Agenten werden mit einem Namensdienst genannt `AgletProxy` global identifiziert und kommunizieren über den Austausch von Nachrichten. Als Kontrollmechanismen für diese mobilen Agenten stellt [AYOM98] (S. 42f.) Methoden zur Lokalisierung von Agenten vor, die einigen von MASIF vorgeschlagenen Techniken ähnlich sind. Diese Techniken sind Brute Force, Logging und Registrieren, die oben erläutert wurden. Es wird wie bei MASIF vorausgesetzt, dass sich der Pfad der Migration dynamisch ändern kann.

8.3 Mole

Mole ist eine rein Java-basiertes Agentensystem, das an der Fakultät für Informatik an der Universität Stuttgart entwickelt wurde. Die Agenten in Mole werden in einer Umgebung ausgeführt, die *Platz* genannt wird. Mole-Agenten sind aktive Entitäten, welche sich von einem Platz zu einem anderen bewegen, um anderen Agenten zu treffen und die Dienste der Plätze bzw. der anderen Agenten in Anspruch zu nehmen. Diese Agenten wurden Multi-Thread entworfen, ihr Zustand und Code wird bei der Migration mittransportiert.

Mole kennt zwei Typen von Agenten: mobile und stationäre. Die stationären Agenten ermitteln die Dienste an einem Platz. Jeder Agent hat einen globalen Agentenidentifizierer, der sich bei der Migration nicht ändert. Ein Agent kann entweder von einer Anwendung, die nicht zum Agentensystem gehört, oder von einem anderen Agenten erzeugt werden.[5] Mole sieht verschiedene Kommunikationsmechanismen im Agentensystem vor:

- Agent/Dienst-Interaktion. Sie basiert auf einem lokalen oder globalen RPC-ähnlichen Mechanismus. Diese Art der Kommunikation ist typisch für die Interaktion zwischen mobilen Agenten und stationären Dienstagenten.
- Interaktion zwischen mobilen Agenten. Sie ist eine *Peer to Peer*-Interaktion, die auf höheren Protokollen wie z.B. KQML basiert.
- Anonyme Gruppenkommunikation. Sie ermöglicht es, mit einer Gruppe von Agenten, die irgendeinen gemeinsamen Schnittpunkt haben, über Nachrichten oder die Erzeugung von Ereignissen zu kommunizieren.
- Kommunikation zwischen mobilen Agenten und menschlichen Anwendern. Sie wurde bisher nicht spezifiziert.

Für die Lokalisierung von Agent hat man das sogenannte *Pfadkonzept* vorgelegt, das auf der Verfolgung der Spuren der Agenten basiert. Für eine implizite Terminierung der Agenten wurde das sogenannte *Energiekonzept* vorgeschlagen. Basierend auf Pfad- und Energiekonzept hat

[5] Ausführliche Informationen über Mole findet man in [MOLE], [BHRRS97] und [BAUJ99].

man dann eine dritte Lösung konzipiert: Das *Schattenkonzept*. Dieses spricht das Lokalisierungs- und das Terminierungsproblem der Mole-Agenten an. Sowohl das Energie- als auch das Schattenkonzept in Mole basieren auf dem sogenannten *Abhängigkeitsprinzip,* das ihre Realisierung ermöglicht. Deshalb wird dieses Prinzip kurz besprochen, und danach werden die eigentlichen Kontrollmechanismen im Mole behandelt.

8.3.1 Abhängigkeitsprinzip

Die Idee der Abhängigkeit stammt von den Beziehungen zwischen den Objekten, die in einem System kooperieren. Solche Beziehungen sind z.B.

- Klient/Server-Beziehung. Sie basiert auf einer Anforderung des Klienten und der zugehörigen Antwort des Servers. Dabei ist die Aktivität eines Servers von den Anfragen der Klienten abhängig.
- Eltern/Kind-Beziehung. Sie ist zwischen den Prozessen in einem verteilten System zu beobachten. In dieser Beziehung hat der Kind-Prozess immer einen Eltern-Prozess, der ihn erzeugt hat.
- Referenz-Beziehung. Sie ist in der verteilten *Garbage Collection* zu beobachten. In dieser Beziehung werden Objekte, die irgendwann gebraucht werden könnten und direkt oder indirekt von den sogenannten Root-Objekten erreichbar sind, nicht gelöscht. Wenn sie nicht erreichbar sind und nicht gebraucht werden, werden sie vom System entfernt.

Eine fehlende Abhängigkeitsbeziehung bei einem Objekt bedeutet, dass das Objekt als *Waise* bezeichnet wird und vom System entfernt werden kann. Somit wird im Mole ein Abhängigkeitsobjekt eingeführt, das in [BAUJ99] (S. 4) folgendermaßen definiert wird:

> Ein Abhängigkeitsobjekt ist ein Objekt, das für ein anderes Objekt eine Elternbeziehung zur Verfügung stellt.

Das Abhängigkeitsobjekt kann ein Agent, ein spezifisches Objekt auf einem Platz oder der Platz selbst sein. Ein Agent kann von mehreren Objekten abhängig sein, er muss aber wenigstens zu einem Objekt eine Abhängigkeitsbeziehung pflegen, sonst hat er kein Existenzrecht und wird vom System entfernt.

Dieses Konzept bedeutet eine zusätzliche Komplexität für ein Agentensystem. Ein interessanter Aspekt ist es, die Idee der Abhängigkeit aus der Sicht der Agentenforschung zu betrachten. Es kann dabei festgestellt werden, dass die mobilen Agenten nicht immer nach ihrer inneren Logik agieren können, die vom Agenteninhaber gemäß der eigentlichen Aufgaben programmiert wurde. Sie benötigen vielmehr die Unterstützung eines äußeren Objekts, das wahrscheinlich mit dem eigentlichen Auftrag des Agenten gar nichts zu tun hat.

Waisenerkennung

Basierend auf dem Abhängigkeitsprinzip wurde der Mechanismus der Waisenerkennung eingeführt. Dabei ist ein Agent ein Waise, wenn kein Abhängigkeitsobjekt für diesen Agenten existiert. Wie bereits angedeutet, wird dieses Prinzip von Mole als ein Kontrollmechanismus für Mole-Agenten eingestuft. Das liegt daran, dass mit der Waisenerklärung der mobilen Agenten ihre Terminierung veranlasst werden kann. Die Funktionalität dieses Prinzips kann bei der Zuordnung der vorhandenen Ressourcen zu Agenten behilflich sein: Die Ressourcen können sorgfältig verteilt und ihre Verschwendung verhindert werden.

8.3.2 Energiekonzept

Das Energiekonzept stellt die Funktionalität für die Waisenerkennung in Mole bereit. Beim Start eines Agenten wird ihm eine bestimmte Menge Energie gewährt. Diese wird verbraucht, wenn der Agent die Ressourcen und Dienste des Systems in Anspruch nimmt. Nach der Abnahme der Energie eines Agenten kann er von seinem Abhängigkeitsobjekt neue Energie anfordern. Die Feststellung, wie viel Energie er noch zum Bearbeiten eines Auftrags braucht, liegt in seiner eigenen Verantwortung bzw. in der seines Programmierers. Diese Feststellung geschieht mit einer Threshold-Funktion.

Alle Plätze, an denen Agenten ausgeführt werden, kontrollieren in bestimmten Zeitintervallen die laufenden Agenten und erniedrigen die Energiemenge dieser Agenten entsprechend ihrem Verbrauch von CPU, Hauptspeicher, usw. Die Agenten können sich schlafen legen, wenn sie nichts zu tun haben oder auf irgendein Ereignis warten müssen. So sparen sie Energie. Sie können bei Bedarf wieder aufgeweckt werden. Dieser Mechanismus ermöglicht die Erkennung von Waisen ohne zusätzliche Kommunikation. Die notwendigen Informationen dafür sind immer lokal vorhanden.

Ein Nachteil verbunden mit diesem Mechanismus ist die Tatsache, dass die Energieanforderung eine Verbindungsaufnahme des Agenten mit seinem Abhängigkeitsobjekt erfordert. Der Agent wird zu dieser Aktivität gezwungen. Ein weiterer Kritikpunkt ist die Fehleranfälligkeit dieses Konzepts. Es gibt drei Schlüsselrollen, deren Funktionstüchtigkeit zu gewährleisten ist: Die Plätze, an denen der Agent und sein Abhängigkeitsobjekt ausgeführt werden, das Abhängigkeitsobjekt selbst sowie der Kommunikationskanal. Wenn einer der Plätze abstürzt, kann ein Agent keine Energie mehr erhalten und stirbt. Das gleiche gilt, wenn das Abhängigkeitsobjekt seinerseits nicht mehr existiert. Ein Agent muss aber auch sterben, wenn die Kommunikationswege mit seinem Abhängigkeitsobjekt gestört sind.

8.3.3 Pfadkonzept

Das Pfadkonzept stellt die Funktionalität für die Lokalisierung und Terminierung von Agenten bereit und ist eine Variante des Logging-Konzepts. Gemäß dieses Konzepts hinterlassen die mobilen Agenten bei der Migration zwischen den Plätzen ihre Spuren. Wenn der Platz bekannt ist, an dem ein bestimmter Agent erzeugt wurde, kann man ihn durch die Verfolgung der hinterlassenen Spuren finden. In diesem Konzept werden die Pfadinformationen nicht in einer Datenbank oder einem Festspeicher gehalten. Vielmehr werden sie im Hauptspeicher und flüchtig gehalten. Folglich gehen die Pfadinformationen verloren, wenn das System beendet wird oder abstürzt.

Eine Spur wird hinterlassen, wenn ein Agent einen Platz verlässt. Beim Ankommen an einem Platz braucht er keine Spuren zu hinterlegen. Mole sieht dabei vor, dass als Spur die eindeutige Identität des Agenten und der Zielplatz hinterlegt werden. Für die Lokalisierung eines Agenten muss man angefangen vom Startplatz feststellen, ob der Agent noch präsent ist. Im negativen Fall muss man feststellen, ob noch irgendwelche Spuren da sind. Danach wird in dem Platz weitergesucht, der in der Spur als nächster vermerkt wurde. Zum Terminieren eines Agenten kann man die Lokalisierung und Terminierung in einem Auftrag abwickeln.

Dieses Konzept schränkt die Autonomie der Agenten hinsichtlich der Abhängigkeiten nicht ein, so dass sie ohne Vorankündigung wandern können. Ein Problem verbunden mit diesem Konzept ist allerdings die Aufbewahrung der Spuren: Die Schwierigkeit ist die Feststellung, wie lange können die Spuren gespeichert werden müssen. Dabei spielt die Verfügbarkeit der beteiligten Objekten, die die Spuren sichern und aufbewahren, eine entscheidende Rolle. Wenn eines dieser Objekte, z.B. ein Platz, ausfällt, und die Pfadinformationen verloren gehen, so ist die Rekonstruktion des Pfads wenn nicht unmöglich, so doch sehr schwierig. Deshalb ist es in solch einer Situation wesentlich einfacher, einen Agenten mit Hilfe der Brute Force-Methode zu lokalisieren. Die Rekonstruktion des Pfads kann anderenfalls sehr teuer werden.

8.3.4 Schattenkonzept

Dieses Konzept basiert wie das Energiekonzept auf dem Abhängigkeitsprinzip, wobei das Abhängigkeitsobjekt ein *Schatten* ist. Es handelt sich dabei um eine bestimmte Datenstruktur. In diesem Konzept werden die Energie- und Pfadkonzept kombiniert und vereinigt. Beim Start jeder Anwendung wird ein bzw. werden mehrere Schatten erzeugt. Die Anwendung und ihr Schatten müssen nicht auf dem selben Platz ausgeführt werden. Die Agenten, die von der Anwendung gestartet werden, sind nicht von der Anwendung, sondern von dem Schatten abhängig.

In bestimmten regulären Intervallen kontrolliert das System, ob die Schatten der laufenden Agenten noch existieren. In dieser Phase dürfen die Agenten nicht migrieren. Existiert der Schatten eines Agenten, wird dem Agenten im Unterschied zum Energiekonzept statt Maßgaben zum Ressourcenverbrauch ein neues Zeitquantum als Einschränkungsfaktor gewährt. Wenn für

einen Agenten keinen Schatten mehr existiert, wird er Waise. Wenn der Schatten eines Agenten nicht erreichbar ist, werden die Versuche so lange wiederholt, bis eine bestimmte Anzahl überschritten wird. In diesem Fall stirbt der Agent nach dem Verbrauch seines verbliebenen Zeitquantums.

Werden neue Kinder-Agenten durch die Agenten erzeugt, ordnet das System den Kinder die Schatten und die verbliebenen Zeitquanten der Eltern zu. Würde das System den Kindern das volle Zeitquantum gewähren, könnten die Agenten unendlich lange leben, in dem sie kurz vor dem Ende ihres Zeitquantums eine Kopie von sich selbst mit dem vollen Zeitquantum anlegen könnten.

Wird dieses Konzept mit dem Pfadkonzept kombiniert, so erreicht man dadurch die folgende Optimierung:

Die Schatten aktualisieren ihre Informationen über den aktuellen Platz, in dem sich ein Agent aufhält, bei jeder neuen Anforderung eines Zeitquantums durch den Agenten. Dabei legen sie einen neuen Pfad an, dessen Startpunkt auf dem aktuellen Platz des Agenten liegt, von dem aus der Agent das Zeitquantum angefordert hat. Der alte Pfad wird nicht mehr benötigt und gelöscht. Gehen die Informationen über einen Pfad verloren, wird die Ungewissheit über den aktuellen Platz eines Agenten nur bis zur nächsten Anforderung eines Zeitquantums seitens des Agenten andauern.

Basierend auf dem geschilderten Schattenkonzept hat man hierarchische und mobile Schatten eingeführt. In einem hierarchischen Schattenkonzept nehmen die Schatten selbst analog zu den Agenten die Rolle der abhängigen Objekte an. Dabei spielt ein einziger Schatten die Rolle eines Anwendungsschattens und die übrigen Schatten sind von ihm abhängig. Ein mobiler Schatten hingegen kann gemäß einer vorprogrammierten Strategie zu den Plätzen migrieren, von denen aus er seine Anwendungen und Agenten besser erreichen kann.

Der Vorteil dieses Konzepts ist, dass die Agenten schnell gefunden werden können. Es wird zu jedem Zeitpunkt nur ein Teil der Informationen über den gesamten Migrationspfad aufbewahrt. Das erleichtert den Aufwand zum Management der Log-Informationen erheblich. Ein Nachteil dieser Methode ist die Komplexität, die dem Agentensystem durch sie zusätzlich hinzugefügt wird. Die Anwendungen benötigen immer ihre Schatten, was eine weitere Belastung des Laufzeitsystems bedeutet. Ein wesentlicher Kritikpunkt dieser Methode ist aber, dass die Agenten sterben müssen, wenn sie keine Verbindung mehr zu ihren Schatten aufbauen können. Ob sie ihre Aufgabe beendet haben, spielt dabei keine Rolle. Es stellt sich die Frage, ob ein Agentensystem nützlich ist, bei dem Netz- bzw. Systemausfälle eine Agentenanwendung nachhaltig behindern können? Eine wesentliche Motivation für die Entwicklung eines Agentenparadigmas ist seine angestrebte Robustheit gegen Netz- und Systemausfälle. In Mole wird dieses allgemeine Problem teilweise auf das Agentensystem übertragen. Ein weiterer Nachteil betrifft die Autonomieeigenschaft der Agenten. Diese Eigenschaft ist für die Bereiche, in denen die mobilen Agenten zum Einsatz kommen, von entscheidender Bedeutung. Die Autonomie

wird unter anderem dadurch abgesichert, dass die Agenten nicht (permanent) in Verbindung mit ihrem Erzeuger oder einer anderen Autorität stehen müssen. Ein Agent ohne regelmäßige Verbindung zu seinem Schatten ist in Mole ein toter Agent.

8.4 Zusammenfassung

Die in den vorherigen Abschnitten behandelten Ansätze von MASIF, Aglets Workbench und Mole stellen die wesentlichen Konzepte dar, die Kontrollmechanismen für mobile Agenten anbieten. Die Abbildung 8.1 zeigt einige mögliche Ansätze, ihre Kategorisierung und Beispiele der Agentensysteme, die die Ansätze implementiert haben.

Geht man davon aus, dass der Migrationspfad der Agenten im voraus nicht bekannt ist, so kann man die Ansätze in deterministische und nicht-deterministische unterscheiden. Beispiele für nicht-deterministische Ansätze sind das Advertising und das Energiekonzept. Die deterministischen Ansätze ohne Log-Informationen können mit Hilfe des Broadcastings implementiert werden. Dies ist aber aufgrund der vielen dadurch verursachten Nachrichten zu vermeiden. Allerdings gibt es Sondersituationen, die die Verwendung des Broadcastings unvermeidbar machen.

Viel interessanter sind die Ansätze, die zusätzliche Informationen über den Migrationspfad der Agenten speichern. Dabei unterscheiden sich die Ansätze, die die Log-Informationen permanent in Datenbanken und sonst wo speichern von den Ansätzen, die diese Informationen flüchtig speichern.

In Betracht gezogen werden müssen auch die Ansätze, die die Agenten registrieren lassen und dadurch jederzeit die aktuellen Stellen festhalten, an denen sich die Agenten aufhalten. Mit dieser Methode kann ein Agent sofort lokalisiert werden, da die Auswertung der Informationen vorher stattgefunden hat.

Bei allen Ansätzen müssen zwei Faktoren beachtet werden: Die Kommunikation mit den mobilen Agenten und die Unterstützung der Laufzeitsysteme. Bei der Logging-Methode ist es z.B. entscheidend, dass die Spurinformationen gesichert werden. Diese Aufgabe kann vom Laufzeitsystem übernommen werden, oder der Agent bearbeitet es selbst. Das gilt ebenfalls für die Registrierung. Der Agent kann sich mit seinen Kommunikationsmitteln registrieren, oder aber er wird stellvertretend vom Laufzeitsystem oder von einem anderen Agenten registriert.

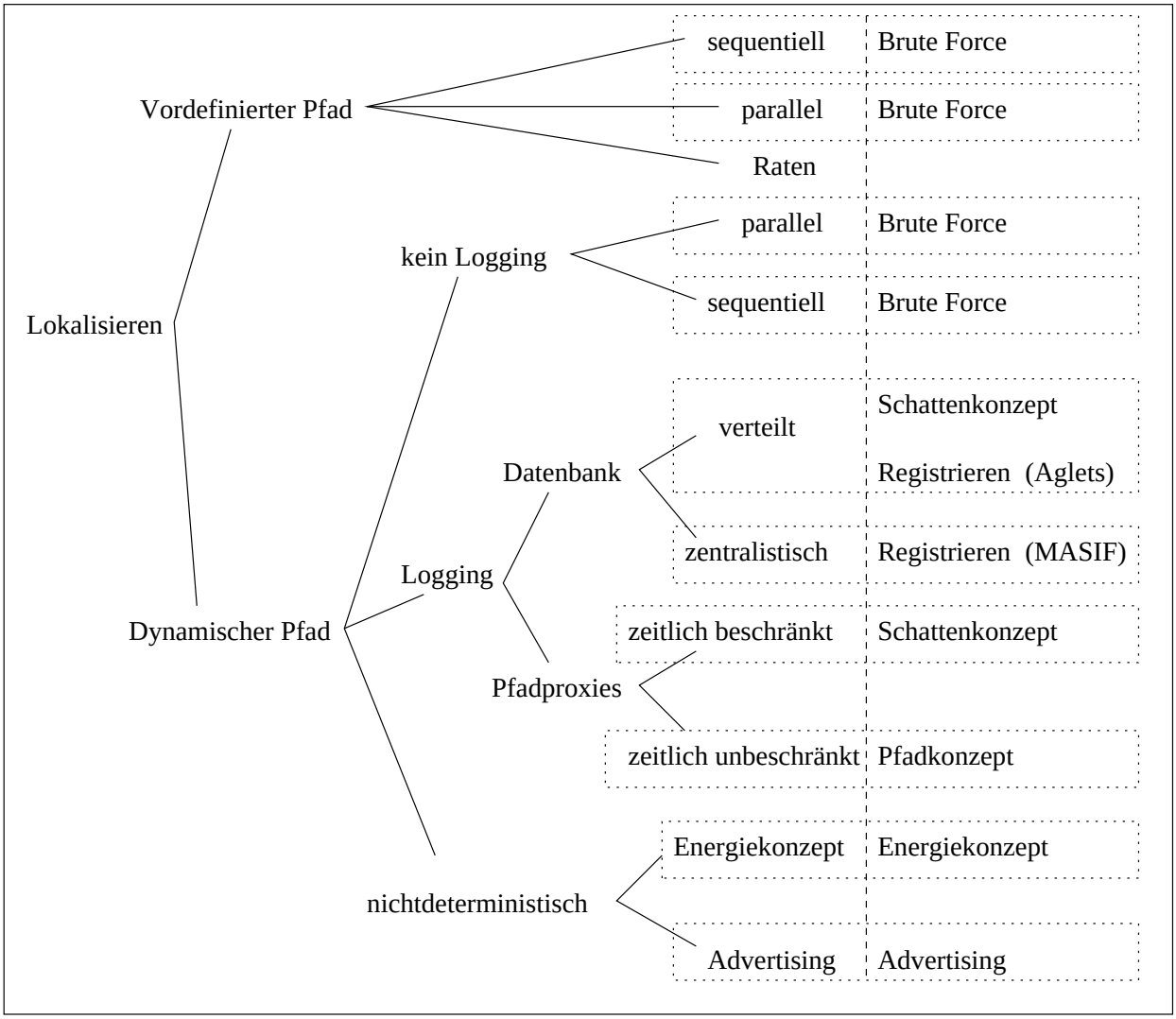


Abbildung 8.1: Klassifikation der Lokalisierungsansätze

Kapitel 9

Lokalisierung von Agenten im *AMETAS*

9.1 AMETAS-Sicht der Kontrollmechanismen

Kontrolle und Autonomie schließen sich prinzipiell gegenseitig aus. Möchte man die Autonomie der Agenten mit systeminternen Mechanismen unterstützen, wird dadurch das Kontrollieren der mobilen Agenten erschwert. Stattet man dagegen ein Agentensystem mit geeigneten systeminternen Mechanismen zum Kontrollieren der Agentenaktivitäten aus, schränkt man damit die Autonomie der mobilen Agenten ein. Die Autonomie ist ein wesentliches Merkmal eines Agenten, aber die Kontrolle der Agentenaktivitäten ist wesentlich für das System, die Systemadministratoren und die Agentenanwender. Der Konflikt der beiden Aspekte - Autonomie und Kontrolle - mobiler Agenten ist im Sachverhalt notwendig vorprogrammiert.

Im Kapitel 8 wurden einige Ansätze für Kontrollmechanismen für die mobilen Agenten behandelt. Sieht man bei der Beobachtung der vorgelegten Lösungen von ihren systembezogenen Eigenschaften ab[1], so stellt man fest: Je effizienter der Kontrollmechanismus eines Agentensystems ist, desto massiver wird in die Freiheit der Agenten eingegriffen.

Ein weiterer Gesichtpunkt ist, dass zum Kontrollieren der mobilen Agenten eine Kommunikation mit ihnen notwendig ist. Zur synchronen Kommunikation mit den mobilen Agenten ist ihre Lokalisierung erforderlich. Somit spielt die Art der Kommunikation bei der Lokalisierung der mobilen Agenten eine wesentliche Rolle.

Im folgenden wird vor diesem Hintergrund die Frage der Lokalisierung von Agenten im AMETAS behandelt. Im Abschnitt 2.2.2 wurde die AMETAS-Sicht der Autonomie besprochen. Auf dieser Basis wurde ein Kommunikationsmechanismus implementiert, der auf dem asynchronen Nachrichtenaustausch basiert (siehe 2.2.3). Für diesen Mechanismus ist die Lokalisierung der Agenten nicht notwendig, denn die Nachrichten können im Postfach des Empfängers hinterlegt werden. Der Empfänger kann diese Nachrichten bei einer passenden Gelegenheit abholen. Die im Postfach hinterlegten Nachrichten können vom Agenten aber auch ignoriert

[1] Gemeint sind z.B. die Einführung der Abhängigkeitsobjekte in Mole oder auch CORBA-bezogene Eigenschaften von MASIF.

werden. AMETAS-Agenten sind also nicht unbedingt von einer Kommunikation abhängig.

Die Notwendigkeit der Lokalisierung der mobilen Agenten ist im Rahmen vom AMETAS somit anders einzustufen als bei den oben behandelten Agentensystemen, die nicht so viel Wert auf der Autonomie der Agenten legen und RPC-basierte Kommunikationen mit den mobilen Agenten zulassen. In solchen Agentensystemen sind die Kontrollmechanismen ein Teil des Systems, ohne die die Funktionalität der jeweiligen Agenteninfrastruktur nicht gegeben ist.

Nichtsdestoweniger können Situationen eintreten, die eine Lokalisierung des AMETAS-Agenten erforderlich machen entsprechend der Ausführungen im vorherigen Kapitel. In den folgenden Abschnitten wird eine AMETAS-Lösung entwickelt und angeboten.

9.2 Systemmechanismen

Zunächst ist zu klären, ob die AMETAS-Agenten mit Hilfe der Werkzeuge, die in der Infrastruktur vorhanden sind, kontrolliert bzw. lokalisiert werden können. Zwei Möglichkeiten sind näher zu betrachten:

1. Der Umgang der AMETAS-Stellen mit den Agenten. Wenn ein Agent die Anforderungen des Sicherheitssystems erfüllt, kann er die ihm gestellten Aufgaben bearbeiten. Es ist ihm nicht vorgeschrieben, sich irgendwo zu registrieren oder seine Spuren zu hinterlassen. Man kann einen bestimmten oder eine Gruppe von Agenten so programmieren, dass sie sich z.B. registrieren sollen. Diese Registrierung kann nur durch den Agenteninhaber erfolgen. Eine Registrierung durch das System ist nicht vorgesehen. Mit anderen Worten verhalten sich die Agenten so, dass von einer Kontrolle keine Rede sein kann.

2. Die Kontaktaufnahme mit den Agenten. Weil die AMETAS-Agenten nicht synchron angesprochen werden können, ist die Hinterlegung von Nachrichten die einzige Möglichkeit, sie anzusprechen. Die AMETAS-Agenten holen ihre Nachrichten aber nach der ihnen innewohnenden Logik ab. Das Abholen der Nachrichten ist keine Vorschrift, was bedeutet, dass nicht alle zugestellten Nachrichten von den Agenten abgeholt werden. Somit ist mit Hilfe der asynchronen Nachrichten ihre Lokalisierung nicht gewährleistet.

Trotz dieser Einschränkungen bietet die Infrastruktur Möglichkeiten, den Agenten auf die Spur zu kommen: Das Vermittlungssystem und das Ereignissystem vom AMETAS.

9.2.1 AMETAS-Vermittlungssystem

Das Vermittlungssystem nimmt geeignet formulierte Anfragen über die Stellennutzer entgegen und liefert Antworten zurück, die Auskunft über die Existenz und den Zustand der Stellennutzer

geben.[2] Man unterscheidet zwei Vermittlungsarten:

1. **Instanzvermittlung**. Diese Art der Vermittlung ermöglicht es, Auskunft über eine oder mehrere aktuell existierende Instanzen eines bestimmten Stellennutzers in einer AMETAS-Stelle zu erhalten. Mit dieser Vermittlungsart kann man feststellen, welche Stellennutzer, also auch welche Agenten, in einer AMETAS-Stelle präsent sind. Der Klient muss dazu eine Beschreibung präsentieren, die mit den Beschreibungen der existierenden Instanzen verglichen wird. Wenn eine Instanz vorliegt, so existiert im AMETAS eine *PlaceUserID*. Diese wird dann als Adresse verwendet, um diesem Stellennutzer eine Nachricht zusenden zu können.

2. **Typvermittlung**. Diese Art der Vermittlung wird von Klienten verwendet, um bestimmte Stellennutzer ins Leben zu rufen. Dazu nimmt der Vermittler Anfragen zum Instantiieren eines Stellennutzers entgegen. Auch in diesem Fall muss der Klient eine Beschreibung abgeben. Es werden ihm dann eine oder mehrere SPU-Container-Namen zusammen mit deren Beschreibung geliefert, deren Beschreibung hinreichend gut mit der abgegebenen übereinstimmt.

Über den Treiber hat jeder Stellennutzer die Möglichkeit, eine Anfrage an den Vermittler zu stellen, wofür er ein Vermittlungsgesuch formulieren und den Treiber anweisen muss, die Anfrage einzureichen. Das Resultat nimmt er dann direkt von der aufgerufenen Methode entgegen.

Eine Schwäche des Vermittlungssystems zur Lokalisierung der Agenten ist seine lokale Begrenztheit und Unzuverlässigkeit: Die Vermittlung findet lokal statt. Sie muss daher in jeder Stelle wiederholt werden. Dies macht eine Entwicklung von geeigneten Diensten und Agenten unumgänglich. Die Informationen über die Instanzen der Agenten werden eine Weile nach ihrer Abreise in der Stellendatenbank aufbewahrt. Das Vermittlungssystem kann daher Agenten als aktiv melden, die sich nicht mehr in der Stelle aufhalten.

9.2.2 Nutzung des AMETAS-Ereignissystems für die Agentenlokalisierung

Der AMETAS-Ereignis-Mechanismus ist eine weitere Möglichkeit, den Agenten auf die Spur zu kommen. Die allgemeine Funktionalität wurde bereits in 2.2.4 vorgestellt.

Die Ereignisklasse, die die notwendigen Informationen über die Stellennutzer und die Agenten unter ihnen enthält, ist eine Instanz der Klasse `AMETASPlaceUserEvent.` Diese Klasse hat ein Feld, das die Art des Ereignisses enthält und als *EventID* bezeichnet wird. Wenn eines der folgenden Ereignisse in einer Stelle eintritt, generiert der Ereignismanager eine Instanz der oben genannten Klasse und setzt das EventID-Feld auf den entsprechenden Wert, der die EventID repräsentiert:

[2] Das neue Vermittlungssystem vom AMETAS bestimmt zusätzlich die Sprache, mit der die Anfragen an Vermittler formuliert werden können.

- Starten eines Stellennutzers
- Terminieren eines Stellennutzers
- Stoppen eines Stellennutzers (Stellennutzer wird von außen beendet)
- Ankommen eines Agenten
- Abreise des Agenten

Die entsprechenden EventID zu den genannten Ereignisse sind: `PU_LOCAL_START_EVENT`, `PU_TERMINATED_EVENT`, `PU_KILLED_EVENT`, `AGENT_ARRIVED_EVENT` und `AGENT_DEPARTED_EVENT`.

Ein weiteres Feld in der Klasse `AMETASPlaceUserEvent` enthält ein Objekt der Klasse `AMETASPlaceUserID`, die die Identität eines jeden Stellennutzers präsentiert. Jeder Stellennutzer besitzt eine Instanz dieser Klasse, um seine Identität auszuweisen. Eine *PlaceUserID* besitzt folgende Felder:

Name	Gruppe	IP-Adresse des Herkunftsrechners	Zeitpunkt der Erzeugung	Starter-ID

Dabei enthält das Feld *Starter-ID* ein Objekt der Klasse `AMETASIdentityID`, das eine Identitätsinstanz identifiziert. Damit können Informationen über den Besitzer des Stellennutzers mit dem empfangenen Ereignis erhalten werden.

Ein Stellennutzer, der ein oder mehrere Ereignisse dieser Art erhalten will, muss sich zunächst im *EventManager* registrieren. Dazu verfügt jeder Stellennutzer in seinem Treiber über die Methode `registerEventListener`. Um die entsprechenden Ereignisse zu erhalten, müssen die Stellennutzer die Schnittstelle `AMETASNotifiable` implementieren, die die Methode `notifyListener` zum Empfangen der Ereignisse bereitstellt.[3]

9.3 Spurensicherung

Das Herausfinden der aktuellen Aufenthalts-Stelle des Agenten und der Kontakt mit dem Agenten sind zwei getrennte Vorgänge. Ein Beobachterdienst kann genau so gut wie die Agenten selbst die Information über den aktuellen Aufenthaltsort des Agenten liefern.

Als Ansatz für die Lokalisierung der Agenten im AMETAS empfehlen sich die Vorgänge, die mit den Agenten zu tun haben und sich durch den Ereignis-Mechanismus der Stelle bemerkbar machen. Diese Vorgänge werden auf den Stellen beobachtet und dabei werden die Spuren der Agenten gesichert. Dieser erste Schritt schafft eine Basis für alle möglichen Mechanismen,

[3] Ausführliche Informationen findet man in [AMTU] und [AMAPI].

die die Spuren auswerten und auf der Basis dieser Auswertung geeignete Kontrollalgorithmen entwickeln.

Es wird auf jeder AMETAS-Stelle ein sogenannter *ManagerService* installiert, der eine abgeleitete Klasse der Klasse `AMETASServiceObject` ist. Die Aufgabe des Managerservices besteht darin, die Ereignisse, die die Stellennutzer betreffen, in jeder Stelle zu empfangen. Dafür sind die AMETAS-Dienste durch ihren *ServiceManager*, der eine Instanz der Klasse `AMETASServiceManager` ist, gut ausgerüstet. Diese Klasse implementiert die Schnittstelle `AMETASNotifiable` und kann die die Stellennutzer betreffenden Ereignisse von dem Ereignismanager erhalten, sobald diese Ereignisse eintreten.

Um sich zu registrieren, ruft der `ManagerService` in seiner `initService`-Methode seine Methode zur Registrierung beim Ereignismanager auf:

```
Object[] argObjekt = {new AMETASPlaceUserIDMask()};
m_ServiceMgr.getDriver().registerEventListener(
              AMETASPlaceUserEvent.ANY_PU_EVENT,
              0,
              argObjekt);
```

Die Argumente sind folgendermaßen zu verstehen:

- `AMETASPlaceUserEvent.ANY_PU_EVENT` bedeutet, dass dieser Stellennutzer sich für alle Ereignisse interessiert, die die Stellennutzer betreffen.
- `AMETASEvent.HIGHEST_PRIORITY` legt fest, dass der ManagerService sofort benachrichtigt werden muss, wenn die spezifizierten Ereignisse eintreten.
- `0` deutet daraufhin, dass keine weitere Optionen benötigt werden.
- `AMETASPlaceUserIDMask` charakterisiert die Menge der Stellennutzer, für die sich der *EventListener* interessiert. Ein Objekt dieser Klasse stellt eine Teilmenge von `AMETASPlaceUserID` dar. Diese Klasse hat die gleichen Datenfelder wie die `AMETASPlaceUserID` und dient dazu, eine Menge von Stellennutzern zu filtern. Sie ist verwandt mit der `AMETASPlaceUserID`, aber unabhängig von ihr, was den Vererbungsbaum angeht. Sie wird hier verwendet, um die Empfänger einer AMETASMessage festzulegen (vgl. [AMAPI, AMTU]).

Nach der Registrierung muss der Dienst nur auf die Ereignisse warten. Wenn ein Ereignis - wie z.B. das Ankommen eines Agenten - eintritt, wird dem Managerdienst eine Instanz der Klasse `AMETASPlaceUserEvent` durch seine `handleEvent`-Methode übergeben, die Auskunft über die Herkunftsstelle und PlaceUserID des ankommenden Agenten geben kann.

Die erhaltenen Informationen müssen so verwaltet werden, dass die Lokalisierung von Agenten mit ihrer Hilfe möglich ist. Im Rahmen der bisherigen Implementierung werden die

erhaltenen Informationen in hierfür vorgesehene Tabellen eingefügt, wobei der Managerdienst auf jeder Stelle über drei getrennte Hash-Tabellen für folgende Zwecke verfügt:

- Speichern der Informationen über ankommende Agenten sowie über die Agenten, die die Stelle verlassen
- Sicherung der Informationen über die Stellennutzer, die in dieser Stelle starten
- Festhalten der Auskünfte über die Stellennutzer, die in dieser Stelle terminieren oder dazu gezwungen werden.

Zur Zeit werden die genannten Tabellen nur im Hauptspeicher gehalten. Ein wesentliches Problem hängt mit der Dauer der Speicherung der Spuren zusammen. Dabei stellt sich die Frage: Wie lange sind die Informationen brauchbar? Dabei spielen zwei Faktoren eine wichtige Rolle. Einerseits hängt die Dauer der Speicherung von den Informationen ab und wann diese als veraltet zu gelten haben. Wenn z.B. der gleiche Agent auf einer Stelle zweimal terminiert, ist das erste Ereignis veraltet. Anderseits aber hängt die erforderliche Dauer von der Auswertungsstrategie ab, die die Informationen benötigt. Möchte man Statistiken über die Agenten sammeln, um bestimmte Aussagen z.B. über die möglichen Migrationspfade machen zu können, so sind alle gesammelten Informationen wichtig, die dann in einer geeigneten Datenstruktur verwaltet werden müssen. In diesem Fall ist die persistente Speicherung der Daten unumgänglich.

9.4 Schnappschussmethode

Die *Schnappschussmethode* ist dem Logging-Mechanismus ähnlich, der im Kapitel 8 behandelt wurde. Es werden dabei die zeitlich beschränkt verfügbaren Log-Informationen verwendet, deren Sicherung im vorangegangenen Abschnitt behandelt wurde.

Wenn ein Agent, z.B. *AgentX* in einer Stelle ankommt, wird das Ereignis in dieser Stelle festgehalten. Die Herausforderung liegt darin, mit Hilfe der Informationen der einzelnen Stellen AgentX zu finden. Eine mögliche Vorgehensweise ist die Beauftragung eines *Detektiv-Agenten*, der die in Frage kommenden Stellen durchsuchen und mit der Abfrage der in den Stellen vorhandenen Informationen AgentX lokalisieren soll.

Für die stationären Agenten und auch die Agenten, die sich lange in einer Stelle aufhalten, ist diese Vorgehensweise zum Auffinden der Agenten geeignet. Wenn aber die gesuchten Agenten häufiger migrieren und ihren Pfad dynamisch ändern, birgt diese Methode ein Problem: Es kann vorkommen, dass ein Agent mit dieser Methode ewig gesucht werden muss, ohne dass er vom Detektiv-Agenten erwischt wird. Der Nicht-Determinismus der Methode ist also nicht ausgeschlossen. Außerdem ist die Methode ineffektiv: Die Suche kann mehrere Rundreisen erforderlich machen.

Idee:

Mit Hilfe des *Schnappschussalgorithmus* ([MAFR89], S. 103ff.) erhält man einen systemweiten[4] Ausschnitt, aus dem man die Stelle herausfinden kann, in der sich AgentX bereits aufhält oder wohin er gerade migriert.

Der Schnappschuss wird durch einen *Schnappschuss-Agenten* angekündigt. Dieser Agent übergibt eine Nachricht an den ManagerService, die den Namen des gesuchten Agenten und eventuell seine Gruppe enthält. An jeder Stelle, in der er ankommt, hinterlässt er diese Nachricht. Erhält der ManagerService die Nachricht, so wartet er eine bestimmte Zeit ab. Die Wartezeit wird durch die Anzahl der Stellen und die Entfernung der Stellen voneinander bestimmt. Sie wird so gewählt, dass der Schnappschuss-Agent alle zu durchsuchenden Stellen besuchen kann. Diese Wartezeit ist notwendig, um folgendes Szenario zu verhindern:

Der Schnappschuss-Agent besucht alle Stellen und kehrt an seinen Ausgangspunkt zurück. Alle Stellen halten ihre Informationen bereit. Diese Informationen machen aber keine Aussagen über die Stelle, in der sich AgentX aufhalten könnte. Denn während der Schnappschuss-Ankündigung ist AgentX zu den Stellen migriert, die die Schnappschuss-Nachricht bereits gesehen haben. Somit liefern alle Stellen veraltete Informationen über AgentX. Ein typischer Fall ist der, dass AgentX hinter dem Schnappschuss-Agenten her migriert. Es wird deutlich, dass mit der Schnappschussmethode auf diese Weise keine zuverlässige Aussage über den Aufenthaltsort von AgentX gemacht werden kann, weil sich der Zustand des System inzwischen bereits wieder verändert hat.

Wenn die Wartezeit abgelaufen ist, hat der Managerdienst in jeder Stelle des Systems einen der folgenden Zustände festgehalten, wobei implizit angenommen wird, dass die veralteten Informationen aussortiert werden:

1. AgentX ist nicht auf dieser Stelle gewesen
2. AgentX war auf dieser Stelle und ist abgereist
3. AgentX war hier und hat auf dieser Stelle terminiert
4. AgentX ist hier und aktiv.

In den Stellen, bei denen der erste Fall vorkommt, hinterlässt der Managerdienst nur eine Nachricht an den Detektiv-Agenten, der eventuell zu dieser Stelle kommen wird. Dahingegen startet der Managerdienst in den Stellen, bei denen der Fall 3 oder 4 eintritt, einen Detektiv-Agenten. Er beauftragt ihn, mit einer Nachricht zu der Stelle zu gehen, die den Schnappschuss initiiert hat. Der Inhalt der Nachricht ist dem Fall entsprechend zu setzen. Dabei hinterlässt der Managerdienst noch eine weitere Nachricht, deren Inhalt die Aktivität des Dienstes mit Blick auf den Detektiv-Agenten betrifft. Die Nachricht ist an den Detektiv-Agenten adressiert.

[4] Mit "System” meint man die Stellen, die als Aufenthaltsort des gesuchten Agenten in Frage kommen.

Wenn in einer Stelle der zweite Fall vorkommt, startet der Managerdienst einen Detektiv-Agenten und beauftragt ihn zu der Stelle zu gehen, zu der AgentX von der aktuellen Stelle aus migriert ist. Hier wird eine weitere Nachricht an den Detektiv-Agenten hinterlassen, der zu dieser Stelle kommen wird.

Wenn ein Detektiv-Agent in der Stelle Y startet, wird ihm seine Nachricht übergeben. Lautet der Inhalt, dass AgentX noch in dieser Stelle aktiv ist oder dass er hier terminiert hat, so heißt dies, dass AgentX gefunden wurde. Der Detektiv-Agent macht sich nun auf den Weg zu der Stelle, die den Schnappschuss initiiert hat. Ansonsten migriert er zur Stelle Z, die als Zielstelle vom AgentX festgestellt wurde.

Wenn nun ein Detektiv-Agent in der Stelle Z ankommt, holt er seine Nachricht ab. Die Nachricht gibt Auskunft über den Zustand, der während der Ankündigung des Schnappschusses festgehalten wurde. Wenn er feststellt, dass von dieser Stelle aus ein weiterer Detektiv-Agent gestartet wurde, so ist seine Aufgabe zu Ende und er terminiert. Ansonsten gilt Fall 1 (AgentX war nicht hier) und der Detektiv-Agent migriert mit der Nachricht zur Initiator-Stelle, dass AgentX gefunden wurde. AgentX muss dann nämlich unterwegs zu dieser Stelle sein. Dabei basiert die Entscheidung des Detektiv-Agenten darauf, dass der Managerdienst in der Stelle Z den Fall 1 festgehalten hat, und ihm von der Stelle Y gemeldet wurde, dass AgentX zur Stelle Z abgereist ist. Damit wird genau die Situation erfasst, die in der Abbildung 9.1 dargestellt ist: Ein gesuchter Agent befand sich während des Schnappschuss-Vorgangs zwischen den Stelle 4 und 5. Aus dem Wissen, dass AgentX auf dem Weg zur Stelle 5 war, kann er folgern, dass AgentX immer noch unterwegs zur Stelle 5 ist.

Algorithmen und Abbildungen

Die Algorithmen 1, 2, 3 und 4 stellen die beschriebenen Ideen algorithmisch dar. Um die Algorithmen zu veranschaulichen, betrachte man die Abbildungen 9.1, 9.2, 9.3, 9.4 und 9.5.

Algorithm 1 Lokalisierung von Agenten

```
1 Lokalisiere(AgentX: String, Stelle_Liste: String[]) {
2   if(AgentX_ist_hier oder AgentX_hat_hier_terminiert)
3     gefunden. Gebe den Namen der Stelle zurück
4   else
5     Schnappschuss_Agent(AgentX, Stellen_Liste)
6 }
```

Algorithm 2 Schnappschuss-Agent.

```
1 Schnappschuss_Agent(AgentX: String, Stellen_Liste: String[]) {
2 neue Nachricht(”SCHNAPPSCHUSS”, AgentX)
3   for(jede Stelle in der Stellen_Liste) {
4          gehe_zu(Stelle) und
5           hinterlege_Nachricht_an_ManagerService(Nachricht)
6     }
7  }
```

Algorithm 3 Schnappschuss erhalten. Dieser Algorithmus wird vom ManagerSevice nach dem Erhalt der Schnappschuss-Nachricht durchgeführt.

```
1 Schnappschuss_erhalten(AgentX:String) {
2      warte_bestimmte_Zeit_ab
3      if(hat_hier_terminiert(AgentX)) {
4          neue Init_Nachricht("AgentX_hat_hier_terminiert")
5          Detektiv_Agent(Init_Nachricht)
6          neue Nachricht("AgentX_hat_hier_terminiert”)
7          hinterlege_an_ankommenden_Detektiv_Agenten(Nachricht)
8       }else if(AgentX_ist_hier(AgentX)) {
9          neue Init_Nachricht("AgentX_ist_hier")
10         Detektiv_Agent(Init_Nachricht)
11         neue Nachricht("AgentX_ist_hier")
12          hinterlege_an_ankommenden_Detektiv_Agenten(Nachricht)
13      }else if(war_hier(AgentX)) {
14          StelleX = zu_welcher_Stelle_ging(AgentX)
15         neue Init_Nachricht("AgentX_war_hier", StelleX)
16         Detektiv_Agent(Init_Nachricht)
17         neue Nachricht("AgentX_war_hier", StelleX)
18         hinterlege_an_ankommenden_Detektiv_Agenten(Nachricht)
19      }else {
20         neue Nachricht("AgentX_war_nicht_hier")
21         hinterlege_an_ankommenden_Detektiv_Agenten(Nachricht)
22      }
23  }
```

Algorithm 4 Detektiv-Agent

```
1  Detektiv_Agent(Nachricht: AMETASMessage) {
2     if(erster_Lauf) {
3        Nachricht_Inhalt[] = Nachricht.getBody()
4        if(Nachricht_Inhalt[0] == "AgentX_hat_hier_terminiert")
5           gehe_zu(Schnappschuss_Initiator)
6        else if(Nachricht_Inhalt[0] == "AgentX_ist_hier")
7                 gehe_zu(Schnappschuss_Initiator)
8        else if(Nachricht_Inhalt[0] == "AgentX_war_hier") {
9                 StelleX = Nachricht_Inhalt[1]
10                  gehe_zu(StelleX)
11       }else  fertig
12    }else
13       if(Stelle_ist_Schnappschuss_Initiator)
14          gebe_die_Nachricht_weiter
15       else {
16          hole_Nachricht : Neue_Nachricht
17          Inhalt_Neue_Nachricht[] = Neue_Nachricht.getBody()
18          if(Inhalt_Neue_Nachricht[] == "AgentX_war_nicht_hier")
19             gehe_zu(Schnappschuss_Initiator)
20          else  fertig
21       }
22 }
```

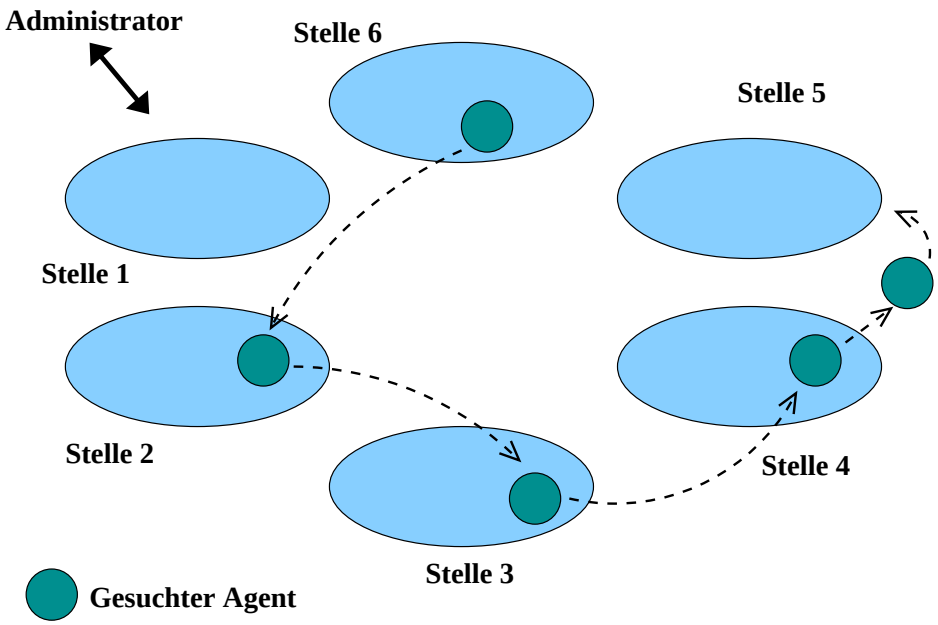


Abbildung 9.1: Lokalisierungsschema - Agentenpfad

9.5 Analyse und Bewertung

Mit der Schnappschussmethode wird ein Schnitt des Systems festgehalten. Dieses besteht aus den AMETAS-Stellen, die als Aufenthaltsort eines gesuchten Agenten in Frage kommen. Die hier vorgestellte Methode basiert auf dem dezentralistischen Ansatz, bei dem die lokalen Informationen in jeder Stelle ausgewertet werden. Dabei wird nur das Ergebnis an die Initiator-Stelle des Schnappschusses gesendet. Der Nachteil der Methode ist, dass für jede Abfrage, die einen Agenten oder eine Agentengruppe betrifft, ein neuer Schnappschuss gestartet werden muss. Die Lokalisierung von Agenten im AMETAS ist aber eine Aktivität, die nicht permanent gebraucht wird. Vor diesem Hintergrund ist der Aufwand als vertretbar einzuschätzen.

Eine berechtigte Frage in diesem Zusammenhang ist: Warum läuft ein Agent im Kreis umher, um den Schnappschuss anzukündigen? Hierzu ist das Szenario näher zu betrachten: In einer Menge von Stellen sind Informationen über einen Agenten lokal vorhanden, die für die Agentenlokalisierung ausgewertet werden müssen. Im Allgemeinen muss man davon ausgehen, dass sich das System dynamisch verändert. Damit man sich nicht dauernd zwischen den Stellen hin- und herbewegen muss, um die Spuren auszuwerten, benötigt man eine Aussage über den globalen Zustand des Systems zu einem bestimmten Zeitpunkt. Das Erreichen eines globalen Zustandes in einem verteilten System wird als *Schnappschuss* bezeichnet (vgl. [MAFR89], S. 103).

Die Art und Weise, wie man den globalen Zustand erreicht, ist eine weitere Frage. Es muss nicht unbedingt ein Agent im Kreis herum laufen, um den Schnappschuss anzukündigen. Die Grundidee besteht darin, dass ein Initiator eine Traversierung der gesamten Knoten startet, die die Stellen repräsentieren. Dabei erwartet er von allen laufenden Stellen ein *Echo*. Die vorge-

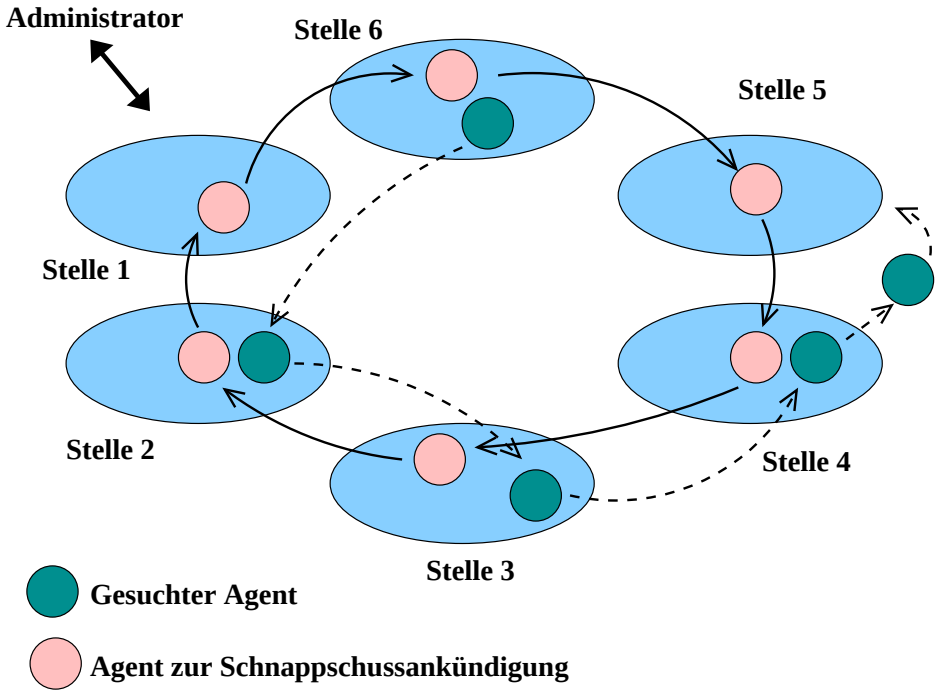


Abbildung 9.2: Lokalisierungsschema - Schnappschuss-Agent

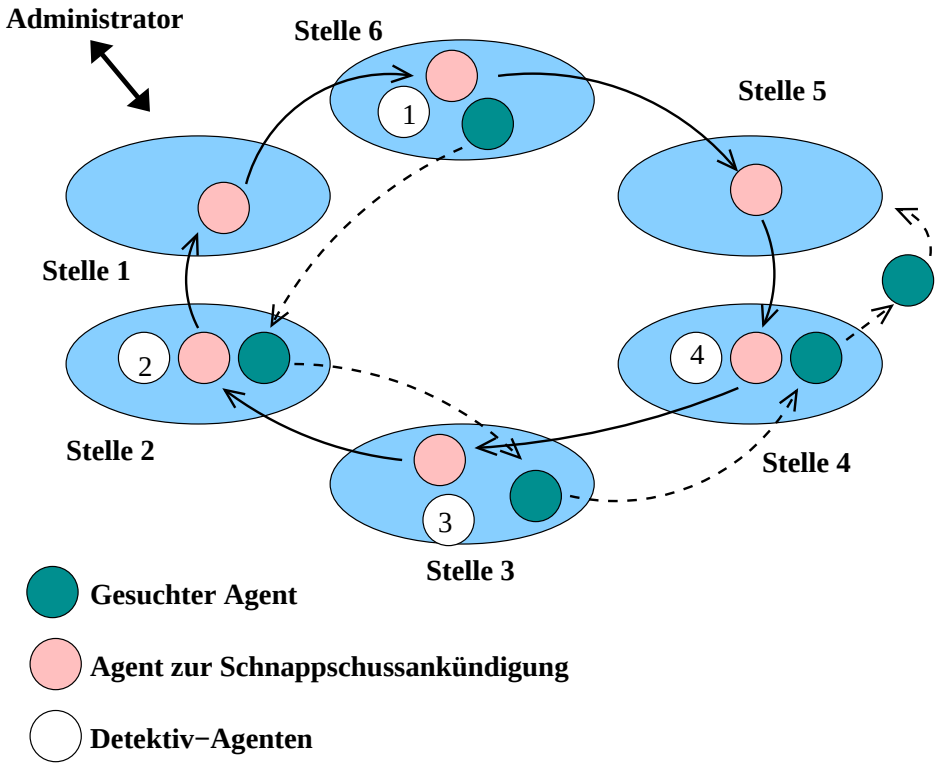


Abbildung 9.3: Lokalisierungsschema - Detektiv-Agenten (erster Start)

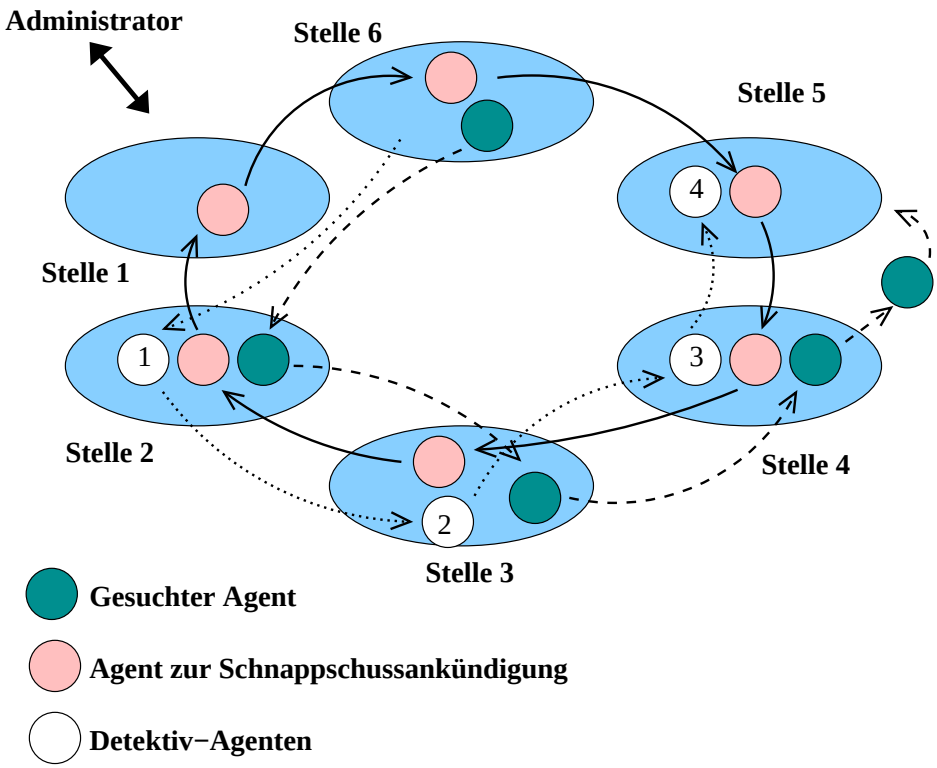


Abbildung 9.4: Lokalisierungsschema - Detektiv-Agenten (nach einer Migration)

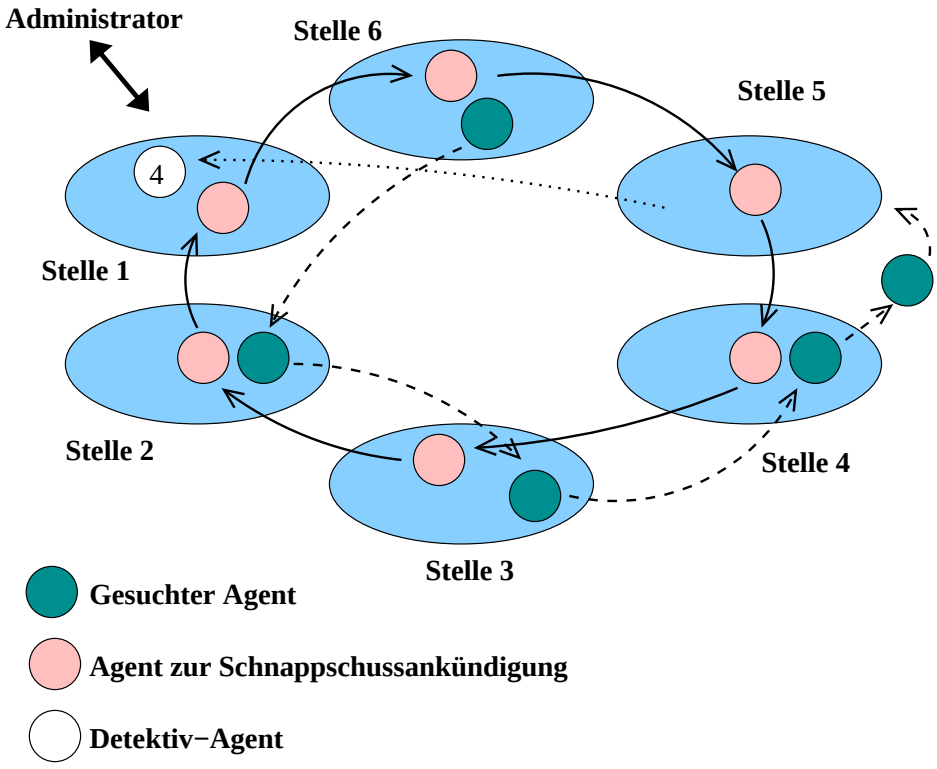


Abbildung 9.5: Lokalisierungsschema - Detektiv-Agent (Terminierung)

stellte Methode ist eine sequentielle Traversierung, die mit dem *Pfad-Verfahren*, das auf der *Depth first*-Strategie, basiert, vergleichbar ist ([MAFR89], S. 75). Der Schnappschuss-Agent kehrt an seinen Ausgangpunkt zurück, wodurch ein Kreis gebildet wird.

Für die Traversierung gibt es auch effiziente parallele Verfahren. Es ist z.B. vorstellbar, dass die Initiator-Stelle für jede mögliche Aufenthaltsstelle des gesuchten Agenten einen Schnappschuss-Agenten beauftragt. Damit kann man auch die Traversierung von Domänen veranlassen, die dem Initiator noch nicht bekannt sind. Diese Idee basiert auf dem *Echo-Algorithmus*, der von einem Initiator gestartet wird ([MAFR89], S. 30). Diese Methode ist zwar schnell, verursacht aber einen größeren Aufwand an Ressourcen, da viele Agenten auf einmal unterwegs sein müssen.

Für die Auswertung der Daten des Schnappschusses ist noch eine andere Vorgehensweise denkbar: Wenn die Stellen die Nachricht des Initiators erhalten, können sie ihre lokalen Information durch einen Agenten an den Initiator schicken. Auf diese Weise kann sich der Initiator selbst ein Bild von dem Zustand des Systems machen und feststellen, auf welcher Stelle sich der gesuchte Agent bei der Zustandsermittlung befand. Der Einsatz der Detektiv-Agenten entfällt. Würden auch statistische Werte in den Informationen enthalten sein, so könnte der Initiator auf dieser Basis probabilistische Angaben machen. Der Initiator könnte dann z.B. aus dem bisherigen Verhalten eines Agenten eine Aussage darüber treffen, welche die wahrscheinliche Aufenthaltsstelle des Agenten ist.

9.5.1 Andere Sichtweise?

Für das gegebene Szenario liefert das Verfahren eine zufriedenstellende Lösung. Ein Nachteil der Methode ist, dass bei jedem Lokalisierungsbedarf eine neue Suche durch einen Schnappschuss gestartet werden muss. Die Frage ist, ob dies umgangen werden kann. Man ist bis jetzt davon ausgegangen, dass die Informationen über Agenten, die zur Lokalisierung von Bedeutung sind, lokal auf jeder Stelle verwaltet werden. Die Alternative dazu wäre ein Verfahren, das auf der Basis der im Kapitel 9 vorgestellten Registrierungsmethode, eine zentrale oder domänenbezogene Registrierung leisten würde. Im AMETAS ist ausgeschlossen, dass seine Agenten zu so einer Aktion gezwungen werden. Eine alternative Möglichkeit wäre, dass man die Registrierung in Vertretung der Agenten vornimmt.

Das Szenario hierfür wäre das folgende: Man setzt für diesen Zweck *Registrierer-Agenten* ein. Wenn der Managerdienst der jeweiligen Stellen durch den Ereignismanager über ein Ereignis in Kenntnis gesetzt wird, schickt er diese Information durch einen Registrierer-Agenten an eine oder mehrere vereinbarte Stellen, anstatt die Informationen selbst zu speichern. Auf diese Weise werden die Informationen über jeden Agenten zentral verwaltet. Ob es eine oder mehrere solche Stellen in einer Domäne geben soll, oder ob es für mehrere Domänen eine solche Stelle geben soll, ist situationsabhängig zu beantworten. Die Fragen, die mit der Speicherung, Ihrer

Dauer und Art verbunden sind, sind noch adäquat zu beantworten.

Der Vorteil dieser Strategie wäre, dass man zu jedem Zeitpunkt eine mit hoher Wahrscheinlichkeit richtige Antwort über die Stelle geben kann, in der sich ein gesuchter Agent gerade aufhält, ohne eine Suche starten zu müssen. Es sind dann nur die zentralen Stellen abzufragen. Der Nachteil der Methode ist aber, dass permanent Registrierer-Agenten unterwegs sein müssen, um die anderen anzumelden. Dies verursacht dauernden Agentenverkehr, der nicht immer notwendig ist. Auch die zentrale Verwaltung von Informationen in einem heterogenen System wirft weitere Probleme auf. Würden aktuelle Informationen über den Aufenthaltsort eines bestimmten AMETAS-Agenten häufig benötigt, kann sich diese schnelle Suchmethode als hilfreich oder sogar notwendig erweisen. Geht man von dem Normalfall aus, dass die Lokalisierung von Agenten nicht häufig erforderlich ist, wird die in dieser Arbeit vorgeschlagene Lösung zur Lokalisierung von Agenten empfohlen.

9.6 Terminierung

Zur Terminierung der AMETAS-Agenten kann der beschriebene Lokalisierungsalgorithmus 1 verwendet werden. Die Detektiv-Agenten müssen nicht aktiviert werden. Der Schnappschuss-Agent hinterlegt in jeder Stelle, in die er migriert, eine Terminierungsnachricht beim Managerdienst bezüglich eines zu terminierenden Agenten. Der Managerdienst verwendet dazu die Methode `KillPlaceUser` seines Treibers. Mit dieser Methode wird ein Stellennutzer auf die schwarze Liste der zu terminierenden Stellennutzer der Stelle gesetzt. Dabei wird eine Zeitspanne in Millisekunden als Argument angegeben, die festlegt, bis wann die Aufforderung zum Terminieren gültig ist. [5]

[5] An dieser Stelle sind zusätzlich einige Werkzeuge zu erwähnen, die zum Stoppen von allen Stellennutzern in den AMETAS-Stellen verwendet werden können. Dazu gehören *StartAndKill*, *StartAndKillAgain*, *Main*. Die beiden ersten sind Benutzeradapter, die das Starten und Stoppen der Stellennutzer ermöglichen. Main ist ein Dienst, der nur zum Stoppen der Stellennutzer verwendet werden kann.

Der Entwickler dieser Werkzeuge ist Michael Zapf, der, als Basis seiner Dissertation über die Typisierung von Agenten, das Gesamtkonzept des AMETAS geprägt hat. Man kann die Werkzeuge runterladen von der Adresse: http://www.ametas.de/runterlad.html.

Kapitel 10

Zusammenfassung

In dieser Arbeit wurde die Implementierung einer JMX-konformen Managementinfrastruktur für das Agentensystem AMETAS vorgestellt. Darauf basierend wurden im Rahmen des Fehlermanagements Kontrollmechanismen der mobilen Agenten im AMETAS untersucht und eine Lösung für die Lokalisierung von AMETAS-Agenten entworfen und implementiert.

Der essentielle Hintergrund für das AMETAS-Management stellt sich folgendermaßen dar: Die Betrachtung des Anwendungs- und Infrastrukturmanagements mit Blick auf die Managementhierarchie stellt die Offenheit und Kooperationsfähigkeit der angestrebten Managementlösung in den Vordergrund. Diese Eigenschaften ermöglichen die Integration der in einem Unternehmen existierenden Managementlösungen. Ziel ist dabei ein kostengünstiges und effizientes Management. Eine Managementarchitektur wird mit Hilfe der informations-, organisations-, kommunikations- und funktionsbezogenen Aspekte beschrieben und modelliert. Anhand dieser Aspekte ist CORBA, DMTF, WBEM und JMX analysiert und ihre Eignung für das AMETAS-Management bewertet worden. Neben den allgemeinen Kriterien sind ihre Teilmodelle, ihre Unterstützung des dezentralen und des dynamischen Managements sowie ihre Integrationsfähigkeit im AMETAS zentrale Punkte. Es zeigt sich, dass die JMX die besten Möglichkeiten für das AMETAS-Management bietet.

Das OSI-Funktionsmodell klassifiziert die Managementaufgaben und -funktionen in fünf Bereiche, die häufig als FCAPS bezeichnet werden: Fehler-, Konfigurations-, Abrechnungs-, Leistungs- und Sicherheitsmanagement. Diese Klassifikation ist orthogonal zu jeder anderen und bietet einen geeigneten Rahmen für die Aufteilung der Managementaufgaben und -funktionen. Das in dieser Arbeit empfohlene AMETAS-Management orientiert sich hinsichtlich der Managementaufgabenaufteilung am OSI-Modell.

Die JMX bietet mächtige Werkzeuge zur Instrumentierung aller Arten von Ressourcen. Ihre Java-Basiertheit bedeutet eine wesentliche Vereinfachung für das Agentensystem. Die offene Architektur von der JMX ermöglicht die Kooperation des AMETAS-Managements mit anderen Managementstandards. Das AMETAS-Management nutzt die Vorzüge der mobilen Agenten insbesondere im Bereich des Konfigurations- und Fehlermanagements aus. Folgende Eigen-

schaften zeichnen das AMETAS-Management aus:

- Verwendung der Agenteninfrastruktur für das Management. Selbiges wird dabei als ein AMETAS-Dienst implementiert und kann alle Möglichkeiten und Dienste der Agenteninfrastruktur nutzen.
- Verwendung der AMETAS-Agenten und Dienste als Managementwerkzeuge.
- Selbstmanagement des Systems. Der Managementdienst ist hierfür mit ausreichender Intelligenz ausgestattet. Er nutzt die Mechanismen der Agenteninfrastruktur aus und erledigt diverse Managementaufgaben selbständig. Das Ereignissystem vom AMETAS spielt hierbei eine wichtige Rolle.

Die Analyse der Kontrollmechanismen von MASIF, Aglets Workbench und Mole liefert hinsichtlich ihrer Eignung für die Lokalisierung von Agenten im AMETAS folgendes Ergebnis: Die untersuchten Ansätze sind teilweise allgemein anwendbar. Man unterscheidet die nichtdeterministischen Ansätze wie Advertising und Energiekonzept von denen, die bestimmte Spuren von Agenten in einer geeigneten Art hinterlegen. In dieser Hinsicht stellte sich das Pfadkonzept als interessant heraus: Bei diesem Konzept können die Informationen über den Pfad der Migration eines Agenten in geeigneter Weise zeitlich beschränkt oder unbeschränkt gespeichert werden. Eine andere Alternative bietet die Registrierungsmethode. Bei dieser Methode wird ein Agent in einer zentralen Stelle registriert, wobei die eindeutige Identität eines Agenten und die aktuelle Stelle, in der sich ein Agent aufhält, gespeichert werden.

Vor dem Hintergrund der erfolgten Analyse empfiehlt sich als Basis für die Lokalisierung von AMETAS-Agenten eine Art Pfadkonzept: Die Spuren der Agenten werden durch einen Managementdienst gesichert. Will man einen bestimmten Agenten oder eine Gruppe lokalisieren, werden die dezentral vorhandenen Informationen innerhalb eines konsistenten Schnitts (Schnappschuss) ausgewertet.

Die Schnappschussmethode empfiehlt sich für die Lokalisierung von Agenten im AMETAS entsprechend den zu Beginn der Arbeit von einem Lokalisierungsmechanismus geforderten Eigenschaften: Sie erlaubt eine zuverlässige Lokalisierung der gesuchten Agenten, deren Autonomie dabei respektiert wird. Die Kosten-Leistungsrelation ist günstig einzuschätzen, da unnötiger Daten- bzw. Agentenverkehr ebenso vermieden wird wie die Pflege umfangreicher, zentralistischer Datenbanken.

Anhang A

Beispiele, Bilder

A.1 Java-Klassen des AMETAS-Managements

Für das AMETAS-Management sind folgende Schnittstellen und Klassen vom Autor implementiert worden:

Java-Schnittstellen:

`ManagerServiceIf`

Java-Klassen:

`AgentPathData`	`DetectiveAgent`
`HTMLBuilder`	`HTTPActor`
`LoginForm`	`ManagerAgent`
`ManagerService`	`MBeanBuilder`
`PlaceBean`	`PortObject`
`ServiceParameter`	`SnappshutAgent`
`SSLAdapter`	`SSLAdapterBean`
`WWWInjector`	`WWWTimer`
`WWWTimeObject`	

A.2 Beispiele

Die folgenden Java-Klassen demonstrieren, wie man eine AMETAS-Ressource als `ModelMBean` instrumentieren und im MBean-Server des AMETAS-Managements registrieren kann. Dabei müssen die Metadaten-Klassen gebildet werden. Mit Hilfe der Metadaten-Klassen wird ein Objekt der Klasse `ModelMBeanInfo` gebildet. Nach der Erzeugung einer Instanz der

Klasse `RequiredModelMBean` mit einem Objektnamen für eine Ressource durch den MBean-Server wird der `ModelMBean` die `ModelMBeanInfo` zugeordnet. Die Zuordnung geschieht durch den MBean-Server.

```
package AMETAS.mgm;

import java.io.*;
import java.util.*;
import java.lang.reflect.*;

/* AMETAS-spezifische Klassen */
import AMETAS.place.*;
import AMETAS.pns.*;
import AMETAS.servicedev.*;
import AMETAS.data.*;

/* JMX-spezifische Klassen */
import javax.management.*;
import javax.management.modelmbean.*;
import javax.management.relation.*;
import com.sun.management.jmx.Trace;

class MBeanBuilder {
    /** Referenz auf den MBeanServer */
    private MBeanServer m_mBeanServer = null;

    /** Referenz auf den Manageragenten */
    private ManagerAgent m_managerAgent = null;

    /** Referenz auf die AMETAS-Stelle */
    private AMETASPlace m_plThePlace = null;

    /** Membervariable für die private Nutzung */
    private Properties m_placeConfigProp = null;

    /** Referenz auf die MBean, die die aktuelle Stelle präsentiert */
    private PlaceBean m_placeBean = null;

    /** Ein Objekt, das die Parameter für die Einstellungen
```

```
    des Managementdienstes und die dazu gehörigen Objekte enthält */
private ServiceParameter m_serviceParm = null;

/** Konstruktor der Klasse */
public MBeanBuilder(ManagerAgent mAgent, MBeanServer mbeanServer) {
    m_managerAgent = mAgent;
    m_mBeanServer =  mbeanServer;
    m_plThePlace = mAgent.getPlace();
    m_placeConfigProp = new Properties();
    m_placeBean = new PlaceBean(m_managerAgent, m_plThePlace);
    m_serviceParm = m_managerAgent.getServiceParm();
 }

/** Methode zum Bilden und Registrieren einer MBean für
    die aktuelle AMETAS-Stelle */
void buildPlaceMBean() {
    String mbeanName = "Place";
    ObjectName objectName = null;
    try {
        String domain = m_mBeanServer.getDefaultDomain();
        objectName = new ObjectName(domain+":type="+mbeanName);
    }catch(MalformedObjectNameException mfonx) {
        String fehler = "<MBeanBuilder> kann kein Objekt bilden.\n";
        fehler += mfonx.getMessage();
        output(fehler);
    }

  String mbeanClassName = "javax.management.modelmbean.RequiredModelMBean";
  try {
      m_mBeanServer.createMBean(mbeanClassName,objectName);
  }catch (Exception e) {
      output("\t!!! <MBeanBuilder> kann die " +
              mbeanName + " MBean nicht erzeugen!!!");
      output(e.getMessage());
      e.printStackTrace();
  }
  ModelMBeanInfo mmBeanInfo = null;
  mmBeanInfo = buildDynamicPlaceMBeanInfo(objectName, mbeanName, mmBeanInfo);
 try {
      m_mBeanServer.invoke(objectName, "setModelMBeanInfo",
```

```
                                new Object[] {mmBeanInfo},
            new String[]{"javax.management.modelmbean.ModelMBeanInfo"});
        } catch (Exception e) {
            output("<MBeanBuilder> kann die invoke-Methode
            zum Setzen von MBeanInfo nicht ausführen!!\n");
            output(e.getMessage());
            e.printStackTrace();
        }

     try {
         m_mBeanServer.invoke(objectName, "setManagedResource",
                    new Object[] {m_placeBean, "objectReference"},
                    new String[]{"java.lang.Object", "java.lang.String"});
     } catch (Exception e) {
         output("<MBeanBuilder> kann die invoke-Methode
         zum Setzen von ManagedRessource nicht ausführen!!\n");
         output(e.getMessage());
         e.printStackTrace();
        }
    }



    /** Methode zum Bilden der MBeanInfo von der AMETAS-Stelle */
  private ModelMBeanInfo buildDynamicPlaceMBeanInfo(ObjectName inMBeanObjectName,
                                  String inMBeanName, ModelMBeanInfo mmBeanInfo) {
        Class placeBean;
        Descriptor placeDescriptor = null;
        ModelMBeanAttributeInfo[] mmbeanAInfo = new ModelMBeanAttributeInfo[6];
        ModelMBeanConstructorInfo[] mmbeanCInfo = new ModelMBeanConstructorInfo[1];
        ModelMBeanOperationInfo[] mmbeanOInfo = new ModelMBeanOperationInfo[7];
        ModelMBeanNotificationInfo[] mmbeanNInfo =
                                            new ModelMBeanNotificationInfo[1];
        String sPersistDir = m_serviceParm.getPath();
        if ((sPersistDir==null)||(sPersistDir.length()==0)) {
            output("<MBeanBuilder> Warnung! Der Pfad zum Speichern
            der MBean wurde nicht eingestellt!.");
            sPersistDir =  m_plThePlace.getLogDir();
        }
```

```
if(!sPersistDir.endsWith(System.getProperty("file.separator")))
   sPersistDir += System.getProperty("file.separator");
sPersistDir += "persistdir";
File f = null;
try {
   f= new File(sPersistDir);
   if(!f.exists())
      f.mkdir();
}catch(NullPointerException npx) {
   String fehler = "<MBeanBuilder><NullPointerException>
   ManagerService kann nicht in Logfile schreiben.\n"+npx.getMessage();
   output(fehler);
}catch(SecurityException sx) {
   String fehler = "<MBeanBuilder><SecurityException>
   ManagerService kann nicht in Logfile schreiben.\n"+sx.getMessage();
   output(fehler);
}

try {
   /*++++++++++++++++++++++++++++++++++++++*/
   /* erzeuge ein Deskriptorobjekt für das Attribut "HostName" */
   Descriptor hostNameDescr =
   new DescriptorSupport(new String[]
                         {"name=HostName",
                          "descriptorType=attribute",
                          "displayName=HostName",
                          "getMethod=getHostName",
                          "currancyTimeLimit=0"})

    /* Bilde ein ModelMBeanInfo für das Attribut "HostName" */
    /* Dieses Attribut beschreibt den Rechnernamen, auf dem die Stelle*/
    /* ausgeführt wird */
    mmbeanAInfo[0] =
            new ModelMBeanAttributeInfo("HostName",
                                        "java.lang.String",
                                        "Name des Computers",
                                         true,false,false,
                                         hostNameDescr);

/* erzeuge ein Deskriptorobjekt für die Operation "getHostName"
```

```
Diese Operation kann den Wert des Attributes "HostName" zurückgeben */
       Descriptor getHostNameDesc =
           new DescriptorSupport(new String[] {"name=getHostName",
                                 "descriptorType=operation",
                                 "class=AMETAS.mgm.PlaceBean",
                                 "role=getter"});

       getHostNameDesc.setField("tagetObject",m_placeBean);
       getHostNameDesc.setField("tagetObjectType","objectReference");

       MBeanParameterInfo[] parms = null;

/* Bilde die ModelMBeanoperationInfo für die Operation "getHostName" */
       mmbeanOInfo[0] =
           new ModelMBeanOperationInfo("getHostName",
                                       "Gibt den Hostname zurueck",
                                       parms,"java.lang.String",
                                       MBeanOperationInfo.INFO,
                                       getHostNameDesc);
       /*++++++++++++++++++++++++++++++++++++++++*/
       /*++++++++++++++++++++++++++++++++++++++++*/
 /* erzeuge ein Descriptorobjekt für das Attribute "PlaceProperty" */
 /* Das Attribut beschreibt die Stelleneinstellungen              */
       Descriptor propDescr =
          new DescriptorSupport(new String[]
              {"name=PlaceProperty",
               "descriptorType=attribute",
               "displayName=PlaceProperty",
               "getMethod=getProperty",
               "currancyTimeLimit=0"});

       /* Bilde ein ModelMBeanInfo für das Attribut "PlaceProperty" */
       mmbeanAInfo[1] = new ModelMBeanAttributeInfo("PlaceProperty",
                                           "java.lang.String",
                                           "Einstellungen der Stelle",
                                           true, true, false,
                                           propDescr);

/* Bilde ein Deskriptorobjekt für die Operation "getProperty".
   Diese Operation muss den Wert vom "PlaceProperty" zurückgeben. */
```

```
    Descriptor getPropDesc =
         new DescriptorSupport(new String[]
             {"name=getProperty",
              "descriptorType=operation",
              "class=AMETAS.mgm.PlaceBean",
              "role=getter"});
   getPropDesc.setField("tagetObject",m_placeBean);
    getPropDesc.setField("tagetObjectType","objectReference");

 /* Bilde die ModelMBeanOperationInfo für die Operation "getProperty" */
    mmbeanOInfo[1] =
       new ModelMBeanOperationInfo("getProperty",
                    "Gibt die Einstellungen der Stelle zurueck",
                    parms, "java.lang.String",
                    MBeanOperationInfo.INFO,
                    getPropDesc);

/* Bilde ein Deskriptorobjekt für die Operation "setProperty".
   Diese Operation muss den Wert vom "PlaceProperty" setzen. */
    Descriptor setPropDesc =
        new DescriptorSupport(new String[]
                             {"name=setProperty",
                              "descriptorType=operation",
                              "class=AMETAS.mgm.PlaceBean",
                              "role=setter"});

    setPropDesc.setField("tagetObject",m_placeBean);
    setPropDesc.setField("tagetObjetcType","objectReference");
    MBeanParameterInfo[] setPropParms =
        new MBeanParameterInfo[] {
            (new MBeanParameterInfo("propertyName",
                                    "java.lang.String",
                                    "Parametername")),
            (new MBeanParameterInfo("propertyValue",
                                    "java.lang.String",
                                    "Wert des Parameters"))};

/* Bilde die ModelMBeanOperationInfo für die Operation "setProperty" */
    new ModelMBeanOperationInfo("setProperty",
                                "Setzt ein Stellenparameter",
```

```
                                        setPropParms, "java.lang.Byte",
                                        MBeanOperationInfo.ACTION, setPropDesc);
 /*+++++++++++++++++++++++++++++++++++++++*/
/* ********************************************************************
   Die weiteren Attribute und die (dazugehörigen) Operationen werden genau so
   wie eben gebildet. Man muss die Größe der Felder nur so wählen, dass
   alle Attribute und Operationen hineinpassen.
 ********************************************************************/
/*+++++++++++++++++++++++++++++++++++++++*/
     /* erzeuge ein Deskriptorobjekt für das Attribute "PlaceName" */
     /* Das Attribut bezeichnet den Stellennamen  */
            Descriptor placeNameDescr =
                new DescriptorSupport(new String[]
                    {"name=PlaceName",
                     "descriptorType=attribute",
                     "displayName=PlaceName",
                     "getMethod=getPlaceName",
                     "currancyTimeLimit=0"});

     /* Bilde eine ModelMBeanInfo für das Attribut "PlaceName" */
         mmbeanAInfo[2] = new ModelMBeanAttributeInfo("PlaceName",
                                             "java.lang.String",
                                             "Name der AMETAS-Stelle",
                                              true,false,false,
                                              placeNameDescr);

      /* Bilde ein Deskriptorobjekt für die Operation "getPlaceName".
         Diese Operation muss den Wert vom "getPlaceName" zurückgeben. */
         Descriptor getPlaceNameDesc =
              new DescriptorSupport(new String[]
                 {"name=getPlaceName",
                  "descriptorType=operation",
                  "class=AMETAS.mgm.PlaceBean",
                  "role=getter"});

         getPlaceNameDesc.setField("tagetObject",m_placeBean);
         getPlaceNameDesc.setField("tagetObjectType","objectReference");

     /* Bilde die ModelMBeanOperationInfo für die Operation "getPlaceName" */
            mmbeanOInfo[3] =
```

```
            new ModelMBeanOperationInfo("getPlaceName",
                                "Gibt den Stellennamen zurueck",
                                parms, "java.lang.String",
                                MBeanOperationInfo.INFO, getPlaceNameDesc);

     /*++++++++++++++++++++++++++++++++++++++*/
    /*++++++++++++++++++++++++++++++++++++++*/
         /* erzeuge ein Deskriptorobjekt für das Attribut "DomainName" */
        /* Das Attribut bezeichnet den Domänennamen  */
          Descriptor domainNameDescr =
             new DescriptorSupport(new String[]
                  {"name=DomainName",
                   "descriptorType=attribute",
                   "displayName=DomainName",
                   "getMethod=getDomainName",
                  "currancyTimeLimit=0"});

       /* Bilde ein ModelMBeanInfo für das Attribut "DomainName" */
           mmbeanAInfo[3] = new ModelMBeanAttributeInfo("DomainName",
                                "java.lang.String","Name der Domäne",
                                true, false, false,domainNameDescr);

          /* Bilde ein Deskriptorobjekt für die Operation "getDomainName".
             Diese Operation muss den Wert vom "DomainName" zurückgeben. */
          Descriptor getDomainNameDesc =
              new DescriptorSupport(new String[]
                  {"name=getDomainName",
                   "descriptorType=operation",
                   "class=AMETAS.mgm.PlaceBean",
                   "role=getter"});

          getDomainNameDesc.setField("tagetObject",m_placeBean);
          getDomainNameDesc.setField("tagetObjectType","objectReference");

    /* Bilde die ModelMBeanOperationInfo für die Operation "getDomainName" */
          mmbeanOInfo[4] =
              new ModelMBeanOperationInfo("getDomainName",
                                          "Gibt den Domainname zurueck",
                                          parms,"java.lang.String",
                                          MBeanOperationInfo.INFO,
```

```
                                        getDomainNameDesc);
        /*++++++++++++++++++++++++++++++++++++++*/
        /*++++++++++++++++++++++++++++++++++++++*/
        /* erzeuge ein Deskriptorobjekt für das Attribut "RelayPlace" */
        /* Das Attribut bezeichnet die Ersatzstelle für diese Stelle. */
        Descriptor relayPlaceDescr =
            new DescriptorSupport(new String[]
                {"name=RelayPlace",
                 "descriptorType=attribute",
                 "displayName=RelayPlace",
                 "getMethod=getRelayPlace",
                 "currancyTimeLimit=0"});

    /* Bilde eine ModelMBeanInfo für das Attribut "RelayPlace" */
       mmbeanAInfo[4] = new ModelMBeanAttributeInfo("RelayPlace",
                            "java.lang.String","Name der Ersatzstelle",
                            true,false,false,relayPlaceDescr);

        /* Bilde ein Deskriptorobjekt für die Operation "getRelayPlace".
            Diese Operation muss den Wert vom "RelayPlace" zurückgeben. */
          Descriptor getRelayPlaceDesc =
            new DescriptorSupport(new String[]
                {"name=getRelayPlace",
                 "descriptorType=operation",
                 "class=AMETAS.mgm.PlaceBean",
                 "role=getter"});

        getRelayPlaceDesc.setField("tagetObject",m_placeBean);
        getRelayPlaceDesc.setField("tagetObjectType","objectReference");

    /* Bilde die ModelMBeanOperationInfo für die Operation "getRelayPlace" */
        mmbeanOInfo[5] =
            new ModelMBeanOperationInfo("getRelayPlace",
                                "Gibt die Ersatz-Stelle zurueck",
                                parms,"java.lang.String",
                                MBeanOperationInfo.INFO, getRelayPlaceDesc);
        /*++++++++++++++++++++++++++++++++++++++*/
        /*++++++++++++++++++++++++++++++++++++++*/
        /* erzeuge ein Deskriptorobjekt für das Attribut "PortNumber" */
        /* Das Attribut bezeichnet die Portnummer der Stelle. */
```

```
        Descriptor portNoDescr =
            new DescriptorSupport(new String[]
                {"name=PortNumber",
                 "descriptorType=attribute",
                 "displayName=PortNumber",
                 "getMethod=getPortNumber",
                 "currancyTimeLimit=0"});

    /* Bilde eine ModelMBeanInfo für das Attribut "PortNumber" */
        mmbeanAInfo[5] =
            new ModelMBeanAttributeInfo("PortNumber",
                                        "java.lang.Integer",
                                        "Portnummer der Stelle für PNS",
                                        true,false,false,portNoDescr);

        /* Bilde ein Deskriptorobjekt für die Operation "getPortNumber".
           Diese Operation muss den Wert vom "PortNumber" zurückgeben. */
         Descriptor getPortNoDesc =
            new DescriptorSupport(new String[]
                {"name=getPortNumber",
                 "descriptorType=operation",
                 "class=AMETAS.mgm.PlaceBean",
                 "role=getter"});

        getPortNoDesc.setField("tagetObject",m_placeBean);
        getPortNoDesc.setField("tagetObjectType","objectReference");

    /* Bilde die ModelMBeanOperationInfo für die Operation "getPortNumber" */
      mmbeanOInfo[6] =
        new ModelMBeanOperationInfo("getPortNumber",
                                    "Gibt die Portnummer der Stelle zurück",
                                    parms, "java.lang.Integer",
                                    MBeanOperationInfo.INFO, getPortNoDesc);
     /*++++++++++++++++++++++++++++++++++++++*/
     /*++++++++++++++++++++++++++++++++++++++*/
        placeBean = Class.forName("AMETAS.mgm.PlaceBean");
        Constructor[] placeMBeanConst = placeBean.getConstructors();

        /* Erzeuge ein Deskriptorobjekt für einen Kostruktur,
           der die Stelle repräsentiert. Die Ausführung der Methoden
```

```
        dieses Objektes nehmen Einfluß auf die Eigenschaften der Stelle.

    Descriptor placeNameConsDesc =
        new DescriptorSupport(new String[]
            {"name=PlaceBean",
             "descriptorType=operation",
             "class=AMETAS.mgm.PlaceBean",
             "visibility=1",
             "role=constructor"});

    MBeanParameterInfo[] mmbeanPInfo = new MBeanParameterInfo[1];
    mmbeanPInfo[0] = new MBeanParameterInfo("place",
                                            "AMETAS.place.AMETASPlace",
                                            "Referenz auf die Stelle");

    /* Erzeuge die ModelMBeanConstructorInfo für die Bean */
    mmbeanCInfo[0] =
        new ModelMBeanConstructorInfo("PlaceBean",
                                      "PlaceBean() : constructs a laceBean",
                                      mmbeanPInfo,
                                      placeNameConsDesc);
    /*++++++++++++++++++++++++++++++++++++++*/
    /*++++++++++++++++++++++++++++++++++++++*/
    /* Erzeuge ein Deskriptorobjekt für ein generisches Ereignis */
    Descriptor genEventDesc =
        new DescriptorSupport(new String[]
             {"descriptorType=notification",
              "name=jmx.ModelMBean.General",
              "severity=4",
              "MessageId=MA001",
              "log=T",
              m_plThePlace.getPlaceName()+"_notification.log"});
    String[] genTypes = new String[1];
    genTypes[0] = new String("jmx.ModelMBean.General");

   /* Bilde die ModelMBeanNotificationInfo für das "jmx.ModelMBean.General"
    mmbeanNInfo[0] =
        new ModelMBeanNotificationInfo(genTypes,
                                       "jmx.ModelMBean.General",
                                       "Generisches Event",
```

```
                                                genEventDesc); // test event
            /*++++++++++++++++++++++++++++++++++++++++*/
            /*++++++++++++++++++++++++++++++++++++++++*/
            /* erzeuge ein Deskriptorobjekt für die Stellen bzw.
               den Repräsentanten der Stelle. */
            placeDescriptor =
                new DescriptorSupport(new String[]
                    {("name="+inMBeanObjectName),
                     "descriptorType=mbean",
                     ("displayName="+inMBeanName ),
                     ("pesistLocation="+sPersistDir),
                     "log=T",
                     ("logFile="+sPersistDir+System.getProperty("file.separator")+
                     m_plThePlace.getPlaceName()+"_mbean.log"),
                     "persistName=placebean",
                     "currencyTimeLimit=10",
                     "persistPeriod=10",
                     "persistPolicy=noMoreOftenThan"});

        /* Bilde jetz die MBeanInfo für die Stelle bzw.
           den Repräsentanten der Stelle */
            mmBeanInfo = new ModelMBeanInfoSupport("AMETAS.mgm.PlaceBean",
                                        "Implementation of ModelMBean for  a Place",
                                         mmbeanAInfo, mmbeanCInfo,
                                         mmbeanOInfo, mmbeanNInfo);

            mmBeanInfo.setMBeanDescriptor(placeDescriptor);
            return mmBeanInfo;
        }catch(RuntimeOperationsException rox) {
            output("<MBeanBuilder>: Descriptor-Error:\n "+rox.getMessage());
            return null;
        }catch(MBeanException mbx) {
            String excep =  "Descriptor-Error or:\n ";
            excep += "<MBeanBuilder> kann "+inMBeanObjectName+
                     " MBean nicht erzeugen!\n";
            output(excep + mbx.getMessage());
            return null;
        }catch(SecurityException sx) {
            output("<MBenBuilder>: Constructor-Error:\n"+sx.getMessage());
            return null;
```

```
            }catch(LinkageError le) {
                output("<MBeanBuilder>: Classfinder-Error:\n"+le.getMessage());
                return null;
            }catch(ClassNotFoundException cnfx) {
                output("<MBeanBuilder>Classfinder-Error:\n"+cnfx.getMessage());
                return null;
            }
        }

         private void echo(String msg) {
            output(msg);
         }

         private void output(String msg) {
            System.out.println(msg);
         }
    }
```

```
/**************************************************************************
   Die Klasse PlaceBean repräsentiert eine AMETAS-Stelle. D.h, ihre Operationen
   wirken auf die Stelle. Wenn eine Managementanwendung auf PlaceBean operiert,
   wird das Ergebnis auf die AMETAS-Stelle bzw. von der AMETAS-Stelle getragen.
**************************************************************************/
package AMETAS.mgm;
import java.lang.*;
import java.util.*;
import AMETAS.place.*;
import AMETAS.pns.*;

public class PlaceBean implements java.io.Serializable {
    /* Referenz auf den ManagerAgent */
    private ManagerAgent m_managerAgent = null;

    /* Referenz auf die AMETAS-Stelle */
    private AMETASPlace m_plThePlace;

    /* enthält temporär die Stellenkonfiguration */
    private Properties m_placeConfigProp = null;

    /* Hostname */
    private String m_shostName = null;

    /* Domänenname */
    private String m_sDomainName = null;

    /* Stellenname */
    private String m_sPlaceName = null;

    /* voller Name der Stelle */
    private String m_sFullName = null;

    /* Ersatzstelle */
    private String m_sRelayPlace = null;

    /* Portnummer */
    private int m_nPortNumber = 0;

    /* Konstuktur der Klasse. Dieser wird vom MBean-Server benötigt */
```

```
public PlaceBean() {
}

/* weiterer Konstuktur der Klasse. */
public PlaceBean(ManagerAgent manager, AMETASPlace place) {
    m_plThePlace = place;
    m_managerAgent = manager;
    m_shostName = m_plThePlace.getHostName();
    m_placeConfigProp = new Properties();
    String sFullName =  m_plThePlace.getPlaceName();
    int index = sFullName.indexOf(".");
    if(index == -1) m_sPlaceName = sFullName;
    else m_sPlaceName = sFullName.substring(0,index);
    getPNSEntry();
}

public String getHostName() {
    return m_shostName;
}

public String getPlaceName() {
    return m_sPlaceName;
}

public String getDomainName() {
    return m_sDomainName;
}

public String getFullName() {
    return m_sFullName;
}

public String getRelayPlace() {
    return m_sRelayPlace;
}

public Integer getPortNumber() {
    return new Integer(m_nPortNumber);
}
```

```
        public String getProperty() {
            return m_plThePlace.configToString();
        }

        public Byte setProperty(String propertyName, String propertyValue) {
            return new Byte(m_plThePlace.updateProperty(propertyName,propertyValue));
        }

        void output(String msg) {
            m_managerAgent.output(msg);
        }
    }
```

A.3 Bilder

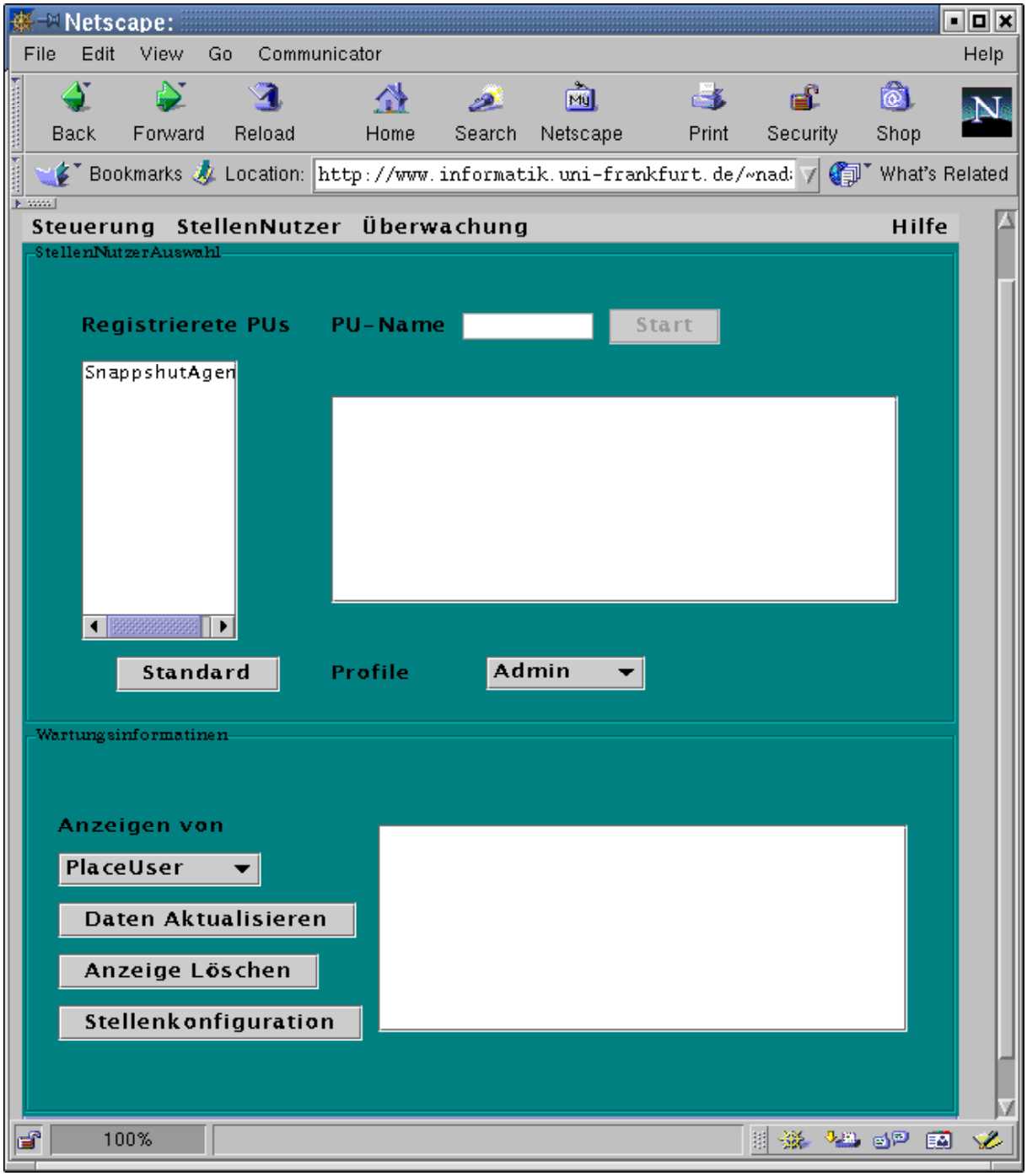


Abbildung A.1: AMAWI-Oberfläche

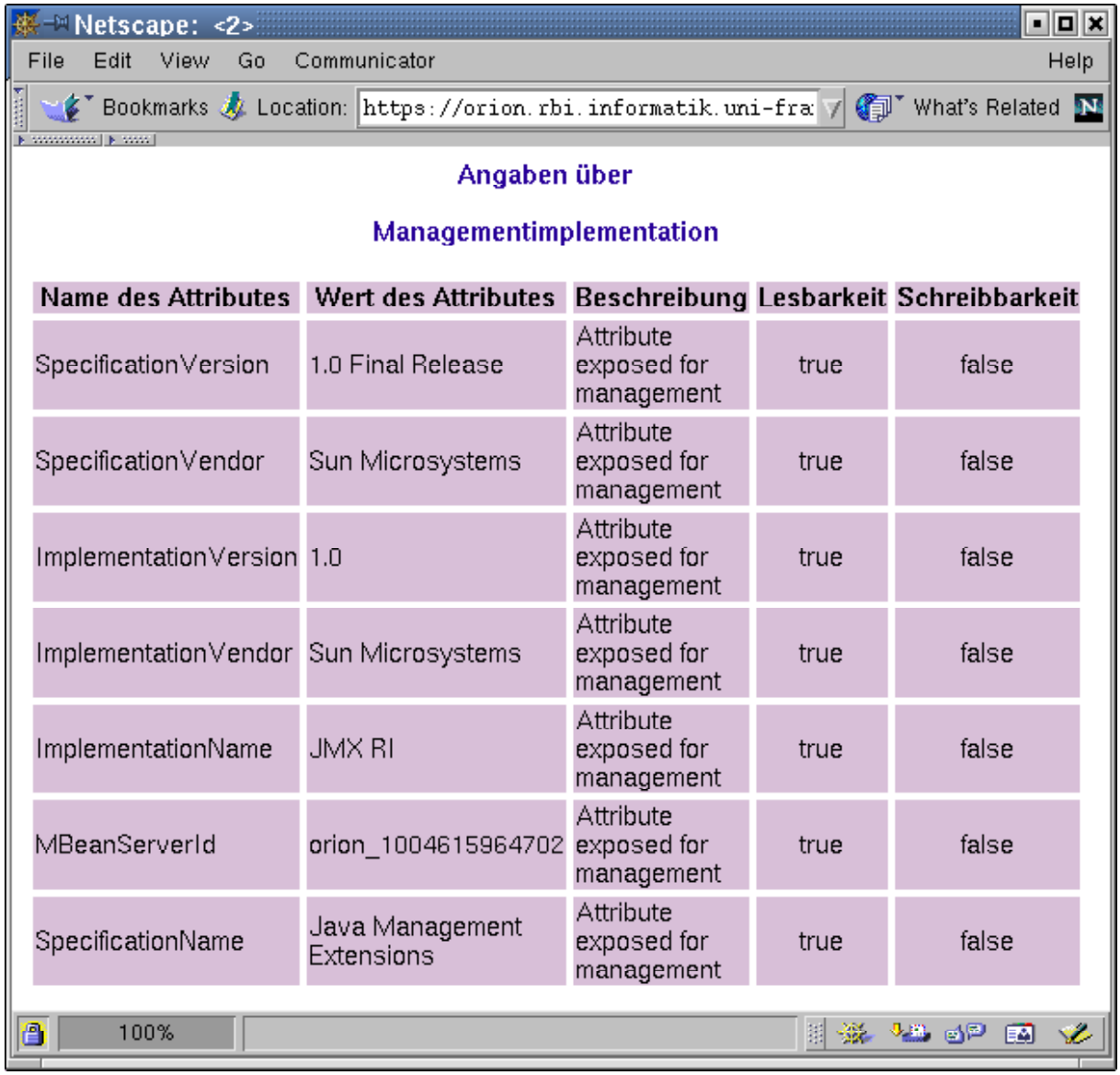
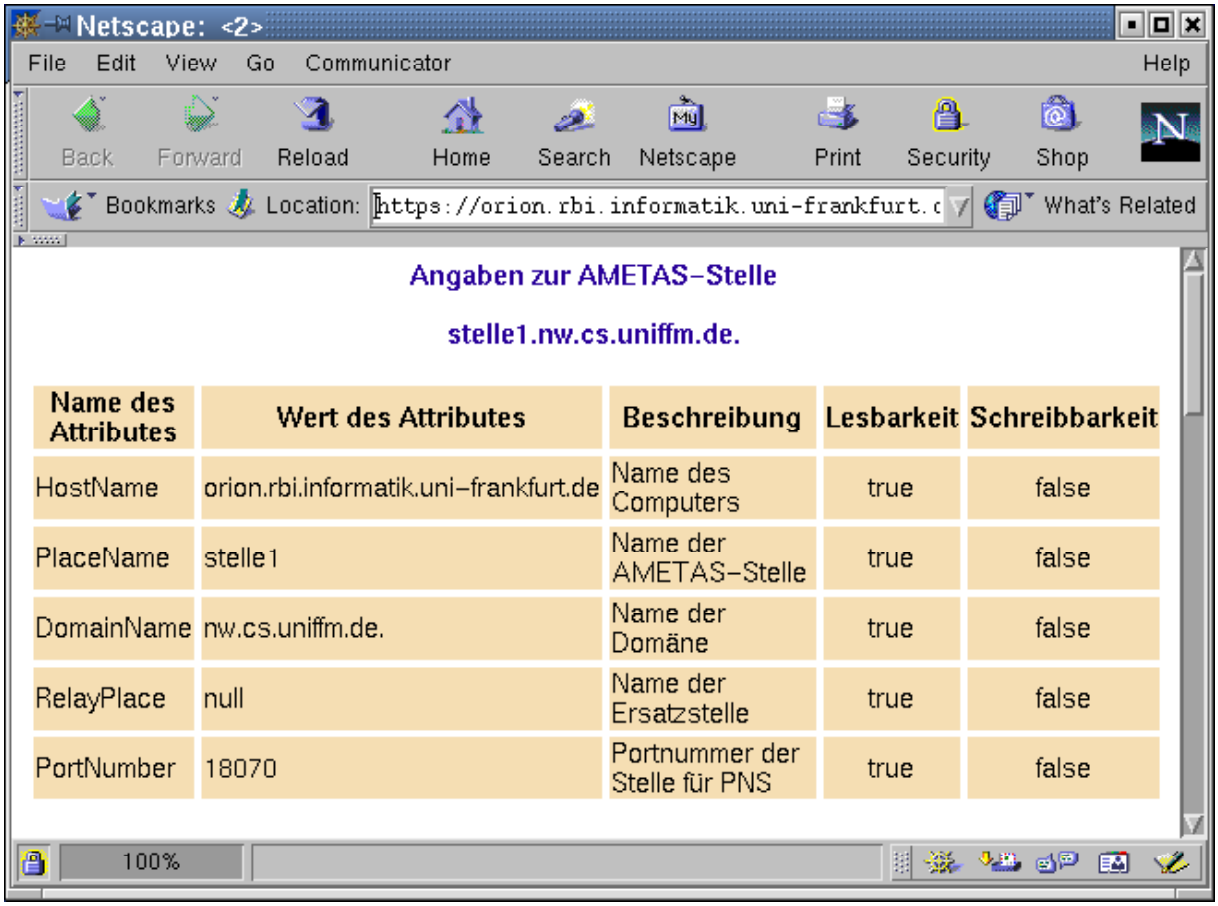


Abbildung A.2: Managementinformationen über den MBean-Server



Name des Attributes	Wert des Attributes	Beschreibung	Lesbarkeit	Schreibbarkeit
HostName	orion.rbi.informatik.uni-frankfurt.de	Name des Computers	true	false
PlaceName	stelle1	Name der AMETAS-Stelle	true	false
DomainName	nw.cs.uniffm.de.	Name der Domäne	true	false
RelayPlace	null	Name der Ersatzstelle	true	false
PortNumber	18070	Portnummer der Stelle für PNS	true	false

Abbildung A.3: Ausgabe der Stelleninformation im Browser

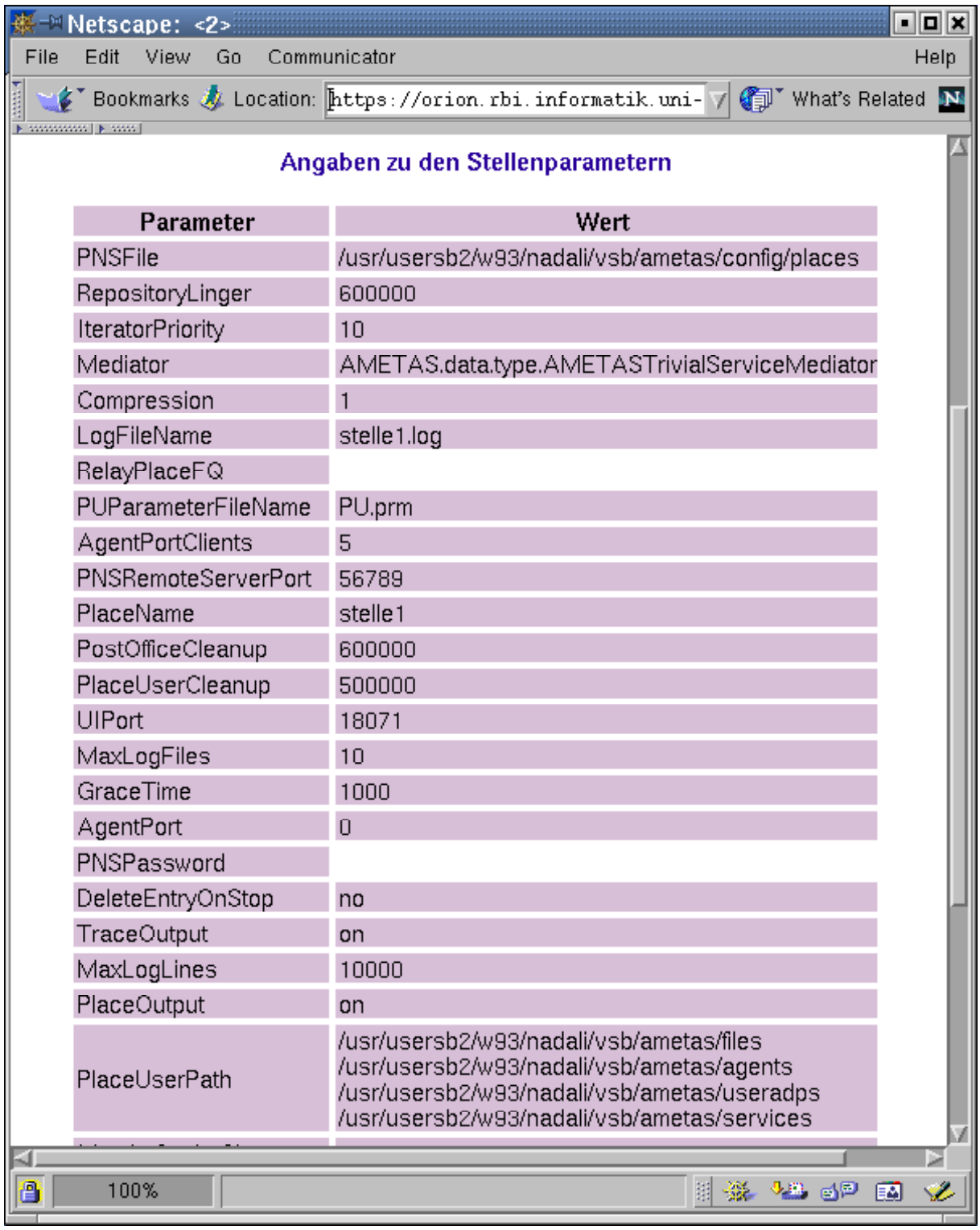


Abbildung A.4: Ausgabe der Stellenparameter im Browser

Literaturverzeichnis

[3WHTTP] *Hypertext Transfer Protocol*, http://www.w3.org/Protocols/.

[AcP200] Austin, Calvin and Pawlan, Monica: *Advanced Programming for the Java 2 Platform*, Addison Wesley, 2000, http://developer.java.sun.com/developer/onlineTraining/Programming/JDCBook/index.html.

[AGLETS] *Aglets Documentation*, http://www.trl.ibm.com/aglets/documentation.html.

[AMAPI] *AMETAS 2.5.4: Öffentliche Paketbeschreibungen* http://www.ametas.de/ametas/doc/api/index.html.

[AMoA] Herrmann, Klaus und Zapf, Michael: *On AMETAS*, July 2000, http://www.ametas.de/ametas/docs/white/OnAMETAS.pdf.

[AMoC] Herrmann, Klaus und Zapf, Michael: *On Communication*, July 2000, http://www.ametas.de/ametas/docs/white/OnCommunikation.pdf.

[AMoP] Herrmann, Klaus und Zapf, Michael: *On Privileges*, July 2000, http://www.vsb.cs.uni-frankfurt.de/ametas/docs/white/OnPrivileges.pdf.

[AMoS] Herrmann, Klaus und Zapf, Michael: *On Security*, July 2000, http://www.vsb.cs.uni-frankfurt.de/ametas/docs/white/OnSecurity.pdf.

[AMTU] Zapf, Michael: *Das AMETAS-Tutorial*, http://www.ametas.de/ametas/doc/Tutorial/

[AMWP] Herrmann, Klaus und Zapf, Michael: *AMETAS* White Paper Series, July 2000, http://www.ametas.de/ametas/docs/white/ALLWhite.pdf.

[AYOM98] Aridor, Yarvi and Oshima, Mitsuru: *Infrastucture for Mobile Agents: Requirements and Design*. In: Kurt Rothermel; Fritz Hohl (Eds.), Mobile Agents, Second International Workshop, (MA '98), Stuttgart, Germany, September 1998, Lecture Notes in Comuter Science 1477, Springer Verlag, Berlin, S. 38-49.

[BAUJ99] Baumann, Joachim: *Control Algorithms for Mobile Agents*, http://elib.uni-stuttgart.de/opus/volltexte/2000/616/.

[BHRRS97] Baumann, J., Hohl, F., Radouniklis, N., Rothermel, K und Straßer, M: *Communication Concepts for Mobile Agent Systems*. Rothermel, K, Popescu-Zeletin, R. (Eds.), Mobile Agents, First International Workshop (MA '97), Berlin, Germany, April 1997. Lecture Notes in Comuter Science 1219, Springer Verlag, Berlin, S. 123-135.

[CIMS] *Common Information Model (CIM) Specification* (Version 2.2) June 14, 1999 http://www.dmtf.org/standards/cim_spec_v22/index.php.

[CORBA-AS] *The Common Object Request Broker: Architectur and Specification*, http://www.omg.org/technology/documents/formal/corba_iiop.htm.

[DMIS] *Desktop Management Interface Specification* (Version 2.0s), 24 June 1998, http://www.dmtf.org/download/spec/dmi20s.zip.

[DMTF] *Ditributed Management Task Force*, http://www.dmtf.org.

[FG96] Franklin, Stan and Gresser, Art: *Is it an Agent, or just a Programm?: A Texonomy for Autonomous Agents*, Proceeding of the Thrid International Workshop on Agents Theories, Architectures, and Lanaguages, Springer-Verlag, 1996.

[FIPA-MA98] FIPA 98 Part 1 Version 1.0: *Agent Management Specification*, http://www.fipa.org/specs/fipa00002/.

[HE98] Herrmann, Klaus: *Netz- und Systemmanagement mit mobilen Agenten im AMETAS-Agentensystem*, Diplomarbeit, August 1998, http://www.vsb.cs.uni-frankfurt.de/lehre/diplom/fertig.html.

[HeAN99] Hegring, H., Abeck, S. und Neumair, B.: *Integriertes Management vernetzter Systeme: Konzepte, Architekturen, und deren betrieblicher Einsatz*, dpunkt Verlag, 1999.

[ISO 10165-4] ISO/IEC 10165-4:1992 Information technology – Open Systems Interconnection – Structure of management information – Part 4: *Guidelines for the definition of managed objects.*

[ISO 7498-4] ISO/IEC 7498-4:1989 Information processing systems – Open Systems Interconnection – Basic Reference Model – Part 4: *Management framework.*

[JDMK] *Java Dynamic Managegement Kit*, http://www.sun.com/software/java-dynamic/.

[JMX] *Java Management Extensions*, http://java.sun.com/products/JavaManagement.

[JMXIAS] *Java Management Extensions Instrumentation and Agent Specification*, v1.0, http://jcp.org/aboutJava/communityprocess/final/jsr003/jmx_instr_agent.pdf.

[JSR3] *JSR3, Java Management Extension (JMX) Spezification*, http://jcp.org/jsr/detail/3.prt.

[JSR48] *JSR48 WBEM Service Specification*, http://jcp.org/jsr/detail/48.prt.

[JSR70] *IIOP Protocll Adapter for JMX Spezification*, http://jcp.org/jsr/detail/70.prt.

[JSR71] *JMX-TMN Specification*, http://jcp.org/jsr/detail/71.prt.

[JSR77] *Java 2 Plattform, Enterprise Edition Management Specification*, http://jcp.org/jsr/detail/71.prt.

[JSSE] *Java Secure Socket Extension (JSSE),http://java.sun.com/products/jsse/*.

[KELL98] Keller, A.: *CORBA-basiertes Enterprise Management: Interoperabilität und Managementinstrumentierung verteilter kooperativer Managementsysteme in heterogener Umgebung*, Technische Universität München, 1998, http://wwwhegering.informatik.tu-muenchen.de/Literatur/MNMPub/Dissertationen/kell98/kell98.shtml

[KuWLuBa] Ku, H., Luderer, G.W.R., and Subbiah, B.: A*n Intelligent Mobile Agent Framework for Distributed Network Management*, Network System Laboratory Telecommunication Research Center, Arizona State University, ftp://ftp.eas.asu.edu/pub/trc/netsys/s5a4.ps.gz.

[LISA98] Lipperts, S. and Park, A.S.: *Managing CORBA with Agents*. 4th International Symposium on Interworking "Interoperability of networks for interoperable services", Interworking98, Otawa, Canada, July 1998.

[LISA99-1] Lipperts, S. and Park, A.S.: *An Agent-based Middleware - Asolution for Terminal and User Mobility*, http://www-i4.informatik.rwth-aachen.de/research/publication/.

[LISA99-2] Park, A.S. and Lipperts, S.: *Mobility Support with a Mobile Agent System*, http://www-i4.informatik.rwth-aachen.de/research/publication/.

[MAFR89] Friedmann Mattern: *Verteilte Basisalgorithemen*, Informatik Fachbrichte 226, Springer-Verlag, Berin Heildelberg, 1989.

[MASIFS] *Mobile Agent System Interoperability Facilities Specification*, ftp://ftp.omg.org/pub/docs/orbos/97-10-05.pdf.

[MIL ET AL.98] Milojicic, D., Breugst, M. u.a.: *MASIF The OMG Mobile Agent System Interoperability Facility*, Mobile Agents, Second International Workshop, (MA '98), Stuttgart, Germany, September 1998, Kurt Rothermel; Fritz Hohl (Eds.), Lecture Notes in Comuter Science 1477, Springer Verlag, Berlin, S. 50-67.

[MOLE] *The Home of the Mole*, http://mole.informatik.uni-stuttgart.de/.

[OMAG] *Discussion of the Object Management Architecture (OMA) Guide*, http://www.omg.org/technology/documents/formal/object_management_architecture.htm.

[OMG] *Object Management Group (OMG)*, *http://www.omg.org/news/about/index.htm*.

[OUKA99] Ouahid, H. and Karmouch, A.: *An XML-based Web Mining Agen*t, Mata '99, First International Workshop on Mobile Agents for Telecommunication Applications, Ottawa, Canada, Ahemed Karmouch, Roger Impe, World Scientific, S. 393-403.

[RFC2246] *The Transport Layer Security (TLS) Protocol Version 1.0* http://www.ietf.org/rfc/rfc2246.txt.

[RO90] Rose, M.T.: *The Open Book, A Practical Perspective On OSI*, Prantice-Hall, 1990.

[RO91] Rose, M.T.: *The Simple Book, An Introduction to Networking Managemnet*. Prantice-Hall, 1991.

[RUM91] J. Rumbauh, J., Blaha, M., Premerlani, W., Eddy, F. and W. Lorensen: *Object-oriented Modelling and Design*. Prantice-Hall, Inc. 1991.

[SSLv3] *The Secure Sockets Layer (SSL) Protocol Version 3,0* http://www.netscape.com/eng/ssl3/draft302.txt.

[STA93] W. Stallings: *SNMP, SNMPv2 and CMIP The Practical Guide to Network Management Standards*, Addison-Wesley, 1993.

[UMLS] *OMG Unified Modeling Language Version 1.3 specification* http://www.rational.com/media/uml/post.pdf.

[WBEM] *Web-Based Interprise Management (WBEM) Initiative*, http://www.dmtf.org/standards/standard_wbem.php.