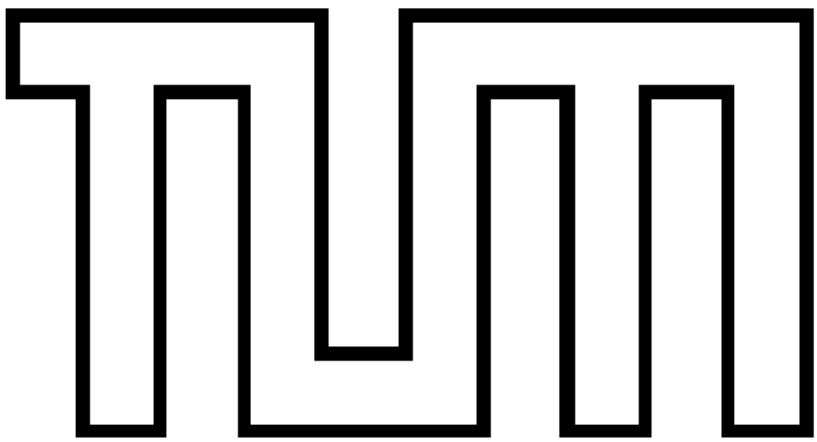
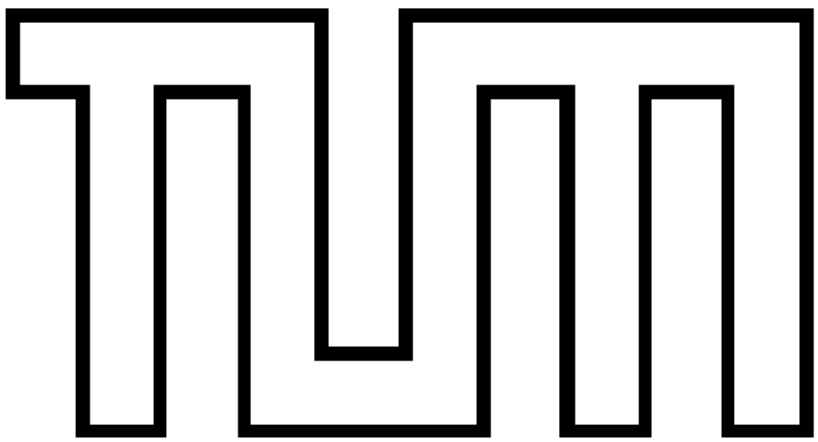


INSTITUT FÜR INFORMATIK
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Diplomarbeit

Erstellung eines Kommunikationsmodells für kooperierende Agenten für das Management von verteilten Systemen

Markus Wennrich



INSTITUT FÜR INFORMATIK
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Diplomarbeit

Erstellung eines Kommunikationsmodells für kooperierende Agenten für das Management von verteilten Systemen

Bearbeiter: Markus Wennrich
Aufgabensteller: Prof. Dr. H.-G. Hegering
Betreuer: Maria-Athina Mountzia
Abgabedatum: 15. Februar 1997

Ehrenwörtliche Erklärung

Ich versichere, daß ich diese Diplomarbeit selbständig verfaßt und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 15. Februar 1997

.....

(Unterschrift des Kandidaten)

Abstract

Bisher baute das Netz- und Systemmanagement auf einem zentralen Manager-Agenten Ansatz auf. Um jedoch die immer größer, komplexer und verteilter werdenden Netze adäquat managen zu können, gehen viele neue Ansätze in Richtung eines verteilten Managements. Einer dieser Ansätze ist das Konzept des Management by Delegation, das durch die Verwendung von kooperierenden Agenten durchgesetzt werden soll. In dieser Arbeit wird eine Kommunikationsmodell und ein auf KQML aufbauendes Kommunikationsprotokoll für solche kooperierenden Agenten entwickelt. Außerdem wird eine prototypische Implementierung vorgestellt.

1 Einführung 1

2 Motivation 2
2.1 Zentraler Ansatz 2
2.2 Vorhandene dezentrale Ansätze 3
2.3 Fazit 5
2.4 Lösungsansatz „flexible Agenten“ 5
2.4.1 Definition 5
2.4.2 Szenarien 6
2.4.2.1 Szenario: Webserver-Logging 7
2.4.2.2 Szenario: Virus-Checking 8
2.4.2.3 Szenario: Monitoring und Recovering eines Webservers 8
2.4.3 Neue Aspekte 10
2.4.3.1 dynamische Erweiterbarkeit 10
2.4.3.2 Modellierung der Managementapplikationen 10
2.4.3.3 Nichtdeterminismus 11
2.4.3.4 Kompatibilität zu vorhandenen Managementarchitekturen 11
2.4.3.5 Sicherheitsaspekte 11
2.4.3.6 Einheitliche Kommunikation und Kooperation 12
2.4.3.7 Management der Agenten notwendig 12
2.4.4 Einordnung der Diplomarbeit 13

3 Kommunikationsmodell 14
3.1 Anforderungen an das Kommunikationsmodell 14
3.2 Randbedingungen 16
3.3 Sicherheit 17
3.3.1 Anforderungen 17
3.3.2 Verschlüsselungstechniken 18
3.3.3 Agent-Key-Server 21
3.3.4 Agent-Resource-Server 22
3.4 Organisationsform 23
3.4.1 Vollstruktur 24
3.4.2 Hierarchische Struktur 25
3.4.3 gemischter Ansatz 26
3.4.4 Organisationskriterien 27
3.4.5 Agent-Domain-Name-System 28
3.5 Dienstlokalisierung 29
3.5.1 Statischer Ansatz 30
3.5.2 Vermittlung 31
3.5.3 Broadcast 33
3.5.4 Intelligent Naming 34
3.5.4.1 Beispiel für Intelligent Naming: 34
3.5.4.2 Definierte Klassennamen 35

3.5.4.3 Instanzabhängige Namen ... 36
3.5.4.4 Freie Namen ... 36
3.6 Ereignismeldungen ... 36
3.6.1 Anforderungen ... 36
3.6.2 Agent Event Server ... 37
3.7 Zusammenfassung ... 38

4 Kommunikationsprotokoll ... 39
4.1 Interaktionsprotokoll ... 39
4.1.1 Anforderungen ... 39
4.1.2 Nameserver-Protokoll ... 41
4.1.3 Eventserver Protokoll ... 43
4.1.4 Key-Server Protokoll ... 44
4.1.5 Resource-Server Protokoll ... 45
4.1.6 Agent - Agent Protokoll ... 47
4.2 Agentensprache ... 48
4.2.1 Anforderungen ... 48
4.2.2 KQML ... 49
4.3 Transportprotokoll ... 50
4.3.1 Anforderungen ... 50
4.3.2 Existierende Transportprotokolle ... 51
4.3.2.1 HyperText Transfer Protocol (HTTP) ... 51
4.3.2.2 Simple Mail Transfer Protocol (SMTP) ... 51
4.3.2.3 Simple Network Management Protocol (SNMP) ... 52
4.3.2.4 Common Object Broker Architekture (CORBA) ... 52
4.4 Internet Management ... 52
4.4.1 Simple Network Management Protocol (SNMP) ... 52
4.4.2 SNMP als Agentensprache ... 53
4.4.3 SNMP als Transportprotokoll ... 53
4.4.4 Fazit ... 54

5 Existierende Agentenmodelle ... 55
5.1 KQML ... 55
5.1.1 Kommunikationsmodell ... 56
5.1.2 KQML Nachrichten ... 56
5.1.3 Protokollstack ... 57
5.1.3.1 Content Layer ... 58
5.1.3.2 Message Layer ... 58
5.1.3.3 Communication Layer ... 59
5.1.4 Beispiele ... 60
5.1.4.1 Beispiel Accounting ... 60
5.1.4.2 Beispiel Verfügbarkeit ... 61
5.2 CORBA ... 62
5.2.1 Überblick ... 62

5.2.2 Einsatz von CORBA in einem Multi-Agentensystem 64
5.2.2.1 CORBA zum Transport von Nachrichten 64
5.2.2.2 Weiterer Einsatz von CORBA 65
5.2.3 CORBA Agent Facility 65

6 Prototypische Implementierung 68
6.1 Ansatz 68
6.1.1 Agenten 68
6.1.1.1 Plattform 68
6.1.1.2 Programmiersprache 68
6.1.2 Funktionalität 69
6.1.3 Entwicklungsumgebung 71
6.1.4 Bibliotheken 71
6.2 Das Java Agent Template 71
6.2.1 Architektur 72
6.2.2 Interpreter 74
6.2.3 Änderungen am JAT 75
6.3 Architektur 76
6.4 Server 76
6.4.1 Class AInterpreter 76
6.4.2 ANSInterpreter 77
6.4.3 AESInterpreter 79
6.4.4 ARSInterpreter 81
6.4.5 AKSInterpreter 82
6.5 Beispiel-Managementanwendung 83

7 Ausblick 84
7.1 Erweiterung des Prototypen 84
7.2 CORBA Mobile Agent Facility 85
7.3 JMAPI 85
7.4 mobile Agenten 87

8 Literaturverzeichnis 88

Abbildungsverzeichnis

Abbildung 2-1 Zentraler Manager-Agenten Ansatz 2
Abbildung 2-2 Dezentraler Ansatz 3
Abbildung 2-3 Szenario Webserver-Logging 7
Abbildung 2-4 Szenario Recovery-Server 9
Abbildung 3-1 Schlüsselverteilung (symmetrisch) 21
Abbildung 3-2 Delegierungsmechanismus 23
Abbildung 3-3 Vollstruktur 24
Abbildung 3-4 Hierarchische Struktur 25
Abbildung 3-5 Teambildung 26
Abbildung 3-6 Auslassen von Hierarchiestufen 27
Abbildung 3-7 Mehrere übergeordnete Agenten 27
Abbildung 3-8 Agent-Name-System 29
Abbildung 3-9 Vermittler 31
Abbildung 3-10 Broadcast 33
Abbildung 3-11 Beispiel eines Klassenbaums 35
Abbildung 3-12 Agent-Event-Server System 38
Abbildung 4-1 Agent ↔ ANS Protokoll 41
Abbildung 4-2 ANS ↔ ANS Protokoll mit Weiterleitung 42
Abbildung 4-3 ANS ↔ ANS Protokoll ohne Weiterleitung 42
Abbildung 4-4 AES Protokoll 43
Abbildung 4-5 Agent-Key-Server Protokoll (symmetrisch) 44
Abbildung 4-6 Agent-Resource-Server Protokoll 46
Abbildung 5-1 Einordnung der Agenten Technologien 55
Abbildung 5-2 KQML Multiagenten-System 56
Abbildung 5-3 KQML Protokoll Stack 57
Abbildung 5-4 CORBA Architektur 63
Abbildung 5-5 CORBA Agent Facility 66
Abbildung 6-1 Architektur des Java Agent Template 72
Abbildung 6-2 Class ANSInterpreter 77
Abbildung 6-3 Class AESInterpreter 79
Abbildung 6-4 Event Datenstruktur 80
Abbildung 6-5 Class ARSInterpreter 81
Abbildung 7-1 JMAPI Architektur 86

Tabellenverzeichnis

Tabelle 5-1: KQML Syntax in BNF 57
Tabelle 5-2 Reservierte Schlüsselworte 58
Tabelle 5-3 Reservierte Performative Namen 59

1 Einführung

Durch die zunehmende Verteilung und Vernetzung von Kommunikationsressourcen, nimmt die Größe und Heterogenität der Netze immer mehr zu. Das führt dazu, daß das Management solcher komplexen Umgebungen immer schwieriger wird.

Die heutzutage eingesetzten Managementsysteme basieren auf einem zentralen Manager-Agenten Ansatz, der nicht flexibel genug ist, um den Anforderungen solcher komplexen, verteilten Systeme genüge leisten zu können.

Da aber die Verteilung von Ressourcen und Diensten einer der wichtigsten Aspekte in der Realisierung von Managementanwendungen ist, gehen viele neue Ansätze in Richtung eines verteilten Managements. Einer dieser Ansätze, um verteilte, flexible Managementsysteme zu ermöglichen, ist das Konzept des Management by Delegation, das durch die Verwendung von kooperierenden Agenten durchgesetzt werden soll. Bestandteile dieses Konzeptes sind sogenannte flexible Agenten, deren Funktionalität dynamisch erweitert werden kann und die mit anderen Agenten zur Verwirklichung der Managementaufgaben kooperieren.

In dieser Arbeit werden die Anforderungen, die an das Kommunikationsmodel und an das Kommunikationsprotokoll eines solchen Multi-Agentensystems gestellt werden, vorgestellt. Außerdem werden einige vorhandene Modelle und Protokolle, die in verteilten Umgebungen eingesetzt werden, auf ihre Verwendbarkeit hin untersucht.

Aufgrund der Ergebnisse wird ein Kommunikationsmodell und ein auf KQML aufbauendes Kommunikationsprotokoll für solche kooperierenden Agenten entwickelt, sowie eine prototypische Implementierung vorgestellt.

2 Motivation

In den folgenden Absätzen werden die momentanen angewendeten Ansätze des Netz- und Systemmanagements und ihre Grenzen kurz analysiert.

Darauf folgend wird das Konzept des „flexiblen, kooperierenden Agenten“ und seine Vorteile gegenüber den anderen Ansätzen vorgestellt, sowie die neuen Aspekte, die mit diesem Ansatz auftreten, anhand einiger Szenarien erläutert.

2.1 Zentraler Ansatz

Heute bauen vorhandene Netzmanagementarchitekturen auf einem zentralisierten Manager - Agenten Ansatz auf. Die zu managenden Objekte (MO) werden durch Agenten repräsentiert und von einem zentralen Manager verwaltet.

Die Nachteile eines solchen Ansatzes werden nun anhand von SNMP verdeutlicht, gelten aber genauso für CMIP.

Zentrales Netz- und Systemmanagement bedeutet, daß die Netzmanagementapplikation, die sich auf dem Manager befindet, von ihren Daten und Diensten, die sie benötigt, getrennt ist, da diese sich auf den Agenten befinden. Somit sind Agenten im heutigen Netzmanagement nichts anderes als „Datenserver“, die Daten und Parameter über die MO’s in ihrer MIB anbieten und dem Manager einen „remote access“ darauf ermöglichen bzw. ihm erlauben, einzelne Parameter des MO zu setzen.

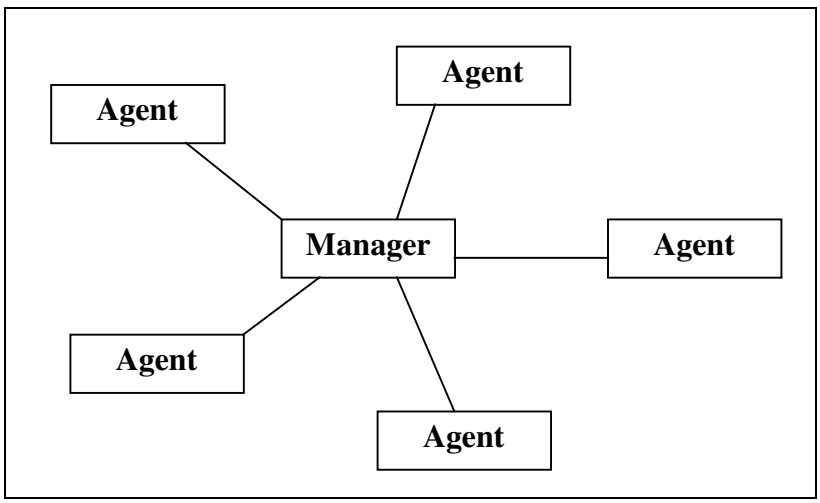


Abbildung 2-1 Zentraler Manager-Agenten Ansatz

Normale Managementaufgaben, wie Erstellen von Statistiken, Überwachen von Verfügbarkeit usw., müssen demzufolge alle auf dem Manager stattfinden.

Daraus folgt, daß das Managen eines solchen Netzes nur bis zu einem bestimmten Grad möglich ist, da irgendwann die Ressourcen - sowohl die Bandbreite und die Rechenzeit, als auch der zur Verfügung stehende Speicherplatz - ausgeschöpft sind.

Die Dienste und Informationen, die ein solcher Agent zu Verfügung stellen kann, müssen schon zum Zeitpunkt der Entwicklung festgelegt werden. Der Agent ist demnach nicht ohne weiteres um neue Funktionalitäten erweiterbar.

Da nun die Managementapplikation nur auf standardisierte Dienste zugreifen kann, müssen Managementprozeduren durch viele, kleine Einzelzugriffe auf die Agenten gelöst werden. Das führt wiederum zu einer Verknappung der vorhandenen Ressourcen.

Ein wesentlicher Bestandteil des Netzmanagement stellt das Wiederherstellen der Netzverfügbarkeit im Fehlerfalle dar. Aber gerade dann, wenn keine oder nur eine schlechte Verbindung zwischen dem Manager und seinen Agenten besteht, ist ein Managen des Netzes nicht mehr möglich, da die Agenten auf Instruktionen des Managers warten müssen.

Gleiches gilt natürlich und besonders für den Fall, wenn der Manager einmal ausfällt.

2.2 Vorhandene dezentrale Ansätze

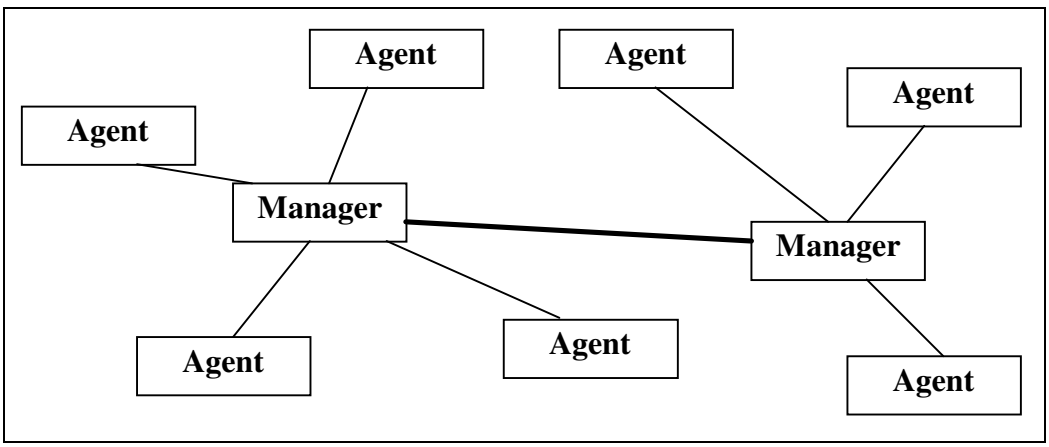


Abbildung 2-2 Dezentraler Ansatz

Diese Nachteile des zentralisierten Managements wurden früh erkannt und es gibt verschiedene Ansätze, um Netzmanagement zu dezentralisieren:

- **RMON-MIB**

  Mit der Hilfe der RMON-MIB [RFC 1757] zum Beispiel wurde erreicht, daß einfache Überwachungsaufgaben von dem Manager in den Agenten übertragen werden können. Dazu gibt es pro Subnetz einen Monitor, der alle Pakete im Subnetz auswertet und statistische Angaben über das Subnetz in der RMON-MIB ablegt. Außerdem kann der Monitor bei Über- bzw. Unterschreiten eines vorgegebenen Schwellwertes den Manager benachrichtigen.

  Die Funktionalität des Monitors muß allerdings schon zum Zeitpunkt des Entwurfs festgelegt werden und ist nicht nachträglich veränderbar.

- **Manager-to-Manager MIB**

  Ein weiterer Ansatz ist die Manager-to-Manager MIB (M2M-MIB) [RFC 1451]. In der M2M-MIB kann ein Manager seine Informationen anderen Managern verfügbar machen, indem er anderen Managern gegenüber die Agentenrolle übernimmt. Hierdurch wird eine gewisse Skalierbarkeit des SNMP-Managements erreicht.

  Ebenso wie bei der RMON-MIB kann der Manager, beim Erreichen bestimmter Schwellwerte andere Manager benachrichtigen, wobei im Gegensatz zu RMON, der Manager sich nicht auf ein Subnetz beschränken muß.

- **Mid-Level-Manger MIB**

  Der Mid-Level-Manager (MLM) kann, genauso wie die M2M-MIB, sowohl die Rolle des Managers, als auch die des Agenten übernehmen. Der MLM-Ansatz ist der erste, bei dem Funktionalität in Form von sogenannten SNMP-Scripts weitergegeben werden kann. Dazu wurde die MLM-MIB definiert, die aus Tabellen besteht, in die die Script-Zeilen eingetragen, sowie die Ausführung der Scripts gesteuert werden kann.

  Obwohl schon ein Schritt in die richtige Richtung getan wurde, ist die Funktionalität der SNMP-Scriptsprache doch sehr eingeschränkt. Außerdem ist die Übertragung der Scripte mit SNMP nicht sehr effizient.

- **Script MIB**

  Die Script MIB stellt eine Erweiterung des MLM-Ansatzes dar, indem dort auf die Verwendung der SNMP-Scriptsprache verzichtet wird und auch andere Scriptsprachen zugelassen werden. Außerdem wird hier die Übertragung der Scripts nicht mehr über SNMP realisiert.

  Eine Kommunikation unter den Agenten bzw. Managern ist jedoch auch hier nicht vorgesehen.

2.3 Fazit

Obwohl die neuen dezentralen Ansätze schon eine deutlichen Fortschritt gegenüber dem zentralen Management darstellen, sind ihre Möglichkeiten immer noch stark eingeschränkt.

Bei den meisten Ansätzen muß der Manager nach wie vor auf vorgegebene Dienste zurückgreifen. Eine dynamische Erweiterung der Funktionalität des Agenten wäre nicht nur wünschenswert, sondern wird zur Bewältigung zukünftiger Aufgaben im Netzmanagement dringend benötigt.

Selbst in den Fällen, in denen eine Erweiterung der Funktionalität möglich ist, beschränkt sich diese meist auf das Ablaufen lassen von Skripten. Diese Abläufe können jedoch, abgesehen vom Beenden, nicht von außerhalb beeinflußt werden.

Zudem bauen die vorhandenen Ansätze immer noch auf ein streng hierarchisches Konzept auf, das heißt, daß - falls eine der „Managerkomponenten“ mal ausfällt - kein Management mehr möglich ist.

In diesem Falle wäre eine Art Eigendynamik des Agenten von Nöten. Diese könnte durch eine kooperative Kommunikation unter den Agenten unterstützt und ermöglicht werden.

All diese Punkte führen zu dem Schluß, daß ein neuer Ansatz gebraucht wird: Der flexible Agent.

2.4 Lösungsansatz „flexible Agenten“

In den folgenden Absätzen wird der Begriff des „flexiblen Agenten“ definiert und seine Funktionsweise anhand von Szenarien verdeutlicht.

2.4.1 Definition

Unter dem Begriff „Agent“ oder „Intelligenter Agent“ werden viele verschiedene Konzepte zusammengefaßt, so daß es unmöglich erscheint, eine klare Definition dafür zu geben.

Statt dessen wird der Begriff „Agent“ in der Literatur häufig durch seine Charakteristika umschrieben:

- **Autonomie**

  Agenten sind Einheiten, die mehr oder weniger autonom arbeiten, das heißt, ohne direkte Einflußnahme eines Benutzers oder eines anderen Agenten. Sie agieren aufgrund von Veränderungen in ihrer Umwelt.

Personal Agents werden von einem Benutzer mit einem Auftrag los geschickt, den sie ohne weitere Einflußnahme des Benutzers erfüllen.

- **Intelligenz**

  Um seine Aufgaben ausführen zu können, benötigt der Agent einen gewissen Grad an „Intelligenz“, das heißt, er benötigt Wissen über seine Umgebung und seine Ziele.

- **Kommunikation**

  Ein Agent kommuniziert mit seiner Umwelt, z. B. mit Benutzern, Datenbanken, Systemressourcen oder anderen Agenten.

- **Kooperation**

  Um sein Ziel/seine Aufgabe erfüllen zu können, kooperiert der Agent mit anderen Agenten. Diese Kooperation kann vom einfachen Client-Server-Konzept über Delegierung von Aufgaben bis hin zu Methoden der verteilten KI reichen.

- **Mobilität**

  Agenten können ihre Aufgaben auf entfernten Netzkomponenten automatisch ausführen, ohne daß es dort einer Installation oder Konfiguration bedarf. Das kann von einem einfachen RPC-Aufruf bis zum „elastic processing“ [YeSi 96] reichen, bei dem ein laufender Agent ohne Unterbrechung seiner Tätigkeiten auf einen anderen Rechner umzieht.

- **Flexibilität**

  Flexiblen Agenten kann dynamisch zur Laufzeit neue Funktionalität hinzugefügt werden, ohne daß die bisherigen Ziele, die der Agent verfolgt, dadurch beeinträchtigt werden.

Somit stellt der „flexible Agent“ im Netzmanagement eine Einheit dar, die

- um neue Managementfunktionalitäten erweitert werden kann,
- eigenständig auf Änderungen im Netz reagiert,
- mit ihrer Umwelt kommuniziert, und
- mit anderen Agenten kooperiert.

2.4.2 Szenarien

Um eine Vorstellungen zu erhalten, wie solch ein Netzmanagementsystem aus kooperierenden, flexiblen Agenten aussehen könnte, werden hier ein paar Szenarien vorgestellt:

2.4.2.1 Szenario: Webserver-Logging

Bei großen WWW-Sites wird der Webserver normalerweise auf verschiedene Rechner verteilt, um so eine gewisse Lastverteilung zu erreichen, und um die Ausfallsicherheit zu erhöhen. Das hat zur Folge, daß die Logfiles dieser WWW-Sites auf verschiedenen Rechnern verteilt sind.

Wenn nun ein Webmaster wissen will, wie oft auf eine bestimmte Seite in einem bestimmte Zeitraum zugegriffen wurde, so muß er sich auf allen Rechnern einloggen und die Logfiles auswerten.

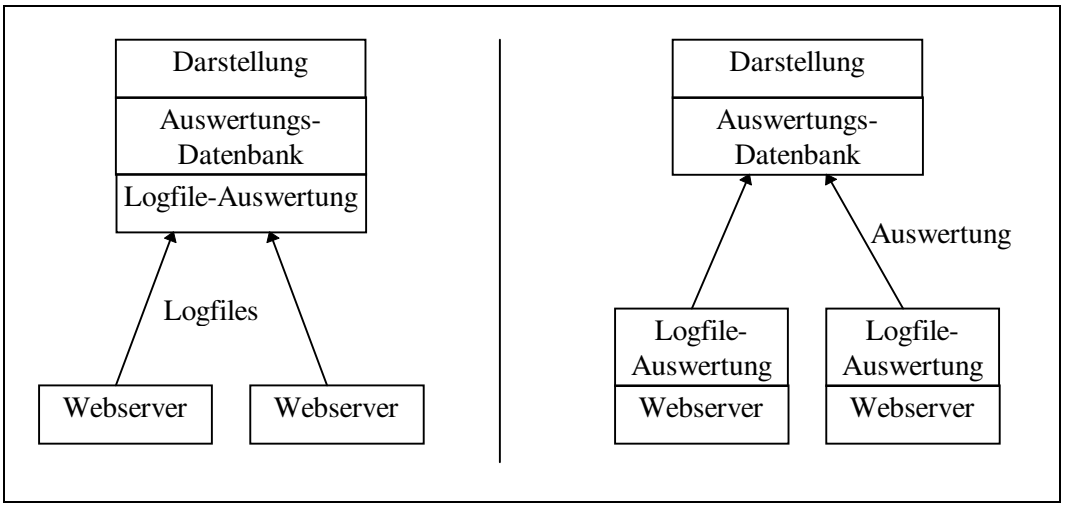


Abbildung 2-3 Szenario Webserver-Logging

Diese Aufgabe soll jetzt automatisch von einer Netzmanagementplattform übernommen werden. Angenommen, die Logfiles werden über SNMP von einem traditionellen SNMP-Agenten angeboten (viele Webserver bieten so etwas an), dann müßte die Plattform sich jedesmal die kompletten Logfiles für diesen Zeitraum von allen Rechnern laden und sie dann auswerten, oder die Logfiles in regelmäßigen Abständen laden und speichern, um sie bei Bedarf bereitzuhalten. Da aber Logfiles in sehr kurzer Zeit sehr groß werden können, ist keines der beiden Vorgehen adäquat.

Viel besser wäre es, wenn die Auswertung der Logfiles schon vor Ort bei den Webservern vorgenommen werden könnte und nur noch die Ergebnisse zum Manager geschickt werden müßten.

Hätte man einen flexiblen Agenten, würde man ihn nur noch um die Fähigkeit, diese Logfiles auszuwerten, erweitern, und ihm dann einen Auftrag mit den dazugehörigen Parametern schicken. Nachdem er dann die Logfiles ausgewertet hat, schickt er die Ergebnisse an den Manager zurück.

Für den Fall, daß man Webserver hat, die kein SNMP anbieten, könnte man auf jedem der Webserver einen flexiblen Agenten positionieren, der die Logfiles direkt aus dem Dateisystem einliest und auswertet. Die Ergebnisse könnte er

dann an den Manager schicken oder an einen weiteren Agenten, der die Ergebnisse der verschiedenen Webserver noch mal zusammenfaßt und an den Manager weiterleitet.

2.4.2.2 Szenario: Virus-Checking

Durch die zunehmende Vernetzung von PCs wird die Gefahr der schnellen Verbreitung von Viren immer größer. Vor allem wenn man an den Trend zum Netzwerk Computer (NC) denkt, der nur noch Programme ausführt, die auf entfernten File-Servern liegen, wird schnell klar, wie wichtig die schnelle Erkennung und Beseitigung von Computerviren ist.

In einem System von flexiblen Agenten könnte das so aussehen, daß auf jedem File-Server ein Agent ist, der in regelmäßigen Abständen nach Viren scannt. Wenn nun ein Virus in einer Datei entdeckt wird, so könnte der Agent folgendermaßen vorgehen:

- Benachrichtigen aller File-Server über die Art des Virus und in welcher Datei er gefunden wurde.
- Auftrag an den Agent senden, der für die Logfiles des File-Servers zuständig ist, ihm alle Clients, die auf diese Datei zugegriffen haben, zu nennen.
- Auftrag an die Agenten senden, auf diesen Clients ebenfalls nach Viren zu scannen.
- Nachricht über den Virus an den Manager schicken. Falls der Virus nicht entfernt werden konnte, Nachricht an den File-Server-Agenten, diese Datei zu sperren.
- Falls der Virus im Speicher entdeckt wurde, Notfall-Nachricht an den Manager schicken, und Auftrag an den System-Agenten, den PC anzuhalten.

2.4.2.3 Szenario: Monitoring und Recovering eines Webservers

Häufig gibt es, vor allem in internen Netzen, Server, deren Verfügbarkeit äußerst wichtig für ein Unternehmen ist. Dazu gehören vor allem File-Server und Authentifizierungs-Server, aber mit der zunehmenden Bedeutung von den sogenannten „Intranets“ auch immer mehr Webserver.

Deshalb werden für solche wichtigen Server sogenannte Recovery-Server eingerichtet, die eingesetzt werden, wenn der eigentliche Server ausfällt. Diese Recovery-Server enthalten die Daten von allen möglichen Servern, so daß sie für jeden Server einspringen können.

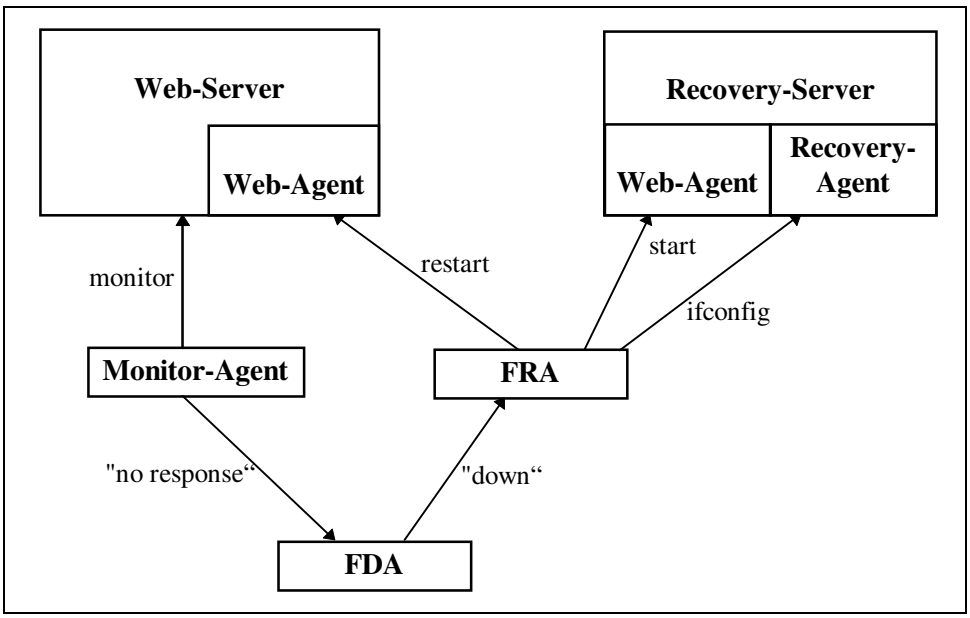


Abbildung 2-4 Szenario Recovery-Server

In einem Multi-Agentensystem könnte der Einsatz von solchen Recovery-Servern, wie folgt aussehen:

- Ein Monitor-Agent überprüft ständig verschiedene Webserver.
- Wenn ein Webserver mal nicht erreichbar ist oder nicht auf einen HTTP-request antwortet, so verschickt der Monitor-Agent einen Event.
- Ein Fehler-Diagnose-Agent (FDA) sammelt diese Events auf und korreliert sie. Nach bestimmten Regeln folgend (im einfachsten Fall zum Beispiel: dreimal hintereinander nicht erreichbar), erklärt er einen Server für „down“ und verschickt wiederum einen entsprechenden Event.
- Diesen Event erhält unter anderem der Fehler-Reperatur-Agent (FRA). Falls nur der Web-Server nicht erreichbar war, der Rechner aber schon, wird der FRA als erstes den Web-Agenten auf dem Web-Server beauftragen, den Server zu restarten.
- Sollte ein restart nicht erfolgreich gewesen oder der ganze Rechner nicht mehr verfügbar sein, so schickt der FRA einen Auftrag an den Recovery-Agenten auf dem Recovery-Server, die entsprechende IP-Adresse zu aktivieren. Außerdem schickt er den Auftrag, den Web-Service zu starten, an den Web-Agenten auf dem Recovery-Server.
- Der Web-Server läuft nun auf dem Recovery-Rechner.

2.4.3 Neue Aspekte

Wie man anhand der Szenarien gut sehen kann, ergeben sich natürlich - trotz aller Vorteile, die dieses Konzept anbietet - neue Aspekte, die beachtet werden müssen:

2.4.3.1 dynamische Erweiterbarkeit

Ein wesentlicher Bestandteil des flexiblen Agenten ist seine Fähigkeit, zur Laufzeit neue Funktionalität aufzunehmen. Dazu gehört natürlich nicht nur der neue Code, der ausgeführt werden soll, sondern vor allem auch die Informationen über diese neue Funktionalität.

Es müssen demnach Wege gefunden werden, wie Funktionalität von einem Agenten zum nächsten übertragen werden kann.

Dazu kommt, daß die Agenten auf den unterschiedlichsten Plattformen laufen. Das bedeutet, daß Funktionalitäten in einer maschinenunabhängigen Sprache entwickelt werden müssen.

Außerdem bedarf es einer Schnittstelle, die die Anbindung der neuen Funktionalität an den Agenten ermöglicht, mitsamt den Informationen, die der Agent darüber benötigt, um sie anzuwenden. Diese Schnittstelle muß natürlich flexibel genug sein, um auch zukünftigen Anwendungen gerecht zu werden.

2.4.3.2 Modellierung der Managementapplikationen

Die meisten Managementaufgaben lassen sich nicht so ohne weiteres in ein dezentrales Konzept eingliedern. Sicherlich gibt es viele, kleine Aufgaben, deren Ausführung man so ohne weiteres auf den Agenten legen kann, wie z. B. das einfache Monitoren von Netzkomponenten.

Aber die komplexen Managementapplikationen, die mit der zunehmenden Wichtigkeit der Netze, benötigt werden, müssen erst für einen dezentralen Ansatz neu modelliert werden. Das heißt, daß die Probleme, die mit der Anwendung gelöst werden sollen, in Teilaufgaben zerlegt werden müssen. Zu diesen Teilaufgaben können dann Funktionalitäten entwickelt werden, die von den einzelnen Agenten ausgeführt werden können.

Diese Neumodellierung aller Managementaufgaben für eine verteilte Umgebung, unter Ausnutzung aller Möglichkeiten der kooperierenden Agenten, wird noch viel Forschung und Entwicklung benötigen. Da aber der Umstieg von zentralem Management zu dezentralem dringend nötig ist, ist diese Neumodellierung ohnehin unumgänglich.

2.4.3.3 Nichtdeterminismus

Netzmanagement mit flexiblen Agenten bedeutet, daß es keinen allmächtigen, zentralen Manager mehr gibt, sondern ein Netz von vielen, kleinen, relativ autonom handelnden Agenten mit unterschiedlichen Zielen, die auf ihre Umgebung reagieren und sich gegenseitig beeinflussen, indem sie sich Nachrichten bzw. neue Aufgaben schicken.

Nun kann es natürlich dazu kommen, daß zwei Agenten unterschiedliche Aufgaben haben, die sich gegenseitig widersprechen. (Im einfachsten Falle wollen beide die gleichen MIB-Variablen mit unterschiedlichen Werten besetzen). Wenn so etwas nicht verhindert wird, kann das fatale Folgen haben. (Man denke nur an zwei Agenten, die gleichzeitig die Routingtabelle ändern wollen und diese dann in einem inkonsistenten Zustand zurücklassen.)

Es bedarf also eines Semaphorenkonzeptes für Managed Objects. Bei MIB-Variablen mag das noch relativ einfach zu entwickeln sein, wie sieht es aber z. B. mit Prozessen aus, die gestartet oder beendet werden sollen?

2.4.3.4 Kompatibilität zu vorhandenen Managementarchitekturen

Im Laufe der Jahre wurden viele, zum Teil äußerst komplexe, Managementanwendungen entwickelt, auf deren Einsatz auch in Zukunft nicht verzichtet werden kann.

Obwohl diese häufig, wie in Kapitel 2.4.3.2 beschrieben, nicht so ohne weiters in ein dezentrales Konzept umgewandelt werden können, können sie unter Umständen dennoch von den neuen Möglichkeiten der flexiblen Agenten profitieren.

Zum Beispiel könnte ein flexibler Agent vor Ort SNMP-Traps filtern, bevor er sie auf ganz normalem Wege zum zentralen Manager schickt.

Auch wäre es möglich, daß die zentrale Managementanwendung auf eine, von einem flexiblen Agenten aufbereitete MIB, zugreift, anstatt auf die von der Netzkomponente tatsächlich angebotene.

Deshalb ist es wichtig, die neuen Agenten so weit wie möglich kompatibel zu anderen/alten Managementarchitekturen zu halten, ohne ihre neuen Möglichkeiten einzuschränken

2.4.3.5 Sicherheitsaspekte

Ein weiteres Problem, das allgemein mit der Einführung von flexiblen Agenten auftaucht, ist die potentielle Gefährdung der Sicherheit.

Einem flexiblen Agenten wird neue Funktionalität zugeschickt, mit dem Auftrag, sie mit gewissen Parametern auszuführen. Nun muß gewährleistet sein, daß diese neue Funktionalität nicht bösartig ist.

Bei „normalen“ flexiblen Agenten könnte dies dadurch gelöst werden, daß die Agenten schon auf Grund ihrer Architektur nichts Schädliches anrichten können. (Wie dies z. B. bei Java-Applets gelöst ist.) In der Umgebung des Netz- und Systemmanagements ist dies aber nicht möglich, da gerade hier auf Dateien (wie z. B. Logfiles oder Konfigurationsdateien) zugegriffen werden muß. Um einige Managementaufgaben ausführen zu können, müßte der Agent sogar Administrationsrechte erhalten.

Es muß also sichergestellt werden, daß kein „bösartiger“ Code in das Agentensystem eingeschleust werden kann.

Genauso muß sichergestellt werden, daß keine „bösartigen“ Nachrichten eingeschleust werden, die den vorhandenen „nützlichen“ Code mißbrauchen.

2.4.3.6 Einheitliche Kommunikation und Kooperation

Wie oben in den Szenarien gezeigt, reicht es nicht aus, daß zwei Agenten mit derselben Funktionalität miteinander kommunizieren können. Häufig ist es auch sinnvoll und notwendig, daß Agenten mit unterschiedlichen Aufgaben und Funktionalitäten miteinander kommunizieren.

Das bedeutet, daß nicht nur die Kommunikation zur Delegierung und Aufgabenerteilung einheitlich sein muß, sondern auch jegliche Kommunikation zur Aufgabenbewältigung und Kooperation.

Dabei ist darauf zu achten, daß die Kooperation zwischen den Agenten möglichst gut gefördert wird, das heißt, daß dem Entwickler von verteilten, kooperierenden Managementapplikationen möglichst viel abgenommen wird.

Dazu wird sowohl ein Naming- bzw. ein Directory-Service benötigt, als auch ein komfortables Event-Handling, bis zur Möglichkeit, Multicast-Nachrichten an eine Gruppe von Agenten zu versenden.[1]

2.4.3.7 Management der Agenten notwendig

Ausgehend von sehr großen Netzen, mit sehr vielen Agenten, wird es schnell klar, daß das Managen der flexiblen Agenten an sich schon eine komplexe Aufgabe darstellt. Es müssen Agenten hinzugefügt bzw. entfernt werden, den Agenten müssen Namen zugewiesen werden, Aufgaben müssen verteilt werden usw.

Man erkennt sofort, daß es erforderlich ist, die Agenten zu organisieren, das heißt zum Beispiel in einzelne, kleine Managementdomänen zusammenzufassen, so daß viele kleine funktionierende Agentensysteme zusammen ein komplexes, funktionierendes Netz von flexiblen Agenten bilden.

[1] Siehe auch Kapitel 3.1, S. 14

2.4.4 Einordnung der Diplomarbeit

Ziel dieser Arbeit ist es, ein Kommunikationsmodell und ein entsprechendes Kommunikationsprotokoll für ein System aus flexiblen, kooperierenden Agenten zu erstellen.

In Kapitel 3 werden die Anforderungen, die bei der Verteilung von Managementaufgaben an das Kommunikationsmodell entstehenden, identifiziert und untersucht.

Anschließend werden in Kapitel 4 die Anforderungen erarbeitet, die dabei an das Kommunikationsprotokoll entstehen. Es werden verschiedene Transportprotokolle auf ihre Verwendbarkeit hin untersucht. Zudem wird geprüft, inwiefern SNMP als Protokoll verwendet werden kann.

In Kapitel 5 schließlich werden zwei bekannte Technologien aus dem Bereich der intelligenten Agenten und der verteilten Anwendungen, KQML und CORBA, vorgestellt und untersucht.

In Kapitel 6 wird dann ein, in Java prototypisch implementierter, flexibler Agent vorgestellt.

Mit einigen Ausblicken in Kapitel 7 wird die Arbeit abgeschlossen.

3 Kommunikationsmodell

Im folgenden soll das Kommunikationsmodell für ein Multi-Agentensystem aus flexiblen, kooperierenden Agenten entwickelt werden.

Im Allgemeinen umfaßt Kommunikation im Netzmanagement folgende Punkte [HeAb 94]:

- Austausch von Informationen, die der Beeinflussung von Managed Objects dienen
- Statusabfragen von Managed Objects
- Asynchrone Ereignismeldungen (Events)

Die Spezifikation eines Kommunikationsmodells muß demzufolge folgende Punkte behandeln:

- Spezifikation der Kommunikationspartner
- Spezifikation der Kommunikationsmechanismen dieser Punkte. Das heißt, Spezifikation der Dienste und Protokolle für Managementanwendungen.
- Definition des Kommunikationsprotokolls (Syntax und Semantik)
- Einbettung des Kommunikationsprotokolls in die Architektur und in die darunterliegenden Protokollschichten.

3.1 Anforderungen an das Kommunikationsmodell

Als erstes müssen die Aspekte, die es bei der Erstellung eines Kommunikationsmodells zu beachten gilt, definiert werden.

- **Sicherheit**

  Wie bei allen Anwendungen in öffentlichen Netzen, ist die Sicherheit immer ein großes Problem, mit dem sich jedes Kommunikationsmodell beschäftigen muß. Man kann nicht davon ausgehen, daß alle Netzkomponenten, die genutzt werden, auch „sicher“ sind.

Sowohl das unbefugte Abhören von Nachrichten zwischen Agenten, als auch das Einschleusen von gefälschten Nachrichten bzw. gefälschter Funktionalität, stellen Gefahren für ein verteiltes Multi-Agentensystem dar.

Man sieht somit, daß es eines Sicherheitskonzeptes bedarf, das solche Angriffe auf die flexiblen Agenten möglichst verhindert.

- **Organisationsform**

  Im zentralen Manager-Agenten-Ansatz gab es nur einen Manager und viele Agenten. Daher war es praktisch nicht möglich, die Agenten in irgendeiner Art und Weise zu strukturieren, das heißt, z. B. zu Gruppen zusammenzufassen oder Hierarchien zu bilden.

  In einer verteilten Umgebung, mit verteilen Managementanwendungen ist nun ein verteiltes Management der Agenten möglich und auch notwendig.

  Obwohl diese Fragestellung zur Problematik des Organisationsmodell gehört, soll sie hier erörtert werden, da sich aus der gewählten Organisationsform Anforderungen an das Kommunikationsmodell ergeben.

- **Dienstlokalisierung**

  Wie bei allen Anwendungen mit verteilten Daten und verteilten Diensten ist die Dienstlokalisierung ein wichtiger Aspekt.

  Es ist somit ein Mechanismus zu entwerfen, der es sowohl dem Manager, als auch dem kooperierenden Agenten ermöglicht, Daten und Dienste im Netz ausfindig zu machen.

- **Ereignismeldungen**

  Neben der Möglichkeit, normale Nachrichten an andere Agenten zu verschicken, z. B. Anfragen, Antworten, Erteilung neuer Aufträge etc., sollte es noch einen Mechanismus geben, um Ereignisse, die bei einem Agenten auftreten, an alle anderen Agenten, die diese Ereignismeldung betrifft, zu verteilen, ohne diese Agenten kennen zu müssen.

- **Kommunikationsprotokoll**

  Ein entscheidender und wichtiger Punkt in der Behandlung des Kommunikationsmodells ist das gewählte Kommunikationsprotokoll.

  In ihm wird festgelegt, in welcher Form Informationen transportiert werden und in welcher Art und Weise sie angefordert werden.

  Die Anforderungen an das Kommunikationsprotokoll werden in Kapitel 4 dieser Arbeit genauer untersucht.

3.2 Randbedingungen

Neben den oben dargelegten Anforderungen an das Kommunikationsmodell, gilt es noch folgende Randbedingungen, die bei Entwicklungen im Rahmen des Netz- und Systemmanagements immer zu berücksichtigen sind, zu beachten:

- **Offenheit**

  All die oben genannten Anforderungen sollten immer unter dem Gesichtspunkt, möglichst konform zu vorhandenen Standards zu bleiben, betrachtet werden.

- **Kompatibilität zu existierenden Managementarchitekturen**

  Wie in Kapitel 2.4.3.4 schon beschrieben, ist es nicht so ohne weiters möglich, alle vorhandenen Managementanwendungen in ein dezentrales Konzept umzuwandeln. Daher sollte der Einsatz flexibler Agenten zumindest auf keinen Fall die bestehenden Managementarchitekturen beeinträchtigen.

  Aufgrund der Tatsache, daß, wie oben schon erwähnt, zentrale Anwendungen durchaus Nutzen aus flexiblen Agenten ziehen könnten, ist zu überprüfen, inwiefern flexible Agenten kompatibel zu bestehenden Managementarchitekturen sein können.

- **Stabilität gegenüber Netzausfällen**

  Neben anderen Aufgaben ist das Fehlermanagement eines der wichtigsten Aufgaben des Netz- und Systemmanagements. Das bedeutet aber, daß eine Managementarchitektur gerade dann besonders gut funktionieren muß, wenn es zu Ausfällen im Netz kommt.

  Gerade in diesem Punkt bietet das Konzept des flexiblen Agenten Vorteile gegenüber dem zentralen Ansatz, da der Agent autonom tätig werden kann, auch wenn keine Verbindung mehr zum Manager besteht.

  Es ist nun zum Teil Aufgabe des Kommunikationsmodells, diesen Vorteil zu nutzen, und keinen single-point-of-failure zu kreieren.

- **Effizienter Ressourcenverbrauch**

  Obwohl Informationen und Dienste - aufgrund des Problems der Netzausfälle - möglicherweise redundant gehalten werden müssen, darf dadurch nicht der Gewinn an Ressourcen, den verteilte Managementanwendungen bieten, wieder verbraucht werden.

  Zudem sollten die gewählten Kommunikations- und Transportprotokolle nicht unnötigen Overhead produzieren.

3.3 Sicherheit

Aufgrund der Tatsache, daß es bekanntermaßen nicht nur wohlgesinnte Mitmenschen gibt, stellt die Sicherheit immer einen wichtigen Punkt dar, besonders aber in verteilten Systemen, da dort zum Großteil unsichere Medien zur Übertragung von Informationen verwendet werden.

Im Netz- und Systemmanagement stellen vor allem das Konfigurations- und das Accountingmanagement gefährdete Bereiche dar, da in diesem Fall die Agenten Administrationsrechte benötigen, um die Aufträge erfüllen zu können.

Ein weiteres Problem bei den flexiblen Agenten ist die Ausführung von delegierter Funktionalität, und somit die Gefahr, „bösartigen“ Code auszuführen.

3.3.1 Anforderungen

Folgende Anforderungen werden von Seiten der Sicherheit an das Kommunikationsmodell gestellt:

- **authentifizierte Nachrichten**

  Da sämtliche Nachrichten und somit auch Aufträge und Funktionalität zum größten Teil über unsichere Medien übertragen werden, muß sichergestellt sein, daß keine Nachrichten verfälscht oder falsche Nachrichten eingespielt werden können.

  Im Konfigurationsmanagement zum Beispiel, könnte sich sonst ein Angreifer so unbefugten Zugriff zu Ressourcen verschaffen.

  Außerdem muß verhindert werden, daß es einem Angreifer gelingt, einen gefälschten Agenten bzw. gefälschte Funktionalität in das Multi-Agentensystem einzuschleusen.

  Das bedeutet demnach, daß sowohl der Absender einer Nachricht, als auch die Korrektheit des Inhaltes der Nachricht gewährleistet sein müssen.

- **Verschlüsselung kritischer Daten**

  In einigen Fällen genügt nicht nur die Authentifizierung von Nachrichten, sondern es müssen die Nachrichten auch vor unbefugtem „Abhören“ geschützt werden, da sie einen vertraulichen Inhalt besitzen.

  Im Abrechnungsmanagement würde das sonst z. B. einem Angreifer ermöglichen, vertrauliche Informationen über die Benutzer des Netzes ausfindig zu machen.

  Somit bedarf es auch eines Verschlüsselungskonzept für vertrauliche Nachrichten.

- **Domänen-Konzept**

  Falls es einem Angreifer doch einmal gelungen sein sollte, in das Multi-Agentensystem falsche Nachrichten oder falsche Funktionalität einzuschleusen, sollte der Schaden, den er anstellen kann, so gering wie möglich gehalten werden. Das heißt, nicht jeder Agent sollte von jedem Agenten jeden Auftrag annehmen.

  Außerdem lassen sich so, die unter Umständen vorhandenen Sicherheitspolicies in das Multi-Agentensystem mit aufnehmen, so daß der Administrator der einen Domäne nicht die Agenten einer anderen Domäne verwalten kann.

- **Authentifizierte Funktionalität**

  Aufgrund der Tatsache, daß ein flexibler Agent zur Laufzeit neue Funktionalität aufnehmen und ausführen kann, muß gewährleistet werden, daß diese neue Funktionalität sicher ist und nicht von einem bösartigen Angreifer eingeschleust wurde.

  Dies ist besonders in Hinsicht auf die möglichen Administrationsrechte der Agenten wichtig. Deswegen läßt sich auch die Ausführung von Funktionalität an sich nicht von vornherein sicher gestalten.[2]

3.3.2 Verschlüsselungstechniken

Im folgendem sollen verschiedene, existierende Verschlüsselungsverfahren mit ihren Vor- und Nachteilen kurz vorgestellt werden, ohne genauer auf Details einzugehen:

- **globale, symmetrisches Verfahren**

  Es existiert ein Master-Key K_M für alle Agenten, mit dem sie ihre Nachrichten verschlüsseln können. Dadurch können die Nachrichten sowohl vor Verfälschungen, als auch vor unbefugtem Abhören geschützt werden.

  Vorteil: keine Schlüsselverteilung notwendig

  Nachteil: Wenn der eine Schlüssel aufgebrochen wurde, liegt das ganze Multi-Agentensystem für den Angreifer frei. Jeder Agent kann jede Nachricht entschlüsseln. Authentizität des Absenders ist nicht gewährleistet.

- **sitzungsabhängige, symmetrische Verfahren**

  Jedes Agentenpaar besitzt einen gemeinsamen Schlüssel $K_{A,B}$, mit dem sie ihre Nachrichten verschlüsseln. Damit können Nachrichten vor Verfäl-

[2] wie dies zum Beispiel bei Java-Applets mit Hilfe der virtuellen Maschine gemacht wurde. [Java 95]

schung und Abhören geschützt werden, außerdem ist die Authentizität des Absenders gewährleistet.

Vorteil: Wenn ein Schlüssel gebrochen wurde, so liegt nur eine Verbindung für den Angreifer offen.

Nachteil: Es werden viele Schlüssel[3] benötigt. Eine Nachricht, die an eine Vielzahl von Agenten versendet werden soll, kann nicht verschlüsselt werden.

- **Public-Key Verfahren[4]**

  Jeder Agent besitzt einen öffentlich bekannten Schlüssel K_p und einen geheimen Schlüssel K_s. Nachrichten, die mit einem von den beiden Schlüsseln verschlüsselt wurden, können nur mit dem anderen entschlüsselt werden.

  Somit kann der Absender A einer Nachricht authentifiziert werden, indem A die Nachricht mit seinem geheimen Schlüssel verschlüsselt. Wenn nun der Empfänger B die Nachricht mit dem öffentlichen Schlüssel des Absender A entschlüsseln kann, so weiß er, daß die Nachricht von A stammen muß, da nur dieser den geheimen Schlüssel kennt.

  Genauso kann der Absender A die Nachricht mit dem öffentlichen Schlüssel des Empfängers B verschlüsseln, und weiß, daß nur der Empfänger B die Nachricht entschlüsseln kann, da nur dieser den geheimen Schlüssel kennt.

  Somit wäre es also möglich, Nachrichten vor Verfälschung und Abhören zu schützen, sowie die Authentizität des Absenders zu sichern. Wenn das Schlüsselpaar eines Agenten gebrochen wurde, so liegen alle Verbindungen, die von dem Agenten weg und zu dem Agenten hin führen, offen, das heißt, jeder der Angreifer kann die Nachrichten an den Agenten mitlesen und falsche Nachrichten unter dem Namen des Agenten verschicken.

  Vorteil: Im Gegensatz zum sitzungsabhängigen, symmetrischen Verfahren werden nur *n* Schlüsselpaare benötigt. Außerdem kann auch für Nachrichten, die an mehrere Agenten versendet werden sollen, die Authentizität gewährleistet werden.

  Nachteil: Wurde das Schlüsselpaar eines Agenten gebrochen, so liegen alle Verbindungen dieses Agenten offen, und nicht nur eine einzige zu einem anderen Agenten. Außerdem ist die Verschlüsselung mit Public-Key-Verfahren äußerst komplex und benötigt relativ viel Rechenleistung.

---

[3] $\left(\frac{n*(n-1)}{2}\right)$

[4] exemplarisch für asynchrone Verfahren

- **Hash-Verfahren**[5]

  Da die Verschlüsselung von Nachrichten häufig sehr aufwendig ist und viel Rechenleistung auf seiten des Agenten erfordert, werden oft mit Hilfe von Hash-Verfahren sogenannte fingerprints von Nachrichten erstellt. Diese haben die Eigenschaft, daß es sehr schwer ist, eine zweite Nachricht zu kreieren, die den gleichen fingerprint hat, wie die erste.

  Diese fingerprints wiederum werden dann verschlüsselt und mit der Nachricht gemeinsam verschickt.

  Vorteil: Es muß nur der (relativ kleine) fingerprint verschlüsselt werden, nicht die ganze Nachricht.

  Nachteil: Die Nachricht liegt immer noch unverschlüsselt im Klartext vor. Der fingerprint gewährleistet nur Authentizität und Unverfälschtheit.

- **Schlüsselverteilung**

  Sowohl bei sitzungsabhängigen Schlüsseln, als auch in Public-Key-Verfahren, werden eine Vielzahl von Schlüsseln verwendet, um zu gewährleisten, daß nicht ein gebrochener Schlüssel zum Verlust der Sicherheit im ganzen Multi-Agentensystem führt.

  Ein Agent kann aber seinen Schlüssel nicht einfach seinen Kommunikationspartnern mitteilen, da er Gefahr läuft, daß diese Nachricht abgehört oder verfälscht wird. Folglich müßten alle Agenten die öffentlichen Schlüssel aller anderer Agenten von vornherein kennen bzw. mit allen anderen Agenten einen gemeinsamen, sitzungsabhängigen Schlüssel besitzen. Diese Schlüssel müßten von einem Bearbeiter direkt den Agenten mitgegeben werden (nicht über das unsichere Netz, sondern über andere Wege, beispielsweise via Disketten).

  Würde nun ein neuer Agent hinzukommen, so müßte der Bearbeiter die neuen Schlüssel erzeugen und allen anderen Agenten mitteilen.

  Da dieser Aufwand nicht tragbar ist, wird in den meisten Fällen ein sogenannter Authentication-Server verwendet. Dieser kennt alle öffentlichen Schlüssel bzw. hat gemeinsame Schlüssel mit jedem Agenten. Wenn nun ein Agent eine sichere Verbindung mit einem anderen Agenten aufbauen will, so kann er den entsprechenden Schlüssel beim Authentication-Server anfordern. Diese Verbindung ist durch die Verwendung des gemeinsamen Schlüssels zwischen dem Agenten und dem Server gesichert.

  Vorteil: Agenten müssen nicht von vornherein alle Schlüssel kennen. Bei neu hinzugekommenen Agenten muß nur der Schlüssel mit dem Authentication-Server auf anderem Wege eingetragen werden.

[5] zum Beispiel MD5

Nachteil: single-point-of-failure beim Authentication-Server. Wurde der gemeinsame Schlüssel eines Agenten mit dem Server gebrochen, so liegen alle Verbindungen dieses Agenten frei.

3.3.3 Agent-Key-Server

Wie im vorherigen Absatz gezeigt, gibt es eine Vielzahl von Verschlüsselungsverfahren und -techniken. Welches Verfahren nun eingesetzt werden soll, hängt von der jeweiligen Managementanwendung und ihren speziellen Anforderungen an die Sicherheit ab. So ist zum Beispiel bei statistischen Aufgaben unter Umständen gar keine Verschlüsselung notwendig, da diese Zahlen nicht sicherheitsrelevant sind. Bei anderen Aufgaben (z. B. Accounting) dagegen, ist aufgrund des Datenschutzes ein sehr hoher Sicherheitslevel von Nöten. Daher sollte es dem Agenten freistehen, ob er eine Nachricht vollkommen unverschlüsselt übertragen oder sie durch einen verschlüsselten Hash-Key authentifizieren will, oder ob sie komplett verschlüsselt und nur für den Empfänger lesbar sein soll.

Welche Verschlüsselungstechnik (symmetrisch oder asymmetrisch) eingesetzt werden soll, und welche Länge die Schlüssel haben sollen, hängt von der jeweiligen Implementierung bzw. von der Rechenkapazität der Agenten ab. Ein Agent kann auch mehrere Verschlüsselungstechniken beherrschen. Wichtig ist nur, daß beide Kommunikationspartner mindestens ein gemeinsames Verfahren kennen.

Schlüsselverteilung

Aufgrund der Tatsache, daß das Multi-Agentensystem aus vielen Agenten bestehen kann, sollte ein Mechanismus zur Schlüsselverteilung eingesetzt werden, um den Aufwand für den Bearbeiter möglichst klein zu halten. Zu diesem Zweck werden sogenannte Agent-Key-Server (AKS) eingesetzt:

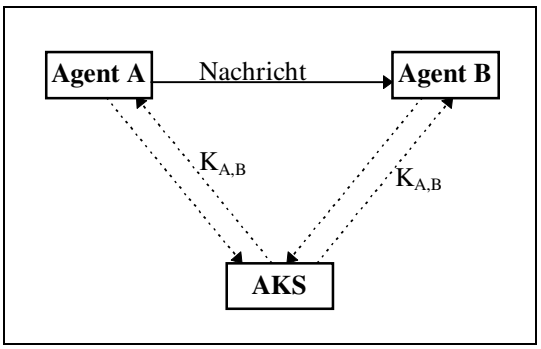


Abbildung 3-1 Schlüsselverteilung (symmetrisch)

- Jeder Agent hat einen gemeinsamen Schlüssel K_A mit dem Agent-Key-Server, der ihm vorliegt (z. B. in einer schreib/lese-geschützen Datei)
- Ebenso hat der AKS zu jedem Agenten in seiner Domäne einen Schlüssel.

- Will ein Agent A eine sichere Verbindung zu einem anderen Agenten B aufnehmen, so fragt er beim AKS nach. (Liegt Agent B in einer anderen Domäne, so fragt der AKS wiederum bei dem AKS der anderen Domäne nach.)
- Im symmetrischen Fall erzeugt der AKS einen neuen Schlüssel $K_{A,B}$, im asymmetrischen Fall wird der öffentliche Schlüssel $K_{B,P}$ des Agenten B verwendet.
- Dieser Schlüssel wird mit K_A verschlüsselt und Agent A zugeschickt.
- Agent A hat nun den Schlüssel und kann seine Nachricht mit dem neuen Schlüssel verschlüsseln.
- Wurde ein asymmetrischer Schlüssel verwendet, so kann Agent B die Nachricht mit seinem geheimen Schlüssel $K_{B,S}$ entschlüsseln. Bei einem symmetrischen Schlüssel muß Agent B beim AKS den Schlüssel $K_{A,B}$ anfordern. Erst dann kann er die Nachricht entschlüsseln.

Filtermöglichkeiten

Natürlich muß ein Agent nicht jeden Auftrag eines beliebigen anderen Agenten akzeptieren. Durch die Möglichkeit, die Authentizität des Absenders zu gewährleisten, kann ein Agent, in Abhängigkeit von der Managementanwendung, nur Aufträge von bestimmten Agenten annehmen (zum Beispiel nur von Agenten aus der eigenen Domäne, oder nur von Agenten einer bestimmten Klasse[6], oder von einigen ausgewiesenen Agenten).

3.3.4 Agent-Resource-Server

Wie oben schon beschrieben, sollte ein Agent nur Funktionalität ausführen, bei der gewährleistet ist, daß sie nicht von einem bösartigen Angreifer stammt.

Ein weiterer Grund, warum ein Agent nicht blind jeden Code ausführen sollte, der ihm delegiert wurde, ist auch, den Agenten vor ungetesteten oder fehlerhaften Code zu schützen.

Daher werden sogenannte Agent-Resource-Server (ARS) eingesetzt, die über gesicherte Funktionalität verfügen. Der Agent führt dann nur Code aus, der von diesen ARS stammt.

Zugang zu diesen Agent-Resource-Servern sollten natürlich nur Administratoren haben.

[6] siehe Kapitel 3.5.4.2 Definierte Klassennamen, Seite 35

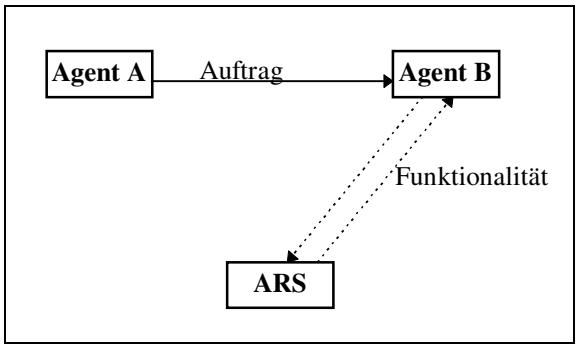


Abbildung 3-2 Delegierungsmechanismus

3.4 Organisationsform

Ab einer gewissen Anzahl an Netz- und Systemkomponenten und somit auch an flexiblen Agenten ist eine Organisation des Netzes, das heißt, eine Unterteilung in Teilnetze, sogenannten Domänen, sowohl aus administrativen, als auch aus organisatorischen und sicherheitsrelevanten Gesichtspunkten unumgänglich. Genauso müssen natürlich auch die flexiblen Agenten in Managementdomänen unterteilt werden. Dabei ist es nicht unbedingt erforderlich, daß sich Netzdomänen und Managementdomänen decken, obwohl dies natürlich Vorteile mit sich bringen kann.

Zu dem Problem der richtigen Organisationsform wurden schon viele Untersuchungen durchgeführt, vor allem aus dem Bereich der Wirtschaftswissenschaften. Tatsächlich läßt sich das Problem der richtigen Organisationsform für flexible Agenten zum Großteil auf das Problem der Organisation im wirtschaftswissenschaftlichem Sinne abbilden. Und von diesem ist bekannt, daß es keine allgemein gültige optimale Lösung gibt, sondern daß sie immer vom Umfeld und von den zu lösenden Aufgaben abhängt.

Daher werden im folgenden die organisatorischen Grundformen, in Bezug auf ein System flexibler Agenten, untersucht, ihre jeweiligen Vor- und Nachteile analysiert, und am Ende ein passendes Modell gewählt.

3.4.1 Vollstruktur

Eine vollstrukturierte Organisationsform zeichnet sich vor allem dadurch aus, daß alle Einheiten gleich sind. Es gibt keine klaren über- bzw. untergeordneten Einheiten. Jede Einheit kann mit jeder anderen interagieren.

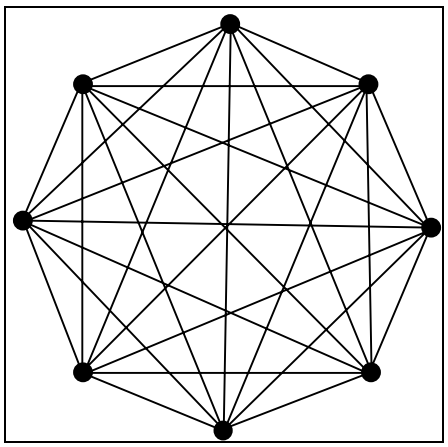


Abbildung 3-3 Vollstruktur

Dieser Ansatz ist der klassische Agentenansatz. Im klassischen Modell der kooperierenden, autonomen Agenten sind alle Agenten gleich und Entscheidungen werden von den Agenten durch gemeinsame Abstimmungsverfahren geregelt.

In unserem Falle würde das bedeuten, daß jeder Agent von jedem anderen Agenten Aufträge entgegennimmt, und daß jeder Agent jede Aufgaben übernehmen kann.

Vorteile der Vollstruktur:

- **Flexibilität**

  Da jeder Agent jede Aufgabe übernehmen kann, wird in dieser Organisationsform die Flexibilität der „flexiblen Agenten“ in keinster Weise eingeschränkt.

- **Ausfallsicherheit**

  Aufgrund der Tatsache, daß jeder Agent mit jedem kommunizieren kann und nicht auf bestimmte Kommunikationswege („Befehlskette“) festgelegt ist, besteht keine Gefahr eines unnötigen Ausfalls des Agentensystem bzw. eines Teils davon.

Nachteile der Vollstruktur:

- **schwierige Verwaltung**

  Der größte Nachteil an der Vollstruktur ist sicherlich, daß sie den Manager bei der Verwaltung der flexiblen Agenten nicht sonderlich unterstützt, da jede denkbare Konstellation möglich ist.

- **Gefahr von Konflikten**

  Dadurch, daß jeder Agent jede beliebige Aufgabe übernehmen kann, sind Zuständigkeiten über Managed Objects nicht klar geregelt. So könnte es also sehr leicht zu dem Fall kommen, daß zwei Agenten versuchen, die selbe Ressource zu bearbeiten.

3.4.2 Hierarchische Struktur

Die (vollständig) hierarchische Struktur stellt das genaue Gegenteil zur Vollstruktur dar. Hier kann jeder Agent nur mit seinem über- bzw. untergeordneten Agenten kommunizieren. Das bedeutet auch, daß jedes Managed Object von genau einem Agenten bearbeitet wird.

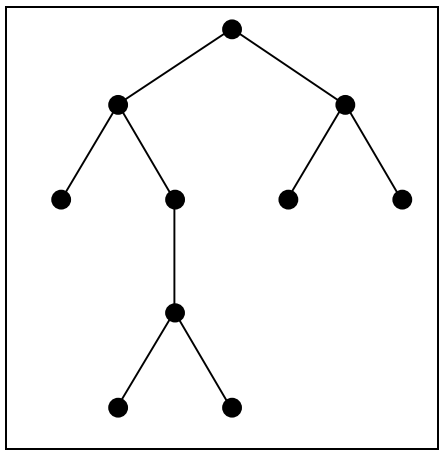


Abbildung 3-4 Hierarchische Struktur

Obwohl dieser Ansatz sehr restriktiv erscheint, stellt er doch den Normalfall im realen, unternehmerischen Alltag dar.

Vorteile der hierarchischen Struktur:

- **geringe Konfliktgefahr**

  Dadurch, daß jeder Agent Aufgaben nur von seinem direkten "Vorgesetzten" annimmt, und jeder Agent nur einen solchen hat, besteht nur eine geringe Gefahr von Konflikten innerhalb des Agenten.

  Da auch die Managed Objects nur von einem Agenten bearbeitet werden, sind auch sie relativ sicher vor Konflikten.

- **übersichtliche Struktur**

  Für den Manager ist die hierarchische Struktur natürlich viel übersichtlicher als die Vollstruktur. Hier ist klar ersichtlich, welcher Agent mit welchen Agenten kommuniziert und kooperiert, womit das Verwalten eines solchen Agentensystems vereinfacht wird.

Nachteile der hierarchischen Struktur:

- **weite Wege**

  Aufgrund der Tatsache, daß Nachrichten nur immer zum über- bzw. untergeordneten Agenten geschickt werden können, müssen Nachrichten, die an entfernte Agenten adressiert sind, viele Hierarchieebenen durchschreiten.

  Auch Aufgaben müssen von Hierarchieebene zu Hierarchieebene weiter delegiert werden, bis sie bei dem Agenten angelangt sind, der sie dann ausführen soll.

- **Überbelastung der höheren Agenten**

  Dadurch, daß jegliche Kommunikation anhand des Hierarchiebaums entlang geführt wird, laufen z. B. alle Nachrichten zu entfernten Teilbäumen immer über die Wurzel des Hierarchiebaums, was dort zu einem vermehrten Nachrichtenaufkommen und somit zu einer Überbelastung führt.

- **Zersplitterung des Baumes bei Ausfall eines Agenten**

  Die übergeordneten Agenten stellen single-point-of-failures dar, da, wenn einer von ihnen mal ausfällt, die Kommunikation in alle anderen Teilbäume dann nicht mehr möglich ist.

- **inflexibel**

  Es ist nicht so ohne weiteres möglich, die Agenten bei Bedarf umzustrukturieren, also z. B. einzelne Agenten zu entfernen oder einzufügen, da sie fest in einer Hierarchiestruktur eingebunden sind.

3.4.3 gemischter Ansatz

Natürlich existieren zu den beiden, gerade vorgestellten, extremen Ansätzen noch eine Vielzahl von Mischformen. Hier sollen nur einige mögliche Varianten vorgestellt werden:

- **Bildung von Teams, die auch untereinander kommunizieren**

  Eine Variante ist die Bildung von Teams, das heißt, daß Agenten auf der gleichen Ebene innerhalb eines Teilbaumes auch untereinander kommunizieren können, diese Teams aber trotzdem noch in eine Hierarchiestruktur eingebunden sind. Das hat den Vorteil, daß die Agenten übersichtlich strukturiert sind und die Wurzeln von Teilbäumen nicht so stark belastet werden. Außerdem zerfällt der Baum nicht in Teilbäume, wenn ein Agent der Gruppe mal ausfällt.

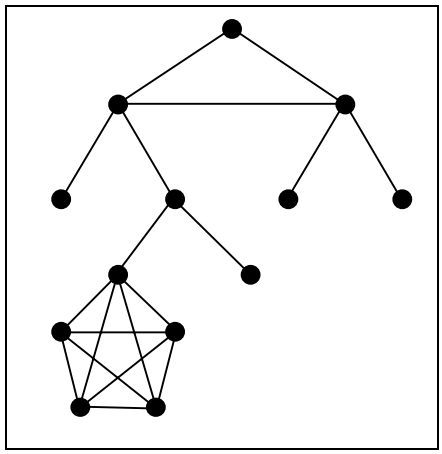


Abbildung 3-5 Teambildung

- **Auslassen von Hierarchiestufen**

  Eine weitere Möglichkeit die Nachteile der streng hierarchischen Struktur zu kompensieren, ist das Auslassen von Hierarchiestufen, das heißt, daß z. B. eine Aufgabe direkt an den Agenten delegiert werden kann, der sie dann auch ausführen soll. Diese Variante wäre deutlich resourcenschonender, birgt aber auch wieder die Gefahr von Konflikten in sich.

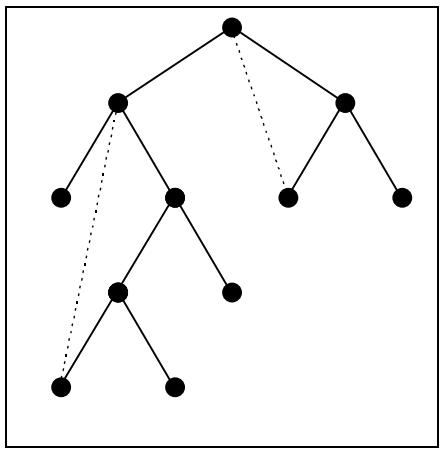


Abbildung 3-6 Auslassen von Hierarchiestufen

- **Gruppenübergreifende Kommunikation**

  Denselben Effekt hat das Zulassen von gruppenübergreifender Kommunikation. Das bedeutet, es wird den Agenten erlaubt, Nachrichten unter Umgehung der Befehlskette direkt an Agenten in anderen Teilbäumen zu schicken. Auch hier muß dafür gesorgt werden, daß keine Konflikte entstehen können.

- **Mehrere übergeordnete Agenten**

  Häufig wird es auch vorkommen, daß ein Agent mehrere übergeordnete Agenten hat, z. B. bei Gateways, die von beiden Seiten administriert werden müssen. Damit keine Konflikte entstehen, müssen die beiden übergeordneten Agenten auch kooperieren.

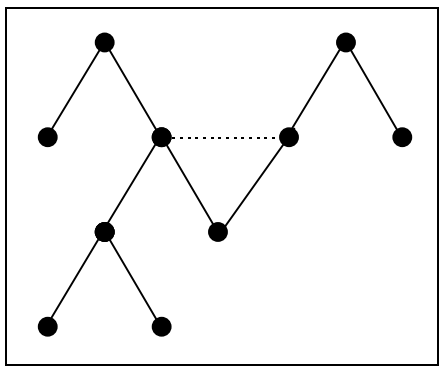


Abbildung 3-7 Mehrere übergeordnete Agenten

3.4.4 Organisationskriterien

Im vorherigen Abschnitt wurden verschiedene Möglichkeiten aufgezeigt, wie flexible Agenten strukturiert und organisiert werden können. Nun sollen mögliche Kriterien, nach denen die Agenten organisiert werden können, vorgestellt werden:

- **Netz-topologisch**

  Eine Unterteilung anhand netz-topologischer Gesichtspunkte bedeutet, daß beispielsweise alle Agenten innerhalb eines LAN-Segmentes zur selben Management-Domäne gehören. Diese Unterteilung ist am naheliegendsten, da hier bei hierarchischen Ansätzen die Belastung des Netzes am geringsten ist.

- **Aufgabenorientiert**

  Eine weitere Möglichkeit, die Agenten zu unterteilen, wäre anhand ihrer Aufgaben. So könnte man alle Agenten, die z. B. für Accounting zuständig sind, in einer Managementdomäne zusammenfassen, oder alle Agenten, die monitoren, etc. Dies würde die Kooperation unter den Agenten fördern, da zumeist nur Agenten, die mit derselben Aufgabe beschäftigt sind, miteinander kooperieren.

- **Managed Objects orientiert**

  Ein ähnliches Kriterium ist die Unterteilung nach Managed Objects, das heißt, man faßt zum Beispiel alle Agenten, die für Router zuständig sind zusammen, oder alle Agenten für einen ganz bestimmten Typ von Drucker, etc. Somit könnte man beispielsweise einen Update von Funktionalität ganz einfach durchführen.

- **unternehmerische Organisation**

  In vielen Fällen wäre eine Unterteilung, anhand der im Unternehmen vorhandenen Struktur, sinnvoll. So könnte man zum Beispiel alle flexiblen Agenten, die zur Entwicklungsabteilung gehören, zu einer Domäne zusammenfassen, oder alle Agenten, die an der Aufrechterhaltung eines bestimmten Dienstes/Produktes beteiligt sind.

Es gibt noch viele, weitere Kriterien (geographische, plattformspezifische, etc.), anhand derer man die Agenten unterteilen und organisieren kann.

3.4.5 Agent-Domain-Name-System

Wie oben gezeigt, ist meist nur ein gemischter Ansatz aus Vollstruktur und Hierarchie sinnvoll. Da im Ansatz der flexiblen Agenten die „Flexibilität“ einen besonders wichtigen Punkt darstellt, wird im auch folgenden Modell auf diesen Punkt besonderen Wert gelegt.

Das in Abbildung 3-8 dargestellte Modell ist an das Modell des Domain-Name-Systems des Internets angelehnt:

- Agenten können zu Managementdomänen (MD) zusammengefaßt werden. Dabei ist dem Manager freigestellt, nach welchen Kriterien er vorgehen will, das heißt, die Agenten können beliebig zusammengestellt werden.
- Jede Managementdomäne wird durch einen Domänennamen eindeutig identifiziert. Agenten einer Domäne werden durch die Kombination aus ihrem Namen und dem Domänennamen angesprochen.
- Für jede Managementdomäne existiert ein Agent-Name-Server (ANS), der die Namen nach Netzadressen auflöst.
- Jeder Agent meldet sich selbst automatisch beim ANS an.

- Jeder Agent kann mehreren Managementdomänen angehören.
- Eine Managementdomäne kann aus Subdomänen bestehen. Es existiert keine Ordnung zwischen Agenten, nur zwischen Domänen, das heißt, es gibt keinen Agenten, der allen anderen Agenten in der Domäne von vornherein übergeordnet ist. Natürlich kann eine (verteilte) Managementanwendung so etwas vorsehen und implementieren.

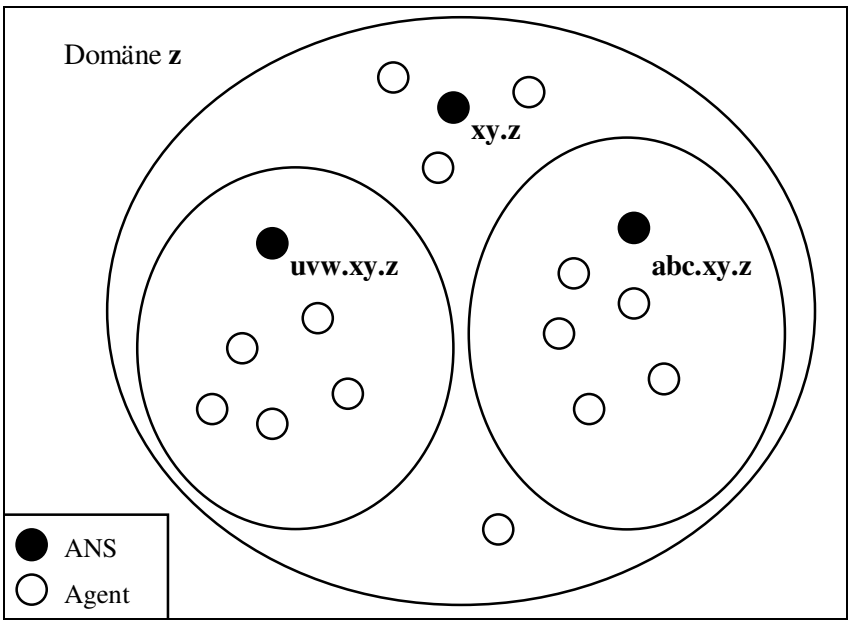


Abbildung 3-8 Agent-Name-System

3.5 Dienstlokalisierung

Damit Agenten miteinander kooperieren können, müssen sie wissen, von welchen Agenten sie welche Managementinformationen beziehen können bzw. welche Agenten welche Informationen benötigen.

Außerdem brauchen kooperierende Agenten Informationen darüber, welche Agenten welche Dienste anbieten, um diese in Anspruch nehmen zu können.

Wenn es um die Verteilung eines Auftrags geht, muß der Agent wissen, welchem Agent er diesen am besten erteilt, das heißt, der Agent muß in Erfahrung bringen können, welcher Agent für eine bestimmte Netzkomponente bzw. Managed Object zuständig ist.

Im folgenden sollen einige Ansätze, die in verteilten Systemen verwendet werden, untersucht werden, um am Ende einen Vorschlag für ein System aus flexiblen Agenten vorzustellen.

3.5.1 Statischer Ansatz

Im statischen Ansatz der Dienstlokalisierung steht bei jeder Aufgabe von vornherein fest, mit welchen Agenten der Agent bei Erfüllung dieser Aufgabe kooperiert. Dazu gibt es zwei Möglichkeiten: Dem Agenten werden bei der Aufgabenerteilung alle benötigten Kooperationspartner mitgeteilt, oder, jeder Agenten hat, unabhängig von der Aufgabe, das Wissen über alle anderen Agenten und „weiß“ somit, mit welchen Agenten er kooperieren muß.

Vorteile des statischen Ansatzes:

- **einfache Implementierung**

  Der Hauptvorteil im Vergleich mit anderen Ansätzen liegt in der einfachen Implementierbarkeit. Die Adressen der Kooperationspartner werden vom Manager festgelegt und mitgegeben, und somit muß im Agenten für das Auffinden von Kommunikationspartnern kein Mechanismus vorhanden sein.

- **geringer Ressourcenverbrauch**

  Dadurch, daß der Agent nicht nach Kooperationspartnern zu suchen braucht, muß er dafür auch nicht auf vorhandene Ressourcen (Bandbreite, Rechenkapazität, etc.) zugreifen.

Nachteile des statischen Ansatzes:

- **relativ geringe Ausfallsicherheit**

  Aufgrund der Tatsache, daß dem Agenten alle Kooperationspartner fest mitgeteilt werden, hat er nicht die Möglichkeit, sich einen neuen zu suchen, wenn der alte Kooperationspartner ausgefallen ist. Somit bleibt nur noch die Möglichkeit, dem Agenten von Anfang an mehrere, mögliche Kooperationspartner mitzuteilen, was dann aber - speziell in größeren System - zu einem hohen Speicherverbrauch auf Seiten des Agenten führt.

- **geringe Übersicht**

  Ab einer gewissen Größe ist es für den Manager nur noch schwer überschaubar, welcher Agent welche Aufgaben ausübt. Es besteht die Gefahr, daß ein gewisser „Wildwuchs“ entsteht. So könnte es leicht passieren, daß manche Dienste doppelt angeboten wurden, weil es dem Manager nicht mehr bewußt war, daß dieser Dienst schon angeboten wurde.

- **hoher Aufwand bei Umstrukturierung/Umkonfiguration**

  Der größte Nachteil der statischen Dienstlokalisierung ist sicherlich der hohe Aufwand bei Umkonfigurationen. Wenn z. B. ein Dienst auf einen anderen Agenten „umgezogen“ wird oder ein neuer Agent hinzukommt, so muß dieser bei allen kooperierenden Anwendungen ebenfalls mit geändert werden.

Als fast schon unmöglich könnte man dann den Versuch bezeichnen, das Agentensystem neu zu strukturieren (z. B. wenn neue Managementdomänen kreiert werden sollen), da hier die Konfiguration jeder Aufgabe oder jedes Dienstes per Hand geändert werden müßte.

3.5.2 Vermittlung

Bei allen vermittelnden Systemen (Broker, Trader, etc.) gibt es einen oder mehrere Vermittler, der die Anfragen über Dienste und Informationen beantwortet.

Das funktioniert im Allgemeinen so, daß die Agenten ihre Dienste beim Vermittler anmelden. Diese Information tauscht dann der Vermittler mit anderen Vermittlern aus.

Wenn nun ein Agent eine Anfrage nach einem Dienst an den Vermittler abgibt, prüft dieser, ob so ein Dienst angemeldet und in vielen Fällen, welcher Dienstanbieter der geeignetste ist und gibt die Adresse an den Agenten zurück.

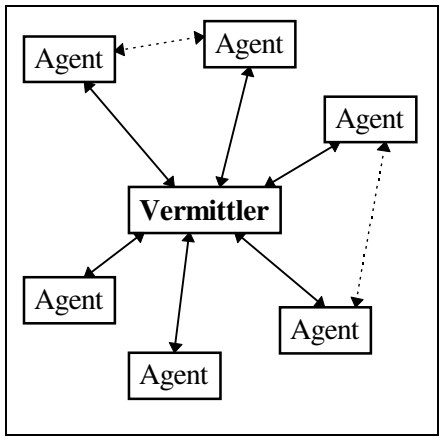


Abbildung 3-9 Vermittler

Bei Brokern sieht das so aus, daß der Broker nicht die Adresse des Dienstanbieters zurückgibt, sondern als Art „Dienst-Proxy“ dient. Das heißt, gegenüber dem Dienstanbieter gibt er sich als Agent aus, und gegenüber dem Agenten als Dienstanbieter. Der Agent stellt also alle Anfragen und Aufgaben an den Broker, der diese dann wiederum an den richtigen Agenten weitergibt.

Vorteile der Vermittlung:

- **Flexibilität bei Umkonfigurationen**

  Wenn das flexible Agentensystem umkonfiguriert werden muß, so bereitet das bei einem vermittelnden Ansatz weniger Probleme als beim statischen Ansatz. Soll zum Beispiel ein Dienst auf einen anderen Agenten umgezogen werden, so muß an die Agenten, die diesen Dienst nutzen, nichts geändert werden, da der neue dienstanbietende Agent sich beim Vermittler angemeldet hat, und so alle anderen Agenten automatisch zu ihm geführt werden.

- **Verwendung von Standards möglich**

  Ein weiterer Vorteil sind die auf diesem Gebiet schon vorhandenen Standards (zum Beispiel X.500, who++, CORBA[7]), so daß diese verwendet werden können und es keiner Neuentwicklung bedarf.

[7] siehe auch Kapitel 5.2 CORBA, S. 62

Nachteile der Vermittlung:

- **Hohe Netzlast/Speicherbedarf bei Vermittler**

  Wenn nun der Vermittler alle Anfragen beantworten können soll, bedeutet das, daß er einen hohen Bedarf an Ressourcen hat, sowohl was Speicherplatz als auch Rechenzeit und Bandbreite betrifft. Die Performanz des Gesamtsystem hängt zumindest zum Teil davon ab, wie schnell der Vermittler Anfragen beantworten und wie gut er den optimalen Kooperationspartner für eine Aufgabe ermitteln kann. Daher wird man versuchen, möglichst viele, gut verteilte Vermittler einzusetzen.

- **single-point-of-failure beim Vermittler**

  Wie oben schon beschrieben, hängt das ganze Agentensystem vom Funktionieren des Vermittlers ab. Fällt dieser aus, oder ist er aufgrund eines Netzausfalles nicht mehr erreichbar, so ist keine Kooperation bzw. Kommunikation mehr möglich. Besonders schwer fällt dies ins Gewicht, wenn der Vermittler als Broker eingesetzt wird, d. h. wenn jede Anfrage über den Vermittler geht, auch wenn vorher schon zwischen den Agenten kooperiert wurde. Da also mit dem Ausfall eines Vermittlers ein ganzes Agentensubsystem ausfällt, ist es auch aus diesem Grunde nötig, möglichst viele Vermittler einzusetzen.

- **Management der Vermittler nötig**

  Um, wie oben gezeigt, den Vorteil eines verteilten Systems gegenüber dem zentralen Manageransatz nicht durch den Einsatz eines zentralen Vermittlers zu verlieren, scheint der Einsatz möglichst vieler Vermittler sinnvoll. Das bedeutet aber auch, daß wiederum die Vermittler an sich verwaltet werden müssen. Das betrifft vor allem die Datenbasis, aufgrund deren die Vermittler ihre Entscheidung treffen. Hier droht die Gefahr, daß der Vermittler über Daten verfügt, die nicht mehr aktuell sind. Der sinnvollste Ansatz ist sicherlich die verteilte Datenbasis, das heißt, nicht jeder Vermittler verfügt über alle Informationen. Damit wäre eine ständige Aktualisierung aller Daten nicht mehr nötig, und es würde den Speicherverbrauch beim Vermittler mindern. Allerdings tritt hier wieder die Gefahr eines Verlust von Daten auf, wenn ein Agent ausfällt und seine Daten verloren gehen. Außerdem wird bei einer verteilten Datenbasis die Vermittlung an sich komplexer.

3.5.3 Broadcast

Der klassische Ansatz zur Dienstlokalisierung in Multi-Agentensystemen ist die Broadcast-Methode. Das bedeutet, daß ein Agent A, der eine Information braucht oder einen Dienst nutzen will, alle anderen Agenten fragt. Jeder Agent, der in der Lage ist, eine Antwort darauf zu geben oder diesen Dienst anzubieten, schickt eine Antwort an Agent A zurück. Agent A sammelt nun alle Antworten, bewertet sie und wählt die beste Antwort bzw. den am besten geeignetsten Dienstanbieter aus.

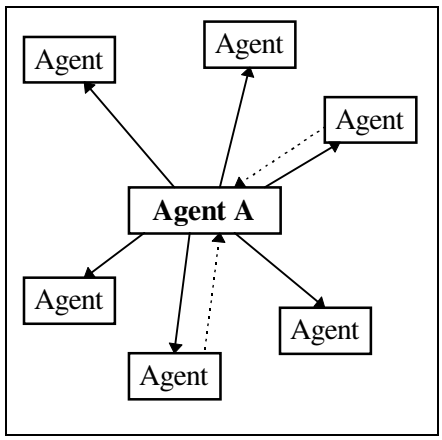


Abbildung 3-10 Broadcast

Vorteile des Broadcast-Ansatzes:

- **Niedrige Antwortzeiten**

  Ein Vorteil gegenüber vermittelnden Systemen sind die schnelleren Antwortzeiten, da jede Anfrage genauso wie jede Antwort immer direkt zu anderen Agenten geht, ohne einen Umweg über einen (vielleicht stark belasteten) Vermittler zu machen.

- **Hohe Ausfallsicherheit**

  Da dieser Ansatz weder von einem festen (statischen) Kooperationspartner abhängt, noch von irgendwelchen Vermittlern, die einen single-point-of-failure darstellen, wird dieses Verfahren am wenigsten von Störungen oder Netzausfällen beeinträchtigt. Wann immer ein Kommunikations- bzw. Kooperationspartner überhaupt erreichbar ist, so wird dieser mit Sicherheit auch gefunden.

Nachteile des Broadcast-Ansatzes:

- **Hohe Netzlast**

  Aufgrund der Tatsache, daß jede Anfrage via Broadcast an alle anderen Agenten verschickt wird, verursacht dieses Verfahren eine relativ hohe Netzlast. Vor allem bei Systemen mit einer hohen Anzahl an Agenten wird die verursachte Netzbelastung nicht mehr tragbar.

- **komplexer Bewertungsalgorithmus nötig**

  Dadurch, daß der Agent alle Angebote, die er auf seine Anfrage erhalten hat, auswerten muß, um dann eine auszuwählen, bedarf es dazu eines Bewertungsalgorithmus, der ihm diese Entscheidung ermöglicht. Soll dazu der Algorithmus für alle verteilten Managementanwendungen der gleiche sein, damit nicht für jede ein eigener Algorithmus entwickelt werde muß, muß dieser Bewertungsalgorithmus generischer Natur sein. Zudem müßte

der Agent, um eine sinnvolle Entscheidung treffen zu können, in seine Bewertung weitere Informationen einfließen lassen, wie z. B. die Netztopologie, etc.

3.5.4 Intelligent Naming

Die im folgenden vorgeschlagene Lösung, die als „Intelligent Naming“ bezeichnet wird, stellt eine Mischung aus den drei gezeigten Ansätzen (statisch, vermittelnd, broadcast) dar:

„Intelligent Naming“ (IN) bedeutet, daß alle Dienste und Informationen, die ein Agent anbietet, durch seine(n) Namen repräsentiert werden, die wiederum durch den Agent-Name-Server (ANS) aufgelöst werden.

Somit stellt der ANS zugleich auch einen Vermittler dar. Das An- und Abmelden von Diensten wird auf das An- und Abmelden von Namen abgebildet. Das hat den Vorteil, daß bei der Entwicklung von verteilten Managementanwendungen keine gesonderte Anfrage bei einem Vermittler nötig ist, sondern die Nachricht einfach an den benötigten Dienst adressiert und versendet wird.

Am einfachsten wird dies durch ein Beispiel erläutert.

3.5.4.1 Beispiel für Intelligent Naming:

Ein Agent „agent@domain“ ist für einen Webserver zuständig. Der Webserver an sich ist der „Enterprise Server“ von Netscape in der Version 2.0. Der Webserver bedient zwei virtuelle Server „www.internet.de“ und „www.blafasel.de“. IN bedeutet nun, daß der Agent z. B. unter den Namen webserver@domain, enterprise-server-2.0@domain, www.internet.de@domain, www.blafasel.de@domain, netscape-server@domain bekannt ist.

Somit braucht also ein anderer Agent, der Informationen über „www.blafasel.de“ benötigt, diese Anfrage nur an „www.blafasel.de@domain“ senden. Befindet sich der Agent in einer anderen Domäne als der Webserver und er kennt die korrekte Domäne nicht, so schickt er seine Anfrage an „www.blafasel.de@*“. Nun ist es Aufgabe des ANS-Systems, alle Adressen, die zu dieser Maske passen, zu ermitteln. (In diesem Beispiel sollte nur ein Name zurückkommen). Die ANS dienen in diesem Fall also als Vermittler. Soll nun ein Auftrag an alle Netscape Enterprise Server der Version 2.0 formuliert werden, z. B. das Logfile-Format zu ändern, so schickt man diesen Auftrag an „enterprise-server-2.0@domain“ und es werden damit alle erreicht.

Genauso braucht ein Agent, der jede Nacht die Auswertung aller Webserver-Logfiles in einer Domäne einsammelt und zusammenfaßt, seine Anfrage nur an „webserver@domain“ schicken und die Antworten einsammeln.

3.5.4.2 Definierte Klassennamen

Damit IN funktionieren kann, bedarf es natürlich definierter, eindeutiger Namen. Dies soll unter anderem durch die Klassennamen des Klassenbaums erreicht werden, der alle möglichen Arten von Managed Objects inklusive ihren Methoden systematisch unterteilt

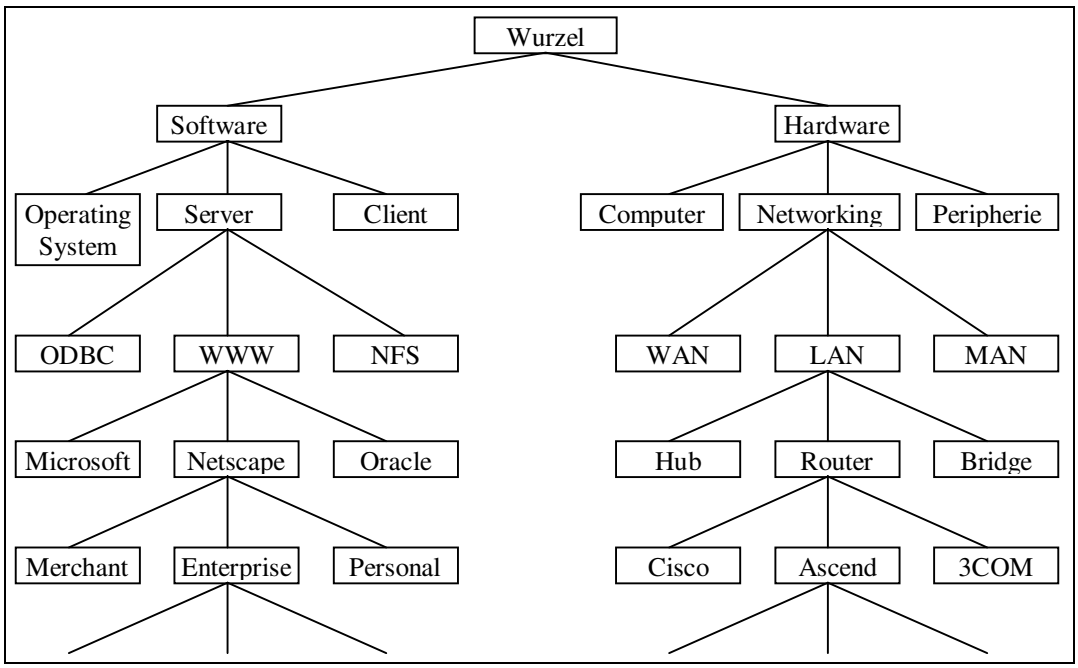


Abbildung 3-11 Beispiel eines Klassenbaums

Zu jeder dieser Klassen im Klassenbaum können Methoden definiert werden, die an die darunterliegenden Klassen vererbt werden. Außerdem kann zu jeder Klasse ein Interface angegeben werden, daß die darunterliegenden Klassen implementieren müssen.[8]

Will man nun alle Agenten einer Klasse innerhalb einer Domäne ansprechen, so adressiert man die Nachricht an den Pfad zur Klasse, also z. B. an Oracle.WWW.Server.Software@Domain, um eine Anfrage an alle Oracle-Webserver einer Domäne zu richten.

Soll z. B. ein Auftrag nur an einen Agenten der Klasse geschickt werden, dessen Name nicht näher bekannt ist, so muß zuerst die ganze Klasse der Domäne „befragt“, und dann der Auftrag an einen der Antwortenden geschickt werden.

Dieses Vorgehen entspricht dem der Broadcastmethode, mit dem Vorteil, daß es nur ein Multicast ist. Somit können die oben genannten Vorteile der Broadcastmethode genutzt werden, ohne übermäßige Netzlast zu erzeugen.

[8] mehr dazu in [Ho 97]

3.5.4.3 Instanzabhängige Namen

Weiterhin soll die Möglichkeit bestehen, jeder Klasse vorzuschreiben, bestimmte Namen beim ANS anzumelden, die von der jeweiligen Objektinstanz abhängig sind. So muß z. B. jeder Agent seine IP-Adresse und seinen FQDN[9] als Namen anmelden. Auch wäre es z. B. sinnvoll, wenn sich jeder Agent, der für einen Webserver zuständig ist, mit dem Namen der Domäne des Webservers anmeldet usw.

3.5.4.4 Freie Namen

Natürlich soll es für die Agenten auch möglich sein, weitere Namen ganz frei zu wählen. Soll z. B. ein Agent aus einer Gruppe von Agenten hervorgehoben werden, um ihn den anderen Agenten überzuordnen, so kann diesem Agent ein bestimmter Name (z. B. chief.www.server.software@domain) zugewiesen werden, obwohl er zur selben Klasse gehört.

3.6 Ereignismeldungen

Jede Managementarchitektur im Netz- und Systemmanagement besitzt die Möglichkeit zu asynchronen Ereignismeldungen. Genauso soll es auch im Flexiblen-Agentensystem diese Möglichkeit geben.

Als asynchrone Ereignismeldung wird eine Nachricht bezeichnet, die ein Agent aufgrund eines Ereignisses, das eingetreten ist, an alle verschickt, die dieses Ereignis betrifft.

Da es nun aber keinen zentralen Manager mehr gibt, an den die Agenten ihre Ereignismeldungen schicken können, muß ein neues Konzept gefunden werden.

3.6.1 Anforderungen

Im folgenden werden die Anforderungen, die an einen Mechanismus zum Versenden von Ereignismeldungen gestellt werden, dargestellt.

- **Subscribe-Mechanismus**

  Aufgrund der Tatsache, daß die Ereignismeldungen (auch Events genannt) nicht mehr an einen zentralen Manager geschickt werden, sondern für andere Agenten gedacht sind, muß einen Mechanismus existieren, mit dem sich ein Agent für einen bestimmten Event registrieren („subscribe“) bzw. austragen lassen kann („unsubscribe“).

[9] Full Qualified Domain Name des Domain Name System des Internets

- **Automatische Registrierung**

  Wenn sich ein Agent für eine Ereignismeldung registrieren lassen will, so sollte er dies nicht für jeden einzelnen Agenten gesondert machen müssen. Außerdem sollte er nicht ständig darauf achten müssen, ob neue Agenten hinzugekommen sind, um sich dann auch bei diesem wieder zu registrieren.

  Es sollte also reichen, wenn sich ein Agent einmal für ein Ereignis anmeldet, und dann ab diesen Zeitpunkt alle Ereignismeldungen dieser Art erhält.

- **Filtermöglichkeit**

  Vor allem in großen Flexiblen-Agentensystemen wird die Zahl an Ereignismeldungen sehr schnell recht hoch werden und ein Agent würde bald Gefahr laufen, von Ereignismeldungen überflutet zu werden. Deswegen sollte für den Agent die Möglichkeit bestehen, sich nur für Ereignisse registrieren zu lassen, die aus bestimmten Domänen bzw. von Agenten einer bestimmten Klasse oder gar nur von einigen ganz bestimmten Agenten stammen.

3.6.2 Agent Event Server

Auf Grund der oben genannten Anforderungen bedarf es eines Eventservers, der alle Events aufnimmt und an die Agenten, entsprechend ihrer Einstellungen, weiterleitet. Da ein zentraler Eventserver aufgrund des single-point-of-failure und der langen Transportwege nicht angebracht ist, muß ein verteiltes Eventserver-System aufgebaut werden.

Das Agent-Event-Server-System baut zum einen auf das Konzepte des Agent-Domain-Name-System[10] auf. Das heißt, in jeder Domäne gibt es einen Eventserver, an den alle Ereignismeldungen geschickt werden und bei dem sich ein Agent für Ereignismeldungen registrieren lassen kann. Jeder Eventserver registriert sich dann wiederum mit diesen Events bei dem nächst höheren Eventserver usw. So wird unter anderem erreicht, daß Ereignisse nicht doppelt in Domänen geschickt werden müssen und auch, daß Ereignisse den Agenten erreichen, die von einem, bis dahin unbekannten Agenten, aus einer weit entfernten Domäne stammen.

[10] siehe Kapitel 3.4.5, S. 28

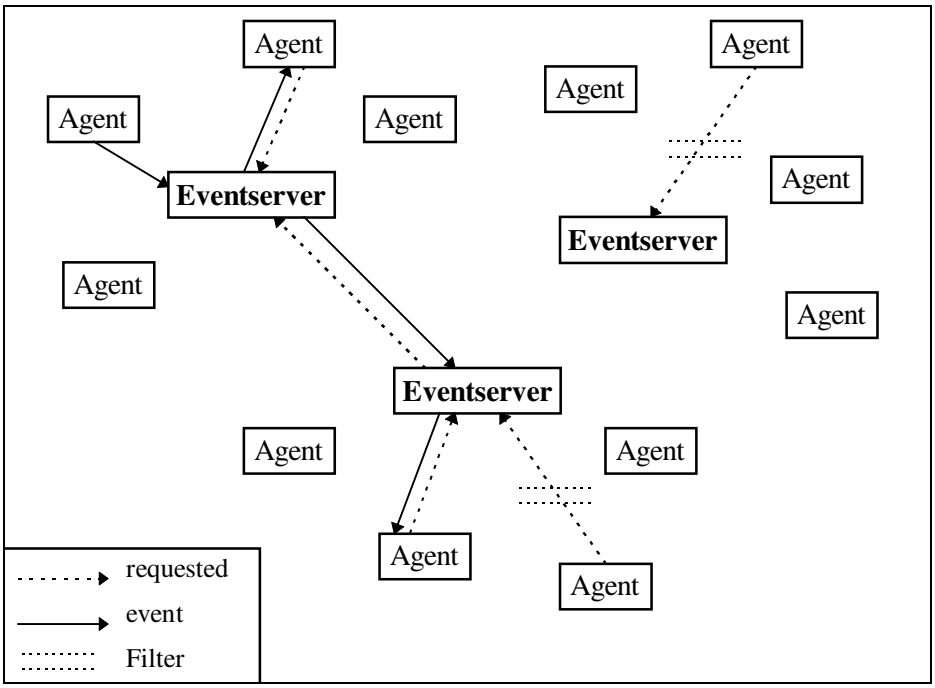


Abbildung 3-12 Agent-Event-Server System

Aufgrund der Tatsache, daß alle Ereignisnamen eindeutig sein sollen, baut das Eventserver-System zum anderen auf das Prinzip des Intelligent Naming[11] auf, das heißt, jedes Ereignis wird einer Klasse des Klassenbaums zugewiesen und ist somit eindeutig.

3.7 Zusammenfassung

In den vorhergehenden Abschnitten wurde ein Kommunikationsmodell entworfen, daß die Zusammenarbeit zwischen flexiblen, kooperierenden Agenten ermöglicht und regelt.

Es wurde ein Sicherheitskonzept entworfen, um die Agenten vor Mißbrauch zu schützen und ein Domänenkonzept entwickelt, daß es ermöglicht, Agenten in Gruppen zu unterteilen. Außerdem wurden Mechanismen zur Dienstlokalisierung und Ereignismeldung vorgestellt.

[11] siehe Kapitel 3.5.4, S. 34

4 Kommunikationsprotokoll

Als nächstes gilt es, ein, zu dem oben entwickelten Kommunikationsmodell, passendes Protokoll zu entwerfen. Zu dem Kommunikationsprotokoll gehören sowohl die genauen Abläufe der Nachrichten (Interaktionsprotokoll), als auch das Format der Nachrichten (Sprache), sowie das zugehörige Transportprotokoll, das die Nachrichten dann tatsächlich überträgt.

4.1 Interaktionsprotokoll

Das Interaktionsprotokoll legt fest, wie ein Agent mit anderen Agenten interagiert, das heißt, welche Nachricht er auf eine bestimmte Anfrage in welchem Fall verschickt, etc.

Solche Interaktionsprotokolle können von einem einfachem Frage-Antwort Schema über Spieltheorien bis hin zu komplexen Verhandlungsmechanismen reichen.

Wie anhand der Szenarien aus Kapitel 2.4.2[12] ersichtlich ist, hängt das jeweilige Interaktionsprotokoll, also der tatsächliche Ablauf der Nachrichten zwischen den Agenten, wer wem wann welche Nachricht sendet, von der jeweiligen Managementanwendung ab. Dennoch müssen einige Interaktionsprotokolle spezifiziert werden, um eine Grundfunktionalität der Agenten bereitzustellen.

4.1.1 Anforderungen

Im folgenden sollen die Anforderungen an die grundlegenden Interaktionsprotokolle, die ein Agent bzw. die die Server beherrschen können müssen, analysiert und definiert werden:

- **Agent ↔ Nameserver Protokoll**

  Es muß ein Interaktionsprotokoll entwickelt werden, das festlegt, wie Agenten ihre Namen beim ANS an- und abmelden, wie Anfragen an den ANS gestellt werden, und wie die Antworten darauf aussehen. Dazu gehören natürlich auch eventuell auftretende Fehlermeldungen.

[12] Seite 6

- **Nameserver ↔ Nameserver Protokoll**

  Für die Kommunikation zwischen den Nameservern aus verschiedenen Domänen muß natürlich auch ein Protokoll existieren. Dies kann, muß sich aber nicht vom Protokoll zwischen einem normalen Agenten und einem Nameserver unterscheiden.

- **Agent ↔ Eventserver Protokoll**

  Genauso muß ein Protokoll für die Kommunikation mit dem Eventserver spezifiziert werden. Dabei werden Mechanismen zum Registrieren und Abmelden, sowie zum Versenden und Empfangen von Ereignissen inklusive entsprechender Fehlermeldungen benötigt.

- **Eventserver ↔ Eventserver Protokoll**

  Ebenso wie beim Nameserver - Nameserver Protokoll kann sich das Protokoll zwischen den Eventservern von dem normalen Event-Protokoll unterscheiden.

- **Agent ↔ Key-Server Protokoll**

  Für die Anfrage nach dem Schlüssel eines Agenten, bzw. für den Auftrag, einen sitzungsabhängigen Schlüssel für zwei Agenten zu kreieren, muß ebenfalls ein Protokoll bestehen.

- **Key-Server ↔ Key-Server Protokoll**

  Das Key-Server - Key-Server Protokoll wird zur Abfrage von Schlüsseln aus fremden Domänen benötigt. Außerdem muß in diesem Protokoll ausgehandelt werden, welcher von beiden Key-Servern einen gemeinsamen Schlüssel kreiert, wenn die beiden Agenten in unterschiedlichen Domänen sind.

- **Agent ↔ Resource-Server Protokoll**

  Wie bei den anderen Protokollen auch, braucht es für das Laden einer Funktionalität ebenfalls ein entsprechendes Interaktionsprotokoll.

- **Resource-Server ↔ Resource-Server Protokoll**

  Dieses Protokoll dient hauptsächlich der Lokalisierung einer Funktionalität, sowie der Sicherstellung ihrer Authentizität.

- **Agent ↔ Agent Protokoll**

  Wie genau die Kommunikation zwischen den flexiblen Agenten abläuft, hängt von der jeweiligen Managementanwendung ab, aber es müssen Protokolle für die Grundfunktionen bereitgestellt werden

  Außerdem werden einige Standard-Fehlermeldungen benötigt, die jeder Agent versteht.

4.1.2 Nameserver-Protokoll

Für das Interaktionsprotokoll zwischen einem Agenten und einem Agent-Name-Server[13] (ANS) gibt es nicht viele sinnvolle Möglichkeiten. Für das An- und Abmelden eines Namens schickt der Agent einfach eine entsprechende Nachricht an den ANS. Um nach einer Adresse zu fragen, schickt er einen request. Um dem Agenten die Adresse dann mitzuteilen, schickt der ANS eine entsprechende response, bzw. eine Fehlermeldung, falls der Name nicht bekannt war.

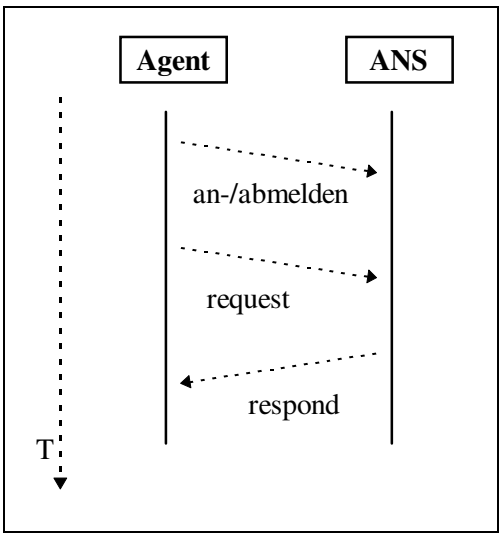


Abbildung 4-1 Agent ↔ ANS Protokoll

Interessanter wird die Frage nach dem Interaktionsprotokoll, wenn die angeforderte Adresse in einer anderen Domäne liegt. Für das Interaktionsprotokoll zwischen den Agent-Name-Servern gibt es verschiedene Möglichkeiten:

- **Weiterleiten der Anfrage des Agenten**

  Eine Möglichkeit besteht darin, daß der ANS die Anfrage des Agenten an den ANS weiterleitet, der am „nächsten“ an der Zieladresse liegt und ihm bekannt ist. „Weiterleiten“ bedeutet, daß der ANS, der die Anfrage beantworten kann, die Antwort direkt an den Agenten zurückschickt, der diese gestellt und nicht an den, von dem er die Anfrage erhalten hat

- **Weiterleiten der Anfrage des ANS**

  Eine weitere Variante des Weiterleitens wäre, genauso wie oben vorzugehen, nur daß die Antwort auf eine Anfrage nicht direkt an den Agenten geht, der die Anfrage gestellt hat, sondern an den ANS, den dieser Agent befragt hat. Dieser ANS wiederum schickt dann eine Antwort an den Agenten.

  Das hätte den Vorteil, daß der ANS die angeforderte Adresse cachen könnte, falls noch weitere Agenten diese benötigen.

[13] siehe Kapitel 3.4.5, S. 28

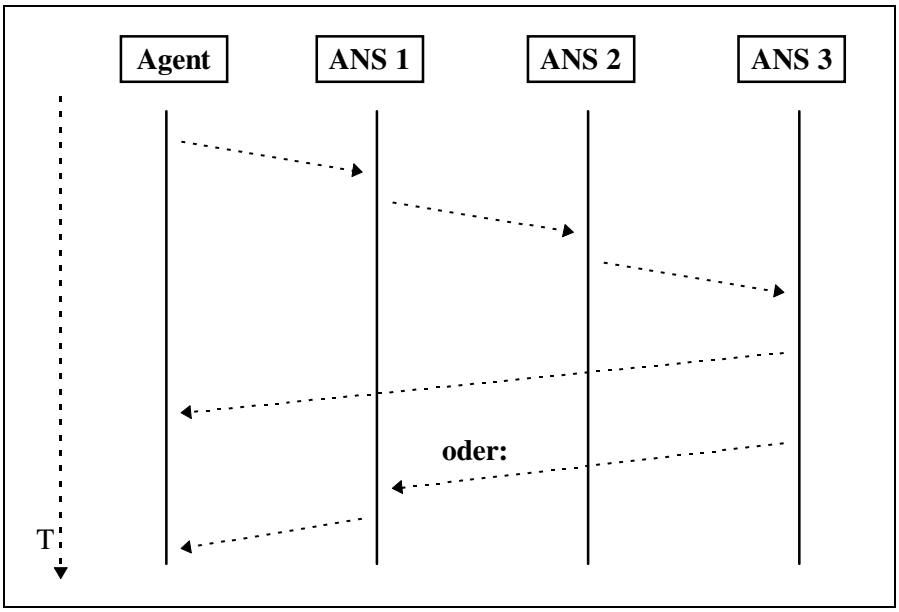


Abbildung 4-2 ANS ↔ ANS Protokoll mit Weiterleitung

- **ohne Weiterleiten**

  Zuletzt bleibt noch die Möglichkeit, daß die Anfrage überhaupt nicht weitergeleitet wird, sondern daß jeder ANS eine eigene Anfrage stellt und darauf einen eigene Antwort erhält, die er dann wiederum als Antwort an den weitergibt, der ihm einen request gesendet hat.

  In diesem Fall könnte jeder ANS die Adresse cachen.

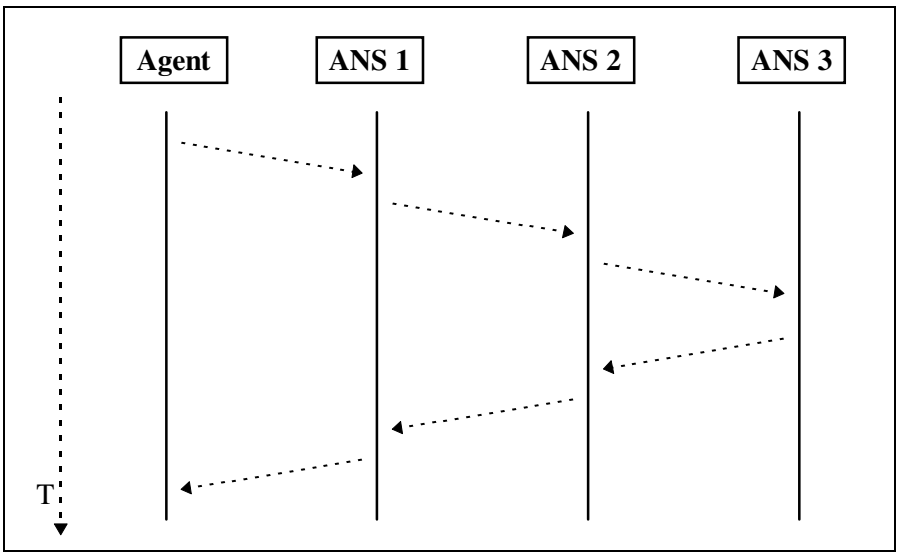


Abbildung 4-3 ANS ↔ ANS Protokoll ohne Weiterleitung

Welches Interaktionsprotokoll nun verwendet werden soll, hängt davon ab, ob Adressen gecached werden sollen oder nicht. Da Agenten ihre Namen beliebig oft ändern können, und hinter einem Namen auch mehrere Agenten stecken können (Gruppenkommunikation!), müßte beim Cachen dafür gesorgt werden, daß

die Einträge immer aktuell sind, das heißt, der ANS müßte bei jeder Anfrage erst die Ziel-ANS befragen, ob die Adresse noch korrekt ist.

Da aber die Namenseinträge in der Regel nicht sonderlich groß sind, wird somit beim Cachen nicht viel gewonnen. (Bei einfachen Agentennamen dürfte die Länge einer Adresse ungefähr der Länge einer Versionsnummer oder eines Zeitstempels entsprechen).

Daher kann auf das Cachen verzichtet werden, zumal der Großteil der Kommunikation ohnehin in der selben Domäne stattfinden wird. Somit ist ein Weiterleiten der Anfrage mit Antwort direkt an den Agenten das effizienteste Verfahren.

4.1.3 Eventserver Protokoll

Das Interaktionsprotokoll für den Event-Mechanismus[14] ist äußerst einfach, da es genau drei Aktionen gibt, zwischen denen aber keine Relation besteht. Sowohl zum subscriben eines Ereignisses, als auch zum unsubscriben wird nur eine einfache Nachricht des Agenten an den Agent-Event-Server (AES) benötigt.

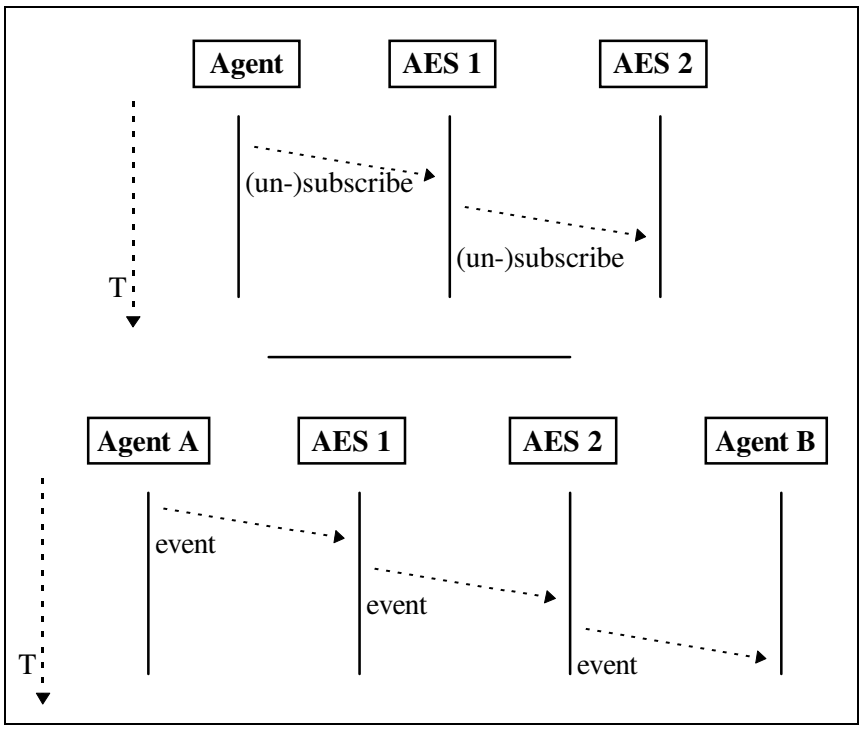


Abbildung 4-4 AES Protokoll

Das gleiche gilt für die Ereignismitteilung. Sie besteht aus einer einfachen Nachricht des Eventservers an den Agenten.

[14] siehe Kapitel 3.6.2, S. 37

Aufgrund der Funktionsweise des Agent-Event-Servers wird für das AES ↔ AES Protokoll kein eigenes Interaktionsprotokoll benötigt, da hier die gleichen Mechanismen verwendet werden wie beim Agent ↔ AES Protokoll

4.1.4 Key-Server Protokoll

Im Prinzip gibt es im Agent-Key-Server[15] (AKS) Protokoll genau eine Interaktion, die aus der Anfrage nach einem Schlüssel und der Antwort auf diese Anfrage besteht.

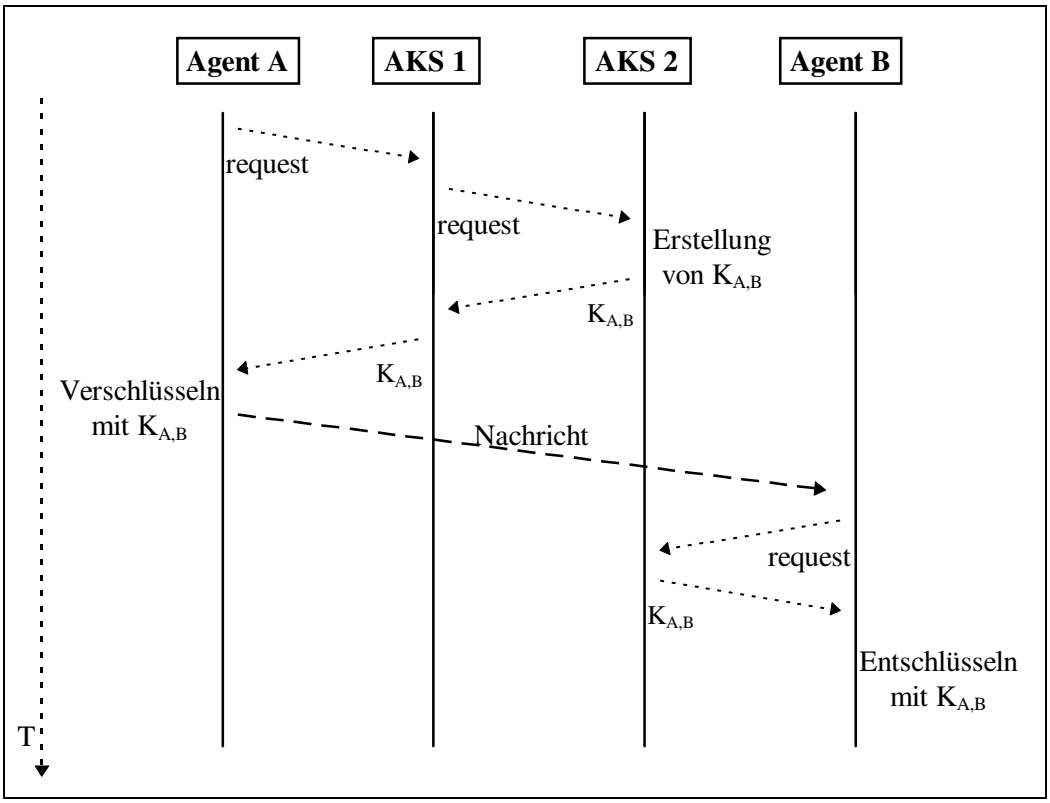


Abbildung 4-5 Agent-Key-Server Protokoll (symmetrisch)

Beim Interaktionsprotokoll zwischen den Key-Servern stellt sich hier, genauso wie beim Agent-Name-Server Protokoll[16], die Frage, ob die Anfrage nur weitergeleitet werden und die Antwort direkt an den Agenten gehen soll, oder ob der Key-Server selbst noch einmal beim Ziel-AKS anfragt und den Schlüssel von ihm erhält und dann diesen an den Agenten schickt.

Im Gegensatz zum ANS Interaktionsprotokoll empfiehlt sich hier ein Weiterleiten der Anfrage jedoch nicht, da ein Agent fremde Schlüssel nur von dem Agent-Key-Server seiner eigenen Domäne annehmen sollte. Da die Schlüssel, die die

[15] siehe Kapitel 3.3.3, S. 21

[16] siehe Kapitel 4.1.2, S. 41

AKS verwenden, um sich gegenseitig Nachrichten zu senden, besonders sicher sein sollten (große Schlüssel, häufiger Wechsel, etc.), ist der Weg über die AKS sicherer als ein Direktversand an den Agenten.

Außerdem ist hier bei asymmetrischen Schlüsseln ein Cachen der Schlüssel durchaus sinnvoll, da sich der öffentliche Schlüssel eines Agenten nach seiner Erstellung im Normalfall nicht ändert.

Im symmetrischen Fall ergibt sich somit die, in Abbildung 4-5 gezeigte Kommunikation. Für den asymmetrischen Fall ergibt sich ein ähnliches Bild, nur daß die öffentlichen Schlüssel nicht erstellt werden müssen, und daß jede Anfrage nach einem Schlüssel immer bis zum Ziel-AKS gehen würde.

Falls dem Ziel-AKS der Agent B nicht bekannt ist, erfolgt eine Fehlermeldung.

4.1.5 Resource-Server Protokoll

Auch beim Agent-Resource-Server[17] (ARS) Protokoll besteht die einzige Interaktion zwischen dem Agenten und dem ARS in der Anfrage nach der Funktionalität und der entsprechenden Lieferung der Funktionalität.

Und auch hier stellt sich in dem Fall, daß die Ressource in einer anderen Domäne liegt, die Frage, ob der request weitergeleitet und die Funktionalität direkt an den Agenten geliefert, oder ob die Ressource von ARS zu ARS durch die Domänen durchgereicht werden soll.

Im Gegensatz zum ANS Interaktionsprotokoll ist hier ein Cachen der Funktionalität durchaus sinnvoll, da diese doch Größenausmaße annehmen kann, die ein Cachen notwendig werden lassen. Außerdem muß bei der Funktionalität nicht auf Aktualität geachtet werden, da bei jeder Anfrage ohnehin immer eine Versionsnummer mit angegeben werden muß, denn es werden schließlich immer mehrere Versionen einer Funktionalität in einem Multi-Agentensystem existieren, deren Abwärtskompatibilität nicht immer unbedingt gewährleistet ist.

Ob nun die Funktionalität von ARS zu ARS gereicht wird und bei jedem ARS gecached wird, oder ob die Ressource direkt zu dem ARS gesendet wird, der diese angefordert hat, hängt von vielen Faktoren ab. Zu diesen zählen unter anderem globale Faktoren, wie die Größe des Multi-Agentensystems (Anzahl der ARS) oder auch die Anzahl an verschiedenen Funktionalitäten (Speicherbedarf eines ARS) oder die zur Verfügung stehenden Bandbreiten. Außerdem gibt es ähnliche lokale Faktoren für jeden ARS (vorhandener Speicherplatz/Bandbreite, Anzahl an Anfragen nach einer Ressource, etc.), anhand derer er, falls er die Anfrage nicht befriedigen kann, für sich selbst entscheiden kann, ob er die Ressource zu sich gesendet haben will, oder ob er die Anfrage nur weiterleitet.

[17] siehe Kapitel 3.3.4, S. 22

Die Sicherstellung der Authentizität einer Funktionalität sollte nicht vom ARS übernommen werden, da sonst ein erfolgreicher Angriff auf einen ARS zur Folge hätte, das bösartiger Code eingeschleust werden kann. Vielmehr sollte diese Aufgabe der Agent übernehmen, in dem er sich den fingerprint[18] direkt (und natürlich verschlüsselt) von einem Master-ARS holt, der nur die fingerprints enthält und keinen Code. Somit müßte es einem Angreifer gelingen, sowohl in einen ARS, als auch in den Master-ARS erfolgreich einzubrechen. Da die Authentizität der Funktionalitäten besonders wichtig ist, könnte man sie z. B. dadurch erhöhen, in dem man mehrere Master-ARS auf getrennten Systemen einsetzt. In diesem Fall holt der Agent den fingerprint von allen Master-ARS und vergleicht diese. Stimmen sie überein, gilt die Funktionalität als authentifiziert.

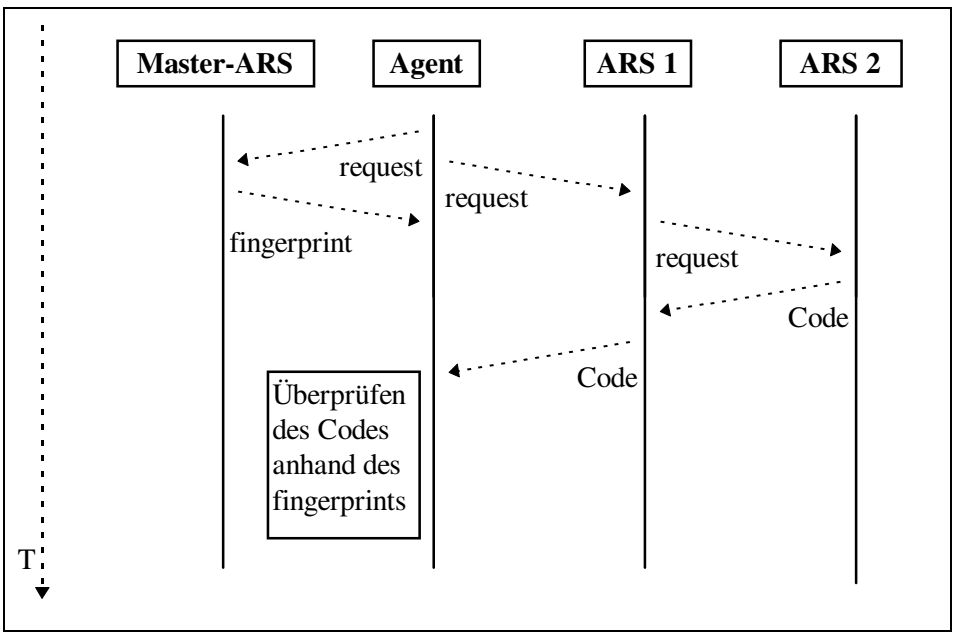


Abbildung 4-6 Agent-Resource-Server Protokoll

Existiert der angeforderte Code oder der angeforderte fingerprint nicht, so schickt der ARS eine Fehlermeldung an den Agenten zurück. Außerdem erzeugt er einen entsprechenden Event, so daß auch der Manager benachrichtigt wird.

[18] siehe Kapitel 3.3.2 Verschlüsselungstechniken, S. 18

4.1.6 Agent - Agent Protokoll

Abgesehen von speziellen Interaktionsprotokollen und dem Eventmechanismus, lassen sich die Interaktionen zwischen den Agenten auf zwei Aktionen abbilden:

- **Aufträge** (eine Anfrage nach Informationen ist auch nur ein Auftrag, Informationen zu senden), und
- **Informationsmitteilung** (Mitteilung von Ergebnissen, einfache Nachrichten, etc.)

Zwischen diesen beiden Aktionen liegt aber kein zwingender, direkter Zusammenhang, da nicht jeder Auftrag ein Ergebnis produziert, das zurückgeschickt werden könnte, und nicht jede Information aufgrund eines Auftrag versendet wird.

Für die weitere, grundlegende Kommunikation zwischen den Agenten (und somit auch zwischen Agenten und Servern) müssen noch einige Standard-Fehlermeldungen bereitgestellt werden:

- **keine Berechtigung**

  Diese Fehlermeldung verschickt ein Agent, wenn er einen Auftrag von einem Agenten erhalten hat, von dem er keine Aufträge annimmt.

- **falscher Schlüssel**

  Diese Nachricht wird verschickt, wenn der Agent die empfangene Nachricht nicht entschlüsseln oder den dazu benötigten Schlüssel vom AKS nicht erhalten konnte.

- **Auftrag kann nicht ausgeführt werden**

  Diese Fehlermeldung wird verwendet, wenn der Agent aus irgendeinem Grund den erhaltenen Auftrag nicht ausführen kann. Sei es, weil er nicht dafür zuständig ist, oder weil er die entsprechende Funktionalität vom ARS nicht erhalten konnte, oder weil er sich momentan in einem angehaltenen Zustand befindet.

- **unkorrekte Nachricht**

  Ein Agent gibt diese Fehlermeldung zurück, wenn die erhaltene Nachricht nicht korrekt war und er sich nicht parsen konnte. Zum Beispiel, wenn die Nachricht nicht vollständig ankam, oder wichtige Schlüsselworte fehlten, etc.

4.2 Agentensprache

Die Agentensprache spezifiziert sowohl die Form und Syntax der Nachrichten, die zwischen den Agenten ausgetauscht werden, als auch ihre Bedeutung (Semantik).

4.2.1 Anforderungen

Aus dem bisher entwickelten Kommunikationsmodell ergeben sich an die Agentensprache folgende Anforderungen:

- **Form**

  Die Sprache sollte von einfacher syntaktischer Form sein, die leicht zu lesen, leicht zu parsen und leicht zu generieren ist. Da die Nachricht bei der Übertragung in einen Datenstrom übersetzt werden muß, sollte sie in einer linearen Form darstellbar sein. Außerdem sollte sie in ihrer Form erweiterbar sein und bezüglich der Länge keine Begrenzungen aufweisen.

- **Inhalt**

  Die gewählte Sprache muß flexibel genug gewählt werden, um alle möglichen Arten von Nachrichten und Informationen in ihr formulieren zu können. Daher ist ein Schichtenmodell nötig, so daß die eigentliche Information als Inhalt in einen „Umschlag“ oder auch Header genannt eingepackt werden, der weitere Informationen über den Inhalt enthält, wie z. B. Sender, Empfänger, Subject, Message-type, Content-type, etc.

  Neben dem benötigten Grundwortschatz für ANS[19], AKS[20], ARS[21] und AES[22]-Anfragen sollte die Sprache auch einen Grundwortschatz anbieten, mit dem die meisten Nachrichten zwischen Agenten formuliert werden können, wie zum Beispiel einfache Informationsanfragen, Erteilung von Aufträgen etc.

  Dieser Grundwortschatz sollte ausreichen, möglichst viele Informationen, bzw. möglichst viele Elemente der zu übertragenden Nachricht schon in der Agentensprache formulieren zu können, da ansonsten die Agenten eine weitere, content-abhängige Sprache kennen müssen.

[19] Agent-Name-Server (siehe auch Kapitel 3.4.5, S. 28)

[20] Agent-Key-Server (siehe auch Kapitel 3.3.3, S. 21)

[21] Agent-Resource-Server (siehe auch Kapitel 3.3.4, S. 22)

[22] Agent-Event-Server (siehe auch Kapitel 3.6.2, S.37)

- **Semantik**

  Die Semantik der vorhandenen Sprachelemente sollte klar definiert und unmißverständlich sein, damit gewährleistet wird, daß ein Agent eine Nachricht in der gleichen Art und Weise interpretiert, wie ein Agent einer anderen Managementanwendungen.

- **Unabhängigkeit**

  Die gewählte Sprache sollte von der Umgebung, in der der Agent läuft unabhängig sein.

  Das betrifft zum einen die verwendete Programmiersprache. Es sollte leicht möglich sein, Schnittstellen zu allen möglichen Programmiersprachen bereitzustellen. Aufgrund der Tatsache, daß die Agenten in einer heterogenen Umgebung eingesetzt werden, sollte diese zudem unabhängig von der Plattform sein, auf der sich die Agenten befinden.

  Schließlich sollte die Agentensprache noch das zu Grunde liegende Transportprotokoll vor dem Agenten verstecken und dem Entwickler eine einheitliche Schnittstelle anbieten.

4.2.2 KQML

Die „Knowledge Query and Manipulation Language“ wurde speziell als Kommunikationssprache für Agenten entwickelt und stellt quasi einen Standard in der Agententechnologie dar.

Eine genauere Beschreibung von KQML erfolgt in Kapitel 5.1[23] dieser Arbeit. Im folgenden Abschnitt soll nur dargestellt werden, warum KQML die oben angegebenen Anforderungen erfüllt:

- **Form**

  KQML-Nachrichten bestehen aus einfachen, linearen ASCII-Zeichenketten und sind daher leicht zu übertragen. Durch die Lisp-Syntax mit dem Key/Value-Format ist die Nachricht außerdem leicht zu parsen, und kann trotzdem einfach in eine andere, zum Beispiel objekt-orientierte Form umgewandelt werden.

- **Inhalt**

  KQML bietet einen Satz von sogenannten performatives an, die verwendet werden können, um alle möglichen Arten von Nachrichten damit zu beschreiben. Dabei wurde versucht einen Mittelweg zu finden, um die Anzahl der performatives möglichst gering zu halten, aber dennoch möglichst viele

[23] Seite 55

Informationen aus dem Content-Layer in den Message-Layer übernehmen zu können.

Zudem definiert KQML die wichtigsten Parameter einer Nachricht, wie Absender, Empfänger etc.

- **Semantik**

  In der momentanen Version von KQML wird die Semantik der Nachrichten nur nebenbei behandelt. Dennoch existieren einige Arbeiten[24] zu diesem Thema, so daß einer Aktualisierung von KQML in dieser Hinsicht nichts mehr im Wege steht.

- **Unabhängigkeit**

  KQML-Agenten wurden schon in einer Vielzahl von Umgebungen mit den unterschiedlichsten Programmiersprachen entwickelt. Obwohl die Syntax an Lisp angelehnt ist, lassen sich KQML-Nachrichten durch ihre einfache Form auch in anderen Sprachen leicht bearbeiten.

  Zudem macht KQML keine Angaben über das zu verwendende Transportprotokoll und es existieren daher Implementierungen, die auf verschiedenen Protokollen aufbauen (HTTP, CORBA, SMTP, TCP, ...)

4.3 Transportprotokoll

Das Transportprotokoll stellt den tatsächlichen Transportmechanismus sowohl von Nachrichten, als auch von der zu delegierenden Funktionalität zur Verfügung. Hierzu könnten verschiedene, vorhandene Transportprotokolle verwendet werden: SMTP, HTTP, TCP, SNMP, etc.

4.3.1 Anforderungen

Aus dem in Kapitel 3 vorgestellten Kommunikationsmodell ergeben sich an das Transportprotokoll folgende Anforderungen:

- **Multicast**

  Aufgrund der Tatsache, daß laut Kommunikationsmodell die Möglichkeit besteht, auch Nachrichten an eine Gruppe von Agenten zu versenden, muß das dem Kommunikationsprotokoll zugrunde liegende Transportprotokoll, dies durch einen Multicast-Mechanismus ermöglichen.

[24] siehe [La 96]

- **Synchron/Asynchron**

  Das Transportprotokoll sollte sowohl synchrone, als auch asynchrone Kommunikation zulassen.

- **Plattform- und Content-Unabhängigkeit**

  Dadurch, daß die Agenten auf den unterschiedlichsten Plattformen laufen, sollte das Protokoll sowohl von dem Betriebssystem, als auch von der Hardware unabhängig sein.

  Zudem sollte das Protokoll beliebige Daten, unabhängig von Form, Inhalt und Größe, transportieren können.

4.3.2 Existierende Transportprotokolle

Das Folgende beschreibt einige ausgewählte, bereits existierende Transportprotokolle, die kurz vorgestellt und untersucht werden.

4.3.2.1 HyperText Transfer Protocol (HTTP)

Das Hypertext Transfer Protocol (HTTP) wurde entwickelt, um Informationen (hauptsächlich Texte und Bilder) von fremden Systemen abzuholen. Mit Hilfe des MIME-Konzepts arbeitet das HTTP-Protokoll content-unabhängig, das bedeutet, es können beliebige Daten damit transportiert werden, welche in einem Header beschrieben werden.

Um normale Nachrichten zu verschicken, eignet sich das HTTP-Protokoll im Grunde eigentlich nicht, da es auf einem request/respond-Mechanismus aufbaut und keine asynchrone Kommunikation zuläßt. Das heißt, auf jede Nachricht würde automatisch eine Antwort folgen, was bei einfachen Mitteilungen an eine große Gruppe von Agenten oder bei Ereignismitteilungen nicht erwünscht wäre.

Außerdem ist im HTTP-Protokoll keine Multicast-Kommunikation vorgesehen.

Jedoch zum Transport von großen Datenmengen, also zum Beispiel Funktionalität, Ergebnisse, Logfiles etc. eignet es sich hervorragend, da solche Daten normalerweise nur an einen Empfänger verschickt werden.

4.3.2.2 Simple Mail Transfer Protocol (SMTP)

Das Simple Mail Transfer Protokoll (SMTP) wurde für den Versand von Nachrichten über das Internet entwickelt.

Obwohl es ursprünglich nur für den Versand von 7-bit codierten Zeichenketten gedacht war, verstehen alle gängigen Implementierungen auch 8-bit Codierungen. Zudem kann mit Hilfe des MIME-Konzeptes beliebiger Inhalt transportiert werden.

Für den Einsatz in Multi-Agentensystemen scheint das SMTP-Protokoll daher besonders interessant, da es eines der wenigen Transportprotokolle ist, das mehrere Empfänger zuläßt und somit voll multicast-fähig ist.

Außerdem läßt es sich, durch den Mail-Header, in das von der Agentensprache verlangte Schichtenmodell leicht eingliedern, so daß die in der Agentensprache geforderten Parameter, wie Sender, Empfänger etc., in den Mail-Header übernommen werden können.

4.3.2.3 Simple Network Management Protocol (SNMP)

Eine Untersuchung zur Eignung von SNMP als Transportprotokoll befindet sich auf Seite 53 in Kapitel 4.4.3 „SNMP als Transportprotokoll“

4.3.2.4 Common Object Broker Architekture (CORBA)

Ebenso wird auf Seite 64 in Kapitel 5.2.2.1 „CORBA zum Transport von Nachrichten“ geprüft, inwiefern sich CORBA als Transportprotokoll eignet.

4.4 Internet Management

Im folgendem soll untersucht werden, inwiefern das im Internet Management verwendete Managementprotokoll SNMP den Anforderungen genügt, bzw. ob es verwendet werden kann, um das Kommunikationsmodell für flexible, kooperierende Agenten zu realisieren.

4.4.1 Simple Network Management Protocol (SNMP)

Das Simple Network Management Protocol gehört zu den am weitest verbreitetsten Managementprotokollen. Die Internet Management Architektur sieht zwei Rollen vor, den Manager und den Agenten, in der der Manager durch einfache requests Werte aus der MIB des Agenten abfragen bzw. neu besetzen kann.

Die meisten Fortschritte im Rahmen des Internet Management, um die Fähigkeiten der SNMP-Agenten zu erweitern, beziehen sich auf die Definierung neuer MIBs. Daneben gibt es aber auch vereinzelte Bestrebungen, sowohl die Agenten leichter erweiterbar zu machen (agentx, script-mib, etc.), als auch ein verteiltes Management zu ermöglichen (MLM, RMON, etc.).[25]

[25] siehe [Ho 97]

4.4.2 SNMP als Agentensprache

SNMP bietet als Sprache nur getRequest, setRequest, getNextRequest, getRespond und trap.

Für einen flexiblen, kooperierenden Agenten sind diese jedoch ungeeignet, da sie nur zum Lesen und Setzen von Variablen bzw. für Ereignismitteilungen gedacht sind, während ein Agent z. B. auch Aufträge erteilen können muß.

Natürlich kann SNMP dazu verwendet werden, um die fehlenden Wörter zu umschreiben. So existiert zum Beispiel eine Lösung von Jürgen Schönwalder von der University of Twente [Schö 96a], um Prozeduren auf entfernten Agenten zu starten, indem der Auftrag in eine Tabelle geschrieben und durch das Setzen einer weiteren Variable gestartet wird.

Doch auch wenn alle fehlende Wörter mit derartigen Methoden umschrieben werden könnten, sind diese Verfahren dennoch unkomfortabel und wenig effizient und ändern nichts an der Tatsache, daß der eigentliche Grundwortschatz von SNMP für eine Agentensprache ungeeignet ist.

Eine weitere Möglichkeit wäre, den trap-Mechanismus nicht nur zum Versendung von Ereignismeldungen zu verwenden, sondern auch, um mit seiner Hilfe zu kommunizieren. Dazu müßten eine Reihe von enterpriseSpezific traps definiert werden, deren Bedeutung den Agenten bekannt ist.

4.4.3 SNMP als Transportprotokoll

Ein anderer möglicher Einsatz von SNMP wäre, SNMP als Transportprotokoll zu verwenden, d. h. Nachrichten, Funktionalität, Ergebnisse etc. an Agenten mit SNMP zu übertragen.

Mit SNMP können Variablen in einer MIB auf einem entfernten Rechner gesetzt werden, das heißt, es können Informationen von einem Rechner zu einem anderen Rechner geschickt werden. Somit kann SNMP als Transportprotokoll für alle möglichen Arten von Daten verwendet werden. Man könnte z. B. auch Mails via SNMP verschicken, indem man eine passende Tabelle besetzt. Fraglich ist nur, ob dies sinnvoll ist.

Genauso könnten einfache kurze Nachrichten übermittelt werden, indem sie in eine Pseudo-Variable geschrieben werden, die eigentlich nicht existiert und auch nicht ausgelesen werden kann.

Eine weitere Variante, Nachrichten mit SNMP zu verschicken, ist mit Hilfe des trap-Mechanismus, der es Agenten erlaubt, asynchrone Ereignismeldungen zu verschicken.

Für den Transport von größeren Datenmengen, wie zum Beispiel Funktionalität, Ergebnisse oder Logfiles eignet sich SNMP nicht sonderlich, da es ein verbindungsloses Protokoll ist und somit jegliche Flußkontrolle fehlt.

4.4.4 Fazit

Auch wenn SNMP von der Syntax her, für den Einsatz in einem flexiblen, kooperierenden Multi-Agentensystem verwendet werden kann, die Semantik von SNMP kann nicht übernommen werden.

Aufgrund der Tatsache, daß die PDUs nicht zu dem verwendet werden, zu dem sie gedacht waren, nämlich Variablen auszulesen und zu setzen, bzw. Ereignismeldungen zu verschicken, besteht kein Grund, SNMP als Agentensprache einzusetzen, vor allem da andere Agentensprachen (wie z. B. KQML) in den Bereichen Grundwortschatz und Semantik deutlich mehr anzubieten haben.

Einzig und allein aus Kompatibilitätsgründen könnte es doch sinnvoll sein, zusätzlich zu der eigentlichen Agentensprache, SNMP zu verwenden. So wäre es zum Beispiel denkbar, daß ein Event-Server jede erhaltene Ereignismeldung zusätzlich in einen trap verpackt und an einen Manager schickt, damit bisherige Managementanwendungen, die als Protokoll nur SNMP kennen, diese empfangen können.

Auch könnten die Agenten klassische SNMP-MIBs anbieten, mit denen es einem Manager ermöglicht wird, die Agenten auch von einer SNMP-Managementstation - zumindest grob - zu konfigurieren. Sinnvoller wäre es jedoch, die Managementstation um die eigentliche Agentensprache zu erweitern.

5 Existierende Agentenmodelle

In den nächsten Abschnitten soll untersucht werden, inwieweit sich die Kommunikationsmodelle von KQML als Standard-Agentensprache und CORBA als Standard für verteilte Anwendungen für den Einsatz in einem Mutli-Agentensystem eignen.

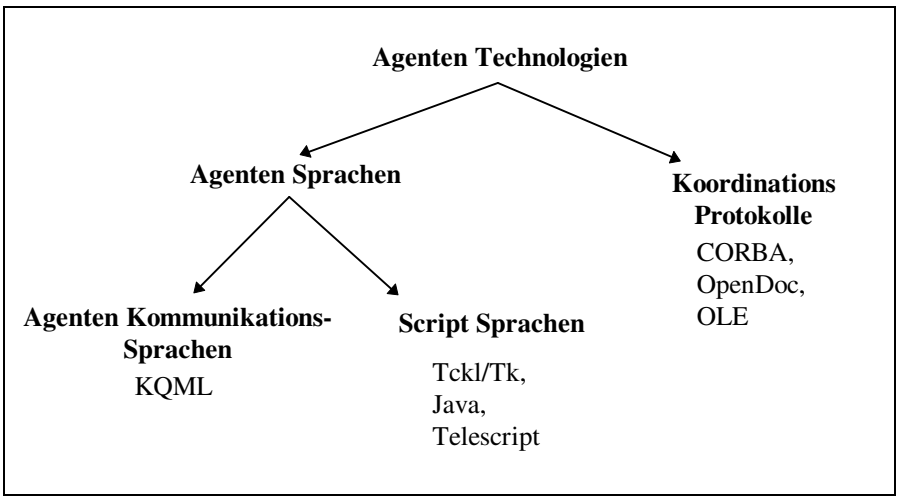


Abbildung 5-1 Einordnung der Agenten Technologien

5.1 KQML

Die „Knowledge Query and Manipulation Language“ (KQML) wurde von der „External Interface Group“ im Rahmen des „Knowledge-Sharing Effort“ (KSE) der „Advanced Research Projects Agency“ (ARPA), des „Air Force Office of Scientific Research“ (AFOSR), der „Corporation for National Research Initiative“ (NRI) und der „National Science Foundation“ (NSF) als Sprache speziell zur Kommunikation und Kooperation zwischen intelligenten Agenten entwickelt. KQML wurde schon für viele Informationssysteme auf unterschiedlichen Plattformen erfolgreich eingesetzt. [KQML 93]

5.1.1 Kommunikationsmodell

An das Kommunikationsmodell an sich stellt KQML keine Anforderungen. Sowohl die Organisationsform als auch die Dienstlokalisierung stehen dem Betreiber eines Agentensystems frei.

Neben den normalen Agenten wird noch ein „facilitator“ definiert, der außer Adressen-Auflösung noch weitere Dienste wie routing und brokering anbietet. Mindestens ein facilitator muß pro Domain existieren.

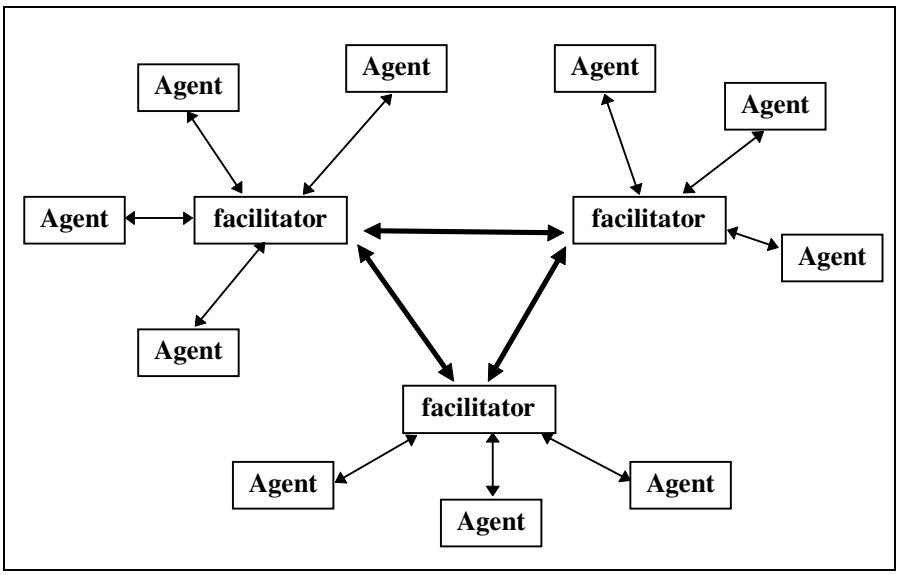


Abbildung 5-2 KQML Multiagenten-System[26]

5.1.2 KQML Nachrichten

KQML Nachrichten werden auch „performatives“ genannt. Eine performative besteht aus ASCII-Zeichen, was den Vorteil hat, daß es sowohl für den Menschen gut lesbar, als auch gut zu parsen und von vielen Plattformen gut zu transportieren ist.

Eine KQML Nachricht besteht aus Schlüssel-Werte-Paaren, welche somit unabhängig von ihrer Reihenfolge sind. Das Format dieser Paare ist an Lisp angelehnt, das heißt, ein Schlüssel wird von einem Doppelpunkt „:“ angeführt und danach folgt der Wert.

[26] aus [MLF]

<performative>	::=	(<word> {<whitespace> :<word> <whitespace> <expression> }*)
<expression>	::=	<word> | <quotation> | <string> | (<word> {<whitespace> <expression>}*)
<word>	::=	<character><character>*
<character>	::=	<alphabetic> | <numeric> | <special>
<special>	::=	< | > | = | + | - | * | / | & | ^ | _ | @ | $ | % | : | . | ! | ?
<quotation>	::=	'<expression> | '<comma-expr>
<comma-expr>	::=	<word> | <quotation> | <string> | ,<comma-expr> | (word {<whitespace> <comma-expr>}*)
<string>	::=	"<stringchar>*" | #<digit><digit>*"<ascii>*
<stringchar>	::=	\<ascii> | <ascii-\-<double-quote>

Tabelle 5-1: KQML Syntax in BNF

Beispiel für eine (typische) KQML-Nachricht:

```
(ask-if
      :sender       A
      :receiver     B
      :language     KIF
      :ontology     foo
      :reply-with   id1
      :content      "bar(a,b)" )
```

5.1.3 Protokollstack

KQML definiert einen Protokollstack, der die Kommunikation zwischen den Agenten ermöglichen soll. Dabei unterscheidet KQML zwischen dem Content Layer, dem Message Layer und dem Communication Layer.

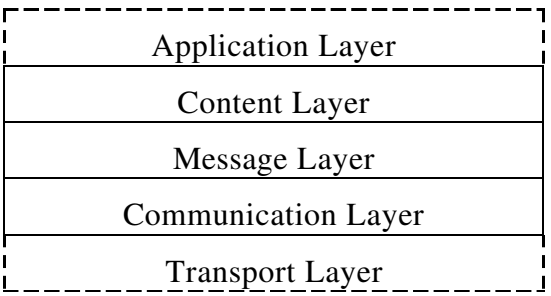


Abbildung 5-3 KQML Protokoll Stack

5.1.3.1 Content Layer

Der Content Layer enthält die eigentliche Information, die verschickt werden soll. Die genaue Form dieser Information interessiert KQML nicht, außer um festzustellen, wo sie beginnt und wo sie endet. Die Information kann z. B. aus ASCII-Strings, Bildern, Maschinencode etc. bestehen, obwohl in den existierenden Implementierungen von KQML nur ASCII unterstützt wird. Häufig enthält der Content eine weitere KQML-Nachricht.

5.1.3.2 Message Layer

Die Aufgabe des Message Layers ist es, die eigentliche Nachricht, das heißt, die Information, die verschickt werden soll, zu beschreiben und in eine Art Umschlag einzupacken.

Der Message Layer bildet somit den Kern von KQML und legt fest, in welcher Art und Weise Agenten miteinander kommunizieren können.

Das Kernstück des Message Layers wiederum bilden die reservierten Schlüsselwörter und performatives.

Von Agenten, die KQML sprechen, wird nicht erwartet, daß sie alle der vordefinierten performatives erfüllen. Sie brauchen nur einen kleinen, von ihnen benötigten Teil, zu verstehen, bei diesem müssen sie sich aber an die Definition halten, um die Interoperabilität zwischen verschiedenen Agentensystem zu gewährleisten.

Schlüssel	**Bedeutung**
:content	Der Inhalt, um den es sich in der Nachricht handelt.
:force	Gibt an, ob der Inhalt von konstanter oder variabler Natur ist.
:in-reply-to	Gibt an, auf welche vorangegengene Nachricht sich diese Nachricht bezieht.
:language	Die Sprache, in der der Inhalt repräsentiert wird.
:ontology	Name der Ontologie, die benutzt wurde, um den Content zu beschreiben.
:receiver	Der Empfänger der Nachricht.
:reply-with	Gibt an, ob eine Antwort erwartet wird, und wenn, worauf sich die Antwort beziehen soll.
:sender	Der Sender der Nachricht.

Tabelle 5-2 Reservierte Schlüsselworte

Zu diesen reservierten performatives existiert kein verabschiedetes Interaktionsprotokoll und keine Semantik, jedoch mehrere Vorschläge.[La 96]

Kategorie	**Name**
Basic query	evaluate, ask-if, ask-about, ask-one, ask-all
Muli-response query	stream-about, stream-all, eos
Response	reply, sorry
Generic information	tell, achieve, cancel, untell, unachieve
Generator	standby, ready, next, rest, discard, generator
Capability-definition	advertise, subscribe, monitor, import, export
Networking	register, unregister, forward, broadcast, route

Tabelle 5-3 Reservierte Performative Namen

Alle reservierten performatives sowie ihre Semantik zu beschreiben, würde den Rahmen dieser Arbeit sprengen. Statt dessen soll anhand von den in Kapitel 5.1.4[27] angegebenen Beispielen der Mechanismus von KQML sichtbar werden.

5.1.3.3 Communication Layer

Die Aufgabe des Communication Layers ist die Aufbereitung und der Versand der KQML-Nachrichten. Dazu fügt er sowohl die :sender und :receiver, als auch die :reply-with und :in-reply-to Werte in die Nachricht ein.

Über das Transportprotokoll an sich macht KQML keine Angaben. Es existieren Implementierungen auf TCP/IP-, HTTP-, CORBA und SMTP-Basis.

Die Form der KQML-Nachricht kann in Abhängigkeit des gewählten Transportprotokolls variieren. So werden z. B. bei der SMTP-Variante die sender/receiver und die reply-with/in-repy-to Werte in den SMTP-Header übernommen. Frühere auf TCP/IP basierte Implementierungen haben diese Werte in einen eigenen Header um die KQML-Nachricht gepackt.

[27] Seite 60

5.1.4 Beispiele[28]

Im Folgenden soll die Verwendung von KQML anhand einiger Beispiele erläutert werden:

5.1.4.1 Beispiel Accounting

Zum Beispiel könnte eine Anfrage an einen Agenten, der für die Benutzerdatenbank einer Domäne zuständig ist, wie folgt aussehen:

Agent1 will von dem Agenten „user-agent“ alle Namen und User-Id’s mit der Group-Id 1003 haben. Angenommen, in der Ontology „user“ wurde festgelegt, daß solche Anfragen in SQL formuliert werden können, so besteht der Inhalt der KQML-Nachricht aus einem select-Statement.

Da der Agent mehrere Antworten erwartet, schickt er eine stream-all KQML-Nachricht, was bedeutet, daß er mehrere Anworten erwartet:

```
(stream-all
      :language     sql
      :ontology     user
      :sender       agent1
      :receiver     user-agent
      :reply-with   63b
      :content      (select name,uid from user where gid=1003))
```

user-agent erhält nun diese Anfrage, führt sie aus, und teilt die Ergebnisse dem agent1 in einer Reihe von KQML-Nachrichten mit. Als language verwendet er KQML und als performative die in der Ontologie user festgelegte performative „tell-user“:

```
(tell
      :language     KQML
      :ontology     user
      :sender       user-agent
      :receiver     agent1
      :in-reply-to  63b
      :content      (tell-user
                    :name     „Hans Meier“
                    :uid      513))

(tell
      :language     KQML
      :ontology     user
      :sender       user-agent
      :receiver     agent1
      :in-reply-to  63b
      :content      (tell-user
```

[28] Bemerkung: In diesen Beispielen sollen relativ viele Möglichkeiten von KQML in einfacher Form dargestellt werden. Tatsächlich würde Kommunikation in dieser Form so nicht vorkommen.

```
                    :name     „Klaudia Huber"
                    :uid      703))
```

Mit der performative eos (end-of-stream) teilt der user-agent dem agent1 mit, daß keine weiteren Ergebnisse folgen:

```
(eos :sender        user-agent
     :receiver      agent1
     :in-reply-to   63b)
```

Gäbe es keine Benutzer mit der Group-Id 1003 würde user-agent mit einem „sorry" antworten, was bedeutet, daß er die Anfrage zwar verstanden hat, er aber keine Informationen dazu liefern kann:

```
(sorry :sender      user-agent
       :receiver    agent1
       :in-reply-to 63b)
```

Eine „error" Nachricht würde bedeuten, daß er die Anfrage nicht verstanden hat.

5.1.4.2 Beispiel Verfügbarkeit

Angenommen, ein Agent A möchte wissen, ob ein bestimmten Rechner von Agent B aus erreichbar ist.

Dazu würde er ihm folgende Nachricht schicken:

```
(ask-if
      :language     KQML
      :ontology     fault
      :sender       AgentA
      :receiver     AgentB
      :reply-with   ping-3
      :content      (ping :host 131.159.27.112))
```

Ebenfalls angenommen, der entsprechende Host ist von Agent B pingbar, so würde er folgende Nachricht zurückschicken:

```
(tell
      :language     KQML
      :ontology     fault
      :sender       AgentB
      :receiver     AgentA
      :in-reply-to  ping-3
      :content      (ping :host 131.159.27.112 :value alive))
```

Wenn nun Agent A aber die Information, ob der Rechner von Agent B aus pingbar ist, nicht nur einmal braucht, sondern immer den aktuellen Stand über die Verfügbarkeit des Rechners haben will, so könnte er den Agent B beauftragen, ihn immer über den aktuellen Stand zu informieren.

Dazu ist in KQML die performative „monitor" vorgesehen, die besagt, daß der Empfänger den Sender immer dann benachrichtigen soll, wenn sich die Antwort auf die Anfrage ändern würde.

Dieser Auftrag würde somit folgendermaßen aussehen:

```
(monitor
      :language     KQML
      :ontology     fault
      :sender       AgentA
      :receiver     AgentB
      :reply-with   m64
      :content      (ping :host 131.159.27.112))
```

Auf diese Anfrage würde der Agent B erst einmal mit obiger tell-Nachricht antworten. Falls der Rechner dann irgendwann mal nicht mehr pingbar sein sollte, so würde sich ja die Antwort auf die Anfrage ändern, und somit würde Agent B dann zum Beispiel folgende Nachricht an Agent A senden:

```
(tell
      :language     KQML
      :ontology     fault
      :sender       AgentB
      :receiver     AgentA
      :in-reply-to  m64
      :content      (ping :host 131.159.27.112 :value timeout))
```

Sobald sich die Antwort auf die monitor-Anfrage wieder ändern würde, also der Rechner wieder erreichbar wäre, würde Agent B natürlich wieder eine „:value alive“-Nachricht an Agent A schicken.

5.2 CORBA

In den folgenden Absätzen wird ein kurzer Überblick über CORBA gegeben und der Einsatz von CORBA bei der Entwicklung eines Multi-Agentensystem untersucht. Außerdem wird das Common Agent Facility der CORBAfacilities kurz vorgestellt.

5.2.1 Überblick

Die „Common Object Requst Broker Architecture“ (CORBA) wurde von der ORB Task Force[29] der Object Management Group (OMG) entworfen. Es wurde ein Objekt Request Broker (ORB) entwickelt, der es Objekten ermöglicht, transparent requests zu stellen und responses darauf zu erhalten. Dadurch wird es Applikationen mit CORBA ermöglicht, in verteilten, heterogenen Umgebungen auf verschiedenen Maschinen zu interagieren. [OMG 91]

Das von der OMG entwickelte Objektmodell sieht Objekte vor, die ihre Dienste anderen Objekten zur Verfügung stellen. Diese Dienste sind in einem sogenann-

[29] Ein Zusammenschluß aus Digital Equipment Corporation, Hewlett-Packard Company, HyperDesk Corporation, NCR Corporation, Object Design, Inc, SunSoft, Inc.

ten „Interface“ definiert und in der IDL (Interface Definition Language) beschrieben. Diese Dienste melden die Objekte bei ORB an.

Ein Client, der diese Dienste nutzen will, stellt einen (in der IDL definierten) request an den ORB, der diesen wiederum an das Objekt weitergibt, das diesen Dienst anbietet. Dabei ist es für den Client irrelevant, auf welcher Maschine dieses Objekt liegt. Ebenso wird die response von dem Objekt an den ORB zurückgegeben, der diese dann wiederum an den Client weiter reicht.

Eine weitere Möglichkeit für den Agenten, einen Dienst zu nutzen, ist das Dynamic Invocation Interface (DII), das es dem Client erlaubt, Dienste zu nutzen, deren IDL er nicht kennt, indem er den Dienstaufruf „beschreibt“. Der ORB findet dann einen, zu diesem Aufruf passenden, Dienst.

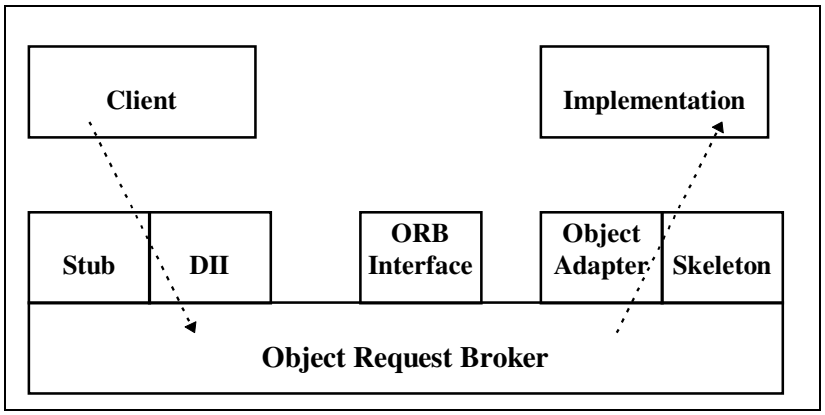


Abbildung 5-4 CORBA Architektur

Zu der CORBA-Architektur wurden noch eine Reihe von standardisierten Diensten definiert, die sogenannten „CORBAservices“ und „CORBAfacilities“. Diese können dazu verwendet werden, um verteilte Applikationen oder neue Services bzw. facilities zu entwickeln

Dazu gehören neben vielen anderen zum Beispiel:

- Naming Service
- Event Service
- Agent Facility [30]
- Task Management Facility

Außerdem wurden Protokolle entworfen, die eine Interoperabilität zwischen den verschiedenen Object Request Brokern ermöglichen.

[30] siehe Kapitel 5.2.3, S. 65

5.2.2 Einsatz von CORBA in einem Multi-Agentensystem

Da CORBA zur Entwicklung von verteilten, objekt-orientierten Systemen auf heterogenen Plattformen entworfen wurde, ist zu prüfen, inwieweit CORBA eingesetzt werden kann, um ein Multi-Agentensystem zu implementieren.

5.2.2.1 CORBA zum Transport von Nachrichten

CORBA wurde entwickelt, um ein transparentes Client-Server-Modell zu ermöglichen und baut aus diesem Grund auf einen synchronen request/response-Mechanismus auf. Daher ist es vom primären Konzept nicht für den Einsatz von Agenten geeignet, die einen asnychronen Messaging-Mechanismus verwenden.

Dennoch gibt es verschiedene Möglichkeiten, wie CORBA zum Transport von Agenten-Nachrichten verwendet werden kann:

- **Asynchrone Nachrichten mit dem Event Service**

  Der Event Service sieht einen asynchronen Nachrichtenversand vor, und könnte daher dazu verwendet werden, Agenten-Nachrichten zu verschikken.

  Dies ist deshalb besonders interessant, da im Event Service ein Multicast vorgesehen ist und somit eine Gruppenkommunikation ermöglicht wird.

- **„Mißbrauchen“ eines requests**

  Eine weitere Möglichkeit wäre, die Nachricht als Parameter in einen request zu verpacken, und diesen dann an den Ziel-Agenten zu versenden. Die response auf diesen request wird mehr oder weniger ignoriert.

- **Adaptieren des CORBA-Konzeptes**

  Natürlich kann das CORBA-Konzept auch so verwendet werden, wie es gedacht war. Das heißt, jeder Agent formuliert seine „Fähigkeiten“, also die Aufträge, die er versteht, in einer IDL und bietet diese als Dienst durch einen ORB an.

  Hierbei könnte das Dynamic Invocation Interface zum Einsatz kommen, das es den Agenten ermöglicht, Aufträge an andere Agenten zu stellen, auch wenn deren exakte IDL ihnen nicht vorliegt.

  Durch den Lifecycle Service wäre es zudem unter Umständen möglich, zur Laufzeit Objekte neuer Klassen zu kreieren, und somit die geforderte Flexibilität der Agenten zu erreichen. Dies gilt es noch genauer zu prüfen.

  Einfache Nachrichten (wie beispielsweise Events) könnten durch den Event Service übertragen werden.

5.2.2.2 Weiterer Einsatz von CORBA

Ein weiterer Schritt wäre, CORBA nicht nur zum Transport von Nachrichten zu verwenden, sondern auch noch andere Dienste von CORBA und den Common Facilites zu nutzen. Allen voran natürlich der Trading Service für den Einsatz des ORB als Vermittler, um im Multi-Agentensystem den richtigen Agenten für einen Auftrag bzw. den passenden Empfänger für eine Nachricht zu finden.

Weiterhin können diverse Dienste verwendet werden:

- Naming Service für die Entwicklung des Agent-Name-Server[31]
- Event Service für den Agent-Event-Server[32]
- Security Service für den Agent-Key-Server[33]
- weitere Dienste wie zum Beispiel der Query Service können für die Entwicklung der Management Applikationen verwendet werden.

5.2.3 CORBA Agent Facility

In den CORBAfacilities ist ein „Agent Facility“ enthalten, das an KQML angelehnt ist.

Im Common Agent Facility werden normale Agenten, sogenannte „static agents“ und Nachrichten zwischen den Agenten, die „mobile agents“, definiert. Durch die Verwendung des „Dynamic Invocation Interface“ aus CORBA v2.0 können die Agenten andere CORBA-Dienste nutzen, wie z. B.: Naming Service, Event Service, Trader Service, Access Control Service, etc.

Ansonsten ist das Kommunikationsmodell nicht weiter spezifiziert. *(„However, the exact profiles and components required need further study“).*

Wenige Wochen vor Fertigstellung dieser Arbeit wurde von der International Buisness Corporation (IBM), The Open Group und GMD FOKUS eine ausführliche „Mobile Agent Facility Spezification“ vorgelegt [MAF 96]. Obwohl diese Arbeit sich ausschließlich mit mobilen Agenten beschäftigt, ist sie dennoch interessant, da sie in drei getrennte Frameworks untergliedert wurde:

- **Agent Framework**

  Im Agent Framework werden mobile Agenten, ihre Identitäten, sowie die Kommunikation und Interaktionen zwischen den Agenten definiert.

[31] siehe Kapitel 3.4.5, S. 28

[32] siehe Kapitel 3.6.2, S. 37

[33] siehe Kapitel 3.3.3, S. 21

Außerdem enthält es ein Sicherheitskonzept zum Schutz eines Hosts vor bösartigen Agenten, sowie zum Schutz der mobilen Agenten vor bösartigen Hosts.

- **Agent Transfer Framework**

  Das Agent Transfer Framework definiert Schnittstellen und Mechanismen zum Senden und Empfangen von mobilen Agenten inklusive ihrer Daten, unabhängig von dem tatsächlich verwendeten Transportprotokoll.

- **Agent Execution Framework**

  Das Agent Execution Framework enthält Schnittstellen, die das Ausführen von Agenten inklusive ihrer mitgebrachten Daten behandeln. Dazu werden unter anderem generische Interfaces, wie das AgentClass Interface, das AgentClassManager Interface oder auch das AgentClassLoader Interface, vorgestellt.

  Diese Interfaces sind unabhängig vom tatsächlichen Format des ausführbaren Codes und dem Format der Daten.

Im Agent Framework werden unter anderem Agents, Agent Proxies und Agent Contexts definiert.

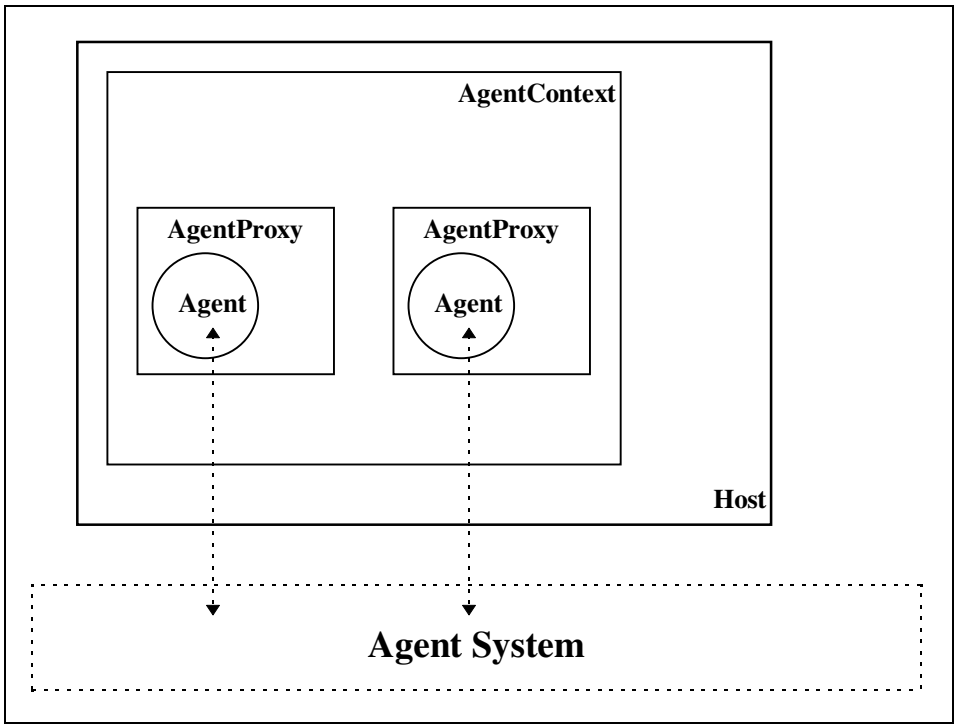


Abbildung 5-5 CORBA Agent Facility

Der AgentProxy schützt den Agenten vor dem Host, und der AgentContext den Host vor dem Agenten. Der AgentContext bietet dem Agenten definierte Schnittstellen an, auf die dieser bei der Erledigung seines Auftrages zurückgreifen kann. Im AgentProxy wird festgelegt, welche Methoden des Agenten, falls überhaupt,

von außerhalb aufgerufen werden können. Normalerweise können beim Agenten keine Methoden aufgerufen werden, sondern es kann nur eine asynchrone Kommunikation mit dem Agenten aufgenommen werden. Diese Kommunikation ist aber nicht weiter spezifiziert.

6 Prototypische Implementierung

Zu dem im bisherigen Teil vorgestellten Kommunikationsmodell, wurde in Zusammenarbeit mit A. Hollerith, aufbauend auf einen vorhandenen Agenten, eine prototypische Plattform für ein Multi-Agentensystem entwickelt.

Dieses wird in den folgenden Absätzen vorgestellt.

6.1 Ansatz

Bei der Implementierung eines flexiblen Agenten muß zwischen dem eigentlichen Agenten, der die Kommunikation mit den anderen Agenten und den Empfang von delegierter Funktionalität bereitstellt, und der zu delegierenden Funktionalität unterschieden werden.

6.1.1 Agenten

Bevor mit der Implementierung begonnen werden konnte, mußte entschieden werden, auf welchen Plattformen die Agenten eingesetzt werden sollen, in welcher Programmiersprache sie entwickelt, und welche vorhandenen Bibliotheken genutzt werden konnten.

6.1.1.1 Plattform

Im Idealfall sollten die Agenten auf den Komponenten laufen, die gemanagt werden sollen. Bei einem Großteil der Managed Objects ist das ohnehin nicht möglich, da diese keine Möglichkeit bieten, fremde Programme ablaufen zu lassen (Router, Drucker, etc.).

Trotzdem sollte die Implementierung des Agenten eine Portierung auf möglichst viele Plattformen ermöglichen und nicht auf ein Betriebssystem zugeschnitten sein.

6.1.1.2 Programmiersprache

Um diese Portierung auf viele Plattformen zu ermöglichen, sollte eine Programmiersprache gewählt werden, die für viele Plattformen erhältlich ist.

Dabei standen folgende Kategorien zur Wahl:

- **Skriptsprachen**

  Hier gibt es zum einem das weite Feld der interpretierten Scriptsprachen, wie Perl, Tcl/Tk, usw. Diese Sprachen bieten zum einen den Vorteil, daß die zugehörigen Interpreter auf vielen Plattformen verfügbar sind und der Source-Code keine Portierung benötigt, zum anderen, daß dafür schon viele Bibliotheken vorhanden sind. Sie haben allerdings den Nachteil, daß sie nicht gerade sehr performant sind.

- **Compiler-Sprachen**

  Dann gibt es die Compiler-Sprachen, die direkten Maschinencode produzieren, wie z. B. C++. Sie haben den Vorteil, sehr performant zu sein, und Compiler existieren ebenfalls für viele Plattformen. Nachteil ist allerdings, daß der Source-Code meistens für jede Plattform extra angepaßt werden muß.

- **Java**

  Zuletzt gibt es noch die Sprache Java, die einen Kompromiß aus interpretierten und compelierten Sprachen darstellt, indem sie einen sogenannten Bytecode herstellt, der von einer virtuellen Maschine ausgeführt werden kann. Diese virtuellen Maschinen gibt es mittlerweile für alle erdenklichen Plattformen. Somit hat Java den Vorteil der kompletten Plattformunabhängigkeit, wie die Scriptsprachen, ist aber performanter und bietet mehr Möglichkeiten.

Zum einen aus diesen Gründen, zum anderen aber auch aufgrund der Wahl, Java für die zu delegierende Funktionalität einzusetzen (siehe Kapitel 6.1.2), wurde die Implementierung des flexiblen Agenten in Java realisiert.

6.1.2 Funktionalität

Einer der Hauptaspekte des flexiblen Agenten ist seine Fähigkeit, neue Funktionalität zur Laufzeit aufzunehmen und auszuführen. [Ho 97]

Hier ist nun die Frage, wie diese Funktionalität aussehen soll, damit sie zur Laufzeit ausgeführt werden kann. Dazu gibt es im Prinzip drei Ansätze:

- **Ausführbarer Code**

  Funktionalität könnte zum einen als ausführbarer Code übermittelt werden, der von dem Agenten einfach in einem eigenen Prozeß gestartet wird. Über diesen Prozeß hat dann der Agent Kontrolle (zum Beispiel über IPC[34]). [WeHo 96]

[34] Inter Process Communication

Dieses Vorgehen hat mehrere Nachteile. Zum einen muß der Code für alle Plattformen, auf denen die Agenten eingesetzt werden, extra compeliert werden. Zum anderen ist die Anbindung der Funktionalität an den Agenten relativ lose. Die neu hinzugekommene Funktionalität kann z. B. nicht direkt (z. B. über einen gemeinsamen Speicher) auf schon vorhandene Objekte im Agenten zugreifen.

- **Scripte**

  Daher verwenden viele Lösungen sogenannte Scriptsprachen, in denen die Funktionalität geschrieben wird.

  In diesem Fall besitzt der Agent einen eingebauten Interpreter, der das Script Schritt für Schritt ausführt. Dadurch erreicht man eine hohe Anbindung der Funktionalität an den Agenten.

  Dazu bedarf es natürlich einer Scriptsprache, die einfach genug ist, um sie in einem Agenten schnell und effizient zu interpretieren, und die mächtig genug ist, um alle möglichen verteilten Managementanwendungen zu realisieren.

  In diesem Bereich wurden mehrere Scriptsprachen für verteilte Managementanwendungen entwickelt, wie z. B. die SNMP Script Language oder die NMP Stack Language. Diese Sprachen bieten neben den nötigen Schleifen- und Bedingungskonstrukten einen komfortable Zugriff auf MIB-Variablen. Für den Einsatz von Managementanwendungen, die nicht nur auf die MIB zugreifen wollen sind sie allerdings nicht geeignet (z. B. keine Datei- oder Netzwerkoperationen möglich). Um den flexiblen Agenten aber wirkliche Flexibilität zu ermöglichen, sind solche Operationen notwendig.

  Neben diesen, für Netzmanagement entwickelten Scriptsprachen, gibt es noch einige Scriptsprachen speziell für Agenten, wie AgentTcl oder Telescript.

- **Java**

  Eine weitere Möglichkeit, um Funktionalität für flexible Agenten zu entwickeln, wäre Java. Java bietet wie die Scriptsprachen den Vorteil absolut plattformunabhängig zu sein. Außerdem läßt sich der neu hinzugekommene Code - im Gegensatz zu anderen compelierten Sprachen - während der Laufzeit durch den ClassLoader einbinden.

  Gegenüber den existierenden Scriptsprachen bietet Java den Vorteil, daß die Sprache mächtig genug ist, um alle möglichen Managementanwendungen zu realisieren. Den Vorteil mancher Scriptsprachen, auf Netz- und Systemmanagement spezialisiert zu sein, kann Java durch die Existenz von entsprechenden Bibliotheken (z. B. für SNMP) kompensieren. Zudem gibt es seit kurzem eine Java Management API [JMAPI 96], die, wie es aussieht, weitreichende Akzeptanz und Einsatz finden könnte.

Da Java für den Einsatz von flexiblen Agenten am besten geeignet scheint, wird Java sowohl für die Implementierung des Agenten, als auch für die Implementierung der zu delegierenden Funktionalität eingesetzt.

6.1.3 Entwicklungsumgebung

Es existieren derzeit auf dem Markt eine Vielzahl von Entwicklungsumgebungen für Java. Da Java aber sowohl beim Sourcecode, als auch im Bytecode plattformunabhängig ist, ist die Wahl der Entwicklungsumgebung letztendlich irrelevant.

In diesem Fall wurde Visual J++ von Microsoft eingesetzt.

6.1.4 Bibliotheken

Neben den normalen Java-Bibliotheken gibt es für die Entwicklung von Agenten in Java einige vorhandene Pakete, die verwendet werden könnten, um obiges Konzept zu realisieren. Viele dieser Pakete beschränken sich jedoch auf einfache Applets, andere legen den Schwerpunkt auf mobile Agenten.

Für die Implementierung des flexiblen Agenten wurde als Grundlage das an der Stanford University entwickelte Java Agenten Template (JAT) in der Version 0.3 verwendet, weil es über die grundlegende Funktionalitäten in der Kommunikation und Ressourcenverwaltung verfügt [JAT 96].

Trotzdem mußten einige Veränderungen an dem Template vorgenommen werden, um die benötigte Flexibilität zu erreichen.

6.2 Das Java Agent Template

Im folgendem Teil wird sowohl die Architektur, als auch die Funktionsweise, sowie die benötigten Änderungen des Java Agent Templates der Stanford University kurz vorgestellt.

6.2.1 Architektur

Der Aufbau des Java Agent Template in der Version 0.3 wird in der folgenden Abbildung dargestellt:

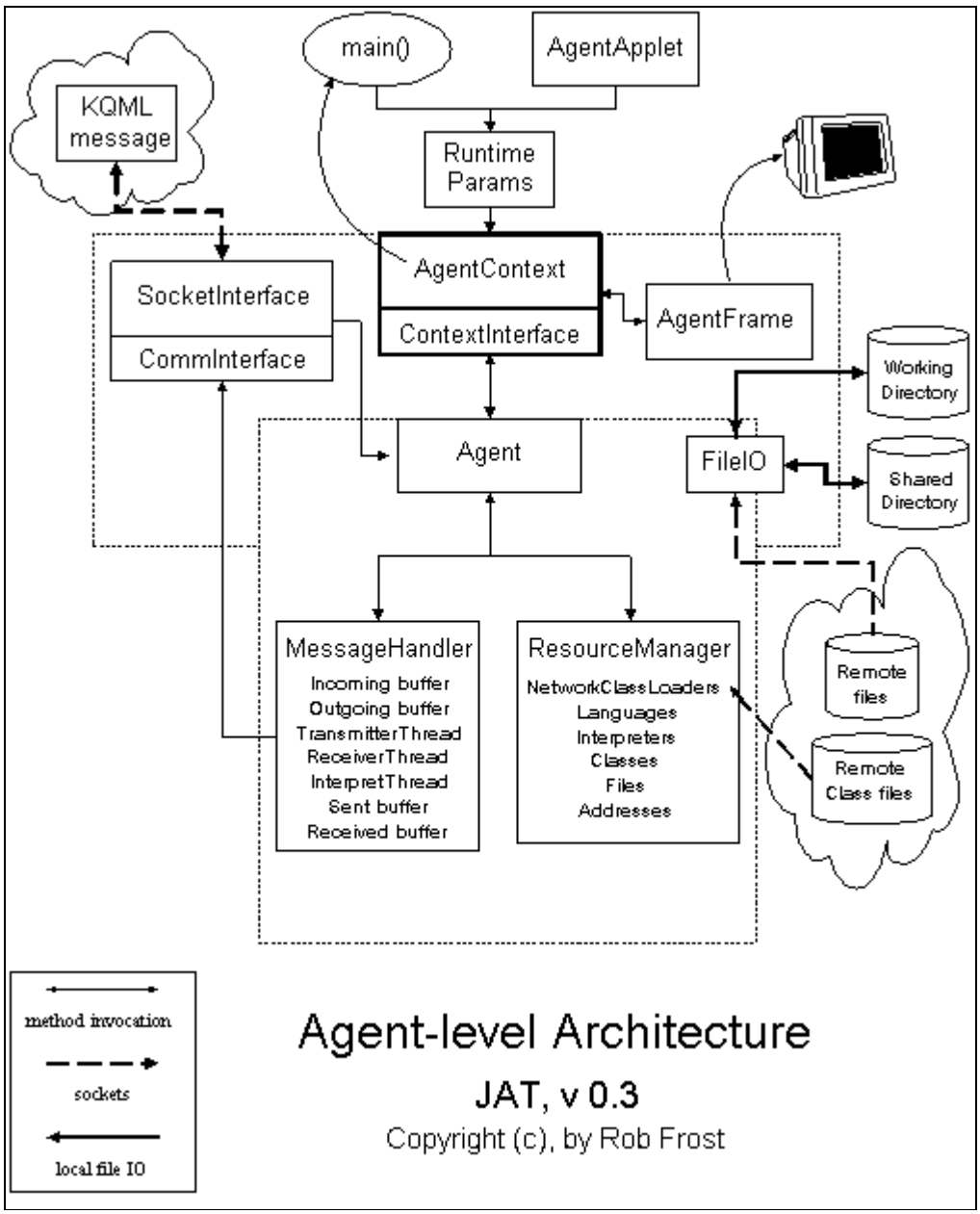


Abbildung 6-1 Architektur des Java Agent Template

Das Java Agent Template besteht im wesentlichen aus folgenden Komponenten:

- **MessageHandler**

  Der MessageHandler stellt die grundlegende Funktionalität zum Versand und zum Empfang von Nachrichten bereit. Dazu verfügt er über incoming und outgoing buffer, in die Nachrichten für den Agenten abgelegt bzw. Nachrichten von dem Agenten abgeholt werden. Außerdem gibt es noch die sent und received buffer, in denen gesendete und empfangene Nachrichten gespeichert werden können.

  Für die tatsächliche Kommunikation verwendet der MessageHandler das SocketInterface.

- **SocketInterface**

  Das SocketInterface bietet die, im CommInterface definierten Schnittstellen zum Versand und Empfang von Nachrichten, über TCP/IP Sockets an. Dabei baut das SocketInterface eine einfache TCP/IP-Verbindung zum Empfänger-Agenten auf und überträgt die Nachricht.

  Neben dem SocketInterface könnten auch beliebig viele andere Interfaces verwendet werden, die die Schnittstellen des CommInterface anbieten.

- **ResourceManager**

  Der ResourceManager verwaltet alle Ressourcen des Agenten. Dazu besitzt er zu jedem Ressource-Typ einen Behälter (storage), in den die Ressourcen abgelegt werden. Wenn der Agent eine Ressource benötigt (z. B. eine Adresse), so versucht er, diese vom ResourceManager zu holen. Wenn der ResourceManager die angeforderte Ressource nicht besitzt, so holt er sich diese von einem anderen Agenten. Dazu verwendet er den, in Java beinhalteten, NetworkClassLoader bzw. die URL-Klasse.

  Als Ressourcen kennt das JAT folgende Speicher:

  - **Interpreters**: Für jede Ontologie gibt es einen Interpreter[35]. Bei dem JAT ist der sogenannte AgentInterpreter dabei, der für die grundlegenden Fähigkeiten, wie Adreß- oder Code-Anfragen zuständig ist.
  - **Languages**: Sprachen, die die Interpreter verwenden können, um Nachrichten zu interpretieren. Die Languages können sowohl Nachrichten in ihrer Sprache erstellen, als auch parsen. Mit dem JAT-Paket wird die Sprache KQML mitgeliefert.
  - **Adresses**: Adressen enthalten die Zuordnung Agentenname ↔ Netzwerk-Adresse. Das JAT-Paket enhält passend zum SocketInterface SocketAdressen, also die Zuordnung Name ↔ TCP/IP-Adresse.

[35] siehe Kapitel 6.2.2, S. 74

- **Classes**: Ein allgemeines storage für Java-Klassen aller Art.
- **Files**: Ein storage für Dateien. Die tatsächlichen Dateien werden allerdings nicht im ResourceManager gehalten, sondern in einem Verzeichnis auf der Festplatte. Der ResourceManager enthält nur die Referenz (URL) auf die Dateien.

- **AgentContext**

  Der AgentContext entspricht der Schnittstelle des Agenten zum Benutzer. Hier können z. B. beim Aufruf des Agenten Parameter mitgegeben werden, oder es kann eine graphische Benutzeroberfläche definiert werden, über die der Agent gesteuert werden kann. Diese kann über die im ContextInterface definierten Schnittstellen auf den MessageHandler und den ResourceManager zugreifen.

  Zudem können Interpreter über das ContextInterface auf lokale Ressourcen des Agenten zugreifen (z. B. lokale Datenbanken, etc.)

  Beim JAT-Paket ist ein einfacher Agent dabei, der eine graphische Oberfläche anbietet, über die man sich die Inhalte des MessageHandlers und des ResourceManagers anzeigen lassen kann. Außerdem enthält das Paket noch einen Agent-Name-Server, sowie eine Beispiel-Applikation, die mit dem JAT entwickelt wurde.

Die im Java Agent Template Paket enthaltenen Agenten sind nicht zum direkten Gebrauch gedacht. Vielmehr stellen sie Beispielanwendungen des Templates dar.

Ein mit Hilfe des JAT entwickelter Agent sollte durch Vererbung, die im JAT-Paket enthaltenen Klassen erweitern.

6.2.2 Interpreter

Die Fähigkeiten eines Agenten und die Kommandos, die er versteht, werden in sogenannten Onotologies definiert.

Wenn nun ein Agent für z. B. einen Drucker zuständig sein soll, so muß er alle Kommandos (performatives) der Drucker-Onotologie verstehen. Das heißt, er braucht als erstes einen Interpreter, der diese performatives versteht und entsprechend reagiert.

Folgendes geschieht, wenn ein JAT-Agent eine Nachricht erhält:

- Überprüfen der Ontologie
- Falls keine entsprechende Interpreter Klasse vorhanden ist, versuchen, diese vom Absender der Nachricht zu erhalten.
- Den Interpreter als neuen Thread starten und ihm die Nachricht übergeben.

- Der Interpreter-Thread parsed die Nachricht, lädt unter Umständen neue Klassen nach und reagiert auf die Nachricht.
- Der Interpreter-Thread wird beendet.

Die Grundfunktionalitäten, wie Nachfragen nach einer Interpreter Klasse, einer Adresse, etc. oder Beantworten solcher Anfragen, werden in dem sogenannten „AgentInterpreter“ definiert, der in jedem Agenten enthalten ist und nicht nachgeladen werden muß.

Eine weitere Möglichkeit, die Fähigkeiten eines Agenten zu erweitern, ist, schon zum Zeitpunkt der Entwicklung neue Fähigkeiten einzubauen, ähnlich dem eingebauten „AgentInterpreter“. Dazu können vorhandene Ur-Klassen durch Vererbung einfach erweitert werden. So wird bei dem JAT-Paket zum Beispiel ein Nameserver-Agent mitgeliefert, dessen Fähigkeiten fest eingebaut sind.

6.2.3 Änderungen am JAT

Das vom Entwickler des JAT-Paketes vorgesehene Konzept der Erweiterung durch Vererbung konnte nicht verwendet werden, da der flexible, kooperierende Agent möglichst flexibel und seine Fähigkeiten nicht schon zum Zeitpunkt der Entwicklung festgelegt sein sollten. Folglich muß sämtliche Funktionalität in die nachladbaren Interpreter gesteckt werden.

Dadurch, daß die Agenten nicht nur flexibel, sondern auch kooperativ sein müssen, benötigen sie die Fähigkeit, mit anderen Agenten zu kommunizieren. Dies ist im JAT-Paket nicht vorgesehen, denn dort wird ein Interpreter gestartet und dann „vergessen“, das heißt, wenn eine weitere Nachricht an den Agenten gesendet wird, so startet dieser wieder einen neuen Interpreter. Für einen kooperierenden Agenten wäre es aber nötig, daß die Nachricht an den bestehenden Interpreter weitergeleitet wird. Deswegen mußte der JAT-Agent um einen Task-Manager[36] erweitert werden, der Kontrolle über die einzelnen laufenden Interpreter hat und Nachrichten, die an sie adressiert sind, weiterleitet.

Ein weiterer Punkt, um den das JAT-Paket erweitert werden mußte, war die Mutlicast-Fähigkeit, das heißt die Fähigkeit, eine Nachricht an eine Gruppe von Agenten zu versenden. Da aufgrund der Probleme mit Multicasts in IPv4 auf echtes Multicasten verzichtet werden sollte, wurde die Beziehung „1 Name $\leftrightarrow$ 1 IP-Adresse“ auf „1 Name $\leftrightarrow$ n IP-Adressen“ erweitert.

Aufgrund dieser Erweiterung mußte auch das „Storage“ für Adressen im „ResourceManager“ deaktiviert werden, damit der Name bei jeder Nachricht neu beim Agent-Name-Server aufgelöst wird, denn die Zusammensetzung einer Gruppe kann sich ja ständig ändern.

[36] siehe [Ho97]

Damit aber bei einer „normalen“ Antwort auf eine Nachricht die Adresse des Absenders beim ANS nicht immer wieder neu aufgelöst werden muß, enthält nun jede Nachricht einen Unified Agent Locator (UAL) des Absenders, die die TCP/IP-Adresse des Absenders beinhaltet. Somit kann bei einfachen Anworten, die normalerweise nicht an eine ganze Gruppe, sondern nur an den Absender gehen sollen, die Adresse direkt verwendet werden, ohne daß sie beim ANS neu angefordert werden muß.

6.3 Architektur

Informationen über die genaue Architektur des flexiblen Agenten und über den Aufbau und die Funktionsweise der prototypischen Implementierung sind in [Ho97] zu finden.

6.4 Server

Wie oben schon beschrieben, werden die Fähigkeiten eines Agenten durch neu hinzukommende Interpreter erweitert. Um das Multi-Agentensystem möglichst flexibel zu gestalten, werden die Server, die nötig sind, um ein System aus flexiblen, kooperierenden Agenten zu betreiben, wie zum Beispiel der Agent-Name-Server, nicht als eigenständiges Programm implementiert, sondern ebenfalls als Interpreter. So ist es beispielsweise möglich, die Funktionalität eines ANS an jeden beliebigen Agenten zu delegieren.

Diese Interpreter sind, wie alle anderen Interpreter, aus der Klasse AInterpreter abgeleitet.

6.4.1 Class AInterpreter

Die Klasse AInterpreter ist die Ur-Klasse aller Interpreter, deren Methoden und Eigenschaften an jeden Interpreter vererbt werden. Sie definiert die Schnittstellen zwischen dem Agenten und den Interpretern.

Genaueres zu dieser Klasse ist in der Arbeit von A. Hollerith [Ho97] zu finden.

6.4.2 ANSInterpreter

Der ANSInterpreter stellt den Interpreter für den Agent-Name-Server[37] dar und ist somit für Auflösung von Agentennamen nach TCP/IP-Adressen zuständig.

Der ANS erkennt folgende performatives der Ontologie ANS:

- ask-resource (:type address :name <Name>)
- ask-resource (:type domain)
- tell-resource (:type address :name <Name> :value <Adresse>)
- tell-resource (:type domain :name <Name>)
- invalidate-resource (:type address :name <Name>)

```
public class ANSInterpreter extends AInterpreter {
    protected String domain;
    protected Hashtable Adresses;

    public void interpretMessage(KQMLmessage message, Agent receiver);
    protected void interpretLanguage(KQMLmessage message, Agent receiver,
    Language language);
    public synchronized void run();
    protected void insertAction(String name, String value, Agent parent);
    protected void removeAction(String name, String sender, Agent parent);
    protected void askAction(KQMLmessage original, String name, String sender,
    Agent parent);
    protected void askDomainAction(String sender, Agent parent);
    protected void tellDomainAction(Agent parent);
}
```

Abbildung 6-2 Class ANSInterpreter

- **protected Sting domain**

  Enthält den eignen Domänennamen.

- **protected Hashtable Adresses**

  Dies ist die Tabelle, die alle Adressen enthält. Implementiert wurde sie als Assoziativspeicher, der als Schlüssel die Namen der Agenten und als Werte Objekte der Klasse SocketAdresses hat. Ein SocketAdresses-Objekt enthält einfach eine Tabelle von TCP/IP-Adressen.

- **run(), interpretLanguage(), interpretMessage()**

  Siehe Class AInterpreter [Ho97].

- **protected void insertAction(String name, String value, Agent parent)**

  Diese Methode wird von interpretLanguage() aufgerufen, wenn eine neue Adresse aufgenommen werden soll.

[37] siehe Kapitel 3.4.5, S. 28

Als erstes wird überprüft, ob der angegebene Name schon existiert. Falls ja, so wird in das entsprechenden SocketAdresses-Objekt die neue TCP/IP-Adresse hinzugefügt. Wenn nicht, wird ein neues SocketAdresses-Objekt erzeugt und als neuer Eintrag zu dem neuen Namen in die Hashtable Adresses eingefügt.

- **protected void removeAction(String name, String sender, Agent parent)**

  Diese Methode wird von interpretLanguage() aufgerufen, wenn eine Adresse aus der Adreßtabelle entfernt werden soll.

  Als erstes wird das SocketAdresses-Objekt aus der Adreßtabelle ausgewählt. Dann wird die TCP/IP-Adresse des Senders daraus entfernt. Ist das SocketAdresses-Objekt nun leer, so wird es ganz gelöscht und der Name aus der Adresses-Tabelle entfernt.

- **protected void askAction(KQMLmessage original, String name, String sender, Agent parent)**

  Diese Methode wird von interpretLanguage() aufgerufen, wenn ein Agent nach einer Adresse fragt. Die Message „original“ ist die Nachricht, wie sie weitergeleitet werden soll, „message“ enthält den content der Orginal-Nachricht.

  - Liegt die gesuchte Adresse in der eigenen Domäne?
    - Wenn ja, paßt eine existierende Adresse der Adresses-Tabelle?
      - Wenn ja, Versenden einer Antwort mit den TCP/IP-Adressen aller passenden Agenten an den Agenten, der sie angefordert hat.
      - Wenn nein, Versenden einer Antwort mit „?“ als TCP/IP-Adresse. (Bedeutet: „Adresse existiert nicht“)
    - Wenn nein, dann nach folgendem Algorithmus, den am besten passenden ANS heraussuchen, dessen Adresse bekannt ist: Domäne der angefragten Adresse solange nach oben durchgehen, bis die Toplevel-Domäne erreicht ist. Wurde kein passender ANS gefunden, von der Toplevel-Domäne solange in Richtung der eigenen Domäne heruntergehen bis diese erreicht wurde.

      Beispiel: Gesucht wird: www@a.b.de, eigene Domäne ist: x.y.z.de

      Suchreihenfolge somit: ANS@b.c.de, ANS@c.de, ANS@de, ANS@z.de, ANS@y.z.de.

      - Wurde ein passender ANS gefunden?
        - Wenn ja, Anfrage an diesen weiterleiten.

- Wenn nein, Nachricht an den Agenten mit „?“ als TCP/IP-Adresse.

- **protected void askDomainAction(String sender, Agent parent)**

  Diese Methode schickt eine tellDomain-Nachricht mit der Domäne des ANS an den Absender.

- **protected void tellDomainAction(Agent parent)**

  Hiermit wird dem ANS seine Domäne mitgeteilt. Diese Nachricht erhält der ANS auf jeden Fall beim Start. Falls der ANS später, im laufenden Betrieb diese Nachricht erhält, so bedeutet das, daß er seinen Domänennamen ändern muß. Daraufhin teilt er allen eingetragenen Agenten (ohne die ANS) die neue Domäne mit tell-domain mit. Den ANS schickt er eine (tell-resource :type address)-Nachricht mit seiner neuen Adresse.

6.4.3 AESInterpreter

Der AESInterpreter stellt den Agent-Event-Server[38] dar und ist somit für das Weiterleiten von Ereignismeldungen an interessierte Agenten zuständig.

Der AESInterpreter erkennt folgende performatives der Ontologie AES:

- subscribe (:type <regulärer Eventausdruck> :filter <reg. Domainfilter>)
- unsubscribe (:type <regulärer Eventausdruck> :filter <reg. Domainfilter>)
- event (:type <Event> [:content <message>])

```
public class AESInterpreter extends AInterpreter {
    protected Hashtable Agents;

    public void interpretMessage(KQMLmessage message, Agent receiver);
    protected void interpretLanguage(KQMLmessage message, Agent receiver,
    Language language);
    public synchronized void run();
    protected void subscribeAction(String event, String filter, String sender);
    protected void unsubscribeAction(String event, String filter, String sender);
    protected void eventAction(KQMLmessage message, String sender);
}
```

Abbildung 6-3 Class AESInterpreter

- **protected Hashtable Agents**

  Die Datenbasis des AES. Diese Hashtable enthält als Schlüssel die eindeutigen Namen (UAL) der Agenten und als Wert eine Hashtable über alle Ereignisse, für die sich ein Agent eingeschrieben hat. Diese Ereignis-

[38] siehe Kapitel 3.6.2, S. 37

Hashtable enthält wiederum pro Ereignis eine Tabelle mit Filterangaben für dieses Ereignis.

Somit ergibt sich folgendes Bild:

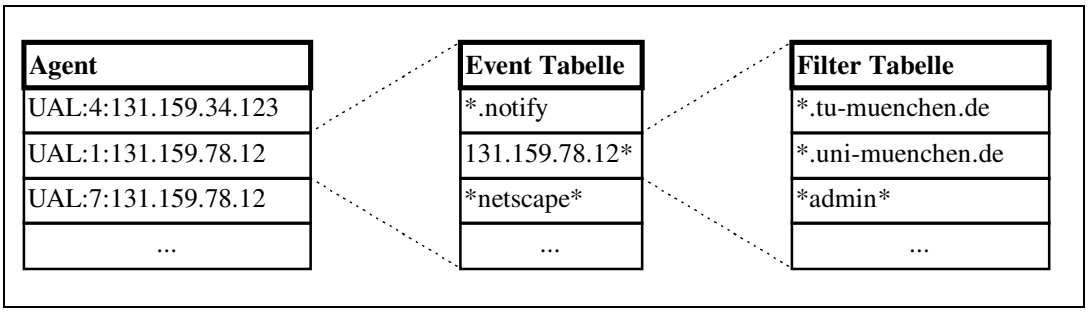


Abbildung 6-4 Event Datenstruktur

- **protected void subscribeAction(String event, String filter, String sender)**

  Diese Methode wird von interpretLanguage aufgerufen, wenn eine subscribe-Nachricht empfangen wurde.

  Als erstes wird überprüft, ob der Absender schon in der Agents-Tabelle eingetragen ist. Wenn nein, wird eine Event-Tabelle kreiert und in die Agents-Tabelle eingetragen.

  Als nächstes wird geprüft, ob dieser Event (bzw. Event-Filter) schon in der Event-Tabelle existiert. Falls nein, wird eine neue Filter-Tabelle erstellt und eingetragen.

  Dann wird der angegebene Filter in die Filter-Tabelle eingetragen.

  Falls der angegebene Filter nicht leer ist, sendet der AES eine subscribe-Nachricht an den AES, auf den die Filterangabe am besten zutrifft. (Beispiel: filter: *.x.y -> AES@x.y)

- **protected void unsubscribeAction(String event, String filter, String sender)**

  unsubscribeAction wird von interpretLanguage aufgerufen, wenn eine unsubscribe-Nachricht empfangen wurde.

  Der angegebene Filter wird aus der Filter-Tabelle des angegebenen Events des Absenders entfernt.

- **protected eventAction(KQMLmessage message, String sender)**

  Diese Methode wird aufgerufen, wenn der AES einen Event empfängt.

  Der AES geht durch die Agents-Tabelle und prüft, ob ein Event-Eintrag eines Agenten auf den angekommenen Event paßt. Falls einer der Filtereinträge auf den Absender der Nachricht zutrifft, wird die Nachricht, so wie

sie ist, an den Agenten weitergeleitet. Dabei kann die Nachricht einen weiteren Inhalt :content enthalten, der das Ereignis näher beschreiben kann.

6.4.4 ARSInterpreter

Der ARSInterpreter ist der Interpreter für den Agent-Resource-Server[39]. Da das JAT-Paket zwischen Classes, Files, Languages und Interpreters unterschieden hat, wurde das hier mit übernommen, obwohl der ARS beliebige Ressourcen verwalten kann.

In der Implementierung des flexiblen Agenten wurde der Vorteil von Java genutzt, Klassen und Dateien direkt über eine URL laden zu können. Daher enthält der ARS nicht die tatsächliche Funktionalitäten, sondern nur Referenzen in Form von URLs darauf.

Der ARS erkennt folgende performatives der Ontologie ARS:

- ask-resource (:type class|interpreter|language|file :name <Identifier>)
- tell-resource (:type class|interpreter|language|file :name <Identifier> :value <URL>)
- invalidate-resource (:type class|interpreter|language|file :name <Identifier>)

Aufgrund des prototypischen Charakters dieser Implementierung unterstützt der ARS in der aktuellen Version keine Domänen, das heißt, momentan gibt es nur einen ARS für das ganze Multi-Agentensystem.

```
public class ARSInterpreter extends AInterpreter {
    protected Hashtable Classes;
    protected Hashtable Interpreters;
    protected Hashtable Languages;
    protected Hashtable Files;

    public void interpretMessage(KQMLmessage message, Agent receiver);
    protected void interpretLanguage(KQMLmessage message, Agent receiver,
    Language language);
    public synchronized void run();
    protected void insertAction(String name, String value, String type, Agent
    parent);
    protected void removeAction(String name, String type, Agent parent);
    protected void askAction(KQMLmessage original, String name, String sender,
    String type, Agent parent);
}
```

Abbildung 6-5 Class ARSInterpreter

[39] siehe Kapitel 3.3.4, S. 22

- **protected Hashtable Classes**

  protected Hashtable Interpreters

  protected Hashtable Languages

  protected Hashtable Files

  Diese Tabellen enthalten die Zuordnung von Ressourcen zu URLs. Daher enthalten die Hashtables als Schlüssel die Resource-Identifier (z. B. Klassennamen, Dateinamen, etc.) und als Werte Objekte der Klasse URL.

- **protected void insertAction(String name, String value, String type, Agent parent)**

  Diese Funktion wird aufgerufen, wenn der Interpreter eine tell-resource Nachricht erhalten hat und fügt die angegebene URL entsprechend des types in die Hashtable zu dem angegebenen Identifier ein.

- **protected void removeAction(String name, String type, Agent parent)**

  Diese Methode trägt die angegebene Ressource aus der entsprechenden Tabelle wieder aus.

- **protected void askAction(KQMLmessage original, String name, String sender, String type, Agent parent)**

  askAction wird aufgerufen, wenn der ARS eine ask-resource Nachricht erhalten hat. Falls die angeforderte Ressource bekannt ist, schickt der ARS die URL zu dieser Ressource in Form einer tell-resource Message zurück, ansonsten ein „?“ als URL.

  In zukünftigen Versionen muß die Methode noch um die Fähigkeit erweitert werden, Ressourcen bei anderen ARS zu suchen, falls sie nicht vorhanden ist.

6.4.5 AKSInterpreter

Der Agent-Key-Server wurde in diesem Prototypen nicht implementiert.

6.5 Beispiel-Managementanwendung

Um die Funktionsweise des flexiblen Agenten erläutern und demonstrieren zu können, wird eine Beispiel-Managementanwendung prototypisch implementiert. Dazu wurde das Szenario „Monitoring und Recovering eines Webserver“ aus Kapitel 2.4.2.3[40] ausgewählt.

Entsprechend dem Szenario, werden die dazu benötigten Interpreter (FRA, FDA, Web, etc.), sowie eine Webserver-Simulation entwickelt. Diese realisieren bzw. simulieren nur die für das Szenario relevanten Funktionen, die hier aber nicht näher besprochen werden sollen.

[40] Seite 8

7 Ausblick

In den folgenden Abschnitten werden einige Möglichkeiten beschrieben, wie der Prototyp erweitert und verbessert werden könnte.

Außerdem werden einige andere Ansätze vorgestellt, die verwendet werden können, um ein Multi-Agentensystem aus flexiblen, kooperierenden Agenten zu realisieren.

7.1 Erweiterung des Prototypen

Der entwickelte flexible Agent stellt nur einen Prototyp dar und muß daher für den realen Einsatz in einer Managementumgebung noch verbessert und erweitert werden.

- **Verschlüsselungsmechanismen**

  In der prototypischen Implementierung wurden alle Sicherheitsmechanismen ausgelassen. Daher muß der MessageHandler noch um Funktionen zur Ver- und Entschlüsselung erweitert werden. Genauso muß auch in der Schnittstelle zu den Interpretern die Methode sendMessage, um einen Parameter erweitert werden, der besagt, ob die Nachricht verschlüsselt, unterschrieben oder im Klartext übertragen werden soll.

  Denkbar wäre, verschiedene Verschlüsselungsalgorithmen genauso wie die Languages, als Ressourcen anzubieten. Somit könnte ein Agent eine unbekannte Verschlüsselungstechnik vom Resource-Server nachladen.

  Zudem muß dann natürlich auch ein Interpreter für den AKS geschrieben werden.

- **weitere Languages**

  Für den tatsächlichen Einsatz des flexiblen Agenten müßten weitere Languages definiert werden. Zumindest eine SNMP Language sollte vorhanden sein, um SNMP-request als Content einer KQML-Nachricht übertragen zu können.

- **weitere Transportprotokolle**

  Weiterhin wäre denkbar, den flexiblen Agenten um weitere Transportprotokolle zu erweitern. So könnte z. B. ein MailInterface für Mails an einen Administrator oder ein CORBAInterface für Nachrichten an CORBA-

Objekte implementiert und zur Verfügung gestellt werden. Möglich wäre auch eine im Agenten eingebaute Schnittstellen zu einem Trouble-Ticket System.

- **Erweiterung des Agent-Resource-Servers**

  Der ARS muß noch um die Fähigkeit erweitert werden, auf verschiedenen Domänen verteilt zu sein. Außerdem müssen noch Sicherheitsmechanismen eingebaut werden (fingerprints, etc.)

7.2 CORBA Mobile Agent Facility[41]

Es wäre zu prüfen, inwieweit das CORBA Mobile Agent Facility [MAF 96] verwendet werden könnte, um den flexiblen Agenten zu realisieren.

Obwohl dieses CORBAfacility explizit mobile Agenten behandelt, definiert es ein Rahmenwerk zu Agenten, das vielleicht verwendet werden könnte. Der mobile Aspekt könnte unter Umständen unberücksichtigt bleiben.

7.3 JMAPI

Nachdem die Entscheidung, den Prototypen mit Hilfe des JAT-Pakets zu entwikkeln, gefallen war, wurde Ende November 1996 von Sun Mircosystems. Inc, die sogenannte „Java Management API“ (JMAPI) herausgegeben. [JMAPI 96]

Thema dieses Rahmenwerkes ist das Management by Delegation im Netz- und Systemmanagement mit Hilfe von Java. Dazu wird hauptsächlich die „Remote Method Invocation“ verwendet, eine Art RPC-Mechanismus.

In der JMAPI sind sogenannte AgentFactories (Appliance) definiert, die ungefähr den hier verwendeten flexiblen Agenten entsprechen, und sogenannte AgentObjects, die in etwa der zu delegierenden Funktionalität gleichen. Diese AgentObjects oder auch die AgentFactories können nicht direkt kommunizieren, sondern nur Methoden der anderen Objekte remote aufrufen. Und auch dies geschieht nicht direkt, sondern über eine ManagedObjectFactory (Admin Runtime Module), die eine Art Proxy für AgentObjects darstellt (vergleichbar mit einem ORB ohne trading Funktionalität). Zusätzlich enthält das Admin Runtime Module, in einer angeschlossenen Datenbank, noch den Code, der zu den AgentFactories delegiert wird.

[41] mehr dazu im Kapitel 5.2.3, S. 65

Außerdem wurde noch ein Eventmechanismus, ähnlich dem hier vorgestellten, definiert, der zwischen EventProducern und EventConsumern unterscheidet. Dabei dient der NotificationDispatcher als eine Art Eventserver.

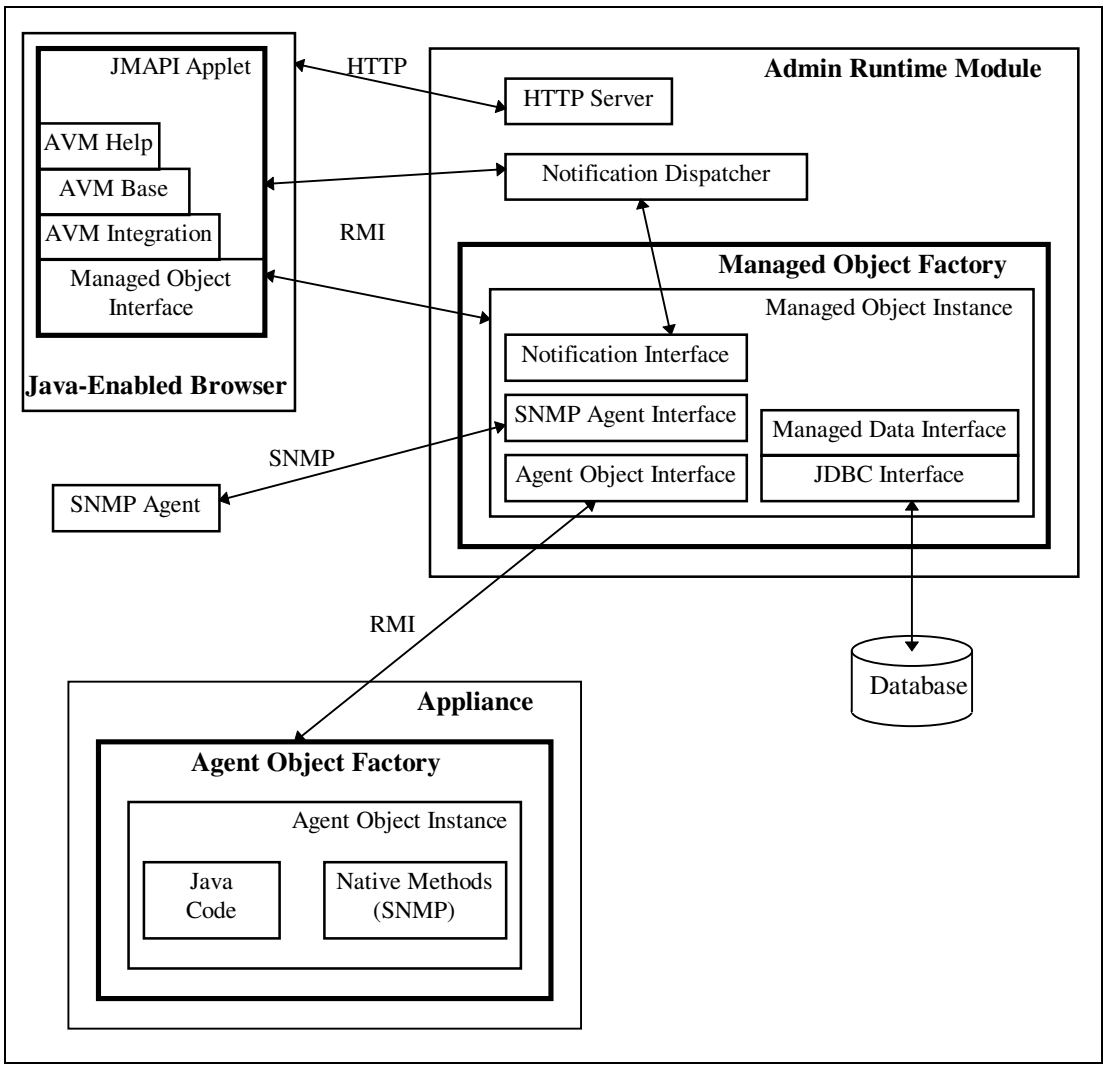


Abbildung 7-1 JMAPI Architektur

Daneben enthält diese API noch umfangreiche Bibliotheken, um diese Objekte über Applets anzeigen und steuern zu können, sowie eine SNMP- und eine Datenbankbank-Bibliothek.

Obwohl die JMAPI nicht für kooperierende Agenten entwickelt wurde, wäre es interessant zu prüfen, ob der enthaltene Eventmechanismus nicht doch zur Kommunikation und Kooperation zwischen den AgentObjects verwendet werden könnte. Auf den ersten Blick sieht es so aus, als ob die JMAPI dazu noch um

einen Agent-Naming-Service erweitert werden müßte. Dazu würde es Änderungen an einigen Teilen der API bedürfen.

Dies müßte jedoch genauer in einer anderen Arbeit untersucht werden.

7.4 mobile Agenten

Mobile Agenten sind Agenten, die von Host zu Host wandern und dort Aktionen ausführen können. Dazu verwenden sie häufig Schnittstellen, die von dem jeweiligen Host angeboten werden, um sich zum Beispiel an die lokalen Datenbank anzuknüpfen.

Interessant wäre zu prüfen, inwiefern mobile Agenten für den Einsatz im Netz- und Systemmanagement eingesetzt werden könnten. Denkbar wäre ein Szenario, in dem ein Agent einen Auftrag bekommt und dann von Host zu Host wandert, um ihn auszuführen.

In diesem Fall wäre das CORBA Mobile Agent Facility sicherlich sehr interessant.

Mögliche Beispiele wären z. B. ein Agent, der von Host zu Host wandert, um dort die Logfiles auszuwerten. Wenn er alle Logfiles ausgewertet hat, kehrt er zur Managementstation zurück, um die Ergebnisse dort zu präsentieren. Oder ein Agent, der von Router zu Router bzw. Rechner wandert, um dort die Filtereinstellung für die Firewall zu aktualisieren.

8 Literaturverzeichnis

[BDM 96] D. Benech, T. Desperats and L. L. Moreau, „A Conceptual Approach to the Integration of Agent Technology in System Management.“ *In Proceedings of the IFIP/IEEE International Workshop on Distributed Systems: Operations & Management, L'Aquilla, Italy*, October 1996

[CaLe 96] J. Case and D. Levi, „SNMP Mid-Level-Management MIB“, Internet Draft, 1994, obsoleted by DISMAN drafts.

[CoMo 96] L. Conradie and M.-A. Mountzia, „A Relational Model for Distributed Systems Monitoring using Flexible Agents“, *In Proceedings of the Third International Workshop on Services in Distributed and Networked Enviroments (ISDNE 96), Macau,* 1996

[DFW 96] M. Daniele, B. Wijnen and D. Franciso, „Agent Extensibility (AgentX) Protocol“, Internet Draft, IAB, November 1996, Work in progress

[Fl 96] D. Flanagen. *Java in a Nutshell.* O'Reilly & Associates, Inc, 1996

[FMF 92] T. Finin, D. McKay, R. Fritzson. „An Overview of KQML: A Knowledge Query and Manipulation Language.“ University of Maryland, Baltimore County, 1992

[Go 95] G. Goldzmidt. „On Distributed System Management.“ Distributed Computing and Communication Lab, Computer Science Department, Columbia University, 1996

[Go 96] Germán Goldzmidt, „Distributed Management by Delegation“, PhD thesis, Columbia University, 1996

[GoYe 95] G. Goldzmidt and Y. Yemini. „Decentralizing Control and Intelligence in Network Management.“ *In Proceedings of the 4th International Symposium on Integrated network Management*, Santa Barbara, CA, 1995

[Gr 95] R. S. Gray, „agent Tcl: a transportable agent system“, *In Proceedings of the ACM CIKM Intelligent Information Agents Workshop,* Dezember 1995

[HeAb 94] H.-G. Hegering and S. Abeck, *Integrated Network and System Management* Addison-Wesley, 1994

[Ho 97] Alexander Hollerith, „Entwurf einer Architektur für flexible Agenten auf der Basis des Konzepts von Management by Delegation.“, Diplomarbeit, Technische Universität München, Februar 1997

[JAT 96] H. R. Frost, *The Java™ Agent Template v0.3*, 1996 http://cdr.stanford.edu/ABE/JavaAgent.html

[Java 95] Sun Microsystems, Inc. *The Java Language Enviromente: A White Paper,* 1995

[JMAPI 96] Sun Microsystems, Inc. *Java Management Application Programming Interface,* 1996 http://java.sun.com/products/JavaManagement/

[JMLP 92] N. R. Jennings, E. H. Mamdani, I. Laresgoiti, J. Perez and J. Corera, „GRATE: A General Framework for Cooerative Prolem Solving“, *IEE-BCS Journal of Intelligent Systems Engineering, 1(2), 1992*

[Ko 95] E. Kovács. „Trader Cutsomization Using Mobile Agents“. University of Stuttgart, 1995

[KQML 93] DARPA Knowledge Sharing Initiative, External Interface Working Group. *Specification of the KQML Agent-Communication Language*, draft specification, 1993

[La 96] Y. Labrou. *Semantics for an Agent Communication Language.* PhD thesis, University of Maryland, 1996

[LaFi 94] Y. Labrou and T. Finin. A semantics approach for KQML - a genreral purpose communication language for software agents. *In Proceedings of the Third International Conference on Information and Knowledge Management,* November 1994

[LeSe 96] D. Levi and J. Schönwalder, „Script MIB: Definition of Managed Objects for the Delegation of Management Scripts“, Internet Draft, IAB, November 1996, Work in progress.

[MaEc1996] T. Magedanz, T.Eckardt. *Mobile Software Agent: A new Paradigm for Telecommuniacations Management.* GMD FOKUS / Technical University of Berlin, Berlin, 1996

[MAF 96] GMD Fokus, International Buisness Machines Corporation, The Open Group. *Mobile Agent Facility Specification.* OMG TC Document cf/96-12-01, 1996

[MLF] J. Mayfield, Y. Labrou, T. Finin. *Evluation of KQML as an Agent Communication Language.* University of Maryland, Baltimore County

[MoDr 96] M.-A. Mountzia and G. Dreo-Rodesek. „Delegation of Functionality: Aspects and Requirements on Management Architectures.“ *In Proceedings of the IFIP/IEEE International Workshop on Distributed Systems: Operation & Management, L'Aquilla, Italy, October 1996*

[MoDr 97] M.-A. Mountzia and G. Dreo-Rodesek. „Using the Concepts of Intelligent Agents in Fault Management of Distributed Systems.“, *submitted to Journal of Network and Systems Management*, 1997

[OMG 91] Digital Equipment Corporation, Hewlett-Packard Company, HyperDesk Corporation, NCR Corporation, Object Desing, Inc, SunSoft, Inc. *The Common Object Request Broker: Architekture and Specification, OMG Document Number 91.12.1, Revision 1.1, 1991*

[RFC 1157] J. D. Case, M. Feder, M. L. Schoffstall and C. Davin, „Simple Network Management Protocol (SNMP)“, RFC 1157, IAB, May 1990.

[RFC 1271] S. Waldbusser, „Remote Network Monitoring Management Information Base“, RFC 1271, IAB, November 1991.

[RFC 1451] J. Case, K. McClogherie, M. Rose and S. Waldbusser, „Manager-to-Manager Management Information Base“, RFC 1451, IAB, April 1993.

[RFC 1757] S. Waldbusser, „Remte Network Monitoring Management Information Base“, RFC 1757, IAB, February 1995.

[RFC 1902] J. Case, K. McClogherie, M. Rose and S. Waldbusser, „Structure of Management Information for version 2 of the Simple Network Management Protocol (SNMPv2)“, RFC 1902, IAB, January 1996.

[Schö 95] Jürgen Schönwalder. „Distributed Network Management - Aproaches and Problems“, TU Braunschweig, 1995

[Schö 96a] Jürgen Schönwalder. „Experiences with the Script MIB“. University of Twente, Enschede - The Netherlands, 1996

[Schö 96b] Jürgen Schönwalder. „Using Mutlicast-SNMP to Coordinate Distributed Management Agents.“, *In Proceedings of the 2nd International IEEE Workshop on Systems Management*, Toronto, Ontario, pages 136-141, June 1996.

[So96] Fergal Somers. „*HYBRID: Unifiying Centralised and Distributed Network Management using Intelligent Agents“,* Broadcom Eireann Research Ltd, Dublin, 1996

[Ta 88] Andrew S. Tanenbaum. *Computer Networks.* Prentice-Hall, 1988

[TFM 95] C. Thirunavukkarasu, T. Finin, J. Mayfield. „Secret Agents - A Security Architecture for the KQML Agent Language“, *In Proceedings of the ACM CIKM Intelligent Information Agents Workshop,* Dezember, 1995

[WeHo 96] M. Wennrich, A. Hollerith, „Flexible Agenten am Beispiel Verfügbarkeit“, Fortgeschrittenen-Praktikum, Technische Universität München, 1996

[YeSi 96] Y. Yemini and S. da Silva, „Towards programmable Networks*“, In Proceedings of the IFIP/IEEE International Workshop on Distributed Systems: Operations & Management*, October 1996