INSTITUT FÜR INFORMATIK
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Diplomarbeit

Einsatz des Java Dynamic Management KIT (JDMK) zur Antwortzeitüberwachung bei der DeTeSystem

Bearbeiter:	Michael Hojnacki
Aufgabensteller:	Prof. Dr. Heinz-Gerd Hegering
Betreuer:	Helmut Reiser Holger Schmidt Rainer Hauck
Externer Betreuer:	Helmut Dürr (DeTeSystem)

INSTITUT FÜR INFORMATIK
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Diplomarbeit

Einsatz des Java Dynamic Management KIT (JDMK) zur Antwortzeitüberwachung bei der DeTeSystem

Bearbeiter:	Michael Hojnacki
Aufgabensteller:	Prof. Dr. Heinz-Gerd Hegering
Betreuer:	Helmut Reiser Rainer Hauck Holger Schmidt
Externer Betreuer:	Helmut Dürr (DeTeSystem)
Abgabetermin:	15. Mai 1999

Hiermit versichere ich, daß ich die vorliegende Diplomarbeit selbständig verfaßt und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 15. Mai 1999

. . . .
(Unterschrift des Kandidaten)

Zusammenfassung

Für die Wertschöpfung in Unternehmen ist die IT (Information Technology) von größter Bedeutung, da sie eine entscheidende Rolle in Bezug auf wirtschaftlichen Erfolg oder Mißerfolg spielt. Marktbedingter Wandel der Unternehmen wie z.B. die Änderung der Produktpalette oder die Gründung neuer Standorte haben zwangsweise eine Veränderung der Infrastruktur der IT und somit des Corporate Network (CN) zur Folge.. Dieser Zusammenhang stellt eine hohe Anforderung an das Netzmanagement dar, das zudem auch unter wirtschaftlichen Aspekten bewertet werden muß.

Die wachsende Komplexität und die strategische Forderung der Unternehmen nach Konzentration auf die Kernkompetenz hat in steigendem Maße zur Folge, daß externe Firmen mit der Bereitstellung und Pflege des Netzwerkes beauftragt werden. Die DeTeSystem GmbH ist ein Anbieter solcher Gesamtlösungen, die sie entwickelt, realisiert und betreibt. Das Anwendungsszenario, das in dieser Arbeit betrachtet wird, repräsentiert die typische Struktur eines solchen Kundennetzes.

Da die technische Sicht nicht mehr ausreicht, um die IT–Systeme eines Unternehmens zu steuern, entfernt man sich immer mehr von der Spezifikation hard– und softwarebezogener Parameter und definiert in sogenannten Service–Level–Agreements Dienste und deren Qualität. Ziel des Ansatzes ist u.a. die Objektivierung von Leistungsmerkmalen aus der entscheidenden Sicht des Kunden, welche als Grundlage für die Planung, den Abschluß und die Überwachung von IT–Services dienen.

Die Analyse des Anwendungsszenarios zeigt, daß der Nachweis dieser Leistungen, der durch die DeTeSystem gewährleistet werden muß, aufgrund der zentralisierten Management–Architektur nur unzureichend oder in Näherung erbracht werden kann. Dies ist insbesondere in solchen Netzen der Fall, in denen Filialen oder Händlerlokationen mittels ISDN–Wählverbindungen an das CN angebunden sind, da hier selbst einfachste Polling–Mechanismen nicht greifen.

Eine Erweiterung dieser Problematik ergibt sich bei der Serviceüberwachung für verteilte und transaktionsbasierte Anwendungen. Vertraglich vereinbarte Antwortzeiten von Applikation und Server–Systemen erfordern eine geeignete Infrastruktur zur Überwachung und Fortschreibung der Service Level, da im Falle einer Abweichung die DeTeSystem als Anbieter in Regreß genommen wird.

Die Betrachtung einer Auswahl am Markt verfügbarer Management–Applikationen zeigt, daß diese Problematik durch die Software–Häuser zwar erkannt wurde, eine brauchbare Lösung allerdings nicht verfügbar ist. Zur Überwachung von Applikationen existieren einige sehr gute Entwicklungswerkzeuge, unter denen insbesondere die Application Responsetime Measurement API der Computer Measurement Group (CMG) auffällt.

Einen mächtigen Ansatz zur Lösung der oben beschriebenen Problematik bieten verteilte Management–Systeme, die Management–Intelligenz dort plazieren, wo sie benötigt wird. Jedoch ist Verteilung allein nicht ausreichend, die notwendige Flexibilität für sich permanent wandelnde Netze bereitzustellen. Die Theorie der flexiblen Management–Agenten (FMA) bietet einen Ansatz, der dieser Anforderung entspricht. Die Delegierbarkeit von Funktionalitäten und die dynamische Veränderbarkeit der FMAs erweisen sich in der Realität komplexer Anwendungsszenarios als wirksame Werkzeuge für ein effizientes Netzmanagement. Das Java Dynamic Management Kit der Firma Sun Microsystems Inc., dient als Entwicklungswerkzeug für solche Agentensysteme. Ziel dieser API ist die Entfernung vom zentralisierten, statisch limitierten Netzmanagement und die Realisierung eines „Service Driven Network“. Obwohl das JDMK an einigen Stellen noch erhebliche Mängel hat, erweist es sich als brauchbares Werkzeug.

Kern dieser Arbeit stellt die Entwicklung und Darstellung eines Konzeptes zur Implementierung einer verteilten Management Architektur mit FMAs unter Berücksichtigung der speziellen Infrastruktur der DeTeSystem dar. Hierbei wird insbesondere eine Lösung zur Messung von Antwortzeiten aus Lokationen, die über eine ISDN–Wählverbindung an ein Extranet angebunden sind, erarbeitet. Die

Diskussion unterschiedlicher Optionen ergibt eine wohlbegründete Architektur der einzelnen Komponenten des dazu notwendigenen Agentensystems, deren Umsetzbarkeit mit den verfügbaren Mitteln in Form einer prototypischen Implementierung gezeigt wird.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einführung

1.1 Motivation

Der Bedarf an Datenkommunikation besteht bereits seit den frühsten Tagen der Computerentwicklung. Heute spielt sie bereits eine derart zentrale Rolle, daß der wirtschaftliche Erfolg großer wie kleiner Unternehmen von der zugrundeliegenden Kommunikationsinfrastruktur in zunehmenden Maße abhängt. Fehler und Leistungsmängel in diesen Systemen haben in der Regel finanzielle Verluste zur Folge. Aus dieser engen Abhängigkeit läßt sich der Anspruch ableiten, die Leistungsfähigkeit der zugrunde liegenden Kommunikations– und Daten–Infrastruktur im Rahmen der benötigten und angestrebten Quality of Services sicherzustellen. Daher ist das Management des Netzwerkes, dessen Komponenten, des Systems und der Applikationen unbedingt erforderlich.

> *Unter Netz–Management versteht man die Gesamtheit der Vorkehrungen und Aktivitäten zur Sicherstellung eines effektiven und effizienten Einsatzes von verteilten Prozessen und Ressourcen, die je nach Managementziel ein Kommunikationsnetz oder ein verteiltes System darstellen können [HeAb 93].*

Die Komplexität dieser Managementaufgabe hängt direkt von der Komplexität des zu managenden Systems ab. Diese ist unter anderem bestimmt [HeAb 93]:

- durch die Anzahl und Typenvielfalt der zu managenden Komponenten.
- von der Heterogenität der Systeme in bezug auf System–Software, Schnittstellen, Protokolle samt ihrer Profile und Versionen.
- von der räumlichen Verteilung der Komponenten.
- von der Anzahl der beteiligten Organisationen bzw. Verantwortungsbereiche.
- vom Grad der Diensteintegration und Teilnetzbildung.
- von der Anzahl und Vielfalt unterstützter Netzdienste und verteilter Anwendungen.

Diese wachsende Komplexität und die strategische Forderung der Unternehmen nach Konzentration auf ihre Kernkompetenzen haben in steigendem Maße zur Folge, daß die Bereitstellung der benötigten Infrastruktur und das damit verbundene Netz– und Systemmanagement an externe Anbieter vergeben wird.
Die DeTeSystem GmbH ist ein Anbieter von solchen Kommunikations– und Systemlösungen für große Unternehmen, deren Bedürfnissen ein zentralisiertes Management großer Netzwerkumgebungen nicht gerecht wird. Zentral konzentrierte Management–Intelligenz bindet zuviele Ressourcen im Netz und kann aus technischen Gründen nicht alle gewünschten Parameter liefern.

Auch die Art der vertraglich vereinbarten Parameter wandelt sich immer mehr von reinen Leistungsbeschreibungen einzelner Komponenten zu dienstorientierten Gesamtlösungen. Es werden immer seltener Teile eines Systems wie Datendurchsatz einer Verbindung oder Festplattenkapazitäten eines Datei–Servers definiert, da diese hard– und softwareorientierte Sicht nicht mehr genügt, um die hochkomplexen IT–Systeme eines Unternehmens zu steuern. Stattdessen findet eine kontinuierliche Trennung von der technischen Sicht der Geräte statt. Sogenannte Service–Level (SL) beschreiben die tatsächliche Leistung eines IT–Systems aus der Sicht des Kunden, so z.B. die maximale Dauer einer komplexen Transaktion.
Insbesondere bei der Überwachung dieser im Rahmen eines Service Level Agreements (SLA) vereinbarten Leistungen sind nur zeitweise per ISDN angebundene Subnetze nicht permanent in einem finanziell sinnvollen Rahmen erfaßbar. Eine Lösung bieten flexible Management Agenten (FMA), die die benötigten Informationen sammeln, auswerten und sobald möglich an eine zentrale Managementplattform übertragen.
Aus der oben bereits dargestellten Problematik des Netzmanagements können zwei Hauptanforderungen an diese Lösung gestellt werden:

1. Verteilung der Managementfunktionalität

2. Flexibilität bei sich ändernden Anforderungen

Das in Abschnitt 3.1 beschriebene Konzept der flexiblen Management Agenten erfüllt diese beiden Ansprüche.

1.2 Aufgabenstellung

Die Aufgabenstellung ist, mit Hilfe des Java Dynamic Management Kit (JDMK) ein Modell zur Realisierung einer Antwortzeitmessung aus Lokationen, die über eine ISDN–Wählverbindung an ein Extranet angbunden sind, zu entwickeln und prototypisch zu implementieren.
Um dieses umzusetzen wird in einem ersten Schritt die Infrastruktur der DeTeSystem GmbH und anhand einer typischen Kundenlösung ein Anwendungsszenario dargestellt. Mit diesem Szenario ist ein Anforderungskatalog zusammenzustellen und bestehende Ansätze, die sowohl komplette Management–Applikationen als auch Entwicklungswerkzeuge umfassen, auf ihre Anwendbarkeit zu prüfen.
Kern der Arbeit stellt die Entwicklung einer Management–Architektur mit dem JDMK und dem Konzept der flexiblen Management–Agenten (FMA) dar, die abschließend prototypisch implementiert werden soll. Dabei ist die Anwendbarkeit des JDMK auf das Szenario der DeTeSystem zu prüfen.

1.3 Gliederungsbeschreibung

Die Diplomarbeit gliedert sich in folgende Kapitel :

Kapitel 2 — Service Level Management bei der DeTeSystem

In diesem Kapitel wird über eine allgemeine Darstellung der DeTeSystem GmbH ein Anwendungsszenario, das die relevanten Charakteristika eines typischen durch die DTS realisierten Kundennetzes beinhaltet, herausgearbeitet. Die Einführung des Begriffes Service Level Management leitet über zur Zusammenstellung der Anforderungen an die Lösung. Insbesondere wird in diesem Kapitel auf die Messung der Antwortzeit und die Überwachung einer ISDN–Verbindung eingegangen. Eine ausführliche Betrachtung der Sicherheitsaspekte eines Agentensystems rundet dieses Kapitel ab.

Kapitel 3 — State–of–the–Art

Hier werden die theoretischen Grundlagen dieser Arbeit und existierende kommerzielle und nicht–kommerzielle Ansätze zur Antwortzeitmessung dargestellt.
Einführend wird die Theorie der flexiblen Management–Agenten erläutert und die Komponenten und Dienste des JDMK als Realisierungsgrundlage dargestellt. Die Beschreibung unterschiedlicher Management–Applikationen und Entwicklungswerkzeuge und deren Bewertung schließen dieses Kapitel ab.

Kapitel 4 — Konzept eines SLM unter JDMK

Im Kapitel 4 wird eine begründete Auswahl aus den im vorhergehenden Kapitel beschriebenen Werkzeugen zur Erstellung eines Konzeptes zum Service–Level–Management im Anwendungsszenario der DeTeSystem getroffen. Anschließend wird eine konkrete Managementumgebung für die gegebene Infrastruktur entworfen und ein Konzept für die Realisierung der wichtigsten Anforderungen vorgeschlagen.

Kapitel 5 — Implementierung eines Prototypen

Basierend auf dem in Kapitel 4 entworfenen Konzept wird nun die Implementierung der einzelnen Software–Komponenten beschrieben. Dabei wird insbesondere auf die Eigenarten der JDMK–API eingegangen. Abschließend werden im Prototypen nicht genutzte Dienste des JDMK auf Möglichkeiten der Anwendung in diesem Szenario untersucht.

Kapitel 6 — Zusammenfassung und Ausblick

Im abschließenden Kapitel werden die Ergebnisse dieser Arbeit kurz zusammengefaßt und ein Ausblick auf künftige Entwicklungen gemacht.

Kapitel 2

Service Level Management (SLM) bei der DeTeSystem

Aus der Beschreibung der Infrastruktur der DeTeSystem, des Anwendungsszenarios und den neuen, aus Service–Level–Agreements resultierenden Anforderungen werden in diesem Kapitel die Anforderungen an eine Management–Architektur hergeleitet. Abschließend wird detailliert auf die Angriffsmöglichkeiten auf eine Agentensystem eingegangen.

2.1 Die DeTeSystem

Die DeTeSystem betreut als hundertprozentige Tochter der Deutschen Telekom AG mehr als 700 Systemkunden. Ihr Geschäftszweck ist das Angebot individueller Systemlösungen für die größten Kunden der Deutschen Telekom AG. Diese Kunden, zu denen Banken, Versicherungen, Industrie– und Handelsunternehmen gehören, haben höchste Anforderungen hinsichtlich Sicherheit und Verfügbarkeit. Die DeTeSystem tritt hier in der Regel als Anbieter für Komplettlösungen (Systemtechnik und -betrieb) auf, der diese kundenspezifisch entwickelt, realisiert und betreibt.

2.1.1 Infrastrukturkonzept der DeTeSystem

Die in diesem Abschnitt beschriebenen Informationen sind im wesentlichen [Duerr1] und [Duerr2] entnommen.
Das Infrastrukturkonzept der DeTeSystem basiert auf einem ausfallsicheren Server-Rechenzentrum (SRZ), regionalen, kundennahen System-Management-Centren (SMC), einem zentralen SMC für den 24-Stunden-Betrieb, und einem Betriebszentrum für die zentralisierte Systemadministration und die zentrale Sicherheits– und Qualitätsüberwachung. Darauf aufbauend dient die Plattform für Management–Applikationen (PMA) als Anwendungsplattform für zentrale Management–Applikationen wie Event–Management, Reporting– und Statistikfunktionen, Service–Level–Management, etc.
Diese betrieblichen Einheiten werden über das sogenannte SMC–Netz verbunden, das einerseits auf dem Corporate Network der DeTeSystem (DTS-CN) basiert und andererseits die Technik der virtuellen LANs (VLAN) nutzt.
Die in Abbildung 2.1 dargestellte Infrastruktur der DeTeSystem besteht aus nachfolgend beschriebenen Hauptkomponenten:

- **Server–Rechenzentrum mit der Plattform für Managementapplikationen**
  Das SRZ ist eine sichere Umgebung für DV– und TK– Systeme und befindet sich in einem strategischen Computer–Zentrum der Telekom. In ihm sind alle wichtigen betriebsunterstützenden

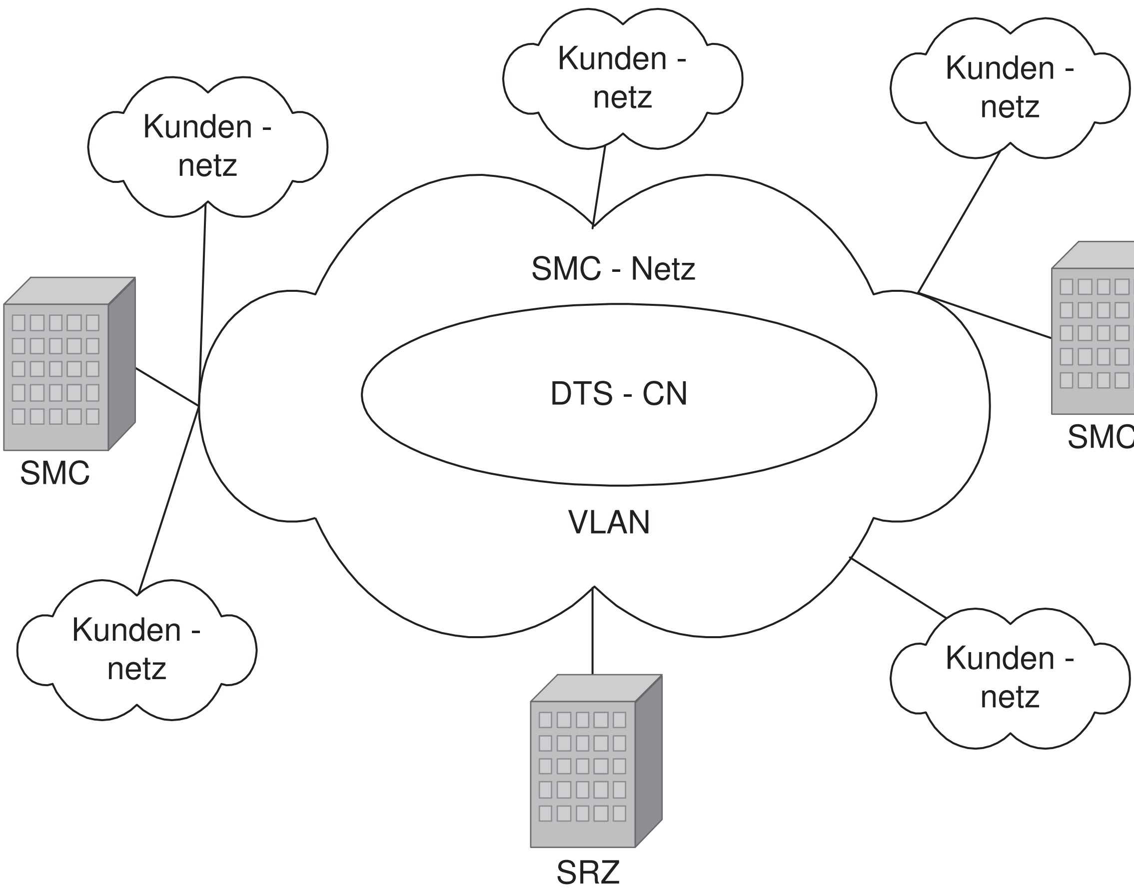

Abbildung 2.1: Das Infrastrukturkonzept der DeTeSystem

Rechner wie Kundennetz-Managementserver, die PMA sowie andere Rechner– und Speichersysteme untergebracht. Aufgabe des SRZ ist das sogenannte Equipment–Housing und Operating, also die Unterbringung und Bedienung der Hardware.
Das SRZ besteht aus zwei durch Brandschutzmauern getrennten, voll redundant ausgelegten Einheiten. Diese ermöglichen im Notfall den Betriebsübergang auf das Ersatzsystem innerhalb einer definierten Zeitspanne.

- **System–Management–Center**
  Kernaufgabe der regional angeordneten SMCs ist der kundennahe Betrieb von Systemlösungen, insbesondere das Netzmanagement der einzelnen Kundennetze.

- **SMC–Netz**
  Die betrieblichen Einheiten der DTS sind durch das DTS-CN mit den Kundennetzen verbunden. Basierend auf der VLAN–Technologie erfolgt eine sichere Trennung der Kundennetze.

- **Kundennetze**
  Da die Komplettlösungen der DTS individuell auf den Kunden zugeschnitten sind, können diese nicht allgemein beschrieben werden. Sie unterscheiden sich in Komplexität, Umfang und Topologie.

Die wichtigste Aufgabe der PMA ist die Verbesserung und Vereinheitlichung der Betriebsqualität und Sicherheit über alle von der DTS betriebenen Systemlösungen hinweg. Durch diese Plattform werden

zentral wichtige Dienste zur Bewältigung von Managementaufgaben bereitgestellt. Dazu gehören z.B. das Service-Level-Management (SLM), das Trouble-Ticket-System (TTS) und die Alarmintegration (PMA-ALI).
Die Management–Applikationen laufen auf eigenen Servern im SRZ. Sie werden auch als Element Management System (EMS) bezeichnet.
Das Netzmanagement an sich wird jedoch in den einzelnen SMCs betrieben, wo über Konsolen der Zugriff auf die jeweiligen Management–Applikationen ermöglicht wird.
Aufbauend auf dieser Infrastruktur ergibt sich nun das Anwendungsszenario dieser Diplomarbeit.

2.1.2 Anwendungsszenario

Obwohl die Realisierungen der Kundenlösungen sich von Fall zu Fall unterscheiden, können doch Parallelen festgestellt werden. Die für diese Arbeit entscheidenden Gemeinsamkeiten werden im folgenden am Beispiel eines Fahrzeugherstellers dargestellt.
Kern des Anwendungsszenarios stellt ein Fahrzeughersteller dar, der ein eigenes Rechenzentrum betreibt, um Händlern die Möglichkeit des Online–Ordering anzubieten. Diese Händler treten dem Fahrzeughersteller gegenüber wiederum als Kunden auf, da sie dieses Online–Ordering gegen Gebühr nutzen.
Die DeTeSystem realisiert und betreibt im Auftrag des Fahrzeugherstellers das dazu notwendige Computernetzwerk.
Die in Abbildung 2.2 modellhaft dargestellte Netzwerktopologie besteht aus folgenden Komponenten:

- **Virtual Private Network**
  Im Auftrag des Kunden betreibt die DTS das Extranet des Fahrzeugherstellers. Dieses Netz dient der Kommunikation im Weitverkehrsbereich und kann zum Beispiel auf der Grundlage der T-InterConnect Plattform der Deutschen Telekom AG basieren. Die Datenströme des Virtuellen Privaten Netzes (VPN) des Fahrzeugherstellers werden logisch von den Datenströmen der anderen Benutzer dieses Netzes getrennt. Diese Trennung wird durch den Einsatz der Tunneling Technik auf IP-Ebene in den aktiven Netzelementen (Routern) der Plattform erreicht.
  Der Zugang zu diesem Netz ist entweder über Standleitungen oder via ISDN möglich. Durch Zugangspunkte, sogenannte Points of Presence (PoP), wird eine Einwahlmöglichkeit angeboten. Die Authentifizierung der Benutzer erfolgt über die ISDN-Rufnummer und einen Kennwort, das mittels Challenge Handshake Protocol (CHAP) übertragen und geprüft wird.

- **Händler**
  Die Händler habe die Möglichkeit, über das Weitverkehrsnetz auf die vom Fahrzeughersteller angebotenen Dienste (z.B. Online Ordering, Internetzugang, ...) zuzugreifen. Hierzu können sie sich mittels Standleitung oder Wählverbindung in dessen Netz anmelden. Realisiert wid dieser Zugang zum VPN mit einem Router.

- **Rechenzentrum des Fahrzeugherstellers**
  In diesem Rechenzentrum werden durch den Fahrzeughersteller sämtliche relevanten Daten und Dienste für dessen Partner angeboten. Insbesondere steht in diesem RZ der für das Online Ordering benötigte Server.

- **Internet Service Area**
  Die Internet Service Area wird durch die DeTeSystem betrieben. Hier wird neben DNS– und Mail–Diensten auch der Zugang in das Internet angeboten.

- **SRZ und SMC der DeTeSystem**
  Das Management dieser Komplettlösungen wird durch die DeTeSystem im SRZ und SMC betrieben.

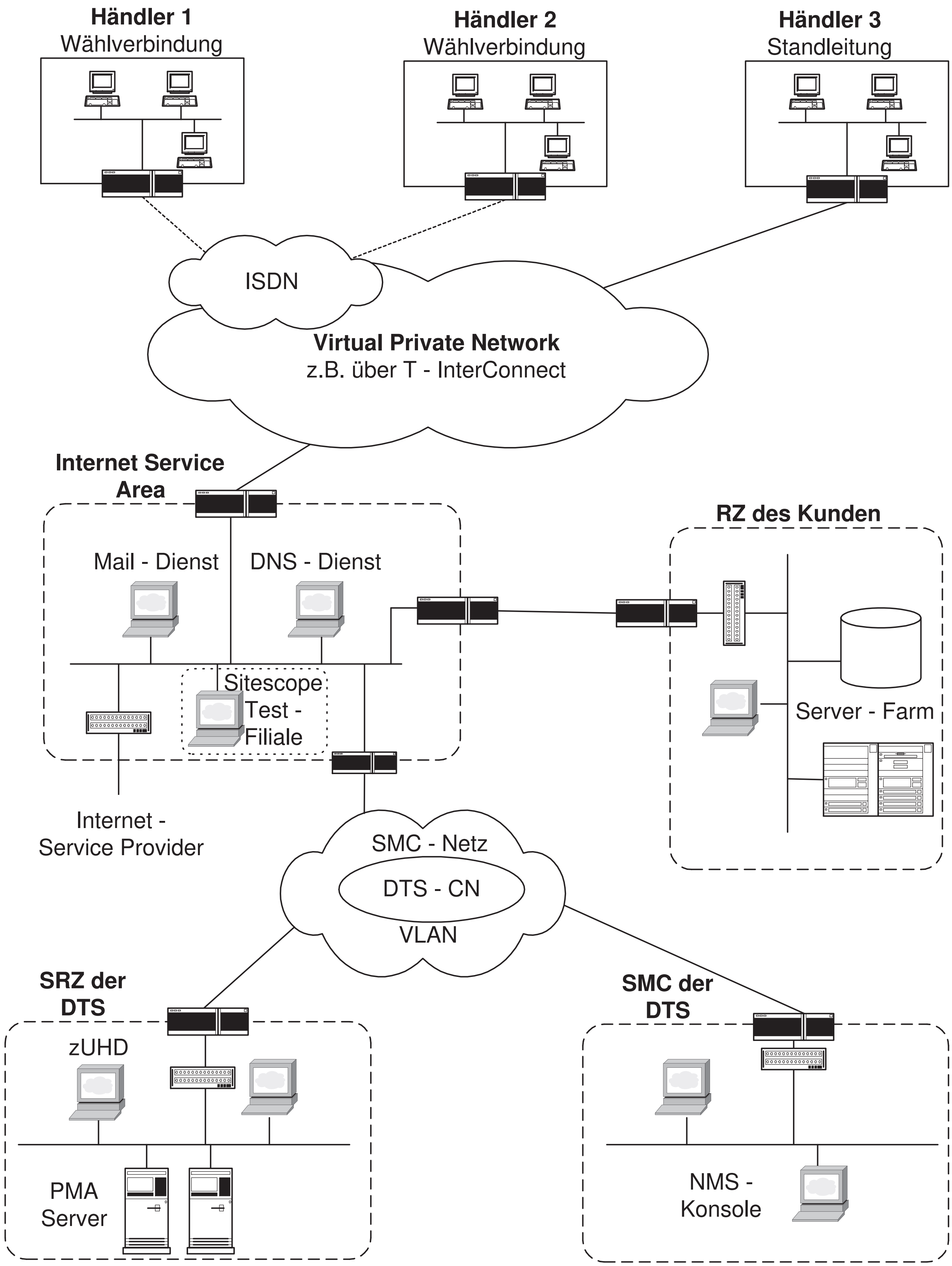

Abbildung 2.2: Die Systemlösung für einen Fahrzeughersteller

- **SMC–Netz**
  Das SMC–Netz verbindet die betrieblichen Einheiten der DeTeSystem.

Es sei an dieser Stelle erwähnt, daß sich für andere Großunternehmen wie Banken, Versicherungen oder Fast–Food–Ketten ein ähnliches Szenario ergeben würde, da es sich in diesen Fällen immer um die Anbindung großflächig verteilter Clients an einen oder mehrere Serverlokationen handelt, die ihrerseits zentrale Dienste anbieten oder Datenhaltung betreiben.

2.2 Service–Level–Management

Die in der Welt der Informationstechnik (IT) in den letzten Jahren zu beobachtende Zunahme von Komplexität und Heterogenität macht eine Planung, Überwachung, Verwaltung und Abrechnung der für diesen Bereich notwendigen Dienste immer schwieriger. Daraus erwuchs in den 90er Jahren der Begriff Service–Level–Management.

2.2.1 Begriff

Grundgedanke dieses Ansatzes ist, daß eine hard– und softwareorientierte Sicht nicht mehr ausreicht, um die hochkomplexen IT–Syteme eines Unternehmens zu steuern.
Service Level Management geht über die technische Sicht solcher System–Parameter hinaus und hilft, durch zusätzliche Service–Definitionen wie z.B. Reaktionszeit bei Störungsmeldung, den neuen Anforderungen gerecht zu werden.
Die Central Computer and Telecommunication Agency (CCTA) definiert SLM wie folgt [ITIL 2] :

> *Service Level Management (SLM) is the process of managing a delivered IT–Service in terms of quality, quantity and cost. ...is formalized by the preparation, agreement and maintenance of formal Service Level Agreements (SLA) which document all the relevant details of an IT service. The Service Level Management can be seen as the bridge between customers and supplier. ...*

Unter einem IT–Service versteht man alle IT–Leistungen, die der Kunde zur erfolgreichen Erledigung seiner Aufgabe (Geschäftsprozesse) benötigt. Es handelt sich hierbei meist um eine Kombination von Hard– und Software sowie Netz– und Telekommunikationsdiensten. Wesentlich ist jedoch, daß sich die Definition ausschließlich aus den Anforderungen des Benutzers oder ganzer Benutzergruppen ergibt.

> *A quality service is a service that lives up to the expectations of costumer. The only possible way to deliver a quality service is by knowing what the customer wants. There is no exception to this. [ITIL 2]*

Ein SLA ist eine schriftliche Vereinbarung zwischen den Kunden, die den geschäftlichen Aspekt eines Unternehmens repräsentiert, und der für die Umsetzung der IT verantwortlichen Institution dieses Unternehmens [McBr 98]. Ziel dieser Vereinbarung ist die Spezifikation dessen, was die unterschiedlichen Benutzergruppen von der IT in Bezug auf Antwortzeit, Systemverfügbarkeit und Prozessquantität von der IT erwarten können [Salm 89]. Solche Benutzergruppen werden meist in Kategorien eingeteilt, die einen unterschiedlichen Service–Level benötigen. Allerdings wird in einem SLA auch definiert, was die IT von dem Kunden z.B. in Bezug auf Systembenutzung erwarten kann.
Durch Service–Level–Agreements entsteht eine neue Schnittstelle zwischen Kunden und Dienstleistern, welche sich nur noch an den vereinbarten Diensten orientiert.
Basierend auf dieser Vereinbarung können beide Parteien ihre Ressourcen planen und einsetzen.
Folgende Komponenten sollte ein SLA enthalten (Auszug) [McBr 98]:

- **Hintergrund**
  Es sollten so viele Hintergrundinformationen dargestellt sein, daß ein nur oberflächlich informierter Leser den Zusammenhang zwischen den ausschlaggebenden Geschäftsindikatoren und der geforderten Dienstgüte versteht.

- **Parteien dieser Vereinbarung**
  Hier werden die Parteien der Vereinbarung und die verantwortlichen Ansprechpartner benannt.

- **Umfang der angebotenen Dienste**
  Dieser Abschnitt quantifiziert den Umfang der angebotenen Dienste. Die Benutzergruppe sollte in der Lage sein, Durchschnitts– und Spitzenwerte zu benennen, sowie Tageszeiten, zu denen diese zu erwarten sind.

- **Zeitfaktoren**
  Hier wird ein qualitativer Aspekt für die Applikationen definiert, und zwar wie schnell der Benutzer mit der Fertigstellung eines Auftrages durch die Applikation rechnen kann (z.B. 90 % der Transaktionen sind innerhalb von 2 Sekunden durchgeführt, Reports sind bis 9 Uhr ausgeliefert, . . .).

- **Verfügbarkeit der Dienste**
  An dieser Stelle des Vertrages muß der Kunde definieren, wann die IT zur Verfügung stehen muß, damit die Arbeit optimal erledigt werden kann.

- **Grenzen der Dienste**
  Dienste können nicht unbegrenzt zur Verfügung gestellt werden. Auslastungsspitzen, Ressourcenbedarf anderer Applikationen und die Gesamtnetzlast setzen der Dienstgüte Grenzen. Diese müssen spezifiziert und abgestimmt werden.

- **Leistungsverrechnung**
  Kostentransparenz und verursachungsgerechte Zuordnung der erbrachten IT–Service–Leistungen sind die Ziele der Leistungsverrechnung. Hier ist insbesondere die Qualität der zugrundeliegenden Basisdaten von ausschlaggebender Bedeutung für die korrekte Verteilung.

- **Messung der Dienste**
  Dieser Abschnitt beschreibt die Prozesse zur Überwachung und Darstellung der entsprechenden Service Level. Hier ist festzustellen, wie diese Dienste überwacht werden und wie häufig Datensammlung vorgenommen werden.

- **Reporting**
  Durch den Service Provider ist in regelmäßigen Abständen durch einen Bericht zu belegen, daß er seinen aus dem SLA hervorgehenden Pflichten nachgekommen ist. Die Form des Berichtes ist vorzugsweise eine permanentes Online–Reporting.

- **Neuverhandlung**
  Da sich die Applikationen in einem permanenten Wandel befinden und sich daraus resultierend die Umgebung verändert, ist es notwendig, daß bereits zu diesem Zeitpunkt eine Neuverhandlung dieser Vereinbarung geplant wird.

Bei der Umsetzung solcher Verträge, deren primäre Aufgabe die Steigerung der gegenseitigen Zufriedenheit und die Erleichterung der Managementaufgaben beider Seiten durch eindeutig spezifizierte Anforderungen darstellt, sind entsprechende Tools zur Überwachung der einzelnen Parameter unerläßlich. Im folgenden Abschnitt werden einige Parameter untersucht, die in solchen Verträgen definiert werden.

2.2.2 Relevante Kenngrößen

Relevante Kenngrößen im Sinne des SLM sind die IT–Leistungen, die ein Anwender benötigt, um seine Aufgabe erfolgreich bewältigen zu können. Diese sind in der Regel eine Kombination aus Netzwerk–, System– und Telekommunikationsdiensten sowie verschiedener Hard– und Softwarekomponenten, welche in dieser speziellen Form für einen Geschäftsprozeß notwendig sind [HASS 1]. Einige dieser Parameter sind in der Tabelle 2.1 aufgeführt.

Bei der in Abbildung 2.2 dargestellten Infrastruktur ist es nur bedingt möglich, solche vertraglich

Begriff	Definition	Bemerkung
Antwortzeit	Zeitintervall zwischen dem Absenden einer Nachricht und dem Empfang der entsprechenden Antwort	s. Netzdurchlaufzeit
Anrufwartezeit	Wartezeit eines Anrufers beim Helpdesk, bevor sein Anruf von einem Mitarbeiter angenommen wird	sollten weniger als 60 Sekunden sein
Funktionszeit	Zeitfenster innerhalb dessen vereinbarte System– und Netz-Services erbracht werden	
Netzdurchlaufzeit	Zeitintervall zwischen dem Absenden einer Nachricht und dem Empfang der Nachricht an Gegenstelle	Kann. u.U. mit Verwendung von PING gemessen werden
Netzmanagement	Definition der Art des Netzmanagement	aktiv, reaktiv, proaktiv, aktive Überwachung
Statistiken	für Statistiken sollte festgelegt werden: Übergabemedium, Berichtszeitraum, Beobachtungsgegenstand, Beobachtungskriterien, Statistikart, Historienlebensdauer	
Verfügbarkeit	gibt das Verhältnis von Zeiten der Verfügbarkeit zu Zeiten der Nicht-Verfügbarkeit einer Resource in Prozent bezogen auf einen definierten Zeitraum	
Dauer des ISDN-Verbindungsaufbaus	Zeitraum zwischen Verbindungswunsch und Übertragung der ersten Benutzerdaten	

Tabelle 2.1: Auszug aus den Betriebsparameter der DeTeSystem nach [Duerr2]

vereinbarten Dienstgüteparameter zu überwachen, da dazu Meß– und Überwachungsmechanismen bei den Händlern vorhanden sein müssen. Dies ist absolut notwendig, da die DeTeSystem für Abweichungen in Regreß genommen werden kann und anderseits in der Lage sein muß, vereinbartes Benutzerverhalten überwachen zu können.

2.3 SLM im Anwendungsszenario der DTS

In diesem Abschnitt wird nun die Frage behandelt, ob ein den Anforderungen der Service–Level–Agreements entsprechende Management–Architektur bei der DeTeSystem existiert, und wenn nicht, welche zusätzlichen Anforderungen erfüllt werden müssen.
Im Mittelpunkt der Betrachtung steht die zu erbringende und zu überwachende IT–Serviceleistung, wie z.B. Verfügbarkeiten, Antwortzeiten oder Transaktionszeiten aus dem Blickpunkt des Dienstnehmers, in unserem Anwendungsszenario also des Händlers. Nur so wird eine objektive Beurteilung von Leistungen möglich, wie sie heute schon in SLAs vereinbart, aufgrund von server–orientierten Ansätzen wie z.B. mittels Sitescope jedoch oftmals nur unzureichend oder in Näherung belegbar sind.
Wie in Abbildung 2.2 zu sehen ist, stellt sich die Situation derzeit so dar: Die DeTeSystem verfügt über Management–Applikationen in ihren SMCs (bzw. im SRZ) und in der ISA. Für die Überwachung von Service–Level–Parametern ist insbesondere eine in der ISA platzierte und mit Sitescope ausgestattete Testfiliale verantwortlich. Durch eine genaue Betrachtung der Architektur läßt sich schnell herausfinden, daß die hier gemessenen Werte nicht auf die Händlerlokationen übertragbar sind. Vielmehr können sie im besten Fall eine Annäherung an die dort gegebene Situation darstellen. Auch können Polling–Mechanismen in die Händlerlokationen kein realistisches Bild darstellen, da sie nur Momentaufnahmen sind und komplexe IT–Leistungen nicht messen können. Zudem sind sie finanziell auch nicht tragbar, da jedesmal eine ISDN–Verbindung aufgebaut werden muß.
Alle Händler sind über ein Weitverkehrsnetz an das VPN des Fahrzeugherstellers angebunden, das nicht dem direkten Zugriff der DeTeSystem unterliegt. Somit sind Störungen wie z.B. Überlastung der PoPs, Ausfälle von Routern o.ä. nicht auszuschließen. Zudem erhöhen sich z.B. Zugriffs– und Antwortzeiten der Dienste der Internet–Service–Area durch die Übertragung via T–InterConnect.
Im Rahmen dieser Arbeit ist eine Managementarchitektur mit Hilfe des JDMK zu entwickeln, die eine Überwachung und Dokumentation der relevanten Dienstgüteparameter in den Händlerlokationen ermöglicht.
Dazu müssen zwei Szenarien unterschieden werden:

1. Standverbindung zwischen Händlerlokation und PoP des Weitverkehrsnetzes

2. Wählverbindung zwischen Händlerlokation und PoP des Weitverkehrsnetzes

In beiden Szenarien ist es notwendig, Managementsoftware auf der Händlerseite zu installieren, da nur so netzwerkrelevante Daten von End–to–End Kommunikationsbeziehungen gemessen werden können. Auf diese Weise kann die unzureichende Messung der oben erwähnten Sitescope–Testfiliale sinnvoll ergänzt werden.
Dies hat neben der eben beschriebenen Notwendigkeit auch viele Vorteile. Einige sind im folgenden aufgeführt:

- Entlastung der zentralen Management–Applikationen

- Geringere Belastung des Netzes und des Managers durch Filterfunktionalität

- Weniger Netzlast in der Nähe der Management–Applikation durch Dezentralisierung der Pollingmechanismen

- Warnmeldung bei drohender Abweichung ermöglicht zeitgerechte Reaktion (proaktives Management)

Im folgenden werden die zentrale Anforderungen an eine Agenten–Architektur im Anwendungsszenario vertieft dargelegt und abschließend klassifiziert.

2.3.1 Anforderungen an die Lösung

Aus technischen, organisatorischen, betriebswirtschaftlichen, funktionellen und sicherheitstechnischen Gründen ergeben sich unterschiedliche Anforderungen an die zu erarbeitende Lösung.
Heterogene Netze erfordern unter *technischen* Gesichtspunkten ein plattformunabhängiges Agentensystem, das in den unterschiedlichsten Umgebungen agieren kann. Da die Agentenplattform auf einem auch für andere Zwecke genutzten Rechner in Händlerlokationen platziert werden soll, ist es notwendig, deren Ressourcenbedarf in Bezug auf Speicherbedarf, Prozessorbeanspruchung und Kommunikationsintensität auf ein Minimum zu reduzieren. Zudem müssen aus einer Protokolldatei (Log–File) ständig mögliche Fehlerursachen z.B. für Meß– oder Verbindungsabbrüche auslesbar sein. Ferner haben Kommunikationsprotokolle zum Informationsaustausch unter den Agenten bzw. zwischen Agenten und dem Manager einem Standard zu entsprechen und leicht austauschbar zu sein.
Der wichtigste *organisatorische* Aspekt, der bei der Implementierung beachtet werden muß, ist die Strukturierung der Agenten. Hier muß die Bildung von Gruppen und Domänen nah an die logische Struktur des Netzes bzw. des Unternehmens angelehnt sein, um Managementfunktionen zu erleichtern. Da sich Anforderungen an das IT–System in direkter Abhängigkeit von Marktveränderungen schnell anpassen lassen müssen, um so die Wettbewerbsfähigkeit eines Unternehmens zu sichern, ist es auch notwendig, die Managementarchitekturen so zu gestalten, daß sie diesem Wandel schnell folgen können. Daraus resultiert für das Agentensystem, daß es dynamisch veränderbar sein muß.
Jedoch nutzt die beste Managementlösung nichts, wenn sie aus *betriebswirtschaftlichen* Gründen nicht tragbar ist. Es sei darauf hingewiesen, daß im Rahmen dieser Arbeit keine Analyse von Lizenzkosten o.ä. vorgenommen wird. Allgemein sind zuerst die Kosten der Einführung eines solchen Konzepts zu untersuchen. In eine bestehende Systemlösung sind die Komponenten eines Agentensystems zu verteilen. Dies sollte ohne großen zeitlichen und personellen Aufwand durchführbar sein. Auszuschließen ist bei der hohen Anzahl an Händlerlokationen die manuelle Installation von Agentenplattformen vor Ort. Vielmehr gilt es automatisierte Mechanismen zu entwickeln, die diese Verteilung über das Netz ermöglichen. Während des Betriebes dieser Managementlösung dürfen für den Kunden keine zusätzlichen Kosten auftreten. Dies bedeutet im Detail, daß die Leistungsfähigkeit des Gesamtsytems nicht spürbar durch das Betreiben des Agentensystems reduziert werden darf und insbesondere in dem vorgegebenen Szenario bei Händlerlokationen keine ISDN–Verbindung selbständig durch den Agenten aufgebaut werden darf. Der Agent muß die durch den Händler etablierten Verbindungen zur Erfüllung seiner Aufgabe nutzen. Unbedingt gering zu halten ist der Aufwand der Implementierung bei sich wandelnden Anforderungen. Hierzu ist eine geeignete Architektur zu entwerfen, die ausreichende Schnittstellen für Anpassungen bereithält.
Eine letzte Forderung ist der Anspruch an die *Funktionalität* des Agenten. Anzustreben ist die Messung der durch den Benutzer *erlebten* Werte, wozu meist eine Instrumentierung der benutzen Applikationen notwendig ist. Des weiteren sollen unterschiedliche Messungen (z.B. Antwortzeiten, Verfügbarkeit, ...) durchgeführt werden und Systemdaten und Benutzerverhalten der entfernten Lokation in dem für die Administration notwendigen Rahmen verfügbar gemacht werden.

2.3.2 Überwachung einer ISDN–Verbindung

Die Überwachung der Service–Level ist eine von der DeTeSystem als Leistungsanbieter durchzuführende Tätigkeit. Es besteht die vertragliche Verpflichtung (s. Abschnitt 2.2), die Einhaltung der vereinbarten Dienstgüte permanent in sogenannten Reports darzulegen.
Aus dieser Perspektive betrachtet stellt eine Architektur, die solche Betriebsparameter mißt, keine Sonderleistung für den Kunden dar und die mit der Erbringung dieser Leistung verbundenen Kosten können folglich nicht auf den Dienstnehmer umgelegt werden. Kosten für die Planung, Umsetzung und Betrieb einer solchen Lösung müssen durch die DeTeSystem getragen werden.
Bei der Vielzahl der Händlerlokationen ist es somit aus betriebswirtschaftlichen Gründen unver-

zichtbar, daß die Agenten bestehende Verbindungen nutzen, und nicht bei Bedarf regelmäßig eigene aufbauen.

2.3.3 Messung von Antwortzeiten

Verfügbarkeit und Antwortzeiten werden in vielen Bereichen der Informationstechnik als Indikator für Leistungsfähigkeit und Benutzerfreundlichkeit eines Systems angesehen. Folglich gibt es mehr als eine Definition für diese Begriffe, die immer wieder kontextabhängig spezifiziert werden müssen. Allgemein können jedoch folgende Definitionen die Gemeinsamkeiten dieser unterschiedlichen Ansätze hervorheben:

> **Definition: Antwortzeit**
> Unter der Antwortzeit eines Dienstes versteht man den Zeitintervall zwischen dem Absenden einer Nachricht und dem Empfang der entsprechenden Antwort.

> **Definition: Verfügbarkeit**
> Eine Ressource ist im Sinne dieser Arbeit dann nicht verfügbar, wenn ihre Antwortzeit x Sekunden überschreitet.

Diese beiden Kenngrößen stellen den zentralen Ansatz dieser Arbeit dar. In Anlehnung an die Service–Level–Agreements sollen diese Parameter auf der Netzwerk– und Applikationsebene gemessen werden. Auf Netzwerkebene soll festgestellt werden, ob für den Händler relevante Netzkomponenten wie

- PoPs
- DNS–Server
- WWW–Server
- E–Mail–Server
- Authentifizierungs–Server
- Router

verfügbar sind. Hier soll mit möglichst einfachen Mitteln festgestellt werden, ob die geprüfte Komponente aktiv ist. Diese Messung soll nicht von dem Benutzerverhalten abhängig sein, sondern sobald möglich in Form eines Polling–Mechanismus durchgeführt werden.
Wie in der Darstellung der Service Level Agreements beschrieben, werden die einzelnen Dienstgüteparameter zwischen dem IT–Dienstleister und dem Benutzer ausgehandelt. Hierbei wird nicht die Kapazität z.B. eines WWW–Servers definiert, da dies aus Sicht des Benutzers Details der Umsetzung sind. Vielmehr stellt der Benutzer fest, mit welchen Antwortzeiten er arbeiten muß, um in vertretbarer Zeit seine Aufgabe zu erfüllen. Für die Durchführung solcher Anforderungen sind zusätzliche Informationen notwendig. Dazu gehören Antworten auf die folgenden Fragestellungen:

1. Arbeitet die Applikation korrekt? Diese Frage liegt allen anderen zugrunde und ist von ausschlaggebender Wichtigkeit.
2. Wie ist die Antwortzeit? Wie ist die Performance der Applikation? Wie hoch ist ihr Datendurchsatz? Es ist anzustreben, daß der aktuelle Service–Level, der durch den Benutzer erlebt wird, gemessen werden kann.

3. Warum ist die Applikation nicht verfügbar? Welche Operation, Transaktion oder entfernte Komponente (Applikation oder Server) ist dafür verantwortlich? Wenn eine Applikation ihre Aufgabe nicht erfüllt ist es unverzichtbar, schnell und effizient diesen Fehler beheben zu können. Informationen zu möglichen Fehlerquellen verringern die Reaktionszeit erheblich.

4. Wer benutzt diese Applikation, wie oft wird sie benutzt und welche Arten von Transaktionen werden durchgeführt? Welche Server bieten diese Dienste an? Diese Informationen erleichtern das sinnvolle Planen und Ausbauen von Kapazitäten.

Um solche Informationen zu erhalten ist es nötig, Messungen auf Applikationsebene vorzunehmen. Es genügt nicht, einzelne Teilaspekte wie Durchsatz in bestimmten Teilstrecken des Netzes, Prozessorauslastung des Servers o.ä. zu erfassen. Vielmehr müssen die vom Benutzer erlebten Werte gemessen werden. Dazu ist es notwendig, die entsprechende Applikation zu instrumentieren, um so Informationen wie z.B. den Beginn und das Ende einer Transaktion zu propagieren.

2.3.4 Sicherheitsanforderungen

Ziel dieses Abschnitts ist die im Umfeld dieser Arbeit relevanten Sicherheitsrisiken für das zu entwerfende Agentensystem zu identifizieren. Zur Definition und Erläuterung der Architektur eines solchen Systemes wird auf das Kapitel 3.1 verwiesen.
Grundsätzlich sind vier Gruppen von Angriffen auf das zu entwickelnde System vorstellbar:

1. Bedrohung des Agenten

2. Bedrohung der Agentenplattform

3. Bedrohung der Kommunikation eines Agentensystems

4. Angriff auf das genutzte SNMP–Framework

Es ist notwendig, mögliche Angriffe zu beschreiben und deren Auswirkungen zu bewerten, um so an sinnvollen Stellen notwendige Schutzmechanismen zu installieren.
Zur Vereinfachung wird in den folgenden Abschnitten das Anwendungsszenario auf einen Service Provider und einen Kunden reduziert. Händlerlokationen sind somit Teil des Kundennetzes.
In diesem Abschnitt wird insbesondere die Betrachtung der Angriffsmöglichkeiten von Agentensystemen von Gertraud Unterreitmeier in [Unter 98] berücksichtigt.

2.3.4.1 Bedrohung des Agenten

Das im Rahmen dieser Arbeit entstehende Agentensystem soll unter anderem einen Leistungsnachweis der DeTeSystem gegenüber ihren Kunden zusammenstellen. Es ist daher notwendig, den Agenten vor möglichen Zugriffen im Kundennetz zu schützen. Insbesondere sind folgende Angriffe denkbar:

- **Ausspionieren des Agenten**
  Der Agent befindet sich während seiner Ausführung vollständig auf einem Rechner im Kundennetz. Weder seine Daten noch der Programmcode dürfen verschlüsselt sein, da sonst eine Ausführung auf der Plattform nicht möglich ist. Daher ist es möglich, den Agenten auszuspionieren. Da jedoch weder die DeTeSytem noch der Kunde Interesse an einer Geheimhaltung der durch den Agenten gesammelten Daten haben, ist diese Bedrohung kaum relevant. Vielmehr wird dem Kunden sinnvollerweise Einblick in die Arbeitsweise und Ergebnisse gewährt.

- **Unterbrechen der Ausführung**
Die Interpretation und somit die Ausführung des Agenten kann durch seine Plattform beendet werden. Dies kann durch den Kunden beabsichtigt initiiert werden, um den Leistungsnachweis der DeTeSystem zu verhindern. Dagegen gibt es keine Schutzmaßnahmen. Selbst durch die Installation eines unabhängigen Rechners ohne Zugriffsmöglichkeiten für den Kunden ist die Trennung von der Stromversorgung nicht zu vermeiden. Allerdings sollte es Mechanismen geben, die das Beenden des Agenten frühzeitig erkennen und einen Neustart initiieren, woduch auch der Fall der unbeabsichtigten Störung erfaßt ist.

- **Codeveränderung bei der Migration**
Schon bei der Migration des Agenten, oder während der für das JDMK typischen dynamischen Änderung der Agentenfuntionalität z.B. durch das Registrieren neuer M-Beans (Erläuterung in Kapitel 3.2) können die Daten und Methoden verändert werden.

- **Manipulation der gesammelten Daten**
Der Agent sammelt während der Messungen im Kundennetz abrechnungsrelevante Daten, wie z.B. Antwortzeiten. Kann der Kunde diese nun verändern, ohne daß die DTS dies zur Kenntnis nimmt, ist finanzieller Schaden die Folge. Dies gilt es zu verhindern.

2.3.4.2 Bedrohung der Agentenplattform

Die Plattform stellt für einen Agenten die Ausführungsumgebung dar. Sie bietet grundsätzlich allen Agenten die Möglichkeit, sie zu nutzten. Daher sollten diese Umgebung die Rechte und Möglichkeiten des ausgeführten Agentencodes begrenzen und überwachen. Nur so kann die Sicherheit der Agentenplattform, der anderen Agenten und der anderen Daten und Prozesse gewährleistet werden.
Dieser Schutz wird in Java–Umgebungen hauptsächlich dadurch realisiert, daß der Programmcode des Agenten interpretiert wird. So kann der Code während der Interpretation auf sicherheitskritische Aspekte geprüft und notfalls gestoppt werden. Dasselbe Verfahren wird auch bei den Java Applets verwendet.
Folgende Angriffsmöglichkeiten auf die Agentenplattform sind denkbar:

- **Ausspionieren der Agentenplattform**
Fremde Agenten, die auf eine Plattform gelangen, könnten versuchen, diese auszuspionieren. Potentielle Ziele sind Daten des Rechners, auf dem sie ausgeführt werden oder Daten anderer Agenten.

- **Ressourcenmißbrauch**
Ein Agent kann auf einer Plattform soviele Ressourcen verbrauchen, daß er andere Agenten oder sogar das System stört.

- **Fluten der Plattform mit fremden Agenten**
Haben fremde Agenten die Möglichkeit, auf eine Plattform zu gelangen, können sie diese in großer Anzahl überfluten. Eine mögliche Folge können wie beim Ressourcenmißbrauch die Störung des Systems oder die Löschung sämtlicher Agenten auf der Plattform sein.

- **Vortäuschen einer falschen Identität**
Ein Agent kann vortäuschen, im Auftrag einer berechtigten Person auf die Plattform gelangen zu wollen.

2.3.4.3 Bedrohung der Kommunikation eines Agentensystems

Während ein Agent mit der Plattform oder mit der Managementapplikation kommuniziert, kann die Sicherheit der übertragenen Daten gefährdet sein.

- **Abhören und Manipulieren der Kommunikation**
  Ein Agent kommuniziert oft nicht nur mit Komponenten der Plattform auf der er sich gerade befindet, sondern tauscht auch Informationen mit anderen Systemen über das Netzwerk aus. Diese Kommunikation kann am Übertragungskanal von anderen abgehört und manipuliert werden.

- **Verweigerung von Diensten**
  Eine Plattform kann dem Agenten während seiner Ausführung alle angeforderten Dienste verweigern, sodaß er keinen Möglichkeit hat, Werte zu messen oder Daten zu übertragen. Vergebliche Versuche sind durch den Agenten zu dokumentieren und wenn möglich später zu übermitteln.

2.3.4.4 Angriff auf das genutzte SNMP–Framework

Wie Sicherheit in ein SNMP Framework paßt, ist in [HPW 98] beschrieben, ein spezifiziertes Access Control Model ist in [WPM 98] dargestellt. Die Bedrohung des SNMP–Frameworks sollen im Kontext dieser Arbeit nicht untersucht werden.

2.3.4.5 Zusammenfassung der Angriffsmöglichkeiten

Zusammenfassend können nun folgende Bedrohungen klassifiziert werden:

Bedrohung
Bedrohung des Agenten
Ausspionieren eines Agenten
Unterbrechung der Ausführung
Codeveränderung bei der Ausführung
Codeveränderung bei der Migration
Bedrohung der Agentenplattform
Ausspionieren der Plattform
Ressourcenmißbrauch
Fluten mit fremden Agenten
Vortäuschen einer fremden Identität
Bedrohung der Kommunikation
Abhören und Manipulieren der Kommunikation
Verweigerung von Diensten

Tabelle 2.2: Klassifizierung der Bedrohungen eines Agentensystems

In der abschließenden Bewertung der entwickelten Architektur ist zu prüfen, inwiefern Sicherheitsmechanismen gegen die hier beschriebenen Angriffmöglichkeiten vorhanden oder realisierbar sind.

2.3.5 Klassifizierung der Anforderungen

In der Tabelle 2.3 wird versucht, die dargestellten Anforderungen zusammenzustellen und zu klassifizieren. Einzelne Anforderungen sind gleichzeitig mehreren Gruppen zuzuordnen. Um die Übersicht zu

Anforderung
Technische Anforderungen
Plattformunabhängigkeit
geringer Resourcenbedarf
Protokolldatei
Organisatorische Anforderungen
Organisation der Agenten
dynamische Veränderbarkeit
Konfigurierbarkeit während der Laufzeit
Integration in bestehende Managementarch.
Betriebswirtschaftliche Anforderungen
geringer Verteilungsaufwand
kein selbständiger Verbindungsaufbau
geringer Implementierungsaufwand
Anforderungen an die Funktionalität
diverse Meßmethoden
Messungen auf Applikationsebene
Auslesen von Systemdaten
Überwachen des Benutzerverhaltens
Anforderungen an die Sicherheit

Tabelle 2.3: Anforderungen an die Lösung

erleichtern wurde hier nur eine Zuordnung durchgeführt.

Kapitel 3

State–of–the–Art

Als Grundlage für das zu entwickelnde Konzept werden in diesem Kapitel relevante theoretische Ansätze und bestehende Entwicklungswerkzeuge und Management–Applikationen unter Berücksichtigung der in Kapitel 2 dargestellten Anforderungen untersucht und bewertet.

3.1 Das Konzept der flexiblen Management Agenten (FMA)

Die Theorie der flexiblen Management–Agenten ist eine Form der Realisierung von verteilten Management–Systemen. Ziel ist es, durch die Platzierung von Management–Intelligenz in die Nähe der zu verwaltenden Ressource komplexe Systeme steuerbar zu machen und zu entlasten.

3.1.1 Der Agentenbegriff

Bis heute ist es nicht gelungen, eine allgemein akzeptierte, umfassende Definition eines intelligenten Agenten zu finden. Das Problem liegt hierbei in dem interdisziplinären Charakter der Agenten, welche sich in Einflüssen unterschiedlicher wissenschaftlicher Forschungseinrichtungen einerseits und den von der Praxis gestellten Anforderungen andererseits widerspiegelt.
Es ist daher notwendig, sich bei der Definition dieses Begriffes auf den Kontext dieser Arbeit zu beschränken und die relevanten Schwerpunkte zu analysieren und herauszuarbeiten.
In [Lang 98] definieren Danny Lange und Mitsuru Oshima einen Softwareagenten anhand der folgenden Prädikate:

- **Stationär**
  Ein Agent läuft in einer bestimmten Ausführungsumgebung ab.
- **Reaktiv**
  Er kann Änderungen in seiner Umgebung feststellen und aufgrund dieser seine Arbeit anpassen.
- **Autonom**
  Der Agent hat die Kontrolle über seine eigenen Aktionen.
- **Zielorientiert**
  Seine Ausführung hat hauptsächlich das Ziel der Problemlösung.
- **Kommunikativ**
  Der Agent kann mit anderen Agenten und seinem Manager kommunizieren.

Außerdem kann er noch folgende Eigenschaften haben:

- **Fortlaufend**
  Der Agent arbeitet ständig an der Verbesserung seines Ergebnisses.

- **Lernend**
  Er paßt seine Handlung an bereits gelernten Fakten an.
- **Vertraut**
  Die Bedienung erscheint dem Benutzer vertraut.

Im Gegensatz zu solchen stationären Agenten haben mobile Agenten die zusätzliche Eigenschaft der **Mobilität**. Dazu wird ihr Programmlauf gestoppt, sie werden auf eine andere Plattform transportiert und dort wird ihre Ausführung fortgesetzt. Die Entscheidung über Zeitpunkt und Ziel liegt allein beim Agenten. Daraus läßt sich folgende Definition für einen mobilen Agenten ableiten:

> **Definition: Mobiler Agent**
> Ein mobiler Agent ist ein Agent, der nicht an die Umgebung gebunden ist, auf der er seine Ausführung begonnen hat. Er kann sich aufgrund einer eigenen Entscheidung von einem System über das Rechnernetz auf ein anderes System begeben. Diesen Vorgang nennt man Migration.

Diese grundsätzliche Darstellung des Agentenbegriffes soll im Rahmen dieser Arbeit genügen. Tiefergehende Behandlungen des Begriffes sind in [Moun 97] und [Lang 98] zu finden.
Diese Definition der Prädikate eines Softwareagenten bildet die Grundlage für das weiterführende Konzept der flexiblen Management–Agenten.

3.1.2 Management by Delegation

Unter dem in der Agentendefinition festgelegten Prädikat *kommunikativ* ist mehr zu verstehen als nur die Fähigkeit, mit anderen Agenten oder einem Manager Informationen auszutauschen. Ein flexibler Software–Agent zeichnet sich durch die Eigenschaft aus, Funktionalitäten an andere Agenten delegieren und selber neue Aufgaben annehmen zu können. Diese Fähigkeit macht den Agenten dynamisch erweiterbar und ermöglicht den flexiblen Einsatz in sich schnell wandelnden Umgebungen.
Die Delegierung einer Aufgabe an einen Software–Agenten umfaßt drei Phasen [Moun 97]:

1. **Before Delegation**
   In der ersten Phase ist festzustellen, welche Aufgabe über welchen Mechanismus an welchen Agenten zu delegieren ist. Dazu können feste Entscheidungsmechanismen — z.B. auf Grundlage der Netzwerktopologie — oder auch Benutzerinteraktionen zugrunde liegen.
   Es kann auch die Situation bestehen, daß ein Agent zur Erfüllung seiner Aufgaben eine zusätzliche Funktionalität benötigt. In einen solchen Fall delegiert er das Bereitstellen dieser Funktionaliät an einen anderen Agenten oder den Manager.
2. **During Delegation**
   Jetzt wird auf der Grundlage der in Phase eins getroffenen Entscheidungen die Aufgabe an den entsprechenden Agenten delegiert. Die dazu benötigte Funktionalität wird dem Agenten in für diesen verständlicher Form übertragen und er beginnt mit der Ausführung.
3. **After Delegation**
   Nach der Delegierung der Aufgabe an einen Agenten ist zu prüfen, wie lange dieser Agent diese Aufgabe durchführen soll, bzw. wann diese als erfüllt gilt. Darauf basierend ist zu prüfen, ob der Agent die neue Funktionalität behalten oder ihm diese wieder entzogen werden soll.

Basierend auf dieser Fähigkeit, neue Aufgaben während der Laufzeit übernehmen zu können, läßt sich nun das Konzept des Management by Delegation (MbD) einführen. Dieses ermöglicht es, flexible auf sich ändernde Gegebenheiten zu reagieren und durch die Verteilung von Aufgaben Ressourcen gleichmäßig zu belasten.
Auf der Grundlage dieser flexiblen Management–Agenten läßt sich nun eine neue Managementstruktur entwickeln.

3.1.3 Ein auf FMAs basiertes verteiltes Managementmodell

Ein verteiltes Managementmodell zeichnet sich insbesondere dadurch aus, daß Managementaufgaben, wie in Abbildung 3.1 dargestellt, „in der Nähe“ der zu verwaltenden Objekte (Managed Objects, kurz MO) plaziert werden.
Es sprechen eine Vielzahl von Gründen für ein Verteilung der Management–Intelligenz und –

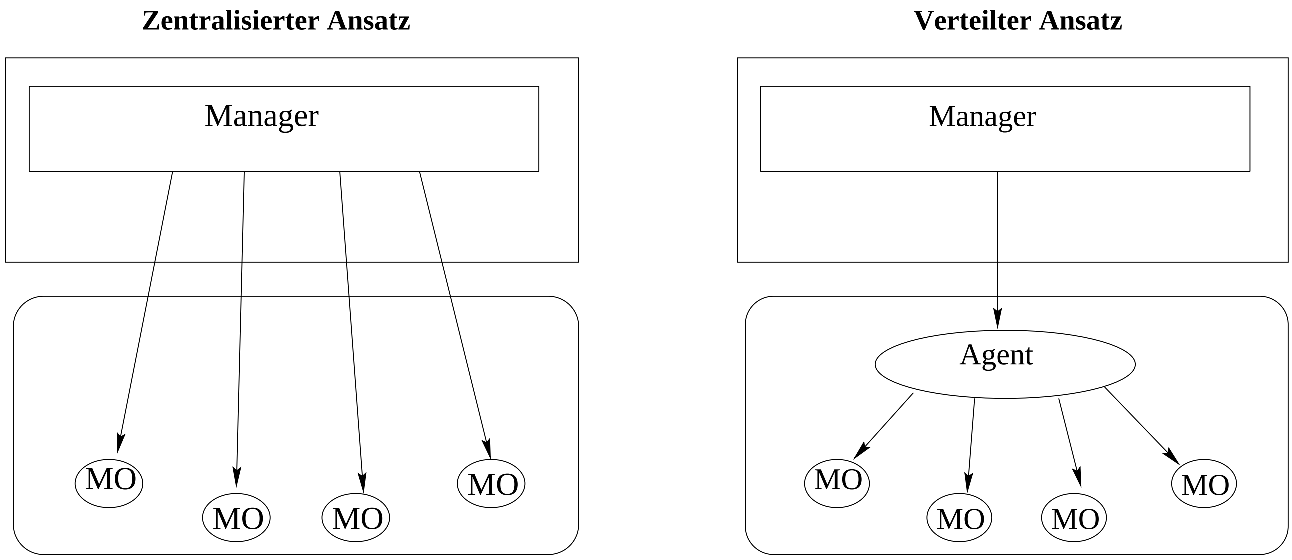

Abbildung 3.1: Das Prinzip eines verteilten Management–Modells

Funktionalität [Moun 97]:

- Notwendigkeit verteilter Management–Intelligenz, z.B. bei zeitweise vom Netz getrennten Systemen
- Hohe Frequenz des Nachrichtenaustausches, z.B. bei Polling Mechanismen
- Hoher Bedarf an Management–Informationen im Verhältnis zu Netzwerkdurchsatz
- Die Management–Applikation befindet sich in einer dynamischen heterogenen Umgebung und muß permanent angepaßt werden
- Häufige Einschränkungen von Ressourcen, die im Extremfall die zeitweise Trennung von Komponenten vom Netz zur Folge haben (Wählverbindungen, nomadische Systeme, . . .)

Die Notwendigkeit für ein dezentrales Management–Konzept ist insbesondere in großen Netzen unübersehbar. In den folgenden Abschnitten wird nun dargelegt, aus welchen Komponenten so ein Konzept besteht und welchen Anforderungen es genügen muß, um in sich geschloßen zu funktionieren.

3.1.3.1 Terminologie

An dieser Stelle ist der Begriff des flexiblen Agenten deutlich von dem der Management–Funktionalität abzugrenzen. Wie im folgenden noch beschrieben wird, unterscheiden sich insbesondere im Konzept des JDMK Funktionalitäten nur bedingt von Agenten.
Grundsätzlich können auf einer Agentenplattform mehrere Agenten laufen, d.h. ihre Programme werden zur gleichen Zeit ausgeführt. Wird nun eine weitere Management–Funktionalität an einen solchen Agenten delegiert, ist diese vom logischen Verständnis her kein weiterer Agent, auch wenn sie, einmal

auf der Agenten–Plattform angekommen und ausgeführt, selbständig ihrer Aufgabe nachkommt und alle Prädikate eines Agenten besitzt.
Im Sinne dieser Arbeit besteht ein flexibler Agent, wie in Abbildung 3.2 dargestellt, aus zwei Kom-

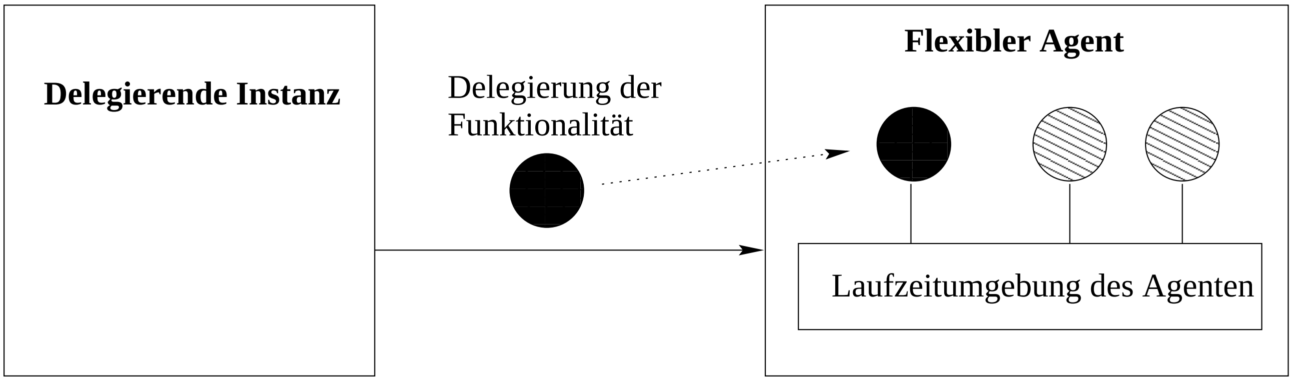

Abbildung 3.2: Das Prinzip der flexiblen Management–Agenten (FMA) nach [Moun 97]

ponenten:

1. Durch die Agentenplattform angebotene Standardfunktionalität (Kommunikationsservices, ...)
2. Sich dynamisch ändernde Funktionalitäten. Diese werden realisiert durch Funktionen, die der Agentenplattform zugewiesen werden.

Bei der Realisierung des Agenten mit dem JDMK wird die Agentenplattform in dem Agenten implementiert. Der Entwickler hat die Möglichkeit, den Umfang der Standardfunktionalität über ein Minimum hinaus selber festzulegen. Auch wenn so die Grenzen zwischen Plattform und Agenten verschwimmen, läuft die Plattform jedoch getrennt von dem Agenten ab, sodaß sie anderen JDMK–Agenten die Möglichkeit bietet, sich dort registrieren und ausführen zu lassen.
Ob nun die in Abbildung 3.2 zugewiesene Funktionalität einen eigenen Agenten darstellt, oder ob sie Teil des bestehenden Agenten werden soll, ist der logischen Betrachtung des Benutzers bzw. Entwicklers überlassen und spiegelt sich z.B. in der Namensgebung wider.
Nach dieser Erörterung der Terminologie kann nun ein Managementmodell dargestellt werden.

3.1.3.2 Das Konzept

Da im Kontext der vorliegenden Arbeit die Delegation von Aufgaben zwischen zwei Agenten nicht gegeben ist, wird dieser Aspekt des FMA–Modells vernachlässigt. Es wird allerdings darauf hingewiesen, daß die Umsetzung dieser Funktionaliät realisierbar ist.
Man unterscheidet im wesentlichen zwischen zwei Management–Einheiten in diesem Konzept:

1. Manager
2. flexibler Agent
3. Agent

Flexible Agenten wurde bereits in dem vorangegangenen Abschnitt erörtert. Diese müssen nun gegen den Manager abgegrenzt werden. Dazu können Kriterien wie die folgenden genutzt werden [Moun 97]:

- Der Manager hat in der Regel eine grafische Benutzerschnittstelle.

- Der Manager befindet sich in der Regel auf einer zentralen Komponente und läuft auf einem bestimmten Rechner.
- Der Manager kann Aufgaben an einen Agenten delegieren, umgekehrt ist dies nicht möglich.
- Der Manager hat eine umfangreichere Informationsbasis als die Agenten (Netzwerktopologie, ...)
- Der Manager einer spezifischen Domäne hat alle Informationen über das verteilte Agentensystem in seiner Domäne (welche Agenten/ Prozesse wo laufen) und kann deren Ausführung steuern.
- Der Manager hat ein Veto–Recht beim Ausführen von Management–Aufgaben. Er kann Agenten im Konfliktfall *überstimmen*.

Im Gegensatz zum flexiblen Agenten hat ein Agent nicht die Möglichkeit, seine Funktionalität während der Laufzeit an sich verändernde Verhältnisse anzupassen. Beispiel für einen solchen Agenten ist ein SNMP–Agent.
Abbildung 3.3 stellt dar, wie sich aus den verwalteten Komponenten und den dazugehörigen Agenten in einer Netzstruktur eine Management–Architektur für ein zu verwaltendes System herleitet.
Die Zuweisung flexibler Agenten zu Domänen geschieht in Abhängigkeit von organisatorischen Aspek-

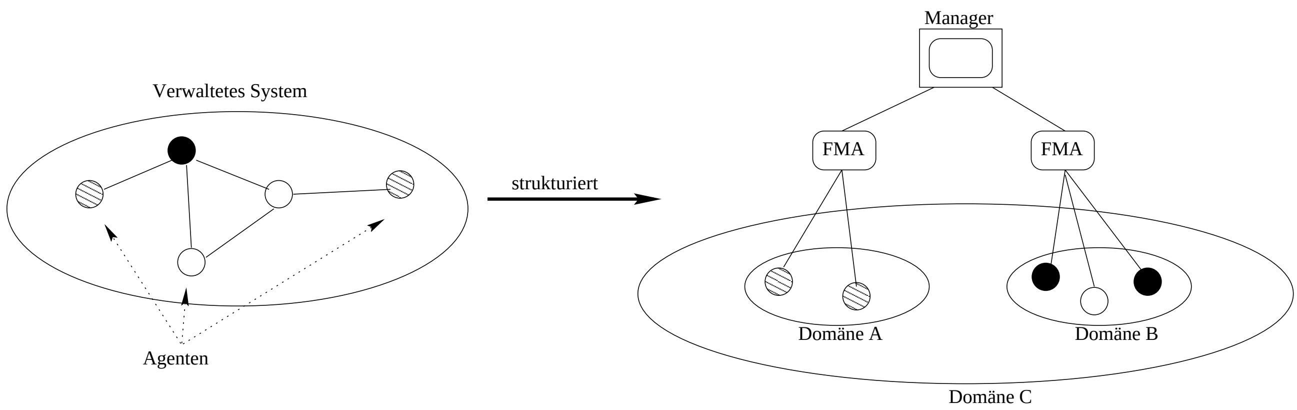

Abbildung 3.3: Logische Struktur eines Agentensystems nach [Moun 97]

ten, die z.B. geographischer Natur sein können oder auf der Art der zu verwaltenden Ressource basieren. Ausgehend von dieser Struktur werden nun Gruppen definiert, die die FMAs innerhalb einer Domäne unter funktionellen Gesichtspunkten gliedern. So können z.B. Gruppen gebildet werden für Agenten, die Messungen vornehmen, oder solche, die Informationen sammeln und auswerten.
Als letztes soll an dieser Stelle die Kommunikation in einem Agentensystem untersucht werden. Man unterscheidet drei unterschiedliche Arten von Datenaustausch:

1. **Aufgaben**
   Aufgaben können Abfragen oder Aktionen sein. Zur Realisierung werden in der Regel Funktionsaufrufe genutzt.
2. **Informationen**
   Informationen können Events, Meßergebnisse oder die Rückgabe eines Abfrage–Ergebnisses sein.
3. **Funktionalität**
   Eine Funktionaliät wird übertragen, wenn der Agent zu der Ausführung einer an ihn gestellten Aufgabe nicht in der Lage ist. Dies können z.B. Skripte oder Klassen von MOs mit deren Methoden sein.

Um diesen Datenaustausch zu ermöglichen, müssen bestimmte Dienste angeboten werden. Die für diese Arbeit relevanten sind im folgenden dargestellt:

- **Delegations–Dienst** (Delegation Service)
  Dieser Dienst ermöglicht den Austausch von Aufgaben und Funktionalitäten.
- **Ereignis–Dienst** (Event Service)
  Über unterschiedliche Mechanismen kann der Agent sich für den Erhalt relevanter Events (Ereignismitteilungen) registrieren lassen.
- **Agenten–Bestimmungs–Dienst** (Location Service)
  Mit diesem Dienst können Agenten in einem System identifiziert und lokalisiert werden.
- **Sicherheits–Dienst** (Security Service)
  Management impliziert generell die Anforderung, den unbefugten Zugriff auf sicherheitempfindliche Ressoucen zu unterbinden. Ein solcher Dienst soll das System durch Mechanismen wie z.B. Authentifizierung vor möglichem Schaden schützen.
- **Management–Dienst** (Management Service)
  Es sollte möglich sein, die verteilten und laufenden Agenten zu überwachen und den Stand ihrer Tätigkeit jederzeit abrufen zu können.

Grundlage für diese Dienste ist ein Basis–Kommunikations–Dienst, der die entsprechenden Protokolle zur Verfügung stellt.
In diesem Kapitel wurde dargestellt, daß das Konzept der flexiblen Management–Agenten eine Architektur anbietet, die den Anforderungen komplexer und moderner Netze gerecht wird.
Das im folgenden Kapitel beschriebene JDMK stellt ein Werkzeug zur Erstellung eines solchen Agenten–Systemes dar.

3.2 Das Java Dynamic Management Kit

In diesem Kapitel wird die zur Entwicklung des Prototypen benutzte Programmiersprache Java und das Java Dynamic Management Kit vorgestellt und die zugrundeliegenden Konzepte erläutert.

3.2.1 Die Sprache Java

Die von der Firma SUN entwickelte Programmiersprache Java hat sich in vielen Bereichen als Standardsprache durchgesetzt. Auch bei den unterschiedlichen Agentensystemen (Voyager, Aglets, JumpingBeans, ...) gibt es kaum noch aktuelle Versionen, die nicht in Java geschriebene Agenten unterstützt.
In einer der ersten Beschreibungen der Sprache benutzte die Firma Sun folgende Aneinanderreihung von Schlagworten [Flan 98]:

> *Java: A simple, object-oriented, distributed, interpreted, robust, secure, architectural neutral, portable, high-performance, multithreaded and dynamic language .*

Im folgenden werden die wichtigsten Attribute kurz erläutert.
Java ist eine moderne, objektorientierte Programmiersprache, bei der die Quelltexte vor der Ausführung in schnell und effizient zu interpretierenden Bytecode übersetzt werden. Der Compiler, der Debugger und der größte Teil der Ausführungsumgebung sind selbst in Java geschrieben. Nur ein kleiner maschinenabhängiger Teil des Interpreters ist an die jeweilige Plattform gebunden. Dieser Interpreter, auch virtuelle Maschine genannt, ist für alle bekannten Plattformen frei verfügbar und kann selbst Programme mit grafischer Benutzeroberfläche auf jeder Plattform ausführen.

Da die Bytecode–Sequenzen interpretiert werden, liegt die Ausführungsgeschwindigkeit deutlich unter der anderer Sprachen wie z.B. C oder Fortran. Daher gibt es auch die Möglichkeit, den Bytecode kurz vor der Ausführung in direkt durch den Prozessor ausführbaren Code zu übersetzen. Dazu wurden sogenannte Just-in-Time Compiler (JIT) entwickelt, die allerdings bis heute für relativ wenige Plattformen verfügbar sind.
Java bietet das praktisch gleichzeitige Ausführen mehrerer Programmfäden an, sog. Threads. So können in Java realisierte Agentensysteme z.B. gleichzeitig mehrere Agenten ausführen und die Kommunikation regeln. Mit dem Exception–Handling bietet Java einen mächtigen und erweiterbaren Mechanismus zur Fehlerbehandlung.
JavaBeans ist ein Software–Komponenten–Modell für Java. Die JavaBeans API–Spezifikation definiert ein Bean wie folgt:

> *Ein JavaBean ist eine wiederverwendbare Software–Komponente, die visuell mittels eines Entwicklungstools manipuliert werden kann. In der JavaBeans API sind eine Reihe von Namenskonventionen vorgegeben, die die Wiederverwendung erleichtern.*

Die Entwicklung von Java ist noch längst nicht abgeschlossen. So wurde noch während der Erstellung dieser Arbeit die Version 1.2 des Java Development Kit freigegeben. Zwar wurde an der Syntax nichts geändert, jedoch wurden zahlreiche Erweiterungen in Bezug auf die Sicherheit und das GUI implementiert. Da diese neue Version allerdings mit dem JDMK3.0 nicht kompatibel ist, wurde sie im Rahmen dieser Arbeit nicht eingesetzt.
Der durchschlagende Erfolg von Java wird allerdings nicht nur auf die Eigenschaften der Sprache an sich zurückgeführt, sondern auch auf die umfangreichen und standardisierten Klassenbibliotheken, die mit dieser Sprache verbunden sind. So gibt es nicht nur Klassen, die unterschiedlichste Datentypen und Algorithmen zu Verfügung stellen. Auch sind zum Beispiel die Entwicklung von grafischen Benutzeroberflächen mit den unterschiedlichsten Komponenten, der Zugriff auf Datenbanken und CORBA–Mechanismen mit Hilfe umfangreicher Klassen leicht realisierbar. Das Java Dynamic Management Kit (JDMK) stellt eine der neuesten Erweiterungen dar.

3.2.2 Das Konzept des JDMK

Das JDMK bietet einen Satz von Klassen und Tools, die die Entwicklung von dynamisch erweiterbaren und intelligenten Agenten vereinfachen.
Die Abbildung 3.4 zeigt die Komponenten des JDMK und deren Verknüpfungen.
Man unterscheidet in einem Agentensystem zwischen Agentenseite und der Managerseite. Auf der Agentenseite wird das zu managende Objekt durch den Agenten überwacht und manipuliert. Der Manager steuert diese Agenten von der anderen Seite in Form einer Applikation oder durch einen Web–Browser (WWW–basiertes Management).
Sämtliche Komponenten des Agentensystems laufen in einer virtuellen Maschine, die für Java als Laufzeitumgebung benötigt wird und dem Agenten als Plattform dient.

3.2.3 Die Komponenten des JDMK

In den folgenden Abschnitten werden die Komponenten des in Abbildung 3.4 dargestellten Agentensystems des JDMK kurz erläutert. Eine detaillierte Beschreibung ist in [JDMK 98] zu finden.

3.2.3.1 Core Management Framework

Das Core Management Framework (kurz Framework, CMF) dient der Registrierung der Objekte eines Agenten. Diese Agenten können durch den Agenten selber oder den Manager angemeldet werden.

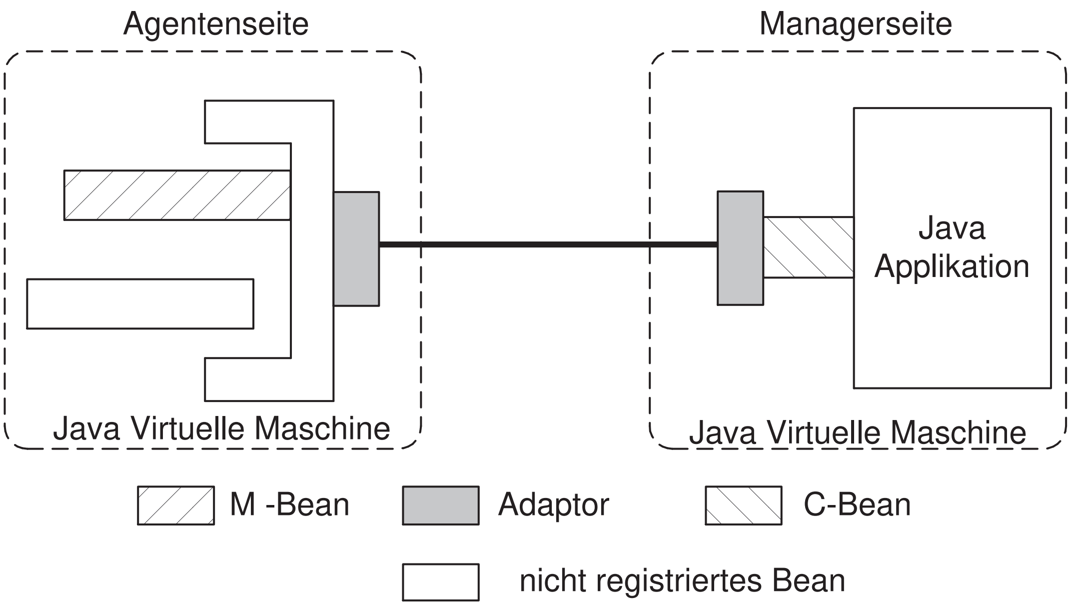

Abbildung 3.4: Komponenten des JDMK und deren Verknüpfungen

Jedes Objekt, das verwaltet werden soll, muß in dem Framework registriert sein.

3.2.3.2 Managed Beans

Ein M–Bean (Kurzform für Managed Bean) ist eine Java–Komponente, die bestimmten Design–Pattern entspricht. Die Instanz eines solchen M–Beans ist managebar, sobald es beim Framework angemeldet ist. Jedes Objekt, das über ein Framework erreichbar sein soll, muß durch ein M–Bean repräsentiert werden. Solche Objekte können Ressourcen sein, die mit dem Agenten verwaltet werden sollen, oder Dienste, die dem System bereitgestellt werden sollen. Es existieren keine Restriktionen bezüglich des Speicherortes des M–Beans, so daß dieser auch auf einem Remote–Server sein kann.

3.2.3.3 Client Beans

Eine C–Bean (Kurzform für Client–Bean) ist ein Stub–Objekt, das auf der Managerseite das M–Bean repräsentiert. Es wird mit dem MOGEN–Compiler generiert. Events, die durch das M–Bean erzeugt werden, werden automatisch an das dazugehörige C–Bean propagiert.

3.2.3.4 Adapter

Ein Adapter verbindet das Framework mit einer externen Applikation und stellt in der Realisierung selber ein M–Bean dar. Es ermöglicht das Betrachten und Manipulieren eines M–Beans mittels eines spezifizierten Protokolls. Ein solcher Adapter ermöglicht einer externen Applikation, Attribute bestehender M–Beans abzufragen und zu setzen, Methoden aufzurufen und neue Objekte zu erzeugen und im Framework anzumelden. Ein Agent muß über mindestens einen Adapter verfügen, kann allerdings auch mehrere haben, um über verschiedene Protokolle ansprechbar zu sein.
Folgende Adapter werden durch das JDMK zur Verfügung gestellt:

- **RMI**
  Der RMI Adapter ermöglicht einem Java-Manager den Zugang zu einem JDM-Agenten mittels der Java Remote Method Invocation (RMI). Mit diesem entfernten Methodenaufruf können Client-Objekte Methoden eines entfernten Server-Objektes Aufrufen. In einem Agentensystem stellt der Agent als Server Methoden (z.B. zum Abfragen oder Ändern von Parametern) zu Verfügung. Diese Methoden werden dann durch die Manager als Client aufgerufen. Dabei kann als Argument und als Rückgabewert jeder primitive Datentyp und jedes komplexe Objekt verwendet werden.

- **HTTP über TCP, UDP oder SSL**
  Diese Adapter ermöglichen die für den Entwickler transparente Benutzung von sogenannten Socket–Verbindungen. Socket–Schnittstellen dienten ursprünglich ausschließlich zur Kommunikation zwischen Prozessen und erlauben neben UDP und TCP auch den Zugriff auf darunterliegende IP–Schichten. Mit SSL kann zusätzlich eine verschlüsselte Übertragung erreicht werden.

- **IIOP**
  Der IIOP–Adapter ermöglicht CORBA–Clients mit einem JDM–Agenten zu kommunizieren wie mit einem CORBA-Server. Er ermöglicht so das Erzeugen und Löschen von M–Beans, das Setzen und Abfragen von Parametern und das Ausführen von Aktionen. Um diesen Adapter nutzen zu können, muß ein Common Object Service (COS) zur Verfügung gestellt werden.

- **SNMP**
  Durch den SNMP–Adapter wird SNMP–Agenten und auch MIB–Browsern der Zugriff auf die Parameter eines M–Beans ermöglicht. Ein Zugriffsschutz geschieht mit einem ACL–File.

- **HTML**
  Wie allgemein bekannt ist, ist HTML kein Protokoll, sondern eine Darstellungssprache für Web–Seiten. Dieser HTML–Adapter ermöglicht die über HTTP die Kommunikation mit einem Web–Browser und stellt somit einen kleinen Web–Server dar, der bei Zugriff auf einen Agenten zur Laufzeit einen Satz von HTML–Seiten generiert, die die M–Beans des Agenten darstellen.

In Kapitel 4.1.3 werden die zur Verfügung gestellten Protokolle genauer dargestellt und gegeneinander abgewogen.

3.2.3.5 Dienste

Zur Vereinfachung der Entwicklungen von Agentensystemen wird durch das JDMK eine Anzahl von Services bereitgestellt.
Eine Auswahl dieser Dienste wird im folgenden kurz vorgestellt:

- **Base Services**
  Diese Services benötigt das Framework, um funktionieren zu können. Sie werden als seperate Komponenten angeboten, um den Entwickler die Auswahl zwischen unterschiedlichen, auch eigenen Implementierungen zu ermöglichen. Diese Services sind der Repository–, der Metadata–, und der Filtering–Service.

  – *Repository–Service*
    Der Repository–Service dient der Registrierung der M–Beans im CMF. Mit einen Objektnamen werden diese Beans dann gespeichert und es kann über diesen auf sie zugegriffen werden. Es gibt die Möglichkeit, die Agenten flüchtig zu speichern — also nur im Arbeitspeicher (Volatile Repository), was bei Beendigung des Agenten zur Folge hat, das ihr aktueller Status verloren geht — oder ihn dauerhaft (Persistent Repository) unter Nutzung von Serialisierungsmechanismen zu speichern. Ein Mixed Repository ermöglicht die unterschiedliche Behandlung verschiedener M–Beans.

  - *Metadata–Service*
    Mit dem Metadata–Service werden die Methoden und Parameter eines M–Beans im Framework bereitgestellt. Dieser Service basiert auf der Reflection API des JDK.
  - *Filtering–Service*
    Dieser Service ermöglicht es, auf eine bestimmte Auswahl von M–Beans zuzugreifen, z.B. auf der Grundlage bestimmter Wertebereiche der Properties.

- **Class Loader**
  Diesen Dienst wird dazu genutzt, eine Klasse von einer entfernten Lokation in das CMF zu laden. Dies kann entweder durch den Agenten selber oder durch einen entfernten Manager initiiert werden. Die geladenen Klassen können sich entweder auf dem Agentenrechner befinden oder auf einer entfernten Maschine, vorausgesetzt, sie werden mittels eines Class–Servers bereitgestellt.

- **Library Loader**
  Der Library–Loader dient dem Laden sogenannter „Native Libraries", also Bibliotheken, die nicht in Java implementiert wurden. Die Anbindung solcher Bibliotheken erfolgt in der Regel über das Java Native Interface (JNI).

- **Monitoring Service**
  Dieser Service ermöglicht das Beobachten des Wertebereichs bestimmter Properties eines M–Beans, wobei die Zeitspannen zwischen den Prüfungszeitpunkten von dem Benutzer festgelegt werden kann. Es werden zwei Arten von Monitoren durch das JDMK angeboten:
  - *Counter Monitor*
    Durch einen Counter Monitor wird ein Zähler überwacht, indem nach Überschreiten einer spzifizierten Schranke ein Event ausgelöst wird.
  - *Gauge Monitor*
    Im Gegensatz zum Counter Monitor kann der Gauge Monitor Schwankungen in einem Wertebereich feststellen. Zur Erstellung entsprechender Events benötigt er eine obere und eine untere Schranke. Der Gauge Monitor wird in Abschnitt 6.2.1.3 anhand eines Beispieles genauer beschrieben.

  Zudem kann der Monitoring Service auch den Grad der Veränderungen eines Wertes propagieren.

- **Cascading Agent Service**
  Der Cascading Agent Service ermöglicht die Entwicklung ein Agentenhierarchie bestehend aus Master- und Sub-Agenten. Der Mechanismus, der hierzu genutzt wird, ist die Spiegelung sämtlicher M–Beans der Sub–Agenten in das Framework des Master–Agenten.

- **Scheduler Service**
  Dieser Dienst ermöglicht das Erzeugen von Alarmen zu bestimmten Zeitpunkten. Diese Alarme (oder Events) werden an alle registrierten Objekte gesandt.

- **Alarm Clock Service**
  Dieser Dienst benachrichtigt die registrierten `AlarmClockEventListener` in regelmäßigen Intervallen. Er wird als Basis des Scheduler Service benutzt.

- **M–Let Service**
  Der Management Applet Service wird durch das M–Bean genutzt, um Klassen aus sogenannten `jar`-Files, also Java–Archiv–Files, zu laden. Diese komprimierten Dateien werden durch den Agenten mittels Auswertung einer HTML–Datei lokalisiert. In dieser HTML-Datei ist durch ein M–Let Tag die URL des Java–Archives angegeben. Zudem bietet dieser Service einen Cache–Mechanismus, der überflüssiges nachladen über das Netz verhindert.

- **Bootstrap–Service**
  Der Bootstrap–Service ist ein ausführbares Programm, das mit Hilfe des M–Let Services das Updaten von Agenten ermöglicht.

- **Launcher–Service**
  Der Launcher–Service bietet die Möglichkeit, sämtliche zu nutzenden M–Beans in einen durch das JDMK zur Verfügung gestellten Basisagenten zu integrieren und zu starten.

- **Discovery–Service**
  Hauptaufgabe des Discovery–Services ist die Ermittlung von Agenten in einem Netzwerk, wozu er in zwei Bereiche aufgeteilt ist:

  – *Discovery–Search–Service*
    Die Komponenten, die benötigt werden, um Agenten in einem Netzwerk zu ermitteln, werden hier festgelegt. Dies ist zum einen ein Discovery–Client, der mittels eines sogenannten Multicast–Discovery–Request, also eines Rundrufes, nach den Agenten sucht und dann auf eine Antwort wartet. Jeder Agent, der durch einen derartigen Rundruf erreicht werden soll, muß über einen Discovery–Responder verfügen.

  – *Discovery–Support–Service*
    Der Discovery–Support–Service bietet einen sogenannten Discovery Monitor an, der das Registrieren und Derigistrieren von Discovery–Responder überwacht.

3.2.4 Zusammenfassung

Dieses einfache Konzept bietet eine Vielfalt von Vorteilen für die Entwicklung eines Agentensystems.

- **Bereitstellung der Hauptkomponenten eines Agentensystems**
  Durch die Bereitstellung der Agentenplattform (Framework), der Adapter und den diversen Services werden die Hauptkomponenten eines Agentensystems in anpaßbarer Form bereitgestellt. Zudem ermöglicht die im JDMK implementierte Architektur auch das Weiterarbeiten der Agenten bei Trennung von der Managementapplikation. Im Rahmen der in dieser Arbeit gestellten Szenarien ist diese Fähigkeit unbedingt notwendig.

- **Dynamische Erweiterbarkeit und Skalierbarkeit**
  Agenten können, auch während ihrer Laufzeit, dynamisch erweitert oder in ihrer Funktionalität eingeschränkt werden. Dies geschieht z.B. durch das Bereitstellen eines neuen Objektes durch einen entfernten Server oder durch die Entfernung eines M-Beans aus dem Agentenframework. Diese Fähigkeit ist aufgrund der Forderung eines dynamischen Netz- und System-Managements im Rahmen der gestellten Aufgabe unabdingbar.

- **Leichtes Management von (Java-)Applikationen**
  Insbesondere JAVA–Applikationen können durch die Hinzufügung eines Frameworks und eines Adapters leicht managebar bemacht werden. Bei anderen Applikationen muß eine Möglichkeit gefunden werden, repräsentative Informationen durch ein M-Bean zugänglich zu machen.

- **Protokoll–Unabhängigkeit**
  Die Entwicklung des M–Beans hängt nicht von dem benutzten Protokoll-Adapter ab, da das M-Bean nur mit dem Framework kommuniziert.

Das JDMK bietet also alle notwendingen Komponenten und Dienste zur Realisierung eines Agentensystems. Im folgenden Abschnitt wird beschrieben, wie die so entwickelte Infrastruktur zur Messung expliziter Werte genutzt werden kann.

3.3 Management–Anwendungen zur Antwortzeitüberwachung von Applikationen

Aus der Vielzahl der Management–Applikationen, die sich die Antwortzeitmessung und die Überwachung von SLAs zu ihrer Aufgabe gemacht haben, werden in diesem Abschnitt zwei Stück herausgegriffen.
Zum einen wird SiteScope von Freshwater untersucht, eine Applikation, die insbesondere durch ihren hohen Marktanteil auffällt und unter anderen auch von der DeTeSystem genutzt wird. Diese Applikation repräsentiert einen Server–basierten Management–Ansatz.
Desweiteren wird INFRA–XS betrachtet, ein innovativer Ansatz der Firma GW–TEL, der sich intensiv auf die Agenten–Architektur stützt.

3.3.1 SiteScope von Freshwater

Sitescope ist eine von der DeTeSystem verwendete Management–Applikation, die primär eine Sammlung von Software–Tools zur Überwachung von Web–Sites anbietet. Sie ist in der Lage, Benutzertransaktionen zu simulieren und deren prompte Ausführung zu verifizieren. Mit Sitescope können verschiedene Server und Komponenten unterschiedlicher Lokationen überwacht werden und Management–Reports in Bezug auf Service–Level von Web–Applikationen erstellt werden.
Das Programm ist eine Java–Server–Applikation, die sich aus den unterschiedlichen Überwachungsmonitoren, dem Sitescope–Framework, einer Browser–basierten Benutzeroberfläche und diversen Diagnosetools zusammensetzt. Es wird auf oder in der Nähe des Web–Servers installiert und überwacht diesen von dort aus.
Unter einem Monitor versteht Freshwater ein Java–basiertes Programm, das, einmal konfiguriert, selbstständig seiner Überwachungstätigkeit nachgeht. So kann z.B. ein URL–Transaction–Monitor so konfiguriert werden, daß er selbständig einer Serie von URLs und Links folgt und dabei die relevanten Antwortzeiten mißt. Bei Überschreiten von Grenzwerten kann dies auf unterschiedliche Weise an andere Applikationen oder verantwortliche Betreiber propagiert werden, z.B. per E–Mail, SNMP oder SMS.
In der Praxis wird dieses Programm bei der DeTeSystem wie folgt eingesetzt: In der Internet Service Area steht ein Rechner, auf dem dieses Programm läuft und die entsprechenden Monitoring–Aktivitäten ausführt. Von hier aus werden Netzkomponenten wie z.B. DNS–Server, FTP–Server und Services wie Mail, Web oder Telnet überwacht.
Es sei noch darauf hingewiesen, daß Freshwater mit SiteSeer ein neues Produkt entwickelt hat, daß den Forderungen des SLM nachkommt, von entfernten Lokationen Services zu überprüfen. Wie allerdings der Freshwater–Homepage zu entnehmen ist, ist dieses Produkt nicht dazu geeignet, eine Gesamtdatenbasis für Statistiken zu erstellen. Des weiteren bietet sie auch nur einen Katalog an Basismessungen an, der die Einbindung Transaktions–basierter Antwortzeitüberwachung in Applikationen nicht vorsieht.

3.3.2 INFRA-XS von GW-TEL

INFRA-XS (Integrating Network–Data For Runtime Analysis) ist eine neue Produktfamilie der Firma GW-TEL, die mit endgeräte–basierten Agenten die Anforderungen an IT–Service–Anbietern aus kundenorientierter Sicht abbilden und nachweisen soll. Hauptanwendungsgebiete dieser Software sind:

- Performance und Laufzeitmessungen in heterogenen Netzwerken
- Verfügbarkeits–Kontrolle und Realtime–Überwachung von zentralen Anwendungen und Servern
- Evaluierung, Überwachung und Auswertung von SLAs

- Aufbereitung und Auswertung unterschiedlicher Netzinformationen z.B. im Bereich von Accounting– und Help–Desk–Systemen

Zur Erreichung dieser Funktionalitäten hat GW-TEL ein Drei–Ebenen–Modell entwickelt, das aus Servern, Clients und Agents besteht. Aufgabe der Server ist es, alle Daten zu sammeln und vorzuverarbeiten. Clients bieten grafische Oberflächen für die Konfiguration des Management–Systems und die Darstellung und Auswertung der in den Servern gesammelten Daten. Die Agenten wiederum werden auf der Seite der Dienstnehmer installiert und führen dort vordefinierte Messungen durch, deren Ergebnisse sie permanent an den Manager übertragen.

3.3.3 Bewertung

Beide Applikationen zeichnen sich durch eine Vielzahl vordefinierter Management–, Überwachungs– und Auswertungskomponenten aus.
Während Sitescope von vielen Seiten wegen des guten Preis–Leistungsverhältnisses gelobt wird und sich als Marktführer bereits einen Namen gemacht hat, ist in der Fachpresse über INFRA-XS nicht viel zu finden.
Sitescope überwacht allerdings von einer zentralen Komponente in Servernähe die Verfügbarkeit und Antwortzeit von wichtigen Servern und ist daher nicht in der Lage, SLA aus Kundensicht zu evaluieren.
INFRA-XS entspricht in seiner Archtektur der in dieser Arbeit angestrebten Lösung. Allerdings ist dem vorliegenden Material nicht zu entnehmen, ob es sich bei den auf Seiten der Dienstnehmer eingesetzten Agenten um Flexible Management Agenten handelt. Vielmehr scheinen diese bereits mit allen Funktionalitäten ausgestattet zu sein. Auch ist nirgends der Einsatz in Verbindung mit ISDN–Routern vorgesehen. Daher ist zu vermuten, daß diese Agenten beim regelmäßigen anpollen des Managers jedesmal eine eigene Verbindung aufbauen.
Zusammenfassend kann festgestellt werden, daß die beiden hier exemplarisch vorgestellten Applikationen den für das Anwendungsszenario gestellten Anforderungen nicht entsprechen.

3.4 Entwicklungswerkzeuge zur Antwortzeitüberwachung von Applikationen

Neben den eben beschriebenen fertigen Applikationen zur Antwortzeitüberwachung gibt es einige sehr gute Ansätze nicht–kommerzieller Nutzervereinigungen wie der Internet Engineering Task Force (IETF), der Desktop Measurement Task Force (DMTF) und der Computer Measurement Group (CMG). Diese Organisationen versuchen, einem Entwickler ein Werkzeug zur Verfügung zu stellen, mit dem sie Applikationen für die Antwortzeitmessung instrumentieren können. Ziel dieser Ansätze ist die Überbrückung der Kluft zwischen den proprietären Geräte– und Applikations–Schnittstellen und standardisierten Managementarchitekturen. Diese Instrumentierung macht diese Ansätze aufwendiger bei der Einführung, eröffnet aber auch neue Perspektiven, die zuvor nicht bestanden.
Anhand der Abbildung 3.5 soll nun die erweiterte Problemstellung dieser Ansätze dargestellt werden. Bereits in der Einleitung dieser Arbeit wurde die Wichtigkeit der IT–Systeme für die Unternehmen dargestellt. Diese IT–Systeme sind in den letzten Jahren immer mehr zusammengewachsen. So verfügt z.B. auch der Fahrzeughersteller über ein Extranet, das ihn mit seinen Händler und mit seinen Lieferanten verbindet.
Startet nun ein Händler eine Transaktion mit dem Ziel, die Lieferzeit für ein zuvor spezifiziertes Fahrzeug berechnen zu lassen, so hat dies eine ganze Reihe von Subtransaktionen zur Folge. Der Fahrzeughersteller muß bei seinen Lieferanten nachfragen, wie lange die Lieferung bestimmte Bauteile (Sitze, Reifen,...) benötigt, um dann aus den Resultaten dieser Subtransaktionen und seinen eigenen Vorgaben eine realistische Lieferzeit zu berechnen.

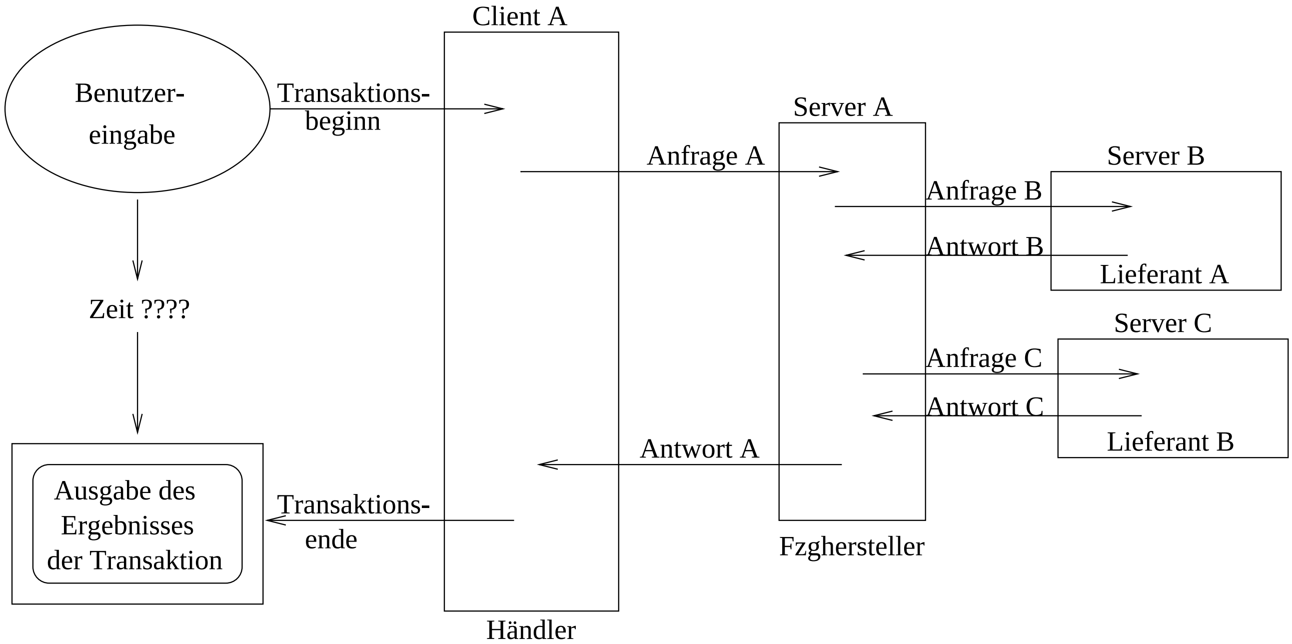

Abbildung 3.5: Verschachtelte Client–Server Transaktionen

Dem Händler, der diese Transaktion initiiert hat, genügt die Tatsache, daß seine Transaktion in wenigen Sekunden beendet ist. Ist dies allerding nicht der Fall, so muß der Service Provider in der Lage sein, den Grund für das Abweichen von diesem vereinbarten Service Level zu finden.
Dieses Wissen um die Verteilung der Antwortzeiten auf Subtransaktionen ermöglicht einem Service Provider die Verbesserung der Leistungsfähigkeit seines Systems. Können „Flaschenhälse“ identifiziert werden, so sind u.U. Perfomance–Steigerungen mit geringem Aufwand zu erzielen.
In diesem Kapitel werden Ansätze analysiert, die diesen Anforderungen z.T. entsprechen und versuchen, Korrelationen von Subtransaktionszeiten durchzuführen.

3.4.1 Application–Management–MIB der IETF

3.4.1.1 Die IETF

Die Internet Engineering Task Force (IETF) ist ein große, offene, internationale Gemeinschaft von Netzwerk–Designern, Operatoren, Händlern und Wissenschaftlern, die mit der Entwicklung und Evolution des Internets beschäftigt sind (siehe `http://www.ietf.org/`.
Ein Bereich der Bestrebungen der IETF ist das Vorantreiben der Möglichkeiten für das Applikations–Management.

3.4.1.2 Zielsetzung

Die Entwicklung der Application Management MIB befindet sich derzeit in einer experimentellen Phase.
Durch diese MIB werden Objekte definiert, die zum Management von Applikationen benötigt werden. Folgende Punkten beinhalten nach [Kalb 98] die zentralen Ziele der Application Management MIB:

- Unterstützung generischer Durchsatzmessungen von Applikationen.

- Bereitstellung von MIBs, die das Management von Applikationen als Arbeitseinheiten möglich machen.
- Unterstützung von generischer Antwortzeitmessung.
- Ressourcen– Management (benutzte Dateien, I/O Statistik, Nutzung von Netzwerkressourcen auf Applikationsebene)
- Bereitstellung von Kontrollelementen für Applikationen (Beenden von Applikationselementen, Rekonfiguration, ...)

Diese MIB ist eingebettet in das SNMP Management Framework und entspricht den Anforderungen von SMIv2.
Die Architektur eines SNMP–Frameworks wird an dieser Stelle als bekannt vorausgesetzt und nicht erläutert. Der interessierte Leser wird auf [Stal 93] verwiesen.

3.4.1.3 Die MIB–Struktur

Die Application Management MIB ist in unterschiedliche Gruppen eingeteilt, die wiederum in Tabellen organisiert sind und für das Applikationsmanagement relevante Monitoring– und Kontrollinformationen beinhalten. Die Gruppen sind wie folgt modelliert:

- Die Service–Level Perspektive einer Applikation
  Die zugehörigen Service–Level Tabellen erlauben die Identifikation einer oder mehrerer Instanzen benannter Services eines Systems und die Assoziation laufender Applikationselemente mit diesen Services.
- Informationen über offene Kanäle der Applikation
  Informationen, die von einer Applikation verarbeitet werden, können mit dem Channel–Konzept (siehe Abbildung 3.6) modelliert werden. Zwei Arten von Kanälen sind z.B. Dateien oder Netz-

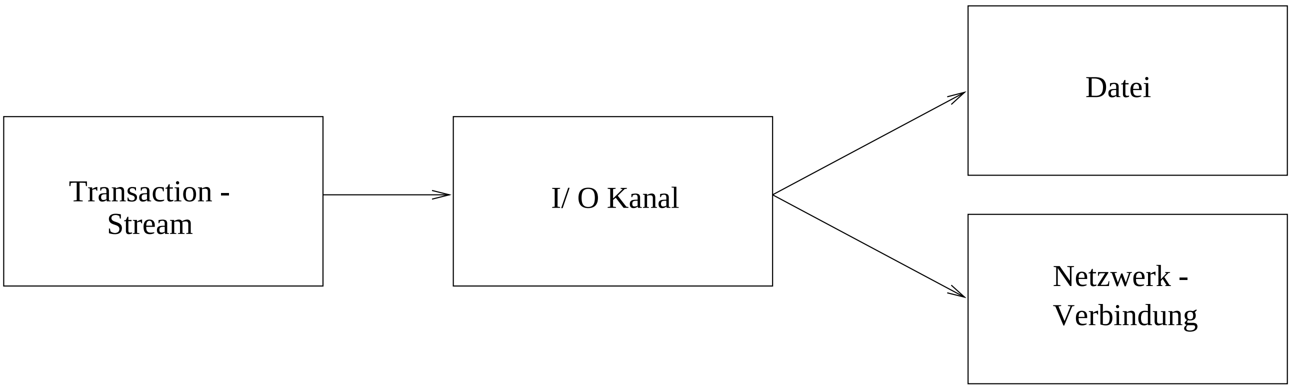

Abbildung 3.6: Das Kanal–Konzept

  werkverbindungen, d.h das zu jedem Eintrag in der Tabelle der offenen Kanäle ein Eintrag entweder in der Tabelle der offenen Dateien oder der offenen Netzwerkverbindungen gehört.
- Historische Informationen über bereits geschlossene Kanäle
  Diese `former channel group` besteht aus mehreren Tabellen und beinhaltet History–Werte zu einer laufenden Applikation oder Service–Instanz.

- Prozess–Status und Kontrollinformationen
  Diese Gruppe besteht aus zwei Tabelle. Die `applEmltStatusTable` beinhaltet wichtige Information über den Resourcenverbrauch der Applikation wie z.B. Anzahl der offene Dateien, Heap–Usage und die letzte Fehlermeldung. Die andere Tabelle ermöglicht einem Benutzer mit entsprechenden Rechten die Kontrolle der Applikation. So kann er z.B. Prozesse beenden und rekonfigurieren.

Die genaue Definition der Management–Application–MIB ist [Kalb 98] zu entnehmen.

3.4.1.4 Beziehungen zu anderen MIBs

Die hier vorgestellte Management–Application–MIB ist, wie bereits erwähnt, aufgesetzt auf einen bereits vorhandenen Satz von früher entwickelten MIBs. Die relevanten Beziehungen werden im folgenden aufgeführt und erörtert:

- System Application MIB [KrSa 98]
  Die System Application MIB definiert Attribute für das Management von Applikationen, die ohne deren Instrumentierung realisiert werden können. Die Spezifikation der Application–Management–MIB erweitert das durch die System Application MIB vorgegebene Framework, da es in der Regel die Instrumentierung der zu verwaltenden Ressource bedingt. Die wichtigste Verbindung zwischen diesen beiden MIBs ist die `sysApplRunElmtIndex`–Variable. Es ist wichtig, daß diese beiden MIBs, die gleichzeitig auf einem Host laufen, konsistent sind um so eine eindeutige Identifizierung der einzelnen Applikationselemente zu gewährleisten.

- Host Resource MIB [Rose 91]
  Die Host Resource MIB bietet Informationen zu Hardware, Betriebssystem und installierter und laufender Software auf einem Host an. Sie besteht im wesentlichen aus drei Hardwaregruppen (`hrSystem`, `hrStorage` und `hrDevice`)und drei Softwaregruppen (`hrSWRun`, `hrSWRunPerf` und `hrSWInstalled`). Von diesen Gruppen sind für die Application–Management–MIB insbesondere die Softwaregruppen von ausschlaggebender Wichtigkeit. Diese definieren Management Informationen zu der Software, die auf dem System installiert ist und ist aufgeteilt in die Bereiche aktuell laufende Software, Performance und installierte Software.
  Auch hier bestehen wieder über diverse Variablen Querverbindungen zur Application–Management–MIB (z.B. muß `hrSWRunIndex` konsistent mit `sysApplRunElmtIndex` sein, ...) die benutzt werden können und beachtet werden müssen.

- Network Services Monitoring MIB [FrKi 98]
  Die Network Services Monitoring MIB ist definiert als ein Basissatz von Attributen zum Management von Netzapplikationen. Die Application–Management–MIB beinhaltet in der Regel nur Informationen bezüglich der verwalteten Ressource und des Systems.

3.4.1.5 Bewertung

Bei der Bewertung dieses Ansatzes der IETF im Kontext mit dem im Rahmen dieser Arbeit gestellten Szenarios fallen folgende Punkte auf:

1. Anwendungsbereich geht weit über das Szenario hinaus (+)
   Ist der Rahmen für die Benutzung solcher MIBs gegeben, können Applikationsmanagementaufgaben sehr leicht erweitert werden. Die Beschränkung auf Monitoring ist nicht gegeben.

2. Eingliederung in das SNMP Framework(+)
   Die Nutzung dieses weitverbreiteten Standards eröffnet weitere Möglichkeiten und erleichtert die Integration in bestehende Managementlösungen.

3. Instrumentierung der Applikationen aufwendig (-)
   Zur Instrumentation der Applikation müssen Monitoring Prozesse und unterschiedliche Agenten implementiert werden. Zusätzlich ist die Nutzung weiterer Interfaces (z.B. AgentX, SNMP–DPI, ...) notwendig.

4. Hoher Ressourcenverbrauch (-)
   Ein Ziel der Implementierung eines Agenten zur Überwachung von SLAs ist eine schlanke Lösung. Es sollen möglichst wenig Ressourcen in Anspruch genommen werden.

5. Keine Korrelation von Sub–Transaktionen möglich (-)
   Die Forderung, komplexe Transaktionen überwachen zu können und deren Antwortzeit auf eventuelle Sub–Transaktionen verteilen, wird von diesem Ansatz nicht erfüllt.

Aufgrund der umfangreichen Implementierung und des hohen Ressourcenbedarfs eignet sich dieser ansonsten hochinteressante Ansatz für das Applikationsmanagement im vorliegende Szenario nicht.

3.4.2 Desktop Management Interface der DMTF

3.4.2.1 Die Desktop Management Task Force

Die Desktop Management Task Force (DMTF) wurde im Mai 1992 von folgenden Firmen gegründet: Digital Equipment Corporation, Hewlett-Packard, IBM, Intel, Microsoft, Novell, SunSoft und SynOptics. Der Zusammenschluß dieser namhaften Firmen und die gemeinsame Arbeit bis heute bezeugt die Akzeptanz des gefundenen Standards (siehe `http://www.dmtf.org/`).
Ziel dieser Gründung war die Entwicklung einer einfachen Lösung für das Management von Desktop–PCs, Notebooks und Servern. Anforderungen aus der Praxis ließen in den letzten Jahren das Management von Applikationen mit in die Forschungen einfließen.

3.4.2.2 Das Desktop Management Interface

Das Desktop Management Interface (DMI) ist ein Framework für das Management von unterschiedlichen Komponenten in Desktop–PCs, Notebooks, Servern und auch Applikationen. Da die DMTF eine neue Infrastruktur für ihr Management–System entworfen hat, wird dieses im Anschluß kurz dargestellt.
Jedes System generiert ein Management Information Format (MIF), das Informationen zu diesem enthält.
Eine zentrale Rolle spielt der DMI Service Provider (SP) auf der verwalteten Plattform, der über die für den entfernten Zugriff auf die MOs benötigten Schnittstellen und Informationen verfügt (Abbildung 3.7). Dieser SP ist ein Programm, das die lokalen Informationen sammelt, die Informationen in der MIF–Datenbank organisiert und Informationen an Manangement–Applikationen auf Anfrage weitergibt. Mittels des Management Interface (MI) kontrolliert es die Kommunikation mit dem Manager, während das Component Interface (CI) den Zugriff auf die MOs ermöglicht. Zusätzlich stellt der SP einen Satz von APIs bereit, die die Entwicklung von Management–Applikationen und die Instrumentierung von Applikationen und Komponenten unterstützen.
Ähnlich einer MIB–Datei stellt eine MIF–Datei (genauere Darstellung einer solchen Datei im folgenden Abschnitt) Informationen bezüglich der durch ein MO angebotenen Attribute dar. Alle MIF–Dateien eines Systems werden durch den SP in einer Datenbank gesammelt und verwaltet.
Zugegriffen werden kann auf einen Service Provider über das MI durch eine auf DMI basierende Management–Applikation. Desweiteren bietet das DMI eine Möglichkeit an, die Informationen per SNMP zugänglich zu machen. Dazu wird in dem DMI–SDK (Software Development Kit) ein `miftomib`–Compiler mitgeliefert, der die MIF–Datei in das SNMP–Format übersetzt. Über einen solchen Mapper kann es z.B. flexiblen Management–Agenten ermöglicht werden, auf eine DMI–Architektur zuzugreifen.
Hier noch einmal die zentralen Komponenten des DMI in einer Übersicht:

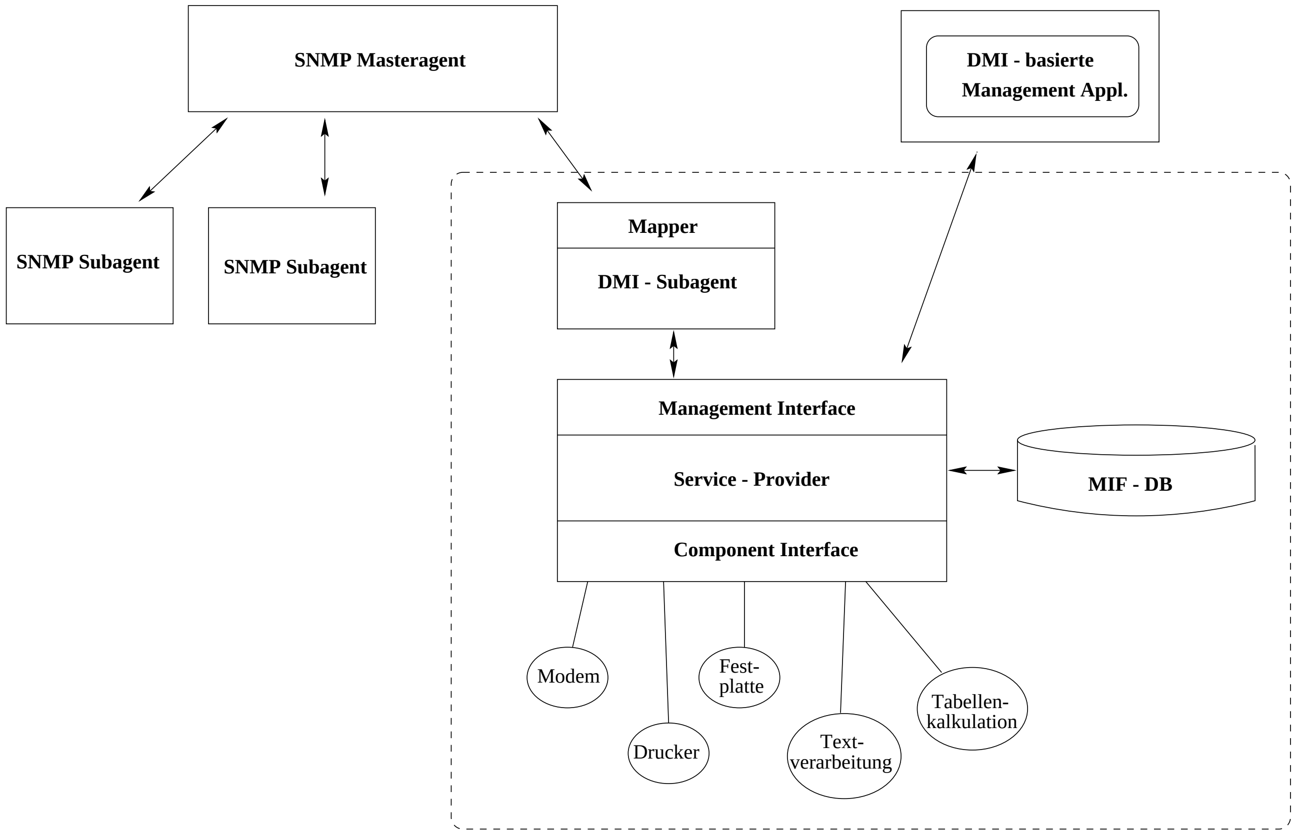

Abbildung 3.7: Die Architektur des Desktop Management Interface (DMI)

- **Service Provider (SP)**
  koordiniert als Teil der DMI–Architektur Anfragen von der Management–Applikation an die Komponenten–Instrumentierung und vermittelt zwischen dem MI und dem CI.

- **Management Interface (MI)**
  ermöglicht Management–Applikationen den Zugang zu den Komponenten–Informationen und registriert diese für die Benachrichtigung bei Events.

- **Component Interface (CI)**
  beschreibt den Zugang zu Management–Informationen und ermöglicht es, eine Komponente zu managen.

- **MIF–Datei**
  beschreibt den Zugang zu Management–Informationen und definiert die mit dieser assoziierten Attribute.

Um sich ein genaueres Bild von der Leistungsfähigkeit dieser Architektur machen zu können wird im folgenden Abschnitt die Darstellung des MOs durch eine MIF–Datei genauer untersucht.

3.4.2.3 Die MIF–Datei

Die verwaltete Information in einem System wird im sogenannten Managed Information Format (MIF) beschrieben. Dieses MIF definiert die MOs und die mit ihnen assoziierte Attribute. Jede Instanz eines MOs muß über eine eigenen MIF–Datei, die seine managebaren Aspekte beschreibt, verfügen.

Schlüsselwort	Teil einer	Beschreibung
`component`	MIF–Datei	definiert eine Komponente; alle anderen Blöcke sind Teil dieser Komponente; es kann nur einen `component`–Block in einer Datei geben.
`path`	Komponente	assoziiert einen String mit Pfadnamen für ein bestimmtes OS (optional)
`group`	Komponente	definiert eine Sammlung von Attributen für eine Komponente
`attribute`	Gruppe	definiert eine Einheit von verwalteter Information innerhalb einer Gruppe.
`table`	Komponente	definiert eine oder mehr Instanzen einer Gruppe unter Verwendung bereits vorhandener Definitionen

Tabelle 3.1: Die Blöcke einer MIF–Datei

Eine MIF–Datei ist in Blöcke aufgeteilt, die in der Tabelle 3.1 beschreiben werden. Aus dieser Gliederung ergibt sich das folgende Grundgerüst:

```
start component
  start path
  end path
  start group
    start attribute
    end attribute
  end group
  start table
  end table
end component
```

Beispielhaft soll dargestellt werden, wie sich die Definition eines Attributes durchführen läßt. Dazu wird angenommen, daß eine MIF–Datei erstellt wird, die unter anderem den Namen einer Applikation und deren Version bereitstellen soll. Zusätzlich soll die Nutzung eines SNMP–Mappers möglich sein. Die dazu nötige Codesequenz sieht wie folgt aus:

```
Start Group
  Name = "Software Template"
  Class = "DTS|BasicSoftware|001"
  Key = 1 // key on Product Name
  Pragma = "SNMP:1.2.3.4.5.6"
  Start Attribute
    ID= 1
    Name = "ApplicationName"
    Description = "The name of the Application"
    Storage = Common
    Type = String(64)
  End Attribute
  Start Attribute
    ID = 2
    Name = "ApplicationVersion"
    Description = "The Application's version number"
    Type = String(32)
```

```
    Value = ""
  End Attribute
End Group
```

Diese Einführung in das Desktop Management Interface und das damit verbundene Management Information Format soll für den Kontext dieser Arbeit genügen. Tiefergehende Informationen sind in [Sol] und auf der Homepage der DMTF (`http://www.dmtf.org`) zu finden.

3.4.2.4 Bewertung

Die Ansätze der DMTF haben sich in den letzten Jahren nicht zuletzt wegen derer namhaften Mitglieder zu Standards entwickelt. Trotzdem werden sie in der Praxis noch lange nicht akzeptiert und immer noch zu selten realisiert.

1. Ein Management–Framework für Notebooks, PCs und Server(+)
   Die DMTF sieht für die DMI eine breite Schnittstelle für unterschiedlichste Systeme vor. Dies macht dieses System zu einem vielseitigen Werkzeug für das Netz– und Komponenten–Management.

2. Management Information Format (+)
   Die einheitliche Beschreibung der MOs erleicht Anbietern, ihre Produkte managebar zu machen und Entwicklern, diese Fähigkeiten zu nutzen.

3. DMI–Standard bis heute selten realisiert (-)
   Wie die in diesem Kapitel beschriebenen Ansätze auch, wird ist die DMI derzeit in nur wenige Produkte integriert, da die Software–Hersteller den Mehraufwand scheuen oder sich nicht auf eine Spezifikation festlegen wollen.

4. Keine Korrelation von Sub–Transaktionen (-)
   Die Forderung, komplexe Transaktionen überwachen zu können und deren Antwortzeit auf eventuelle Sub–Transaktionen verteilen, wird von diesem Ansatz nicht erfüllt.

3.4.3 Application Response Measurement API der CMG

3.4.3.1 Die Computer Measurement Group

Die Computer Measurement Group (CMG) (siehe `http://www.cmg.org`)ist eine nicht–kommerzielle und weltweite Organisation bestehend aus Unternehmen und Fachleuten, die sich mit Messungen und Management von Computer–Systemen beschäftigen. Die meisten CMG–Mitglieder interessieren sich für die Messung und Maximierung von Leistungsfaktoren wie Durchsatz und Antwortzeit. Ziel ist es, bestehende Systeme so zu verbessern und neue so zu entwerfen, daß diese für eine angemessene Leistungsfähigkeit benötigten Ressourcen zu adäquaten Kosten angeboten werden können.
Aufgegliedert ist die CMG in sogenannte Working Groups. Eine dieser Gruppen, die ARM–Working–Group, hat das in diesem Abschnitt beschriebene Konzept entwickelt. Bekannte Mitglieder dieser Gruppe sind z.B. BMC, Candle, HP, Landmark, Sun, Tivoli/IBM, Oracle, Unify, SAS, SES, Boeing, Citicorp, Wells Fargo, ...

3.4.3.2 Das Konzept

Der Kerngedanke des Ansatzes der CMG ist Markierung der für die zu überwachenden Transaktionen zuständigen Codesequenzen (siehe `http://www.hp.com/go/arm`). Der Aufruf entsprechender Methoden der ARM API am Anfang und am Ende der Transaktionen ermöglicht es Agenten, die ebenfalls

`arm_init`	Initialisiert die ARM–Umgebung für die Applikation.
`arm_getid`	Mit `arm_getid` wird eine Transaktion in ARM registriert und bezeichnet.
`arm_start`	`arm_start` signalisiert den Beginn der Transaktion.
`arm_update`	Diese optionale Funktion kann nach `arm_start` und vor `arm_stop` aufgerufen werden, um genauere Informationen über die Transaktion zu erhalten.
`arm_stop`	`arm_stop` signalisiert das Ende der Transaktion.
`arm_end`	Am Ende der Applikation kann `arm_end` aufgerufen werden, um z.B. allokierten Speicher wieder freizugeben.

Tabelle 3.2: Übersicht über die ARM–Funktionen

mit der ARM API bestückt sind, diese Applikation zu beobachten. Die Architektur eines solchen Ansatzes ist in Abbildung 3.8 dargestellt.
Die ARM API [ARM 97] besteht aus einem Satz von Funktionen, die in einer Shared Library zur

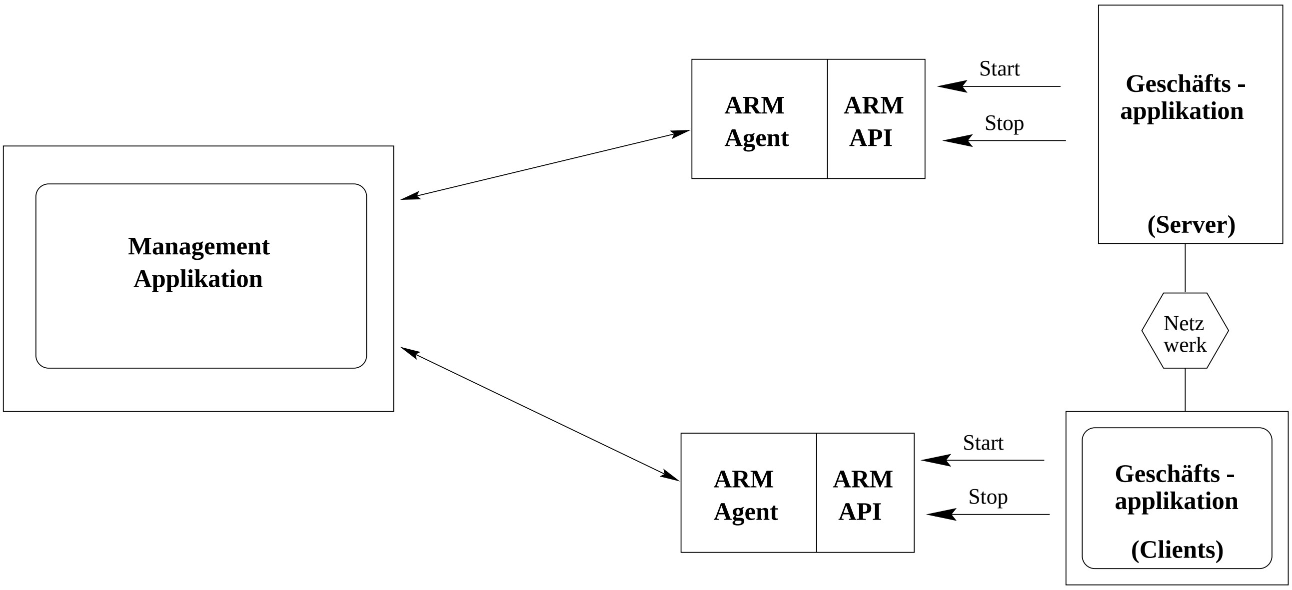

Abbildung 3.8: Die Architektur der ARM–API

Verfügung gestellt werden. Jeder Agent verfügt über eine eigene Implementierung dieser Library. Die zur Verfügung stehenden Funktionen werden in Tabelle 3.2 kurz dargestellt. Nun soll anhand eines Beispieles dargestellt werden, wie eine Applikation instrumentiert wird. Dazu wird angenommen, eine Geschäftsapplikation mit diversen Server–Anfragen soll so bestückt werden, daß einzelne Transaktionen gemessen werden können.
Vereinfacht würde das wie folgt aussehen:

```
APPL = arm_init("Application Name", "User Name")
    A = arm_getID("Transaction A", APPL)
    B = arm_getID("Transaction B", APPL)
    C = arm_getID("Transaction C", APPL)

    loop until program ends
```

```
        arm_start(A)
            arm_start(B)
            // do some work
            arm_stop(B, status)

            arm_start(C)
            loop until transaction ends
                // do some work
                arm_update(C)
            end loop
            arm_stop(C, status)

        arm_stop(A, status)
    end loop
arm_end
```

Zu Beginn der Applikation wird die Methode `arm_init` mit den Parametern `Application Name` und `User Name` aufgerufen. Ziel dieser Initialisierung ist die Vereinfachung der Organisation der später gemessenen Werte. Diese Methode gibt einen Identifikator zurück, der durch den ARM–Agenten vergeben wird und eine Klasse von Transaktionen durch ihre Beziehung zu einer Applikation definiert.
`arm_getID` wird typischerweise in der Initialisierungsphase einer Applikation aufgerufen, um einen Transaktionsnamen zu erstellen. Die Kombination von Transaktionsnamen und Applikationsidentifikator ergibt einen eindeutigen Schlüssel zur Identifikation einer Transaktion. Der Rückgabewert der Methode `arm_getID` stellt einen eindeutigen Identifikator für eine Transaktion dar und wird später den einzelnen `arm`–Aufrufen als Parameter übergeben.
`arm_start` signalisiert dem Agenten nun den Beginn der zu messenden Transaktion. Hierbei können sich diese Transaktion ineinander verschachteln. `arm_update` stellt eine Methode dar, deren Benutzung optional ist. Sie teilt dem Agenten mit, daß die betreffende Transaktion noch läuft. Typischerweise wird sie bei längeren Transaktionen eingesetzt, und zwar in bestimmten Zeitintervallen (Heartbeat, z.B. einmal pro Minute) oder nach dem Abarbeiten einer bestimmten Menge der Transaktion (z.B. nach je 1000 abgearbeiteten Datensätzen).
`arm_stop` indiziert das Ende einer Transaktion und kann von einem neuen Prozess oder Thread bekanntgegeben werden. Diese Methode übergibt neben dem Transaktionsbezeichner auch eine Statusmitteilung, die folgende Werte annehmen kann:

- **GOOD**
  Die Transaktion wurde erfolgreich ausgeführt und der beabsichtigte Dienst wurde erbracht.

- **ERROR**
  Die Transaktion wurde mit einem Fehler beendet und der beabsichtigte Dienst wurde nicht erbracht.

- **ABORT**
  Die Transaktion wurde nicht beendet und der Dienst wurde nicht erbracht. Ein möglich Fehler könnte ein Time Out Fehler o.ä. sein.

`arm_end` teilt dem Agenten mit, daß die Applikation keinen Aufruf der ARM–API mehr tätigen wird. Diese Methode wird typischerweise beim Beenden der Applikation aufgerufen. Alle nicht gestoppten Transaktionen dieser Applikation werden durch den Agenten verworfen.

3.4.3.3 Korrelation von Subtransaktionen

Eine der hervorstechenden Fähigkeiten der ARM–API ist die Korrelation von Subtransaktionen. Wie bereits in der Einführung dieses Kapitels beschrieben wurde, ist es sehr nützlich, aufgerufene Subtransaktionen (Kinder) einer Transaktion (Elter) zuordnen zu können. Insbesondere da der Abbruch einer solchen Kindtransaktion den Abbruch der gesamten Transaktion zur Folge haben kann, die die Zuordnung von Fehlern von ausserordenlichem Wert.
Aus diesem Grunde bietet nun die ARM–API die Möglichkeit, bei dem Aufruf der Methode `arm_start` einen sogenannten Korrelator anzufordern, den sie dann mit den Transaktionsparameter der Applikation an ihre Subtransaktion weitergibt.
Anhand der Abbildung 3.9 wird dieser Mechanismus hier kurz erläutert.

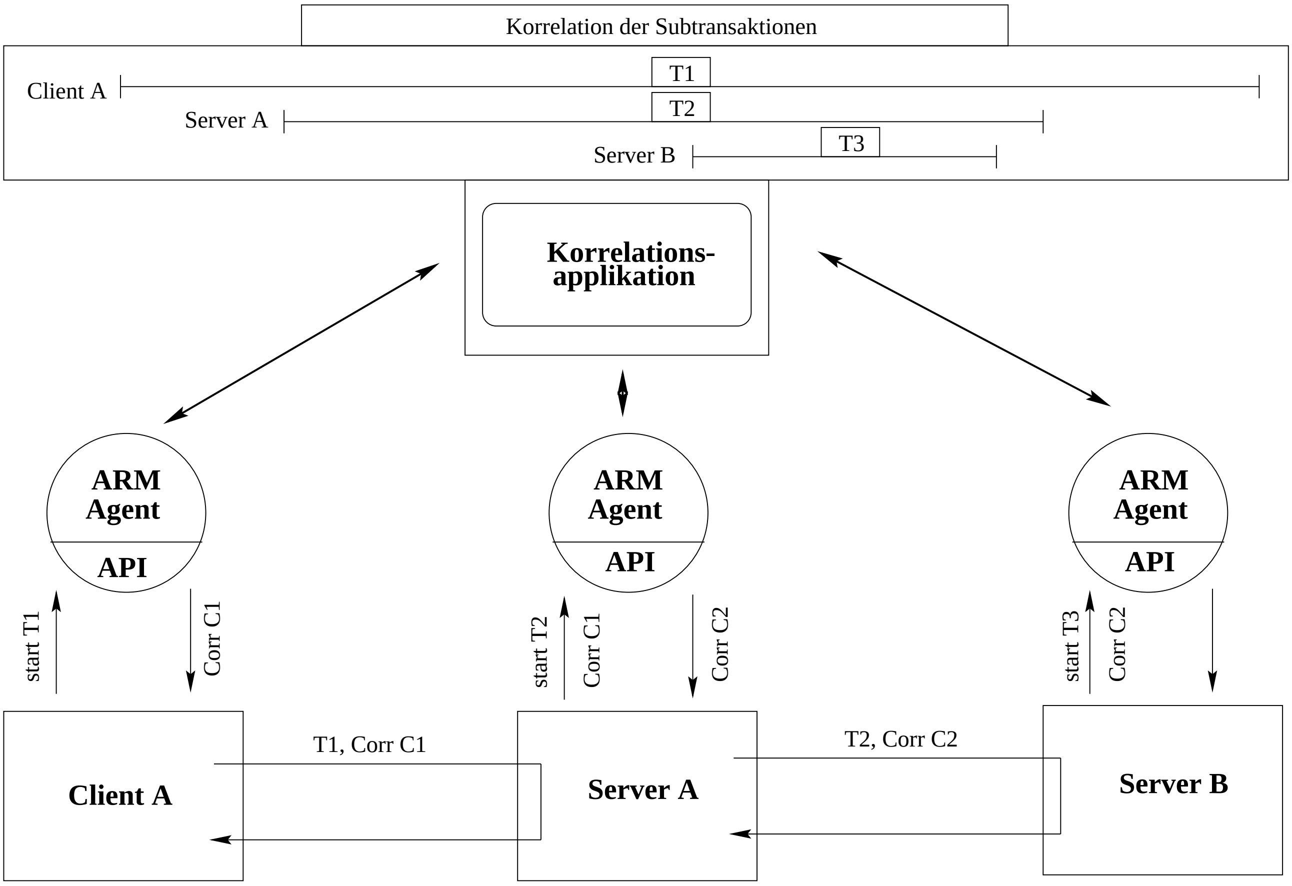

Abbildung 3.9: Korrelation von Subtransaktionen durch die ARM–API

1. Client A (Fahrzeughändler) startet eine Transaktion `T1` (Lieferzeitanfrage für ein Fahrzeug). `arm_start` fordert dazu einen Korrelator an und erhält `C1`.

2. Client A sendet eine Anfrage an Server A (Fahrzeughersteller) und übergibt dabei den Korrelator `C1`.

3. Server A startet eine Transaktion `T2` (Lieferzeitanfrage an Reifenlieferant). Dabei übergibt er `C1` als Hinweis auf die Elterntransaktion und fordert einen neuen Korrelator an. Er erhält `C2`.

4. Server A sendet eine Anfrage an Server B und übergibt dabei den Korrelator `C2`.

5. Server B startet eine Transaktion `T3` und übergibt den Korrelator `C2` als Hinweis auf die Elterntransaktion.

6. `T3` stoppt, `T2` stoppt, `T1` stoppt.

Auf diese Weise läßt sich feststellen, welchen Anteil `T2` und `T3` an der Antwortzeit von `T1` haben.

3.4.3.4 Bewertung

1. Flexibilität (+)
   Jeder Agent (jedes Agentensystem) kann ohne Änderung der Applikation genutzt werden, wenn diese einmal instrumentiert ist.

2. Genaue Abgrenzung der Transaktionen (+)
   Transaktionen können beliebig weit oder eng definiert werden. Die relevanten Transaktionen können exakt eingegrenzt und gemessen werden.

3. Korrelation von Subtransaktionen möglich (+)
   Im Gegensatz zu den vorhergehenden Ansätzen bietet die ARM–API einen Mechanismus zur Korrelation von Sub–Transaktionen.

4. Beschränkung auf Transaktionszeit–Messung (-)
   Die ARM–API ist bewußt einfach gehalten, um die Akzeptanz unter den Software–Herstellern zu erhöhen. Allerdings beinhaltet dieser ansonsten positive Aspekt den Nachteil, für sonstige Management–Anfoderungen nicht geeignet zu sein.

Abschließend kann festgestellt werden, daß die ARM–API den Anforderungen des Anwendungsszenarios entspricht, auch wenn sich ihre begrenzte Funktionalität für die Nutzung im Rahmen zukünftiger Aufgaben als zu gering erweisen kann.

3.5 Zusammenfassung

Allen Ansätzen gemein ist das Ziel, durch einen Standard proprietäre Schnittstellen zu ersetzten und Hersteller–übergreifende Management–Möglichkeiten für Applikationen zu schaffen. Um die Chance einer breiten Unterstützung durch Software–Hersteller zu erhöhen, wurden alle Ansätze unter dem Design–Kriterium der möglichst einfachen Implementierung in bestehende und neue Software–Systeme entwickelt.
Während die ARM–API gezielt für die Messung von Antwortzeiten in Appliaktionen einsetzbar ist, verfügen die übrigen Werkzeuge auch über die Möglichkeit, andere Funktionsbereiche abzudecken.
Im Kontext dieser Arbeit wird für eine Implementierung die ARM–API gewählt, weil sie einen Mechanismus zur Korrelation von Sub–Transaktionen bietet. Dieser schlanke und zielgerichtete Ansatz entspricht den Anforderungen voll.
Es sei an dieser Stelle darauf hingewiesen, daß die Nutzung der ARM–API in Verbindung mit der Application Management Specification (AMS) vorgesehen ist. Die AMS ergänzt den engen Funktionsbereich der ARM, wird allerdings an dieser Stelle nicht weiter vorgestellt, da sie für die Antwortzeitmessung keine weiteren Funktionalitäten enthält.

Kapitel 4

Konzept eines SLM unter JDMK

In diesem Kapitel werden die Erkenntnisse und Folgerungen der vorhergehenden Kapitel in einen konzeptionellen Entwurf der Architektur eingebracht.
Dabei sind insbesondere die Anforderungsanalyse in Kapitel 2 und die Bewertung der einzelnen Werkzeuge im Kapiteln 3 von besonderer Bedeutung.

4.1 Entwicklungsumgebung

In einem ersten Schritt ist festzustellen, ob die für diese Arbeit vorgegebenen Werkzeuge den Anforderungen aus dem Anwendungsszenario genügen. In den folgenden Abschnitten werden nun die Sprache Java, das JDMK als Werkzeug für die Umsetzung einer Agentenarchitektur und die Nutzung bestimmter Protokolle als Grundlage für die Kommunikation zwischen Agenten und Manager evaluiert.

4.1.1 Implementierungssprache

Als Implementierungssprache wird Java in Verbindung mit dem JDMK verwendet. Die wichtigsten Eigenschaften werden hier noch einmal bezug nehmend auf den Anforderungskatalog in Abschnitt 2.2.4 kurz zusammengefaßt:

- **Plattformunabhängigkeit des Bytecodes**
  Für den Einsatz des Agenten in dem gegebenen heterogenen Netzwerk ist die Lauffähigkeit des Codes auf Plattformen unterschiedlichster Beschaffenheit unverzichtbar.

- **Objektorientiertheit der Sprache**
  Dynamische Erweiterbarkeit, Sicherheit und Kommunikationsfähigkeit werden durch eine strukturierte, modulare Implementierung erleichtert.

- **Automatisierte Serialisierung**
  Der Implementierungsaufwand wird durch die Bereitstellung von Serialisierungsmechanismen deutlich reduziert. Die Kommunikation kann durch den Austausch ganzer Objekte vereinfacht werden.

- **Threads**
  In einzelnen unabhängig voneinander ablaufenden Prozessen können parallel durch unterschiedliche M–Beans unterschiedliche Aufgaben wahrgenommen werden.

- **Applets**
  Die Nutzung von Browsern als Laufzeitumgebung für Java–Applikationen ermöglicht das Administrieren der Agenten über das World Wide Web.

- **SNMP–Management–Schnittstellen**
  Das Bereitstellen einer SNMP–Management–Schnittstelle durch das JDMK ermöglicht die Überwachung einzelner Netzwerk–Komponenten. Insbesondere kann so im gegebenen Szenario ein Router überwacht werden.

- **Eignung für das Applikationsmanagement**
  Durch das Beans–Konzept lassen sich Java–Applikationen sehr leicht für Messungen durch einen JDMK–Agenten instrumentieren. Nicht Java–Applikationen können z.B. über die bereits erwähnte SNMP– Schnittstelle und die Application MIB oder mit Hilfe des Java Native Interfaces (JNI) und ARM verwaltet werden.

- **JDBC, CORBA–Schnittstelle, JNI, ...**
  Durch verschiedene Mechanismen wird die Integration in bestehende Managementlandschaften erleichtert.

Zur Implementierung wird das Java Development Kit 1.1.7b von Sun Microsystems verwendet, da die aktuelle Version 1.2 nicht mit dem JDMK3.0beta kompatibel ist.

4.1.2 Evaluierung des JDMK

Wie bereits in Kapitel 3.1 ausgeführt, erfüllt das Konzept der Flexiblen Management Agenten die durch das Anwendungsszenario gestellten Forderung. In diesem Abschnitt soll nun kurz geprüft werden, ob sich mit dem JDMK ein Agentensystem entwickeln läßt, das auf diesem Konzept basiert. Eine ausführliche Evaluierung ist in Diplomarbeit von Harald Knöchlein [Knoe 99] zu finden.
Grundlage für die Erstellung einer Architektur, die den Anforderungen der Flexiblen Management Agenten entspricht, ist die Realisierbarkeit ihrer wichtigsten Komponenten:

- **Agenten**
  Die Realisierung von herkömmlichen Agenten wie z.B. SNMP–Agenten ist durch die Bereitstellung der entsprechenden Adapter gegeben. Es ist sowohl möglich, eigene SNMP–Agenten zu entwickeln, als auch bestehende zu nutzen.

- **Flexible Management Agenten**
  Flexible Management Agenten lassen sich mit Hilfe des Core Management Frameworks leicht implementieren. Durch die Möglichkeit, M–Beans — und somit Funktionalitäten — während der Laufzeit hinzuzufügen oder zu entfernen, ist eine Hauptforderung des FMA–Konzeptes erfüllt.

- **Manager**
  Durch das JDMK wird die Entwicklung eigener Manager insbesondere durch die Bereitstellung von Protokoll–Adaptern unterstützt. Komponenten für grafische Benutzeroberflächen werden über das JDK bereitgestellt.
  Explizit sei hier der HTML–Adapter genannt, der auf der Agentenseiten einen kleinen Web–Server bereitstellt und die Informationen des Agenten in Form einer HTML–Seite präsentiert. Übertragen wird diese Seite natürlich mittels HTTP.

Die in Abschnitt 3.2.2 dargestellten Dienste werden alle durch das JDMK angeboten oder können indirekt implementiert werden. Auch in Bezug auf die Organisation der Agenten sind mit Hilfe des Objektnamen alle Forderungen erfüllt.
Eine umfangreichere Evaluierung des JDMK wurde impliziet in dem Abschnitt 3.2 vorgenommen. Zur Implementierung wird das Java Dynamic Management Kit 3.0beta benutzt.

4.1.3 Auswahl der Protokolle

Durch das JDMK, das als Entwicklungswerkzeug festgelegt ist, werden eine Reihe von Kommunikations–Protokollen in Form von Adaptern vorgegeben. In diesem Abschnitt werden diese kurz dargestellt und Vor– und Nachteile abgewogen.
Die Entscheidung für oder gegen die Nutzung eines bestimmten Protokolles ist in erster Linie von der spezifischen Umgebung abhängig. Da sich allerdings in den meisten Netzen die wichtigsten Protokolle als Standard etabliert haben, sind weitere Aspekte bei der Auswahl zu untersuchen.
Grundlage für diese Diskussion bilden die durch das JDMK zur Verfügung gestellten Adapter. Dadurch fällt z.B. das *Knowledge Query and Manipulation Protocol (KQML)* aus dieser Darstellung heraus, obwohl es für die Kommunikation mit und zwischen flexiblen Agenten entwickelt wurde. Es ist allerdings im Rahmen des JDMK möglich einen neuen Adapter zu implementieren, der dieses Protokoll nutzt.
Die auf TCP bzw. HTTP basierende *Remote Method Invocation (RMI)* ermöglicht Entwicklern das entwerfen verteilter Java–Anwendungen, in denen er Methoden entfernter Objekte aufrufen kann. Das dieser Mechanismus nur zwischen Java–Applikation funktioniert, stört in dem gegebenen Szenario nicht. Kritisch zu betrachten ist jedoch, daß Methodenaufrufe nicht verschlüsselt werden, was zur Folge hat, daß die als Aufrufparameter übergebenen Objekte direkt auslesbar sind. Allerdings bieten die durch Java bereitgestellten Sicherheitsmechanismen und –bibliotheken eine Vielzahl von Möglichkeiten, dem entgegenzuwirken.
Das *Hypertext Transfer Protocol (HTTP)* bildet die Basis des World Wide Web, und ist als solche weit verbreitet. Es ist in der Regel auf TCP oder UDP gesetzt und benötigt nur wenig Prozessor–Leistung, ist allerdings weniger effizient bei der Übertragung großer bzw. vieler kleiner Dateien, da es für jedes Paket eine neue TCP–Verbindung aufbauen muß. Sicherheitsaspekte wurden bei der Grundimplementierung des HTTP nicht beachtet.
Ein sehr modernes Protokoll stellt das JDMK mit dem *Secure Sockets Layer Protocol (SSL)* [DiAl 99], das sichere Kommunikation im Internet gewährleisten soll, bereit. Dieses Protokoll erlaubt Client/Server–Applikationen miteinander zu kommunizieren und dabei Lauschangriffe und Nachrichtenfälschungen von dritten auszuschließen. Basis dafür bildet ein auf einem zuverlässigem Protokoll wie TCP aufgesetztes SSL Record Protocol, daß in erster Linie der Zusammenführung anderer Protokollblöcke wie dem SSL Handshake Protocol dient. Dieses Handshake Protocol ermöglicht es dem Client und dem Server, sich vor dem Austausch von Daten zu authentifizieren und einen Algorithmus für die Verschlüsselung der Übertragung auszuhandeln. Vorteil dieses Protokolls ist, daß höher angesiedelte Protokolle wie das HTTP SSL nutzen können, da es für sie transparent ist. Bei der Entwicklung des SSL–Protokolls wurden folgende Ziele definiert [SSL]:

1. **Kryptographische Sicherheit**
   SSL soll dazu benutzt werden, eine sichere Verbindung zwischen zwei Parteien herzustellen.

2. **Interoperabilität**
   Unabhängige Programmierer sollen unter Nutzung des SSL Applikationen entwickeln können, die ohne Kenntnis des Partners kryptographische Informationen wie z.B. Parameter und Algorithmen austauschen können.

3. **Erweiterbarkeit**
   Neue Sicherheitsmechanismen sollen jederzeit eingearbeitet werden können.

4. **Effizienz**
   Kryptographische Mechanismen benötigen in der Regel eine hohe Prozessorleistung, die es zu minimieren galt.

Mit dem *Internet Inter–Orb Protocol (IIOP)* stellt das JDMK nicht nur ein Protokoll, sondern eine ganze Architektur zur Verfügung, und zwar die Common Object Broker Architecture (CORBA).

Diese Architektur bietet viele Dienste an, die jedoch an anderer Stelle des JDMK bereits enthalten sind. Zudem benötigt sie auch sehr viele Ressourcen, z.B. für einen Name–Service.

Zusammenfassend ist festzustellen, daß sich insbesondere die RMI unter den genannten Sicherheits–Vorbehalten für die Kommunikation zwischen Java–Komponenten eignet. Da SSL über TCP eine sichere Kommunikation gewährleistet, ist der Einsatz dieses Protokolles insbesondere in gefährdeter Umgebung, z.B. im Extranet , vorzuziehen.

4.2 Struktur der konkreten Managementumgebung

In diesem Abschnitt wird nun herausgearbeitet, wie sich ein Agentensystem in die bestehende Infrastruktur der DeTeSystem einordnen läßt. Anhand des Anwendungsszenarios um den Fahrzeughersteller wird anschließend eine Struktur dargestellt, in der die Vielzahl der Agenten mit ihren unterschiedlichen Aufgaben organisiert und verwaltet werden können. Abschließend ist die Frage zu klären, welche Komponente in diesem System welche Funktionaliät erhalten soll.

4.2.1 Einordung in das Anwendungsszenario

In diesem Abschnitt soll nun dargelegt werden, aus welchen Komponenten sich so ein Agentensystem zusammensetzten muß, und wie diese in einer Architektur wie die der DeTeSystem anzuordnen sind. Aufbauend auf die Ausführungen im Kapitel 3.1 setzt sich das System aus folgenden Komponten zusammen:

1. **Manager**
   Der Manager ist die Instanz in einem Agentensystem, die über die höchsten Rechte verfügt, an der die Daten zusammenlaufen und von der aus Einfluß auf das gesamte System genommen werden kann. Der Manager setzt sich in einem verteilten System explizit aus einem laufenden Prozess (Programm) und einer grafischen Benutzeroberfläche (z.B. über Web–Browser oder X–Window) zusammen. Des weiteren muß ein Manager eine Möglichkeit haben, größere Datenmengen zu verwalten. Dies könnte durch ein Datenbanksystem gewährleistet werden.

2. **Agenten**
   Im Gegensatz zum FMA ist ein einfacher Agent eher passiv und erfüllt selten mehr als eine Aufgabe. So stellt z.B. ein SNMP–Agent eine einfache Management–Schnittstelle bereit.

3. **Flexible Management Agenten(FMA)**
   Eine flexibler Management Agent verfügt über ausreichend Management–Intelligenz, um seine Aufgabe ohne permante Steuerung durch einen Manager zu erfüllen.

Basierend auf diesen Komponenten läßt sich nun das in Abbildung 4.1 dargestellte Konzept für eine Management–Architektur zur Antwortzeitüberwachung aufbauen.
Der *Manager* wird auf einer zentrale Komponente der DeTeSystem plaziert, zu der sämtliche Agenten bei geschlossener Verbindung Zugriff haben. Dies ist in Anlehnung an die bestehende Architektur eine Management–Server im Service Rechenzentrum der DTS. Hierher werden die Meßdaten der Agenten übertragen. Zur Verwaltung der Messdaten muß der Manager eine Datenbank zur Verfügung stehen. Grafische Auswertungen, Reports und sonstige Berechnungen können entweder direkt durch die Management–Applikation oder durch zusätzliche Applikation angeboten werden.
Bedient wird der Manager in einem der SMC der DeTeSystem. Dazu kann dann entweder ein einfacher Web–Browser benutzt werden, der über eine entsprechende Schnittstelle auf den Manager zugreift, oder die grafische Benutzeroberfläche wird umgeleitet.

Agenten werden immer dort eingesetzt, wo sie als Schnittstelle erforderlich sind. Dies ist z.B. notwendig, wenn auf MIBs zugegriffen werden muß, oder wenn ARM zum Einsatz kommt. Um die Möglichkeit zu nutzten, die Zusammensetzung von Transaktionszeiten (Antwortzeiten) zu analysieren, ist es notwendig, neben den Applikationen der Händler auch die Applikationen im Rechenzentrum des Fahrzeughersteller zu instrumentieren. Wie in Abschnitt 3.4.3 dargestellt wird, ist dies einfach zu realisieren und aussagekräftig für bei der Fehlersuche.

Flexible Management Agenten kommen nun insbesondere dort zum Einsatz, wo die permanente Verbindung zum Manager nicht gewährleistet ist, also wo eine autonome Datensammlung erforderlich ist. Dies ist im Anwendungsszenario bei den Händlern der Fall ist. Diese FMAs führen selbständig Antwortzeit–Messungen durch, um so den Service Level aus Sicht der Endbenutzer zu überwachen. Sie entscheiden selber wann eine Übertragung der Daten zum Manager möglich ist. Desweiteren ist ihr Einsatz immer dort sinnvoll, wo Meßdaten von mehreren Agenten ähnlich dem Multiplexerprinzip zusammengefaßt werden können, um so die Netzbelastung zu reduzieren.

Mit den so verteilten Komponenten kann ein auf flexiblen Management–Agenten basierendes verteiltes Management–System realisiert werden. Die in den Händlerlokationen plazierten FMAs beinhalten die notwendige Intelligenz, autonom zu handeln und selbständig Verbindung zum Manager aufzubauen. Einfache Agenten dienen diesen FMAs als Werkzeug zur Meßdaten–Sammlung (z.B. ARM–Agenten) und zum Auslesen wichtiger Systeminformationen wie z.B. den Status der ISDN–Verbindung.

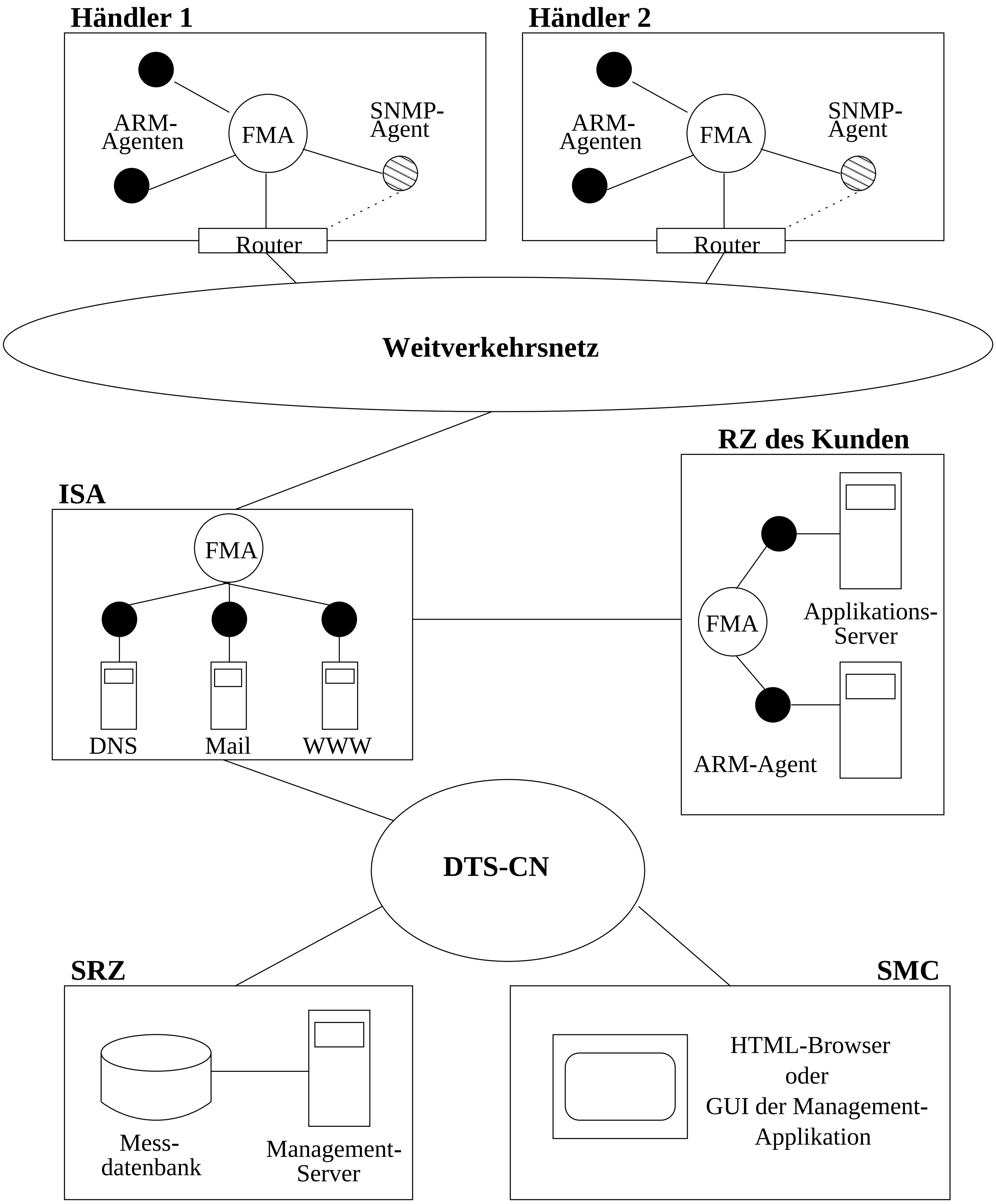

Abbildung 4.1: Konzept einer Management–Architektur zur Antwortzeitüberwachung

4.2.2 Organisation der Agenten

In Abschnitt 3.1.3 wurde die Forderung des Modells der flexiblen Management Agenten dargestellt, Agenten in Domänen und Gruppen gemäß Abbildung 3.3 zu strukturieren.
In dem gegebenen Szenario ist es sinnvoll, Domänen unter Berücksichtigung regionaler Aspekte zu definieren. So können z.B. die Agenten in den Filialen in Nordrhein–Westfalen in einer so bezeichneten Domäne zusammengefaßt werden.
Neben dieser eher organisatorischen Unterteilung in Domänen sollte es eine weitere unter Berücksichtigung funktioneller Eigenschaften geben. Diese Aufteilung in Gruppen spiegelt in einer komplexer werdenden Architektur die Rollen der einzelnen Agenten wieder. So kann es Agenten geben, die bestimmte Messungen vornehmen, an unterschiedlichen Arten von Lokationen plaziert sind (normadische Systeme, große Niederlassungen, ...) oder unterschiedliche Aufgaben erfüllen (messen, auswerten, steuern, ...).
Abschließend muß es ein Merkmal geben, das den einzelnen Agenten benennt und identifiziert. Hierzu kann im einfachsten Fall eine fortlaufende Nummerierung benutzt werden, aber auch eine weitere Verfeinerung des regionalen Konzepts durch die Spezifikation der Stadt und der Filialbezeichnung. Auch die IP–Bezeichnung des Agenten–Hosts wäre hier sinnvoll.
Eine Strukturierung der Agenten, also eine logische und organisatorische Anordnung reduziert die Netzlast durch Eingrenzung der notwendigen Kommunikation und ermöglicht das Delegieren der Aufgaben an die jeweiligen Agenten.
Folgende Strukturen lassen sich in einer solchen Organisation realisieren:

1. Peer to Peer Ansatz

2. Hierarchischer Ansatz

Die Hauptcharakteristik des *Peer to Peer* Ansatzes ist die Tatsache, daß es keine Hierarchie gibt. Alle Agenten können mit der Managementplattform und auch untereinander frei kommunizieren. Die Architektur ist in Abbildung 4.2 vereinfacht dargestellt.
Dieser Ansatz hat viele Vorteile. Er bietet dem System die höchstmögliche Flexibiliät aufgrund der

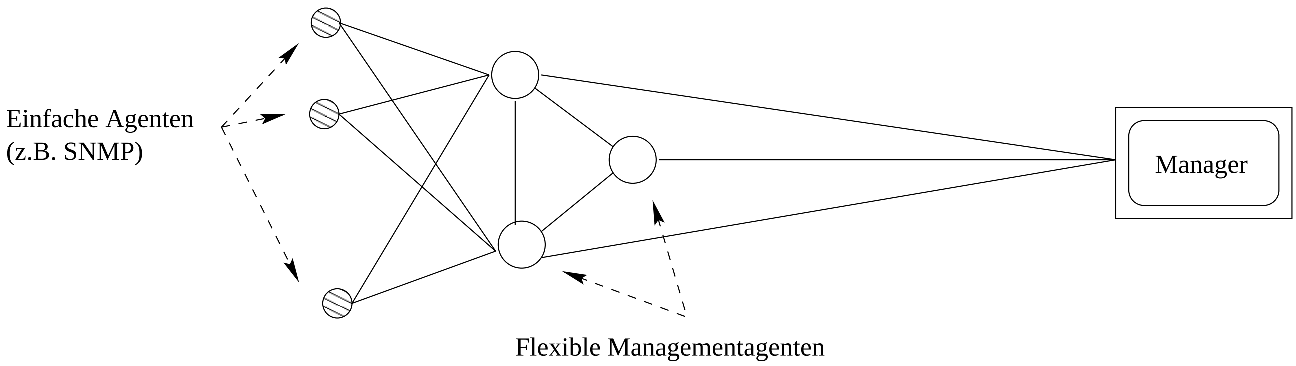

Abbildung 4.2: Peer to Peer Ansatz für die Organisation eines Agentensystems

kompletten Kommunikationsmöglichkeit. Der Ausfall einer einzigen Komponente würde das System nur begrenzt stören. Nachteil ist der hohe Managementaufwand, der mit diesem Ansatz verbunden ist. Jeder Agent muß durch den Manager gesteuert werden und die Kommunikationintensität ist deutlich höher als in dem *hierarchischen Ansatz*, der in Abbildung 4.3 schematisch dargestellt ist. In diesem Modell ist die Kommunikation strikt auf Agentenpaare eingeschränkt, die in der Hierarchie direkt benachbart sind. Vorteile dieser Architektur sind die bessere Strukturierung der Agenten und der damit verbundene verminderte Managementaufwand, eindeutige Kommunikationswege und Entlastung des

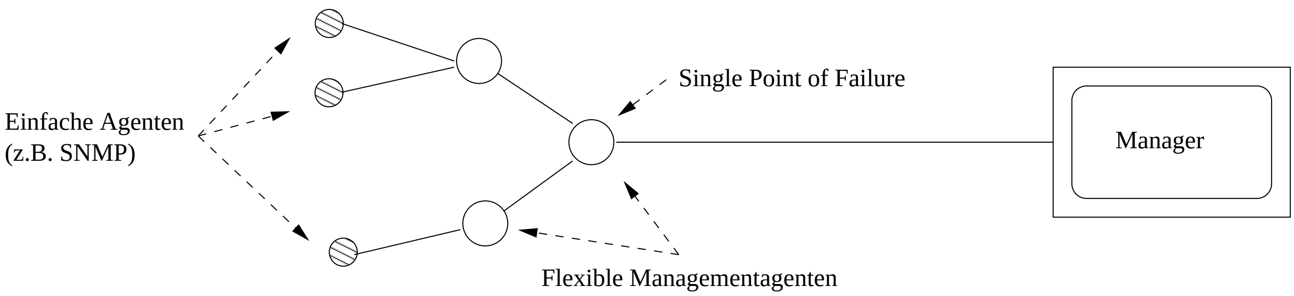

Abbildung 4.3: Hierarchischer Ansatz für die Organisation eines Agentensystems

Netzes in der Umgebung des Managers. Nachteile sind die Anfälligkeit des Systems auf den Ausfall eines einzelnen Agenten (Single Point of Failure) und geringer Flexibiliät. Eine ausführliche Betrachtung dieser Thematik ist in [Moun 97] nachzulesen.
Bei der Organisation eines Agentensystems ist ein schnelles Wachsen der Komplexität und des Aufgabenspektrums einzubeziehen. Insbesondere die Flexibilität der Agenten machen sie zu einem vielseitigen und mächtigen Werkzeug für Management–Aufgaben aller Art.
Insofern ist bei einem großen Netz wie dem des Fahrzeugherstellers im Anwendungsszenario mit über 10.000 Händlerlokationen der logischen Struktur besondere Aufmerksamkeit zu schenken.
Die Gruppierung von Agenten unter geografischen Aspekten im vorhergehenden Abschnitt ist als Beispiel zu betrachten. Eine weitere sinnvolle Gliederung könnte z.B. in Anlehnung an Benutzerkategorien, denen unterschiedliche Service–Level zugesagt wurden, geschehen.
Die Strukturierung innerhalb einer Domäne bzw. Gruppe hat wesentlichen Einfluß auf den Informationsfluß. In der Praxis wird sich wohl eine *Mischform* zwischen dem Peer to Peer Ansatzes und einer hierarchischen Struktur ergeben, wie sie in Abbildung 4.4 zu sehen ist. Die strukturelle Hierarchie ist

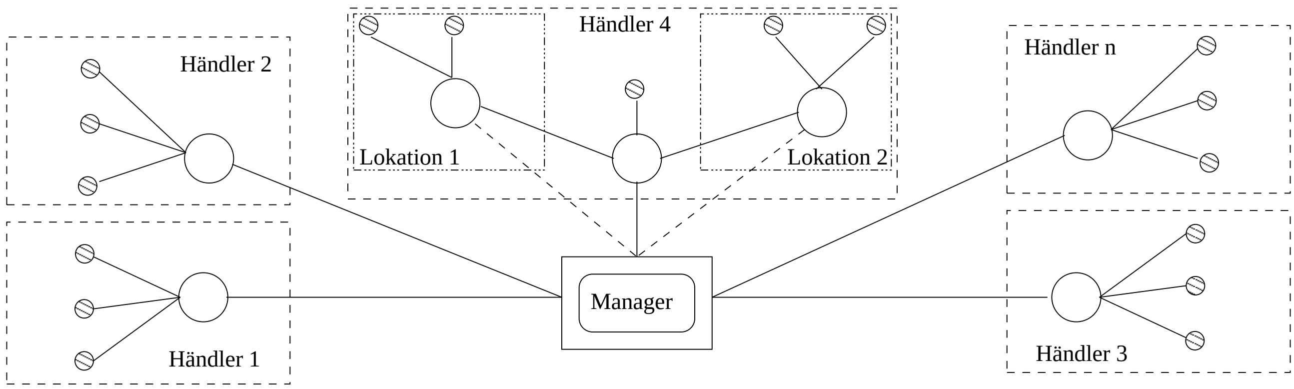

Abbildung 4.4: Struktur des Agentensystem für das Anwendungsszenario der DeTeSystem

im Anwendungsszenario ist sehr flach, aber vorhanden. Daher wird sich eine solche Struktur auch in weiten Bereichen durchsetzen. Insbesondere da es nicht sinnvoll ist, daß Agenten der Händlerlokationen untereinander Informationen austauschen.
Ein Sonderfall des Anwendungsszenarios ist ein Händler, dem mehrere Lokationen zuzuordnen sind, der aber aus Kostengründen nur eine Schnittstelle (Router) zur Verbindungsaufnahme mit dem Fahrzeughersteller betreibt. In diesem Fall ist es sinnvoll, in jeder Niederlassung einen FMA zu installieren, die Ihre Informationen an den in der Händlerzentrale weiterreichen. Dieser wertet die Meßergebnisse u.U. aus und leitet sie bei nächster Möglichkeit an den Manager weiter. In einem solchen Szenario ist die Möglichkeit der direkten Kontaktaufnahme zwischen dem Manager und den Agenten der einzelnen

Lokationen offenzuhalten, da so die Gefahr des „Single Point of Failure“ reduziert werden kann.

4.3 Konzeptionelle Umsetzung der Anforderungen

Im folgenden ist ein Konzept zu erarbeiten, das die Umsetzung der zentralen Forderungen aus Tabelle 2.3 ermöglicht. Hierzu werden folgende Anforderungen im Detail konzeptionell erörtert:

1. Überwachung der ISDN–Verbindung

2. Überwachung der Antwortzeit eines Web–Browsers

Abschließend werden die dabei nicht erörterten Anforderungen auf Umsetzbarkeit überprüft.

4.3.1 Überwachung der ISDN–Verbindung

In dem Anwendungsszenario dieser Arbeit ist die Überwachung einer ISDN–Verbindung zwischen einer Händlerlokation und dem dazugehörigen PoP von ausschlaggebender Bedeutung. Die Forderung nach Minimierung der Kosten impliziert, daß der Agent keine eigene Verbindung aufbauen darf. Daher ist es notwendig, daß er den aktuellen Status der Verbindung kennt.
Um dies zu realisieren bieten sich zwei Kategorien von Lösungen an:

1. Hardware–abhängige Lösungen

2. Hardware–unabhängige Lösungen

Während Hardware–abhängige Lösungen direkt auf die für den Verbindungsaufbau verantwortliche Netzkomponente (Router, Modem) zugreifen, folgern Hardware–unabhängige Lösungen den derzeitigen Status der Verbindung aus Informationen ihrer Umgebung.
Die Entscheidung für eine dieser Lösungsarten ist abhängig von den Rahmenbedingungen des Anwendungsszenarios und dem damit verbundenen Implementierungsaufwand bzw. der Flexibilität für nachträgliche Änderungen.
Eine Hardware–unabhängige Lösung erscheint auf den ersten Blick vorteilhaft, da bei der Realisierung des Agentensystems nicht auf die Ausstattung der Händler geachtet werden muß. Dieser Lösungstyp ist also unbedingt zu wählen, wenn eine Vielzahl unterschiedlicher Geräte zum Verbindungsaufbau betrieben werden. Verbunden ist allerdings mit einer HW–unabhängigen Lösung eine Infrastruktur, die das erlangen der benötigten Informationen ermöglicht. Soll z.B. der Agent benachrichtigt werden, wenn eine Verbindung besteht, so muß eine andere Komponente bestehen, die dieses feststellt und eine entsprechende Nachricht versendet.
Auch kann der Agent Informationen seines Umfeld auswerten, die sich mit dem Aufbau einer ISDN–Verbindung verändern. Diese Faktoren werden dann zwar nicht mehr durch die Verbindungskomponente beeinflußt, andere Änderungen im Umfeld müßten dann jedoch jederzeit kritisch beobachtet werden. So müßte z.B. bei einem „Sniffer“–Konzept permanent die beobachteten IP–Adressen auf ihre Aussagekraft überprüft werden.
Als Anbieter von Gesamtlösungen hat die DeTeSystem sämtliche Händlerlokationen mit denselben Verbindungskomponenten ausgestattet. So findet man überall Cisco–Router zur Umsetzung eines Dail on Demand Konzeptes. Diese Tatsache legt es aus pragmatischen Gründen nahe, eine Hardware–abhängige Lösung zu realisieren, da man davon ausgehen kann, überall ähnliche oder gar gleiche Bedingungen vorzufinden. Auch wenn dies nicht der Fall wäre, könnte auf Grund der hohen Modularisierung des Bean–Konzeptes ohne großen Aufwand für eine neue Verbindungskomponente ein dazugehöriges M–Bean entwickelt und eingebunden werden.
Auf jeden Fall sollte es durch ein Event–Konzept, wie es in Abbildung 4.5 zu sehen ist, bei dem

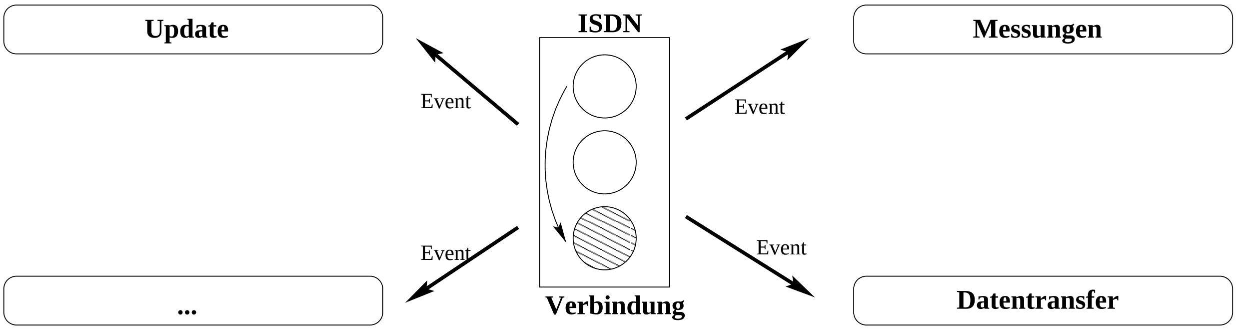

Abbildung 4.5: Benachrichtigung angemeldeter Instanzen

Aufbau einer Verbindung zu einer Benachrichtigung sämtlicher interessierter Instanzen kommen. Hierzu gehören z.B. eine für die Verschickung der Messdaten verantwortliche Instanz, eine, die für die Durchführung verbindungsabhängiger Messungen zuständig ist oder eine Instanz, die prüft, ob es Updates der Funktionalität gibt.
Abschließend kann nun festgestellt werden, daß bei der Implementierung eine Hardware–abhängig Lösung vorzuziehen ist. Möglichkeiten zur Realisierung werden im nächsten Kapitel erörtert. Durch ein Event–Konzept wird die Integration weiterer M–Beans erleichtert.

4.3.2 Messung von Antwortzeiten

Wie bereits in den Beschreibung der Anforderung dargelegt, sollen Antwortzeitmessungen auf Netzwerk– und auf Applikationsebene vorgenommen werden. Um diese zu realisieren, bieten sich verschiedene Konzepte an. Diese werden im folgenden dargestellt.

4.3.2.1 Netzwerkebene

Soll die Verfügbarkeit und Antwortzeit einer Netzkomponente geprüft werden, so bieten sich verschiedene Methoden dazu an. Über ein sogenanntes *passives Monitoring* des Netzes, also die reine Beobachtung der über das Netzwerk verkehrenden Protokollpakete, ist es möglich, Pakete mit Anfragen an bestimmte Server zu identifizieren und dazugehörige Antwortpakete zuzuordnen. Ein solches „Sniffer–Konzept“ läßt sich allerdings nur sehr aufwendig realisieren und kommt daher für einen Einsatz im Anwendungsszenario nicht in Frage.
Jede Form der *aktiven Messung* von Antwortzeiten wird so ablaufen, daß eine Anfrage (`request`) versendet und eine dazugehörige Antwort (`response`) erwartet wird. Kommt diese innerhalb eines zuvor spezifizierten Zeitraumes, so ist diese Komponente verfügbar. Die Zeit, die zwischen Absenden der Anfrage und Erhalt der Antwort verstreicht, ist die Antwortzeit.
Um eine solche Interaktion zu realisieren, wurden in herkömmlichen Ansätzen kommerzieller Produkte meist sogenannte Meßserver eingesetzt, die auf die Anfrage einer Meßroutine ein `response`–Signal senden. Da die Bestückung sämtlicher Komponenten des Anwendungsszenarios mit solchen Meßservern nicht gewünscht ist, werden im folgenden eine Reihe von dynamischeren Ansätzen dargestellt:

- **Mobiler Agent**
  Ein mobiler Agent migriert zum ersten Meßpunkt. Dort angekommen, teilt er seine Bereitschaft mit und antwortet auf die an ihn gerichteten Anfragen. Ist die Messung abgeschlossen, so migriert er zum nächsten Meßpunkt und erfüllt dort seine Aufgabe.

- **Delegierung der Echo–Funktionalität**
  Dem Konzept des MbD folgend ist es möglich, einem auf der Zielplattform laufenden Agenten die

für die Messung notwendige Echo–Funktionalität zu übertragen. Nach Abschluß der Meßsequenz wird ihm diese wieder entzogen, da die Aufgabe erfüllt ist, und ein unnötiges Ansammeln von Funktionalitäten zu vermeiden ist.

- **Stationärer Echo–Agent**
  In diesem Fall besitzt ein Agent auf der Meßplattform bereits die für die Beantwortung von Anfragen zur Antwortzeitmessung notwendige Funktionalität. Der Agent in der Händlerplattform muß sich nur noch dort anmelden und kann die Messung durchführen.

- **Nutzung bestehender Internetwerkzeuge**
  Es existieren Standardwerkzeuge des Internet–Managements, die für die vorliegende Aufgabe genutzt werden können. Basierend auf entsprechend standardisierten Protokollen stehen sie in modernen Netzen überall zur Verfügung und ermöglichen so das Messen von Antwortzeiten.

Um eine Messung durchzuführen wird eine Verbindung aufgebaut, eine bestimmte Datenmenge übertragen und wieder zurückgesendet. Nach dem Schließen der Verbindung wird dieser Vorgang wiederholt. Dieser Test ist z.B. für Webserver interessant, da so ermittelt werden kann, wie lange der Benutzer eines Clients warten muß, bis er eine Antwort erhält.
Die Agentenlösungen beinhalten alle den Nachteil, daß sie auf dem Zielrechner nicht unwesentliche Ressourcen beanspruchen. Die Laufzeitumgebung der Agentenplattform, die Agentenplattform selber und die entsprechende Anzahl der Agenten können für einen Server eine unzumutbare Belastung darstellen. So sendet in der ersten Möglichkeit jede messende Instanz einen mobilen Agenten in das Netz, im schlimmsten Fall entspricht also die Anzahl der Agenten auf einem Server der Anzahl der Händlerlokationen im Anwendungsszenario. Ein ähnliches Bild zeichnet sich durch das Delegieren einzelner Funktionalitäten. In Bezug auf den Resourcenverbrauch eines solchen Konzeptes erweist sich ein fester Echo–Agent als der sparsamste Ansatz, da er nur wenig und eine konstante Menge an Speicher benötigt. Allerdings kann es hier sehr schnell zu einer Überlastung eines solchen Objektes kommen, falls zu viele Anfragen gleichzeitig gestellt werden. Dies würde die Messungen verfälschen und den Echo–Agenten überlasten.
Das Überprüfen von Komponenten wie Routern und Gateways wird in einem solchen System grundsätzlich schwierig sein, da es nur in den seltensten Fällen möglich ist, hier eine Agentenplattform zu installieren.
Die Nutzung eines bestehenden Internet–Werkzeuges stellt eine weitere Möglichkeit dar, die Verfügbarkeit und Antwortzeit auf Netzwerkebene zu messen. Diese Werkzeuge gehören z.T. zur Standardausstattung von Internet–Protokollen und bieten eine für diesen Kontext ausreichende Funktionalität. Zudem ist es nicht notwendig, die Zielplattform zu instrumentieren. Nachteil dieser Methode ist, daß kein Einfluß auf die Art der Durchführung der Messung möglich ist, eigene Spezifikationen also auszuschliessen sind.
Es stellt sich heraus, daß für die Realisierung der Antwortzeitmessung auf Netzwerkebene ein bestehendes Internet–Werkzeug am besten geeignet ist. Dieses stellt die benötigte Funktionalität bereit und verfügt auf jeder aktiven Netzkomponente über einen Empfänger.

4.3.2.2 Applikationsebene

Die Anforderung für die Messung von Antwortzeiten auf Applikationsebene wurde bereits in Kapitel 2.3.3 hergeleitet. In diesem Abschnitt soll nun ein Konzept für die Realisierung dargestellt werden.
Zu Beginn ist zu überprüfen, welche Applikationen durch ein Service–Level–Agreement festgelegte Dienstgüteparameter aufzuweisen hat. Ist dies geschehen, wird anhand der folgenden Schritte eine Überwachung der entsprechenden Werte realisiert.

1. **Definition der Transaktionen**
   Beginnend bei den für den Nutzer sichtbaren Transaktionen werden Blöcke definiert, die in der Regel auch Service–Level–Agreements zugrunde liegen.

2. **Analyse der Transaktionen**
   Wie ist diese Transaktion aufgebaut? Bestehen Abhängigkeiten zu externen Komponenten? In diesem Fall liegt eine Client/ Server Beziehung vor, wie sie in Abbildung bla dargestellt ist.

3. **Bestimmung der zu messenden (Sub–)Transaktionen**
   Welche (Sub–)Transaktionen sind relevant für die Messung des Service Level? Wer kann die gemessenen Daten verwerten? Welche korrigirenden Maßnahmen können aufgrund nicht zufriedenstellender Werte ergriffen werden? Es ist sinnvoll, nur die Werte zu messen, die auch genutzt werden können. Durch diese einfachen Fragen kann schnell festgestellt werden, welche Transaktionen gemessen werden sollen.

4. **Instrumentierung der Applikation**
   Ist die Applikation nicht bereits durch den Entwickler mit einer Management–Schnittstelle ausgestattet, so ist sie in diesem Schritt zu instrumentieren. Zur Auswahl stehen mittlerweile eine Vielzahl von Entwicklungswerkzeuge, von denen einige in Abschnitt 3.4 dargestellt worden.

5. **Entwicklung einer Management–Applikation oder Integration in eine bestehende Plattform**
   Abschließend müßte eine Möglichkeit implementiert werden, auf die gemessenen Daten zuzugreifen und diese auszuwerten. Hierzu kann eine eigene Applikation entworfen werden, oder man integriert das zuvor entworfene System in eine bestehende Management–Plattform.

Im folgenden wird nun eine Konzept für die Überwachung der Antwortzeit eines WWW–Dienstes dargestellt.
Da nicht bekannt ist, ob die durch die BMW–Händler genutzten Applikationen bereits Management–Schnittstellen aufweisen, wird hier auf der Grundlage der in Abschnitt 3.4.3.4 erfolgten Bewertung die ARM–API der CMG als Entwicklungswerkzeug ausgewählt.
Ziel der Instrumentierung ist es, die Dauer einer Transaktion zu messen, in der eine Web–Seite geladen wird. Dabei soll es möglich sein, die dafür benötigte Zeit einzelnen Subtransaktionen zuzuordnen.
Dieser Vorgang läuft wie folgt ab [Tanen 98] (vereinfacht):

1. Der Browser ermittelt eine URL (Benutzereingabe, ...).

2. Der Browser fragt DNS nach der IP–Adresse des zu der URL gehörigen Web–Servers.

3. DNS gibt die gewünschte IP–Adresse zurück.

4. Der Brower baut eine TCP Verbindung auf.

5. Der Browser fordert mittels HTTP eine Datei an.

6. Der verbundene Server sendet die gewünschte Datei.

7. Die TCP–Verbindung wird getrennt.

8. Der Browser zeigt die empfangene Datei an.

Interessant ist in diesem Ablauf die Zeit die benötigt wird, um eine URL aufzulösen (2.-3.) und die Zeit, die benötigt wird, eine Datei aus dem WWW zu empfangen (4.-7.). In Abbildung 4.6 ist dargestellt, wie mit Hilfe eines Agenten, der die Anwendung überwacht, eine solche Zeitmessung zu realisieren ist.

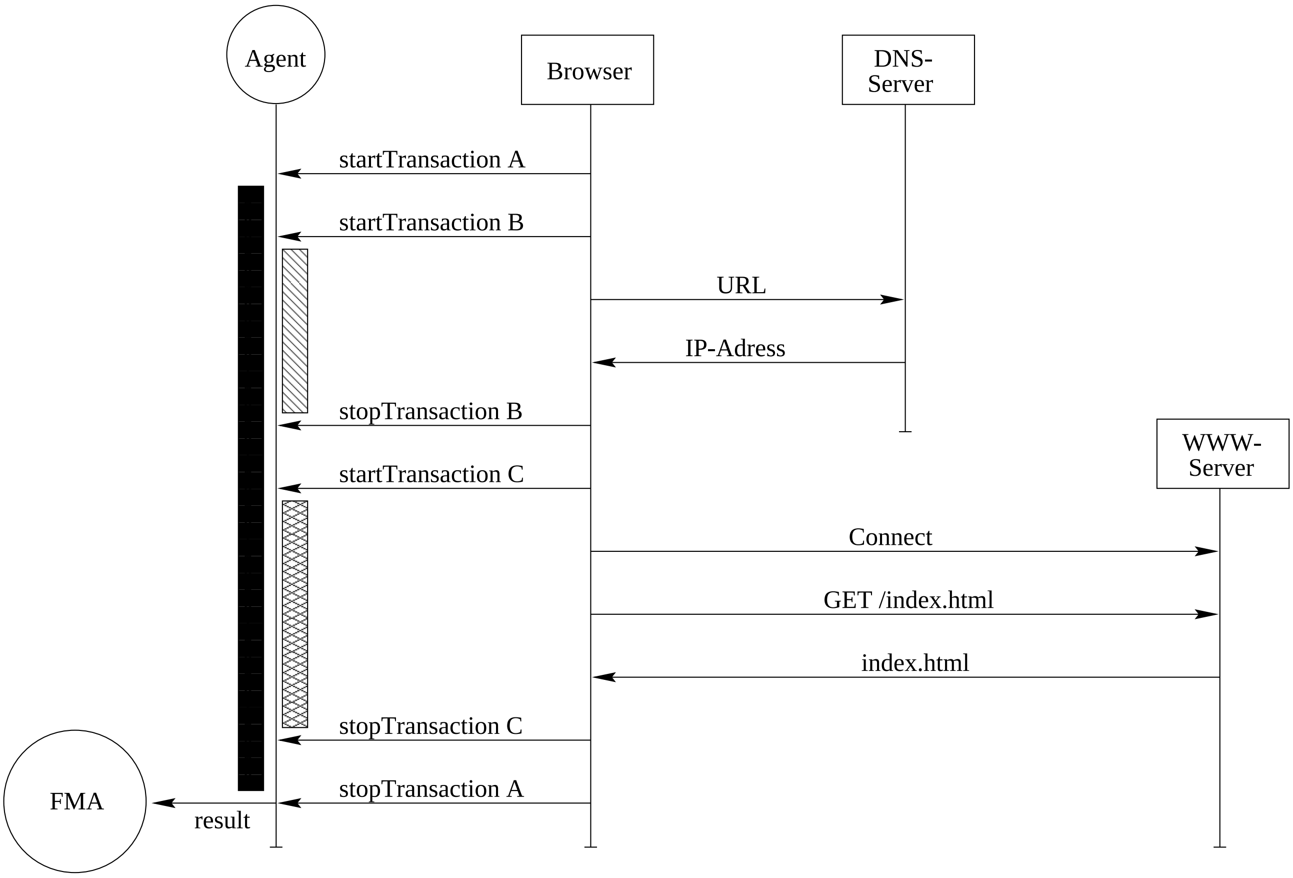

Abbildung 4.6: Konzept einer transaktionsorientierten Antwortzeitüberwachung eines Web–Browsers

Die benötigte Architektur besteht aus einem Flexiblen Management Agenten, der neben der Überwachung der Applikation auch andere Aufgaben erfüllt, einem Agenten, der direkt über ein entsprechendes Interface mit dem Web–Browser verbunden ist und dem Web–Browser selber.
Der Browser wird so instrumentiert, daß er die oben beschriebenen Zeitspannen kennzeichnet und dem Agenten jeweils Anfang und Ende dieser Transaktionen mitteilt. Daraus resultieren drei unterschieldiche Transaktionen:

1. **Transaktion A**: beginnt mit der Bestätigung der URL durch den Benutzer und endet nach der Darstellung der dazugehörigen Seite. Diese Transaktion beinhaltet die (Sub–)Transaktionen B und C.

2. **Transaktion B** beginnt mit der Anfrage an den DNS–Server und endet mit der Erhalt der dazugehörigen IP–Adresse.

3. **Transaktion C** beginnt mit dem Verbindungsaufbau zu dem Web–Server und endet mit der Darstellung der HTML–Seite. Es ist darauf zu achten, daß die Länge dieser Transaktion von der Größe der übertragen Datei abhängt. Diese Information muß bei der Auswertung berücksichtigt werden. Unter Umständen ist es sinnvoll, den Verbindungsaufbau zum Web–Server getrennt zu Messen.

Die Ergebnisse dieser Messungen werden anschließend dem FMA übergeben, der sie dann entsprechend seiner Vorgaben an den Manager weiterreicht. Hierbei ist zu beachten, daß der FMA die Daten in einer Form übergeben bekommt, in der die Korrelation der Sub–Transaktionszeiten zu der Haupttransaktion möglich ist.

4.3.3 Realisierung der Sicherheit

Es ist nicht möglich, im begrenzten Rahmen dieser Arbeit eine eigene konzeptionelle Sicherheitarchitektur für ein komplexes Agentensystem zu entwickeln oder ein bestehendes umfassend zu evaluieren. Grund hierfür sind die Vielzahl der genutzten Dienste und Protokolle auf den unterschiedlichsten Schichten des OSI-Modells.
Es wird am Schluß des Kapitels 5 eine oberflächliche Bewertung der genutzten Sicherheitsmechanismen in der konkreten Entwicklungsumgebung durchgeführt.

4.4 Bewertung des Konzeptes

Abschließend ist nun zu bewerten, inwiefern das hier entwickelte Konzept den Anforderungen im Abschnitt 2.3.1 entspricht. Eine Zusammenfassung dieser Bewertung ist in Tabelle 4.1 dargestellt.
Durch die Nutzung plattformunabhängiger Werkzeuge werden kürzere Entwicklungszeiten in heterogenen Netzen und die Vermeidung von Anpassungen bei Plattformumstellungen in Teilnetzen realisiert. Java ist eine solche *plattformunabhängige* Programmiersprache. Zusammen mit dem JDMK ist so die Entwicklung eines Agentensystems mit skalierbaren Komponenten möglich, was einen *geringen Ressourcenbedarf* sicherstellt. Anders als bei Komplettlösungen wie durch die Applikation Sitescope, die eine Vielzahl benötigter und unbenötigter Funktionalität bereitstellt, können so Agenten entwickelt werden, die den Anforderungen angepaßt sind. Durch das JDK ist es auf einfachem Wege machbar,

Anforderung	erfüllt	realisierbar	nicht realisierbar
Technische Anforderungen			
Plattformunabhängigkeit	X		
geringer Ressourcenbedarf	X		
Protokolldatei	X		
Organisatorische Anforderungen			
Organisation der Agenten	X		
dynamische Veränderbarkeit	X		
Konfigurierbarkeit während der Laufzeit	X		
Integration in bestehende Managementarch.		X	
Betriebswirtschaftliche Anforderungen			
geringer Verteilungsaufwand		X	
kein selbständiger Verbindungsaufbau	X		
geringer Implementierungsaufwand		X	
Anforderungen an die Funktionalität			
diverse Messmethoden		X	
Messung von Antwortzeiten auf Applikationsebene		X	
Auslesen von Systemdaten		X	
Überwachen des Benutzerverhalten		X	

Tabelle 4.1: Bewertung der Lösung

Protokolldateien (Log–Files) anzulegen und persistent zu speichern. Eine sinnvolle *Organisation*der Agenten ist konzeptionell entwickelt worden und durch das JDMK realisierbar. Dies stellt eine Überschaubarkeit des Systems auch bei steigender Agentenzahl sicher.
Dynamische Veränderbarkeit der Agenten während der Laufzeit wird insbesondere durch eine strikt komponentenbasierte Programmierung umgesetzt. Durch engen Schnittstellen (wenige Parameter) und

geringen Interdependenzen zwischen den einzelnen Objekten kann das Austauschen oder Löschen einzelner Funktionalitäten leicht realisiert werden.
Die *Integration in bestehende Management–Architekturen* ist eine Forderung, die durch Werkzeuge des JDK realisierbar ist. Schnittstellen wie JDBC zur Nutzung externer Datenbanken oder JNI zur Anbindung an nicht–Java Programme bieten ausreichend Möglichkeiten, das Agentensystem in einer bestehenden Management–Plattform zu nutzen.
In einem Extranet mit einer hohen Zahl an externen Lokationen ist es nicht möglich, die Agenten bzw. neue Funktionalitäten „per Hand“ zu verteilen. Hier muß das Netz genutzt werden, um die Agenten auf ihre Zielplattform zu bringen. Die dazu benötigten *Verteilungsmechanismen* stehen in der Architektur zur Verfügung.
Ein Konzept zur Vermeidung des *selbständigen ISDN–Verbindungsaufbaus* durch einen Agenten wurde in Abschnitt 4.3.1 ausführlich diskutiert.
Um den Aufwand der Implementierung zu reduzieren, werden mehrere Ansätze empfohlen, die auf das eigentliche Konzept keinen Einfluß haben. Insbesondere sei hier die Entwicklung wiederverwendbarer Softwarekomponenten erwähnt. Obwohl die Entwicklung wiederverwendbarer Software einen erhöhten Aufwand bei der Erstimplementierung hat, zahlt sich diese Investition in der Regel aus. Kürzere *Entwicklungszeiten* für Weiterentwicklungen, höhere Korrektheit, bessere Effizienz durch eine Vielzahl von Entwicklungsiterationen sind nur ein kleine Aufzählung der Vorteile einer solchen Entwicklungsstrategie [Reif 97] [MH 98].
Meßmethoden können jederzeit einem flexiblen Management–Agenten in Form einer zusätzlichen Funktionalität übergeben werden, ebenso wie Objekte für das Auslesen von Systemdaten und um das SLA–konforme Verhalten der Anwender zu überwachen. *Messung von Antwortzeiten auf Netzwerk– und Applikationsebene* wurde in Abschnitt 4.3.2 ein ausführliches Konzept entworfen.
Somit kann festgestellt werden, daß das vorliegende Konzept den in Abschnitt 2.3.1 gestellten Anforderungen entspricht.

Kapitel 5

Prototypische Implementierung

5.1 Die Entwicklungs– und Testumgebung

Die Entwicklungs– und Testumgebung kann trotz der Komplexität des Anwendungsszenarios sehr einfach gehalten werden, da weite Bereiche für das Agentensystem transparent sind.
Das SMC–Netz, die ISA und die Verbindung von dort in das Weitverkehrsnetz ist für den Agenten transparent und stellen sich somit wie eine Standleitung dar. Relevant für das System ist die Verbindung zwischen dem PoP und dem Router der Händlerlokation. Aus diesen Überlegungen hat sich für die Entwicklung die in Abbildung 5.1 zu sehende Umgebung herauskristallisiert. Die Manager–

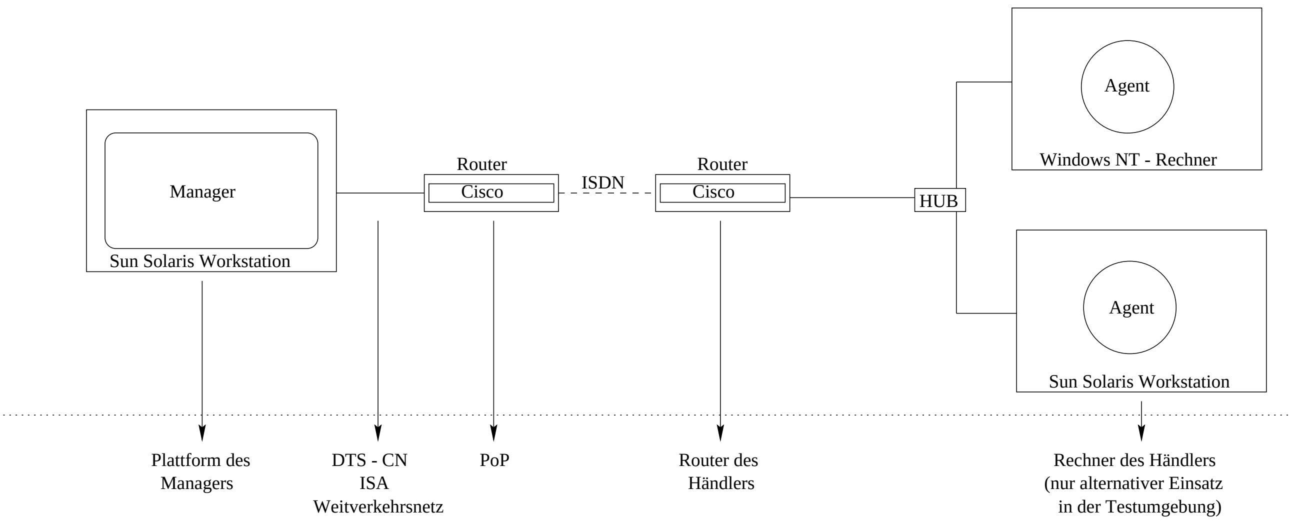

Abbildung 5.1: Die Entwicklungsumgebung und deren Projektion auf das Anwendungsszenario

Applikation wird auf einer Sun–Workstation betrieben, die über eine Ethernet–Verbindung (Twisted Pair) mit einem Router verbunden wird. Diese Workstation würde im Anwendungsszenario z.B. im SRZ stehen. Die Verbindung zum Router entspricht der Verbindung zwischen dem SRZ über das VLAN des SMC–Netzes, ISA und VPN des Weitverkehrsnetzes zum PoP. Die ISDN–Verbindung findet sich real in der Entwicklungsumgebung wieder. Diese ermöglicht die Kontaktaufnahme zu einem zweiten Router, der die Einwahlkomponente der Händlerlokation darstellt.
Das dargestellte Netz wurde so eingerichtet, daß die durch die Router getrennten Komponenten in unterschiedlichen Subnetzen plaziert sind. Dies ermöglicht eine leichte Konfiguration der Router, sodaß diese bei Adressierung eines Paketes in das jeweilig andere Subnetz eine ISDN–Verbindung aufbauen (Dail on Demand).

Als Testumgebung für den Agenten wurde ein Windows–PC und eine weitere Sun–Workstation genutzt. Aufgrund der nachgewiesenen Plattformunabhängigkeit des JDK wurde zum Nachweis dieser Fähigkeit auf die Nutzung weiterer Betriebssysteme verzichtet.
Initiert wird der ISDN–Verbindungsaufbau z.B. durch einen Telnet–Zugriff in das jeweils andere Netz. Dieser Zugriff entspricht dem Verbindungsaufbauwunsch eines Händlers mittels einer dritten Applikation.
Durch die hier beschriebene und in Abbildung 5.1 dargestellte Projektion der Entwicklungs– und Testumgebung auf das Anwendungsszenario sind die Voraussetzungen für die Entwicklung und Evaluierung des Prototypen gegeben.
Zur Implementierung wird das Java Development Kit 1.1.7b von Sun Microsystems verwendet, da die aktuelle Version 1.2 nicht mit dem JDMK3.0beta, das zur Realisierung des Agentensystems benutzt wird, kompatibel ist.

5.2 Umsetzung der Architektur

Bei der Entwicklung des Prototypen stand insbesondere die Erweiterbarkeit und Stabilität der Architektur im Vordergrund. Es galt, den Anforderungen einer heterogenen und dynamischen Umgebung gerecht zu werden. Grundsätzlich sind zur Implementierung des Prototypen ein Manager und ein Agent zu realisieren. Zur Namensgebung der Klassen und zur Notation seien vorab folgende Bermerkungen gemacht:

- Klassen, deren Bezeichnungen mit den Buchstaben `Ag` beginnen, sind dem Agenten zuzuordnen.
- Klassen, deren Bezeichnungen mit den Buchstaben `Mg` beginnen, sind dem Manager zuzuordnen.
- Klassen, deren Bezeichnungen weder mit `Ag` noch mit `Mg` beginnen, werden von beiden Komponenten des Agentensystems genutzt.
- Klassen, deren Bezeichnungen die Buchstabenfolge `GUI` beinhalten, dienen der Visualisierung von Zuständen oder Daten oder bieten Funktionalitäten graphisch an.

Desweiteren werden bekannte Designaspekte von Java–Applikationen beachtet. Klassen beginnen mit großen Buchstaben, Objekte mit kleinen, Kommentare werden entsprechend den `javadoc`–Anforderungen gesetzt und Java–Beans Design–Pattern wenn notwendig angewendet.
In den folgenden zwei Abschnitten wird aus den konzeptionellen Entscheidungen des vorangegangenen Kapitel ein Objektmodell für den Agenten und den Manager entwickelt. Dazu wird die Unified Modelling Language (UML) benutzt, deren genaue Spezifikation in [UML] nachgelesen werden kann.

5.2.1 Der Agent

Wie dem Klassendiagramm in Abbildung 5.2 zu entnehmen ist, bildet die Klasse `AgBasis` den Kern des Agenten. In ihr werden alle für den Start und die Initialisierung des Agenten notwendigen Vorgänge implementiert und die benötigten Objekte instanziiert. Besondere Aufgabe dieser Klasse ist der Start der Agentenplattform und der damit verbundenen Dienste.
`AgDataHandler` verwaltet die im Rahmen der unterschiedlichen Messungen auflaufenden Informationen, die in Form des `DataObject` zusammengestellt werden. Desweiteren ist die Klasse `AgDataHandler` für die Initiierung des Versendens der Daten an den Manager bzw. deren Sicherung auf die lokale Festplatte verantwortlich.
Die Grundfunktionalität für alle Messungen wird in der abstrakten Klasse `AgMeasurement` angeboten. Mit der Implementierung der Methode `performMeasure` wird die jeweilige Messung spezifiziert. Die Klasse `AgConnectivity` ist ein Beispiel für die Umsetzung einer solchen Messung.

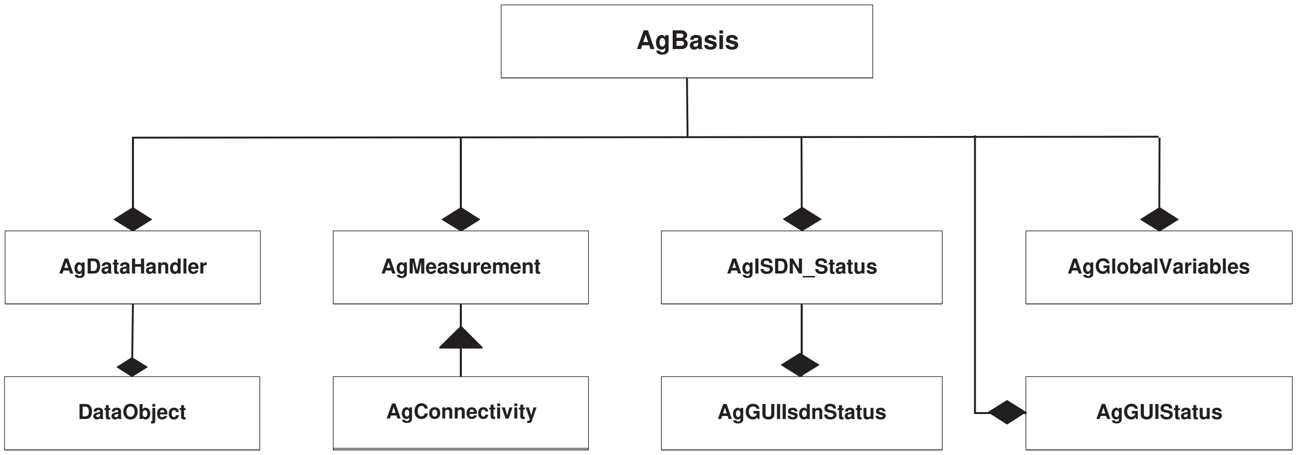

Abbildung 5.2: Klassendiagramm des Agenten (vereinfacht)

Mit `AgISDN_Status` wird der Status eines ISDN–Routers überwacht. Hierzu wird dessen SNMP–Agent genutzt und die entsprechende MIB–Variable in regelmäßigen Abständen überprüft. Zu Demonstrationszwecken wird der jeweilige Status mit Hilfe der Klasse `AgGUIIsdnStatus` in Form einer Ampel visualisiert.
`AgGlobalVariables` beinhaltet zentral sämtliche für den Agenten wesentliche Parameter wie z.B. den IP–Namen und SNMP-Port des lokalen Routers, die Adresse des Managers und die Prüffrequenz der Meßmethode.
Nicht abgebildet ist in Abbildung 5.2 die Klasse `IoHandler`, die die Methoden für sämtliche Schreib– und Lesevorgänge (Log–File, Datensicherung) auf die lokale Festplatte implementiert.
Weitere, z.T. durch Tools des JDMK generierte Klassen, und Details der Implementierung werden später anhand der Aufgaben dargelegt.

5.2.2 Der Manager

Das Dach der Manager–Applikation bildet die Klasse `MgBasis` (siehe Abbildung 5.3), da sie für das Starten und Initialisieren des Managers zuständig ist. Eine grafische Benutzeroberfläche wird durch die Klassen `MgGUIZentrale`, `MgGUIAgParam` und `MgGUIDataBrowser` bereitgestellt. Abweichend von der Vorgabe, grafische Darstellung und Funktionalität zu trennen, wurde die zur Manipulation der Agentenparameter notwendigen Methoden aus Vereinfachungsgründen auch in `MgGUIAgParam` platziert.
Zur Realisierung der Anforderung, daß der Agent seine Daten aus eigenem Antrieb an den Manager übergeben soll (pushen), wurde auf der Manager–Seite eine Schnittstelle entworfen, die diese Daten entgegen nimmt. Die dazu notwendige Plattform wird in `MgCMF` gestartet, wo auch das M-Bean `MgDataReceiver` erzeugt und registriert wird.
Die Datenverwaltung wird durch die Klasse `MgDataHandler` geregelt, die die ihr übergebenen `DataObjects` aus später zu erörternden Gründen in `MgMeasure`–Objekte umformt.

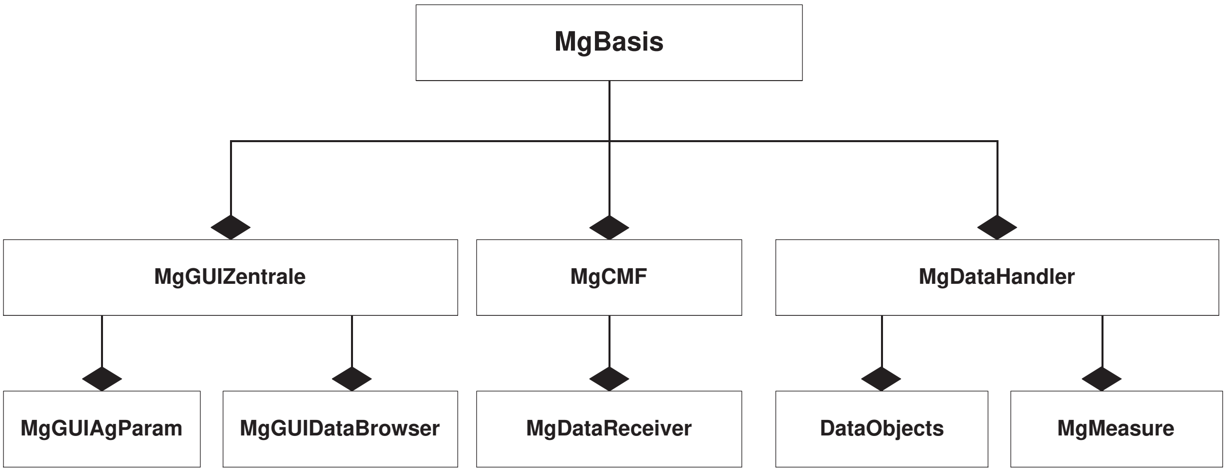

Abbildung 5.3: Klassendiagramm des Managers (vereinfacht)

5.3 Start der Agentenplattform

Zur Bereitstellung der durch den Agenten benötigten Plattformen und Dienste wird das sogenannte *Common Management Framework (CMF)* des JDMK genutzt und der in Abschnitt 3.2.3 beschriebene Metadata–Service gestartet. Dies geschieht wie folgt:

```
// Framework starten
cmf = new Framework();

// MetaDataService starten
String mtdSrvClass = "com.sun.jaw.impl.agent.services.light.MetaDataSrv" ;
String mtdSrvName = domain + ":" + ServiceName.META ;
cmf.newObject(mtdSrvClass, mtdSrvName, null) ;
```

Durch das Erzeugen eines neuen Objektes der Klasse `Framework` mit einem leeren Konstruktor werden zwei Vorgänge automatisch durchgeführt. Es wird diesem Framework automatisch eine Standard–Domänenbezeichnung (`defaultDomain`) zugewiesen und der Repository Service gestartet. Soll nun ein Framework in der Domäne „Fahrzeughersteller“ gestartet werden, so ist der Konstruktor wie folgt aufzurufen:

```
// Framework starten
cmf = new Framework("Fahrzeughersteller");
```

Nach dem Erzeugen des CMF wird eine Instanz der Klasse `com.sun.jaw.impl.agent.services.light.MetaDataSrv` erzeugt und dem Framework zugewiesen. Es wird hier darauf hingewiesen, daß zwar weitere Services im Vorfeld der Prototyperstellung getestet wurde, tatsächlich allerdings nicht alle implementiert wurden. Diese Dienste und ihr potentielles Einsatzgebiet werden im Kapitel 6 geschildert. In einem letzten Schritt werden nun die M–Beans erzeugt und im Framework registriert. Exemplarisch wird die dazu notwendige Vorgehensweise am Beispiel der Klasse `AgISDN_Status` dargestellt.

```
...
// MBean erzeugen
isdnStatus = new AgISDN_Status();
...
// MBean registrieren
try {
  ObjectName i = new ObjectName(domain + ":" + "ISDN_Status.SerialNo=1");
  cmf.addObject(isdnStatus, i);
}
catch (InstanceAlreadyExistException e){
  System.out.println("BasisAgent.initBasisAgent(): InstanceAlreadyExistException !!");
  e.printStackTrace();
}
...
```

Um ein M–Bean registrieren zu können, wird zunächst ein Objektname benötigt. Dieser setzt sich aus folgenden Komponenten zusammen:

1. **Domänenbezeichner**
   Standardmäßig wird die Bezeichnung `defaultDomain` angelegt. Diese kann allerdings durch einen Parameter im Konstruktor des CMF geändert werden.

2. **Klassenbezeichner**
   Hier wird die Klasse angegeben, welche durch das M–Bean repräsentiert wird. Obwohl die Bezeichnung der konkreten Klasse nicht unbedingt mit diesem Bezeichner übereinstimmen muß, ist es nicht sinnvoll, hier abzuweichen.

3. **Suchschlüssel**
   Der Suchschlüssel beinhalt die Möglichkeit, das einzelne M-Bean, und somit den Agenten, genauer zu bezeichnen. Er kann einen oder mehrere Attribute umfassen. Ein mögliches Wertepaar wäre hier Stadt=Duisburg, Filiale=Ottostraße.

Im Prototypen wurde ein Schlüssel mit der Bezeichnung `Serialno` und dem Wert `1` gewählt. Zu beachten ist noch, daß beim Registrieren eines M–Beans eine `InstanceAlreadyExistException` abgefangen werden muß, die verhindern soll, daß mehrere M–Beans mit gleichem Objektnamen erzeugt werden.

5.4 Prüfung des ISDN–Status

Um die Anforderung zu erfüllen, zur Datenübertragung an den Manager durch den Agenten keine eigene ISDN–Verbindung aufzubauen, muß der Agent in der Lage sein, den Zustand dieser Verbindung überwachen zu können.
Zur Umsetzung dieser Forderung wurde im Rahmen des Prototypen die Nutzung des SNMP–Agenten des entsprechenden Routers gewählt, der die notwendigen Informationen bereithält. Ein weiterer Vorteil dieser Methode ist, daß dieser Agent durch die eingesetzten Cisco–Geräte bereits als Standard–Management–Schnittstelle vorhanden ist.
Somit übernimmt der JDMK–Agent die Rolle eines SNMP–Managers. Zur Realisierung dieser Architektur sind folgende Schritte durchzuführen:

1. **Identifikation der benötigten MIB–Variable**
   Die Identifikation der benötigten MIB–Variable hat sich in der Praxis als schwierig erwiesen, da zu den Cisco–Routern unterschiedliche Versionen von Betriebssystemen in Betrieb sind, die

unterschiedliche Informationen bereitstellen. Somit scheiden die Cisco–spezifischen MIBs aus, und die Entscheidung fiel für die in RFC1213 definierten Variablen, die durch alle Ciscosysteme unterstützt werden.

2. **Erzeugung eines MIB–Stores**
   Mit Hilfe des `mibgen`–Tools des JDMK wird eine sogenannte MIB–Store erstellt, also eine Java–Klasse, die als SNMP–Schnittstelle die Informationen für die Zuordnung von Variablenbezeichnung zu OID bereitstellt.

3. **Implementierung einer Methode für Abfrage des SNMP–Agenten**
   Um das Abfragen der SNMP–Variablen zu ermöglichen, muß eine Methode implementiert werden, die eine Verbindung zu dem Agenten des Routers herstellt, die entsprechende Variable ausliest, die Verbindung wieder schließt und den erlangten Wert für alle Komponenten zugänglich macht.

4. **Polling–Mechanismus**
   Da der SNMP–Agent des Routers keine Traps versendet, wenn eine Verbindung aufgebaut wird, ist es notwendig, die entsprechende MIB–Variable mittels der in Punkt 3 implementierten Methode in regelmäßigen Abständen abzufragen (Polling).

5. **Erzeugung eines Events bei Änderung des Status**
   Sollte im Rahmen des Pollings der entsprechenden Variable eine Änderung des ISDN–Verbindungsstatus festgestellt werden, muß dies an die angemeldeten Objekte mittels eines Events propagiert werden.

Realisiert wird dieses Vorgehen mittels der Klasse `AgISDN_Status`, die in Abbildung 5.4 zu sehen ist. Zur Überwachung des ISDN–Status wird in dem vorliegenden Prototypen der Inhalt von `ifOperStatus`

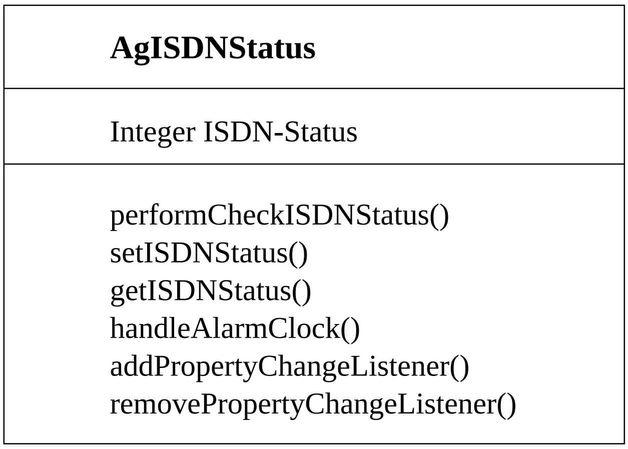

Abbildung 5.4: Diagramm der Klasse `AgISDN_Status`

der RFC1213–MIB zugrunde gelegt. Spezielle Cisco–MIBs stellen zwar noch detaillierte Informationen wie den Kommunikationspartner, bisherige Dauer der ISDN-Verbindung usw. bereit, werden allerdings nicht von allen Ciscosystemen belegt.
Mit Hilfe des so gefundenen MIB–Files kann nun die für den Manager notwendige Klasse für die Zuordnung von MIB–Bezeichnungen und OIDs erzeugt werden. Mit dem JDMK–Tool `mibgen` wird die Klasse `RFC1213_MIBStore.java` generiert.
Die so erzeugte Klasse enthält nun die für den JDMK–Agenten lesbaren Metadata–Definitionen der RFC1213–MIB:

```
...
public class RFC1213_MIBStore extends MibStore implements Serializable{
  /**
   * Default Constructor. Initialize the Mib tree.
   */
  public RFC1213_MIBStore() throws SnmpStatusException {
    loadMib (varList);
  }
  static String varList[][] = {
...
    {"ifTable", "1.3.6.1.2.1.3.1", "TA" },
    {"ifOperStatus", "1.4.6.1.2.1.2.2.1.8", "I"},
...
  }
```

Um die benötigten Informationen abfragen zu können, muß eine Methode implementiert werden, die auf der Basis dieser generierten Klasse eine Verbindung zum SNMP–Agenten herstellt und den Wert der Variable `ifOperStatus` ausliest. Die Implementierung dieser Methode, die als Rückgabewert den jeweiligen ISDN–Status in Form einer Ganzzahl bereitstellt, sieht wie folgt aus:

```
  int status = -1;
  String host = AgBasis.globalVariables.getrouterIpName();
  int    port = AgBasis.globalVariables.getrouterSnmpPort();
```

Zu Beginn der Methode werden die zentralen Parameter mit den jeweiligen Werten belegt. Insbesondere sind hier die Adresse des lokalen Routers und der Port dessen SNMP–Agenten interessant. Die Variable `status` wird mit dem Wert -1 initialisiert, der die Nicht–Überprüfbarkeit des ISDN–Status signalisiert. Nun folgt der Aufbau der Verbindung zum Agenten:

```
     // Initialisierung des SNMP-Frameworks und
     // einer Verbindung zu SNMP-Agenten
     SnmpMain.initializeSNMP(new RFC1213_MIBStore());
     SnmpPeer agent= new SnmpPeer(host, port);

     // Erzeugung einer SNMP-Session
     SnmpSession session= new SnmpSession("performCheckISDNStatusSession");
     session.setDefaultPeer(agent);
```

Dazu wird der das SNMP–Framework, das durch das JDMK bereitgestellt wird, mit den zu prüfenden MIBs initialisiert eine SNMP–Session gestartet. Nun muß eine Liste zusammengestellt werden, in denen die Variablen angegeben werden, deren Werte tatsächlich abgefragt werden sollen. Dazu wird ein Objekt der Klasse `SnmpVarbindList` genutzt, das anschließend in der `snmpGet`–Methode der Session übergeben wird.

```
     // Erstellung einer Liste mit den abzufragenden Variablen
     SnmpVarbindList list = new SnmpVarbindList("performCheckISDNStatusVarbindList");
     list.addVariable("ifOperStatus.1");
     list.addVariable("ifOperStatus.2");
     list.addVariable("ifOperStatus.3");

     // Abfrage der in list aufgefuehrten Variablen
     SnmpRequest request = session.snmpGet(null, list);
```

```
// Falls der Agent nicht verfuegbar ist (Router abgschaltet)
boolean completed = request.waitForCompletion(10000);
```

Mit `waitForCompletion(10000)` wird nun auf die Beendigung der SNMP–Abfrage, die durch einen Thread realisiert ist, gewartet. Wird diese nicht beendet, so wird der Wert -1 aus dieser Methode zurückgegeben. Wird hier durch `completed` das korrekte Ausführen der Methode signalisiert, so müssen die Ergebnisse aus diesem `request` ausgelesen werden. Dies geschieht mit der Methode `getResponseVbList()`:

```
// Neue Liste mit dem Resultat der Abfrage erzeugen
// und einzelne Variablen auslesen
SnmpVarbindList result = request.getResponseVbList();
SnmpVar ifOperBRI  = result.getSnmpVarAt(0);
SnmpVar ifOperB0   = result.getSnmpVarAt(1);
SnmpVar ifOperB1   = result.getSnmpVarAt(2);
```

Abschließend müssen die erlangten Ergebnisse interpretiert und ein Rückgabewert ermittelt werden.

```
//ISDN Verbindung besteht, wenn mind. ein ISDN Kanal benutzt wird !
if ((ifOperB0int==1)||(ifOperB1int==1))
  status = 1;
//ISDN Verbindung besteht nicht, wenn beide Kanaele getestet werden !
else
  status = 2;
return status;
```

Benutzt werden in dieser Methode die SNMP-Funtionalitäten, die in dem JDMK–Paket `com.sun.jaw.snmp.manager` bereitgestellt werden. Zur Klarstellung wird darauf hingewiesen, daß diese Methode Teil des Management–Agenten ist. Dieser JDM–Agent tritt allerdings dem SNMP–Agenten des Routers gegenüber als Manager auf.
Ein Polling–Mechanismus wird mit Hilfe des Alarmclock–Service implementiert, der in regelmäßigen Abständen ein Event erzeugt.

```
alarmClock = new AlarmClock();
alarmClock.setTimeoutAsLong(new Long(10));
alarmClock.performStart();
alarmClock.addAlarmClockListener(this);
```

`setTimeoutAsLong` initialisiert diesen Service mit 10 Millisekunden. Dieser Wert ist zu Beginn deshalb so gering, da praktisch umgehend der Status überprüft werden muß. Die Behandlung des durch die Alarmclock erzeugten Events sieht wie folgt aus:

```
public void handleAlarmClock (AlarmClockEvent e){
   int status = performCheckISDNStatus();
   setISDNStatus(status);
   alarmClock.setTimeoutAsLong(new Long(10000));
   alarmClock.performStart();
 }
```

Durch das `AlarmClockEvent` wird die Methode `handleAlarmClock` aufgerufen. Diese prüft mittels der oben beschriebenen Methode `performCheckISDNStatus` den ISDN–Status, und aktiviert die Alarmclock erneut. Durch den Wert 10000 für den Parameter Timeout wird jetzt der Intervall auf 10 Sekunden

verlängert.
Die in Abbildung 4.5 dargestellte Forderung der Benachrichtigung aller angemeldeten Instanzen bei Änderung des ISDN–Status wird mit einem Objekt der Klasse `PropertyChangeSupport` umgesetzt. Diese Klasse ist ein Teil des `java.beans`–Paketes und hat die Aufgabe, ein Event zu erzeugen, wenn der ISDN–Status sich verändert. Dies geschieht wie folgt:

```
private void setISDNStatus (int nISDNStatus){
     int oldISDNStatus = ISDNStatus;
     this.ISDNStatus = nISDNStatus;
     // Event erzeugen und alten und neuen Status uebergeben
     propertyChangeSupport.firePropertyChange("ISDNStatus",
                                          new Integer(oldISDNStatus),
                                          new Integer(ISDNStatus));
}
```

Jede Klasse, die sich als Listener für dieses Event registriert hat, wird so benachrichtigt. Dabei wird durch das Event sowohl der alte als auch der neue Statuswert übertragen.
Somit wird durch die Klasse `AgISDNStatus` die für das Anwendungsszenario zentrale Anforderung der Überwachung der ISDN–Verbindung erfüllt. Durch die Nutzung des `PropertyChangeEvent` wird die geforderte Benachachrichtigung registrierter Klassen realisiert.

5.5 Realisierung der Messung

Wie in Abschnitt 2.3.3 beschrieben, ist es notwendig, zur Überwachung von Service–Level–Agreements Antwortzeiten aus Benutzersicht zu messen. Durch eine Agentenarchitektur wird diese Anforderung erfüllt, da so die Messung aus der Händlerlokation getätigt wird. Implementiert wurde im Prototypen ein einfaches, auf dem Internetwerkzeug `ping` und ICMP (Internet Control Message Protocol) basierendes Verfahren zur Prüfung der Verfügbarkeit eines IP–Hosts und Messung seiner Antwortzeit.
Ziel der Realisierung auf der Agentenseite ist, den Implementierungsaufwand für neue Meßmethoden möglichst gering zu halten. Daher wurde eine Architektur gewählt, die eine abstrakte Klasse `AgMeasurement` mit allen für eine intervallgesteuerte Messung benötigten Funktionalitäten bereitstellt. Die eigentlich Meßmethode, also `performMeasure`, wurde abstrakt gehalten. Die Umsetzung dieser Methode geschieht in der Klasse `AgConnectivity`. Der durch diese Klassen gesteuerte Ablauf sieht so aus, daß nach Erhalt des `PropertyChangedEvent` der bereits dargestellten Klasse `AgISDN_Handler` die AlarmClock gestartet wird:

```
public void propertyChange(PropertyChangeEvent e){
   alarmClock.performStart();
   }
```

In dem erstem Intervall werden keine Messungen durchgeführt, da hier die Ressourcen für Benutzerdaten freigehalten werden müssen. Während der Initialisierung der Messung werden die IP–Adressen der Hosts aus der Konfigurationsschnittstelle `GlobalVariables` des Agenten ausgelesen und eine neues Objekt der Klasse `DataObject` zur Verwaltung der messungsrelevanten Daten erzeugt:

```
Vector adressen = AgBasis.globalVariables.gethostsToBeChecked();
DataObject cdo = new DataObject(now , true, getownIPAdress(), false,
                            "Connectivity&ResponseTimeWithPing",
                                      adressen, getownIPAdress());
```

Nun wird anhand des Adressenvektors `adressen` eine Erreichbarkeitsanalyse mit dem Internetwerkzeug `ping`, also mit ICMP–Echo–Paketen, durchgeführt. Vorteil einer solchen Realisierung ist, daß dieser

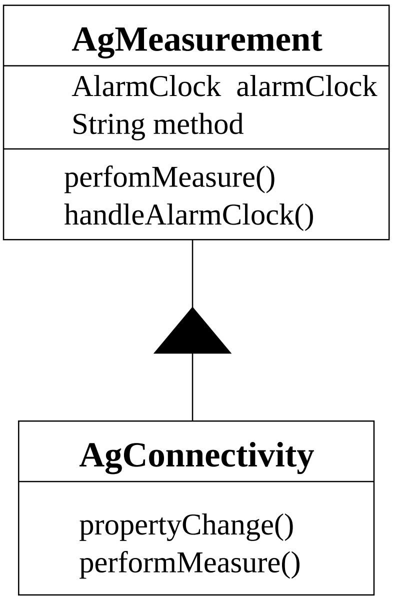

Abbildung 5.5: Diagramm der Klassen `AgMeasurement` und `AgConnectivity`

Mechanismus als Standard überall verfügbar ist. Auf praktisch jeder aktiven Netzkomponenten ist ein Port festgelegt, an das dieses Werkzeug Pakete senden kann, und die dann zurückgeschickt werden. Um nun dieses externe Programm nutzen zu können, wird ein Runtime–Objekt erzeugt, das den Ausführungsrahmen innerhalb des Java–Agenten darstellt.

```
// Rahmen fuer den PingProzess
Process pingproc;
Runtime runtime = Runtime.getRuntime();
```

In der darauffolgenden Meßsequenz wird nun pro IP–Adresse eine Anfangszeit aus dem System ausgelesen, der Ping–Prozeß ausgeführt, und mit Hilfe der Endzeit die Prozeßdauer in Millisekunden berechnet.

```
// Pingsequenz
for (int i=0; i<anzahl; i++){
    String adresse = (String)adressen.elementAt(i);
    try{
        // Startzeit
        long begin = System.currentTimeMillis();
        pingproc = runtime.exec("ping "+ adresse);

        // hier kann eine Maximale Wartezeit ergaenzt werden
        pingproc.waitFor();

        // Endzeit
        long end=System.currentTimeMillis();
        // benoetigte Zeit
        Integer result = new Integer((int)end-(int)begin);

        // wurde Ping erfolgreich durchgefuehrt ??
```

```
        if (pingproc.exitValue()==0)
           results.addElement(result);
        else
            results.addElement(new Integer(-1));
            // -1 bedeutet Host nicht erreichbar
    }
    catch(Exception e){
        e.printStackTrace();
        cdo.setresults(results);
    }
}
```

Der Rückgabe–Wert signalisiert, ob die Messung erfolgreich war oder nicht. Abschließend wird das Ergebnis dem `DataObject` übergeben und dieser an den `DataHandler` weitergereicht.

```
cdo.setresults(results);
AgBasis.dataHandler.addDataObject(cdo);
alarmClock.setTimeoutAsLong(AgBasis.globalVariables.getcheckFrequence());
alarmClock.performStart();
```

Das erneute Setzen der `CheckFrequenz` gewährleistet eine sofortige Reaktion auf eine Änderung der Agenten–Konfiguration.
Der Aufruf `alarmClock.performStart()` veranlaßt die `alarmClock`, nach dem zuvor übergebenem Zeitraum ein neues Event zu erzeugen, das wiederum eine neue Meßsequenz einleitet. Das bedeutet, daß der Zeitraum zwischen den Messungen durch diesen Intervall festgelegt wird. Grund für diese Konstruktion ist die Tatsache, daß die Messungen unterschiedlich lange dauern können. Es es ist aber gewünscht, daß die Zeitspanne zwischen den Messungen konstant bleibt, um so dem lokalen Router die Möglichkeit zu geben, seinen Timeout zu durchlaufen und somit die Verbindung zu beenden, wenn keine Benutzerdaten mehr übertragen werden.
Abweichend von dieser Meßmethodik wurde im Vorfeld der Implementierung ein Zwei–Agenten Modell getestet, daß hier noch kurz vorgestellt wird. Basierend auf den in Abschnitt 4.3.2.1 geschilderten Überlegungen wurden ein Agent entwickelt, der eine Messung der Antwortzeit einen Echo–Agenten nutzt. Zu diesem Agenten, der an einem vorgegebenen Port lauscht, wird ein Socket aufgebaut und ein Datenpaket gesendet, das umgehend zurückgeschickt wird. Dazu wird durch den Echo–Agenten ein in Java verfügbares `ServerSocket`–Objekt nutzt. Eine ähnliche Realisierung ist mit Hilfe der JDMK–Adapter zu realisieren.
Das Prinzip dieser Messung ist ähnlich dem des Internetwerkzeuges `ping`, allerdings wird nicht ICMP genutzt, sondern eine TCP–basierte Socket–Verbindung. Von einer Implementierung dieser Meßmethode in den Prototypen wurde aufgrund der in Abschnitt 4.3.2.1 geführten Diskussion abgesehen.
Auf die Instrumentierung einer Applikation und der damit verbundenen Realisierung der Messung von Antwortzeiten auf Applikationsebene mußte aufgrund des zeitlich begrenzten Rahmens dieser Arbeit verzichtet werden.

5.6 Übertragung der Meßdaten

Die Übertragung der Meßdaten zwischen Agent und Manager basiert auf der Remote Method Invocation (RMI). Der Manager beinhaltet eine Klasse, die Methoden zur Übergabe von Daten an dessen Datenverwaltung beinhaltet. Der Agent ruft nun mittels RMI diese Methoden mit den Datenobjekten als Parameter auf. Dabei wird das Datenobjekt serialisiert und über die durch das JDMK bereitgestellte Infrastruktur übergeben. Zur Realisierung dieses Ansatzes müssen unterschiedliche Mechanismen und Klassen auf der jeweiligen Seite zum Einsatz kommen.

5.6.1 Beschreibung

Die zugrunde liegende Architektur und die dazu implementierten Klassen sind in Abbildung 5.6 dargestellt.
Der Manager stellt als Übertragung–Schnittstelle das M–Bean `MgDataReceiver` zur Verfügung. Um

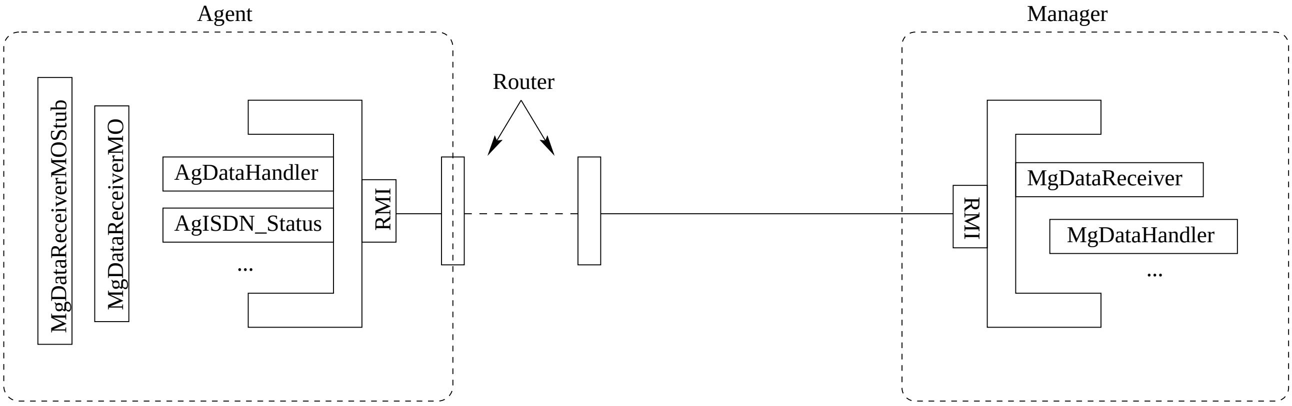

Abbildung 5.6: Architektur zur Meßdatenübertragung vom Agenten zum Manager

für den Agenten erreichbar zu sein, wird durch die Klasse `MgCMF` ein Common Management Framework instanziiert, in dem nur ein `MgDataReceiver`–Objekt registriert wird. Einzige Aufgabe dieses M–Beans ist die Bereitstellung der Methode `performReceiveData`, die beim Aufruf ein `DataObject` übergeben bekommt und dieses an den `MgDataHandler` weiterreicht. Von der Möglichkeit, den `MgDataHandler` direkt in das CMF einzuhängen wurde aus Sicherheitsgründen abgesehen, da über diese Klasse der Zugriff auf sämtliche Daten möglich ist.
Mit dem JDMK–Tool `mogen` werden die Klassen `MgDataReceiverMO` und `MgDataReceiverMOStub` erzeugt werden. Diese C–Beans müssen dem Agenten verfügbar gemacht werden, da sonst die durch das entfernte M–Bean bereitgestellte Methoden nicht kennt. Der Prozess der Meßdatenübertragung vom

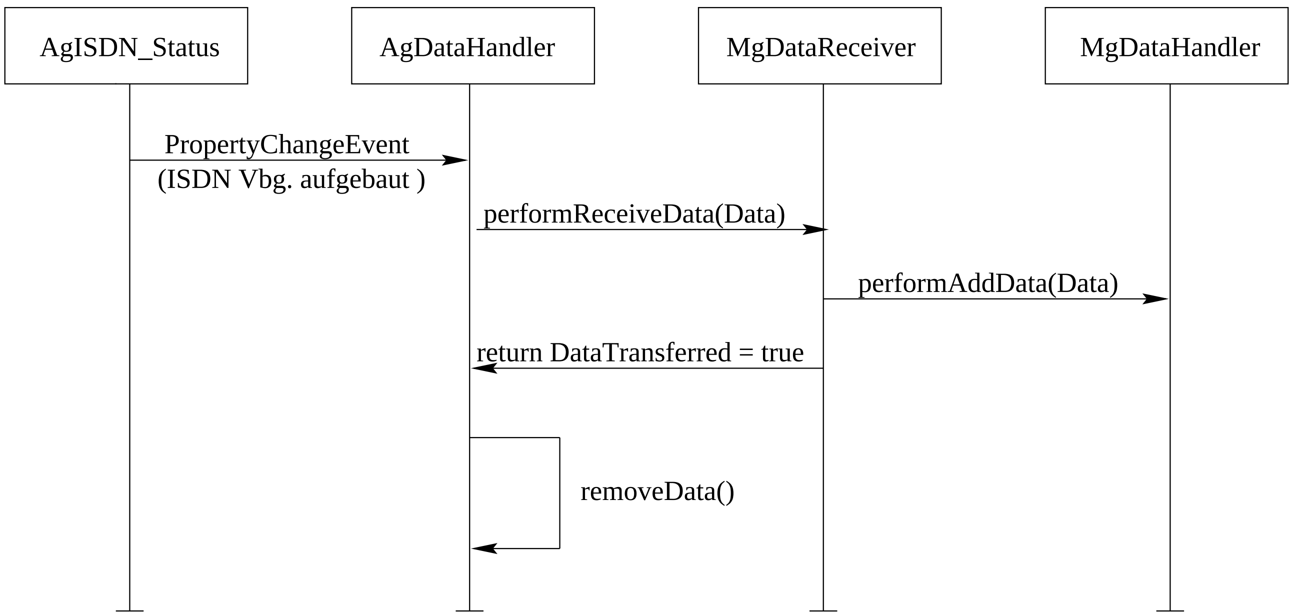

Abbildung 5.7: Prozeß der Meßdatenübertragung vom Agenten zum Manager

Agenten zum Manager ist leicht vereinfacht in Abbildung 5.7 dargestellt. Initiator der Übertragung ist die Klasse `AgISDN_Status`, die sobald die ISDN–Verbindung etabliert wird ein Event propagiert, das durch `AgDataHandler` registriert wird. In der Behandlung dieses Events wird die Methode `handleData` augerufen, die feststellt, ob Daten zu übertragen sind, und wenn ja, dies durch den Aufruf der Methode `performTransferData` mit dem entsprechenden `DataObject` durchführt.
`MgDataReceiver` nimmt über den oben beschriebenen Mechanismus die Daten entgegen und reicht diese mittels `performAddData` an die für die Datenverwaltung des Managers zuständige Klasse `MgDataHandler` weiter. Signalisiert diese Methode, daß die Daten ihr Ziel erreicht haben, wandert diese Nachricht über die einzelnen Rückgabewerte der Methoden zurück zum `AgDataHandler` und ermöglicht so das Löschen der übertragenen Daten aus der lokalen Datenhaltung.
Einige Details der Implementierung werden im folgenden Abschnitt dargestellt.

5.6.2 Realisierung

Die Umsetzung der oben beschriebenen Architektur in ein Klassenmodell ist in Abbildung 5.8 dargestellt. Erhält die Klasse `AgDataHandler` die Nachricht, daß die ISDN–Verbindung offen ist, wird durch

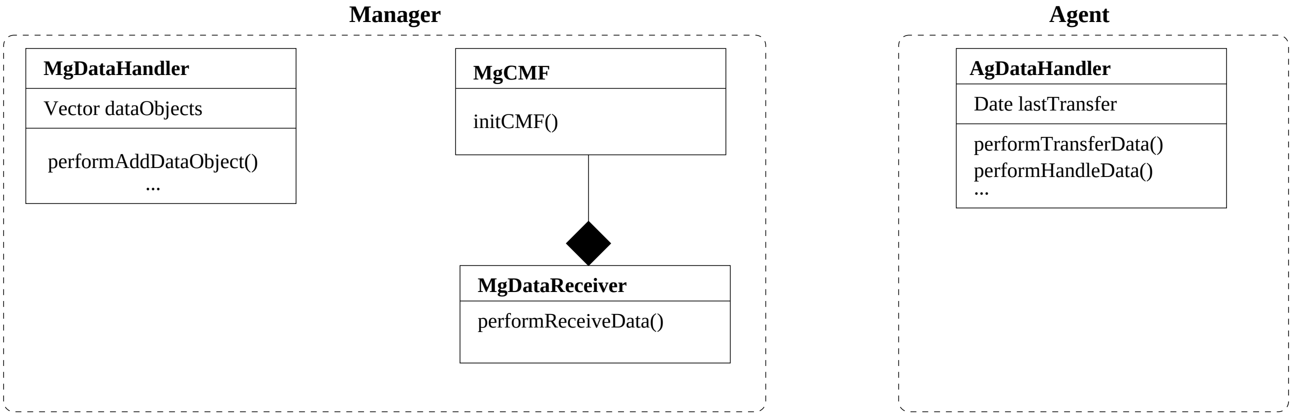

Abbildung 5.8: Diagramm der für die Meßdatenübertragung notwendigen Klassen

die Methode `handleData` geprüft, ob zur Übertragung vorhanden sind.

```
if (AgBasis.isdnStatus.performCheckISDNStatus() == 1) {
    int ok = 1;
    while ((dataObjects.size() > 0) && (ok == 1)){
        ok = performTransferData((DataObject)dataObjects.elementAt(0));
        if (ok == 1 )
            dataObjects.removeElementAt(0);
    }
}
```

Ist dies der Fall, werden sämtliche Datenpakete einzeln übertragen. Erst wenn die Methode `performTransferData` die korrekte Übergabe der Daten bestätigt hat, werden diese aus der lokalen Datenhaltung im Vektor `dataObjects` entfernt.
Um eine Verbindung zwischen dem Agenten und dem Manger herzustellen, wird ein sogenannter Adapter–Client erstellt. Dieser meldet sich bei dem RMI–Adapter des Managers an und erhält so Zugriff auf die dort platzierte Klasse `DataReceiver`.

```
String managerHost     = AgBasis.globalVariables.getmanagerHostName();
```

Dazu wird zunächst der Name des Managers aus der Konfigurationsklasse ausgelesen. Dann wird der Objektname des zu kontaktierenden M–Beans angegeben, der im Rahmen des Prototypen fest implementiert wird.

```
ObjectName name        = new ObjectName("defaultDomain:MgDataReceiver.SerialNo=1");
AdaptorClient adaptor  = new AdaptorClient();
adaptor.connect(null, managerHost, 1099, ServiceName.APT_RMI);
```

Die hier gezeigten Adapter–Clients entsprechen dem C–Bean Konzept des JDMK. Um nun auf den `MgDataReceiver` zugreifen zu können muß zunächst ein `AdaptorClient`–Objekt erzeugt werden. Mit Hilfe dieses Clients kann im folgenden Schritt die Verbindung zum Manager hergestellt werden. Dazu sind neben der Bezeichnung des Hostnamen auch die Art des Adapters (hier wurde ein RMI–Adapter genutzt) und der dazugehörige Port anzugeben.
Nun wird ein C–Bean mit Hilfe der `getObject`–Methode instanziiert.

```
MgDataReceiverMO dataReceiver
        = (MgDataReceiverMO)((Vector)adaptor.getObject(name,null)).firstElement();
```

Über dieses C–Bean kann auf die entfernte Klasse wie auf eine lokale zugegriffen werden. Dies geschieht im Prototypen wie folgt:

```
boolean transferOK = dataReceiver.performReceiveData(da);
```

Bei dieser Art des Zugriffes über den Adapter–Client handelt es sich um einen sogenannten High–Level Zugriff.
Diese Form der Realisierung ermöglicht das Pushen der Daten vom Agenten zum Manager, d.h. der Agent entscheidet aus eigenem Antrieb, wann und welche Daten er übertragen will. Im Prototypen ist der ausschlaggebende Faktor der Aufbau der ISDN–Verbindung. Solange diese besteht, wird jedes Datenpaket nach der Zusammenstellung direkt übertragen. In der Methode `AgDataHandler.performHandleData()` besteht die Möglichkeiten, Regeln zu ergänzen. Diese können z.B. basierend auf der Variable `lastTransfer` eine Mindestpause zwischen zwei Übertragungen (nur eine Übertragung innerhalb von 24 Stunden) festlegen, eine Mindestdatenmenge definieren (mindestens 20 `DataObjects`) oder bestimmen, wie lange die ISDN–Verbindung bestehen muß bevor die Daten übertragen werden dürfen. Schließt man z.B. aus, daß der Agent in den ersten 60 Sekunden einer ISDN–Verbindung nicht tätig werden darf, erhöht dies die Wahrscheinlichkeit, daß die Übertragung des Benutzer nicht gestört wird.

5.7 Konfiguration des Agenten

Die Konfiguration eines Agenten wird über die gleichen Mechanismen realisiert, wie die Datenübertragung um vorhergehenden Abschnitt. Daher wird hier auf eine Darstellung der Realisierung verzichtet. Abbildung 5.9 zeigt die für die Konfiguration des Agenten durch den Manager zuständigen Klassen.
In diesem Fall verfügt der Manager über das C–Bean der Agentenklasse `AgGlobalVariables`. Da diese Klasse bei dem Agenten im CMF registriert ist, kann der Manager mit Hilfe eines Adapter–Clients auf diese Klasse zugreifen und die Parameter mit den implementierten Getter– und Setter–Methoden auslesen und anpassen. Dieser Zugriff ist in den Methode `initElements` und `applyElements` der Klasse `MgGUIAgParam` implementiert.

5.8 Management-Möglichkeiten

Durch die unterschiedlichen Adapter und Protokolle, die mit dem JDMK zur Verfügung gestellt werden, bieten sich mehrere Möglichkeiten für die Administration der Agenten an. In den folgenden Abschnitten

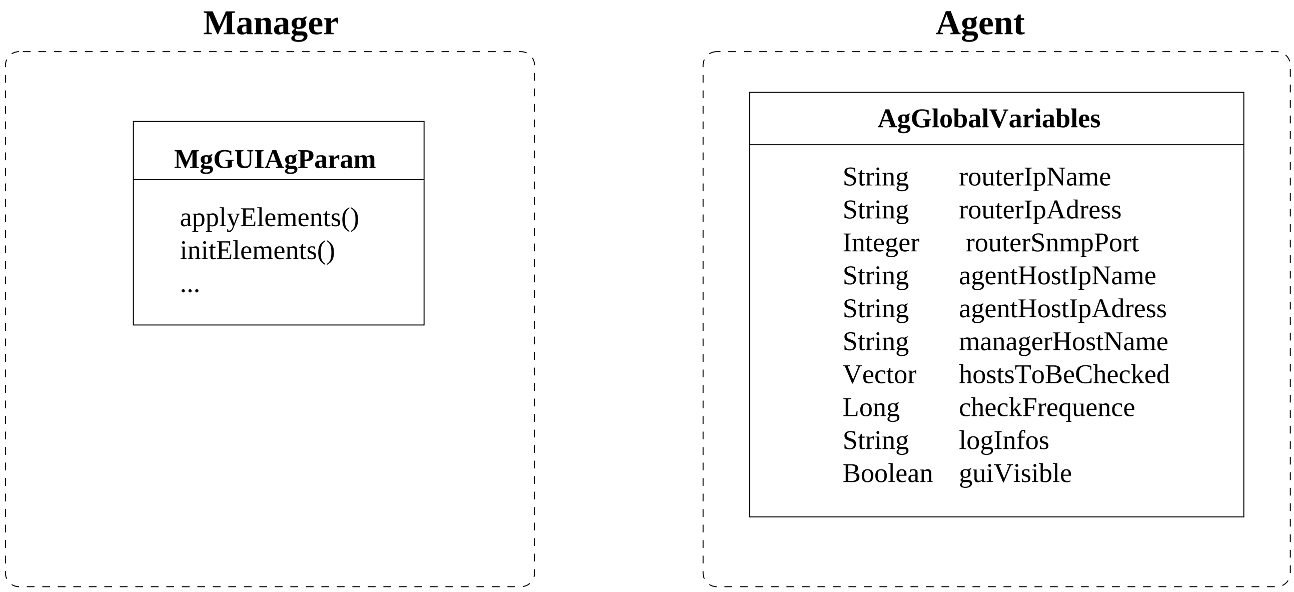

Abbildung 5.9: Diagramm der für die des Managers notwendingen Klassen

werden die unterschiedlichen Schnittstellen dargestellt und die Möglichkeiten für eine Integration in bestehende Managementsysteme dargestellt.

5.8.1 HTML - Browser

Das Management der Agenten ist im JDMK mittels eines HTML–Adapters möglich. Dieser agiert wie ein Web–Server und generiert einen Satz von HTML–Seite für den Manager, der die Parameter und Aktionen der verfügbaren M–Beans darstellt. Die Navigation zwischen den einzelnen Seiten wird über URLs realisiert. Über diese Schnittstelle sind nur die Klassen des Agenten sichtbar, die im CMF registriert wurden.
Interessant für den Nutzung dieser Schnittstelle ist, daß Methoden zur Verfügung stehen, die das Anpassen dieser HTML–Seiten erlauben. Dies ist insbesondere dann sinnvoll, wenn ein Nutzer zwar die Parameter eines M–Beans auslesen können soll, aber nur eingeschränkt auf dessen Funktionalität. Außerdem wird standardmäßig das Löschen einzelner M–Beans auf solchen HTML–Seiten möglich gemacht, was in den seltensten Fällen erwünscht ist.
Folgende Methoden, die mit dem `HtmlStreamableIf`–Interface angeboten werden, erlauben das Anpassen der HTML–Seiten:

- `public boolean isCustomizedViewOnly()`
  Diese Methode bestimmt das Verhalten des Algorithmus für das Generieren der HTML–Seiten, die den Agenten repräsentieren. Gibt diese Methode `true` zurück, so wird *nur* die benutzerdefinierte Seite angezeigt, gibt sie `false` zurück, so wird die generierte Seite um die benutzerdefinierten Informationen ergänzt.

- `public String writeToHtml()`
  Der in dieser Methode zurückgegebene Wert wird zur Erstellung der HTML–Seite interpretiert. Wird durch die Methode `isCustomizedViewOnly() false` zurückgegeben, so wird diese Information vor der generierte Seite angezeigt.

Diese Managementschnittstelle kann durch ein Authentifizierungsverfahren vor unberechtigtem Zugriff geschützt werden. So wird durch Objekte der Klasse `AuthInfo` einzelne Benutzer angelegt, denen der Zugriff auf diese HTML–Seiten erlaubt werden soll. Diese Objekte werden wie folgt angelegt:

```
// HTML Adapter mit Authentifizierung
AuthInfo ai1 = new AuthInfo("root","strengGeheim");
html.addUserAuthenticationInfo(ai1);
AuthInfo ai2 = new AuthInfo("hojnacki","nurGeheim");
html.addUserAuthenticationInfo(ai2);
```

In diesem Beispiel werden zwei Benutzer angelegt, einer mit dem Namen `root` und dem Passwort `strengGeheim` und einer mit dem Namen `hojnacki` und dem Passwort `nurGeheim`. Diese Informationen werden dann mit der Methode `addAuthentificationInfo` an den HTML–Adapter übergeben und aktiviert.
Es gibt allerdings keine Möglichkeit, unterschiedliche Benutzergruppen mit unterschiedliche Rechten anzulegen. Zudem werden diese Informationen bei dem Anmelden unverschlüsselt an den Agenten übertragen. Diese zwei Nachteile machen den Einsatz des ansonsten reichlich ausgestatteten HTML–Adapter in der Praxis fast unmöglich.

5.8.2 Management–Applikation

Im Rahmen des Prototypen wurde, wie bereits beschrieben, eine Management–Applikation entwickelt. Grund dafür ist die Notwendigkeit eines Mechanismus zur Entgegennahme der Daten der einzelnen Agenten. Es ist nicht ausreichend, ähnlich wie bei SNMP–Agenten, die Daten der Agenten abzufragen und dann z.B. über einen Web–Browser zu visualisieren. Vielmehr muß es einen Datenpool geben, der die Informationen, die zum Manager gepusht werden, entgegen nimmt und verwaltet. Sinnvoll über eine HTML–Schnittstelle ist z.B. das Konfigurieren der Agenten. Allerdings ist zu überprüfen, ob aufgrund der Sicherheitsmängel dieses Adapters eine Manager–Applikation nicht zu bevorzugen ist.
Folgenden Anforderungen wird diese prototypischen Implementierung des Managers gerecht:

- Graphische Benutzeroberfläche zur Bedienung des Managers
- Konfiguration des Agenten
- Schnittstelle für die Datenübertragung
- Verwaltung und Darstellung der Daten

Der Manager verfügt über eine graphische Benutzeroberfläche, die die Bedienung sämtlicher Komponenten umfaßt. Nach dem Start erscheint die in Abbildung 5.10 dargestellte „Manager Zentrale“, die in folgende fünf Bereiche aufgeteilt ist:

1. **Prozeßkontrolle**
   In diesem Bereich wird die Möglichkeit gegeben, das CMF zu stoppen oder zu starten. Dies ist insbesondere dann sinnvoll, wenn man über diesen Manager nur die Konfiguration eine Agenten ändern oder gesammelte Daten betrachten will.
   Desweiteren können hier Agenten über ein gesondertes Fenster konfiguriert werden.

2. **Datenbank**
   Dieser Bereich bietet Funktionen zum Abspeichern, Wiederherstellen und Betrachten von Daten an. Das Speichern und Laden der Daten wird über die Klasse `IoHandler` implementiert, deren Methoden so durch die anderen Klasse genutzt werden, das Anpassen oder Überschreiben der jeweiligen Methode das Speichermedium geändert werden kann. Im Prototypen werden die serialisierten Datenobjekt im lokalen Dateisystem gesichert. Leicht zu realisieren ist alternativ die Nutzung einer Datenbank.
   Die Anzeige der Daten wird durch den „Meßdaten–Browser“ realisiert.

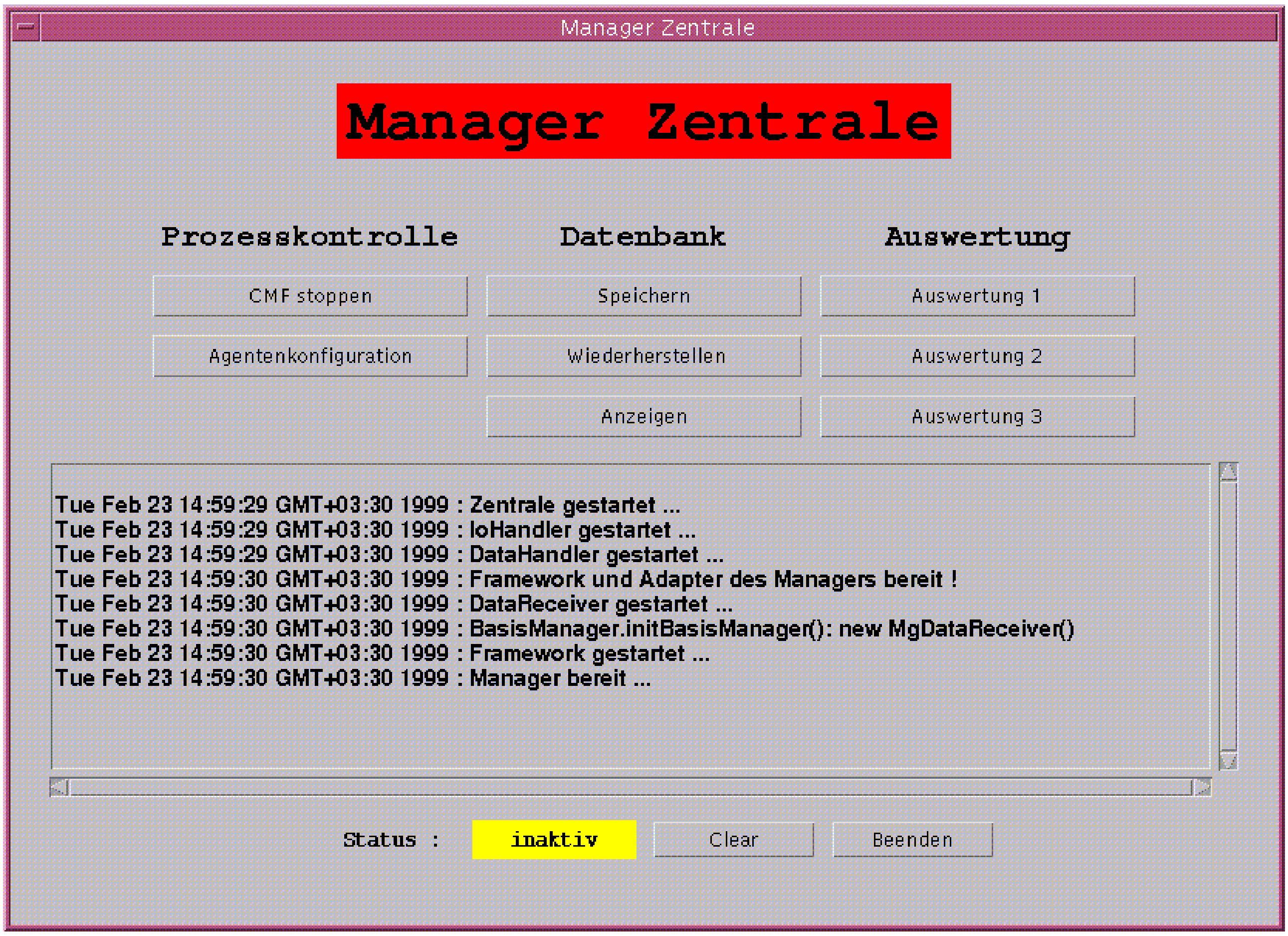

Abbildung 5.10: Die Zentrale des Managers

3. **Auswertung**
   Hier können über verschiedene Buttons Auswertungsmethoden wie z.B. Durchschnittswerte, Ausreißer etc. angeboten werden. Diese wurden allerdings im Rahmen des Prototypen nicht implementiert.

4. **Console des Managers**
   In dem Console–Fenster werden Informationen über die derzeitigen Aktivitäten des Managers angezeigt. So kann man während der Initialisierungsphase den jeweiligen Fortschritt verfolgen und jede Art von Kontakt zu Agenten ablesen.

5. **Allgemeiner Bereich**
   Im Fußbereich der Zentrale ist der Status des Managers angezeigt. Diese Information beinhaltet, ob ein Kontakt zu Agenten besteht (aktiv) oder nicht(inaktiv). Desweiteren besteht die Möglichkeit, die Console zu leeren (Log–Informationen werden in einer Datei gespeichert) und den Manager zu beenden.

Zur Konfiguration der Agenten wird ein Fenster geöffnet, das über die Eingabe des jeweilige Hostnamen eine Verbindung herstellt und die aktuelle Konfiguration abfragt. Mechanismen wie ein Cache, der die letzte bekannte Konfiguration speichert, Überprüfung des Verbindungsstatus zur Vermeidung unnötiger ISDN–Kosten oder Speicherung neuer Konfigurationen, die sich der Agent beim nächsten Kontakt abholt, wurden geprüft, aber nicht implementiert, d.h. eine Realisierung ist leicht in die ge-

gebene Infrastruktur einzubauen.
Zur Konfiguration der Agenten wird ein Fenster geöffnet, das über die Eingabe des jeweilige Host-

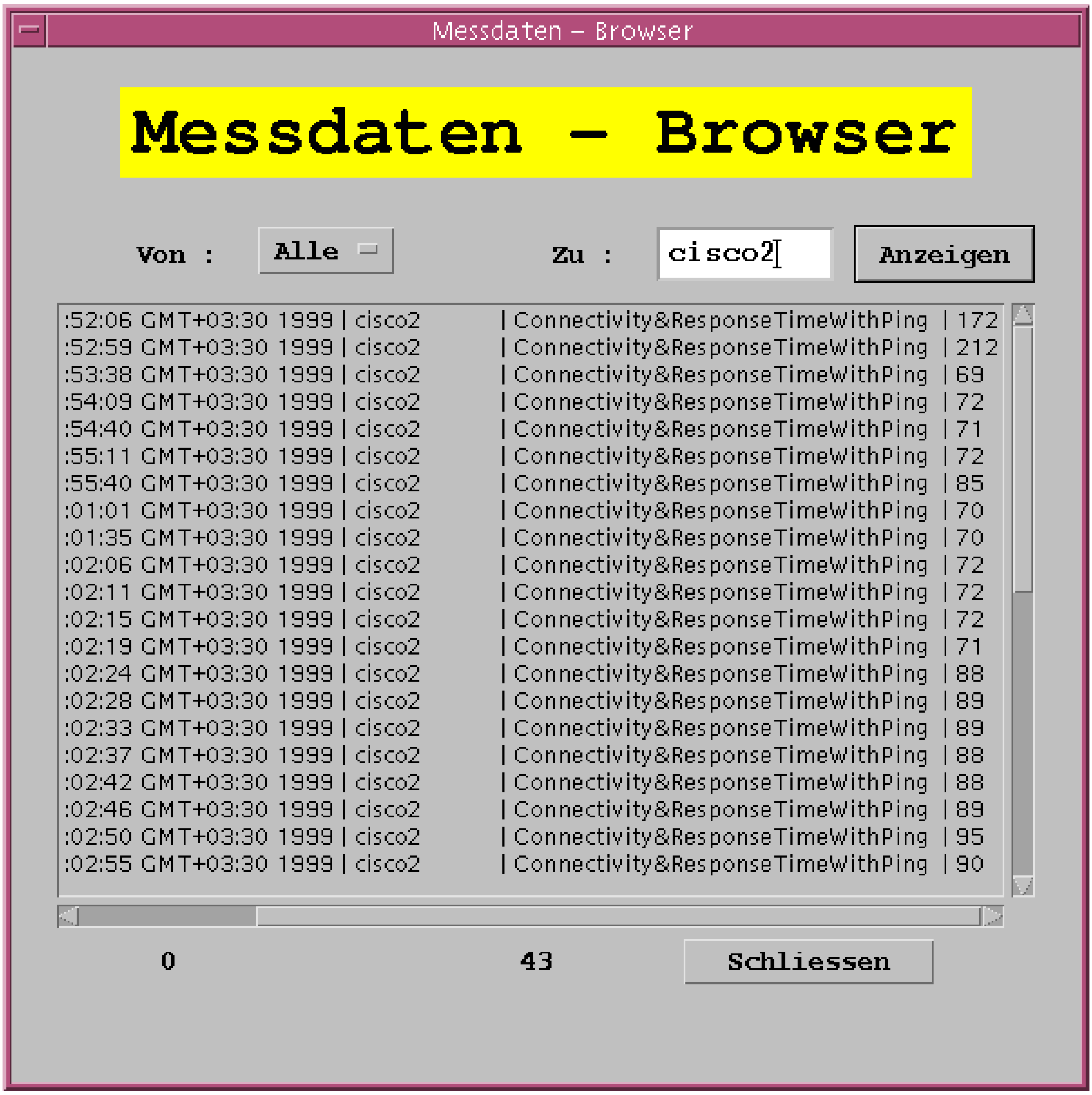

Abbildung 5.11: Der Datenbrowser des Managers

namen eine Verbindung herstellt und die aktuelle Konfiguration abfragt. Mechanismen wie ein Cache, der die letzte bekannte Konfiguration speichert, Überprüfung des Verbindungsstatus zur Vermeidung unnötiger ISDN–Kosten oder Speicherung neuer Konfigurationen, die sich der Agent beim nächsten Kontakt abholt, wurden geprüft, aber nicht implementiert, d.h. eine Realisierung ist leicht in die gegebene Infrastruktur einzubauen.
Der in Abbildung 5.11 zu erkennbare Meßdaten–Browser ermöglicht eine einfache Form der Betrachtung der in der Datenbank vorhandenen Meßdaten. Hier können alle Daten angezeigt werden, aber auch durch Auswahl der Start– und Zielkomponenten eine Auswahl getroffen werden. In der Abbil-

dung sind z.B. alle Messungen dargestellt, die von jedem beliebigen Rechner zur Komponente mit der Bezeichnung `cisco2` durchgeführt wurden. Angezeigt wird neben der Uhrzeit der Messung auch die Meßmethode, hier `Connectivity&ResponseTimeWithPing`, und das Ergebnis in Millisekunden. Eine -1 würde hier die Nicht–Verfügbarkeit signalisieren.
Auf die Implementierung der graphischen Benutzeroberfläche wird im Rahmen dieser Arbeit nicht eingegangen.

5.8.3 JDMK–Tool JOB

Das JDMK–Tool JOB bietet eine grafische Oberfläche zur Kontaktaufnahme mit einem Agenten und zur Visualisierung der bei ihm registrieten M–Beans. Er ist in erster Linie als Debugging–Tool vorgesehen, ist allerdings in Bezug auf die Funktionalität gegenüber einem JDMK–Agenten eine gute Demonstration für ein einfaches Management–Tool. In Abbildung 5.12 ist die Oberfläche des job–Tools zu sehen, daß einen Kontakt zu einem JDMK–Agenten auf dem IP–Host `iller` aufgebaut hat. Dieses Tool bietet die Möglichkeiten, Parameter des Agenten anzupassen und Aktionen auszuführen,

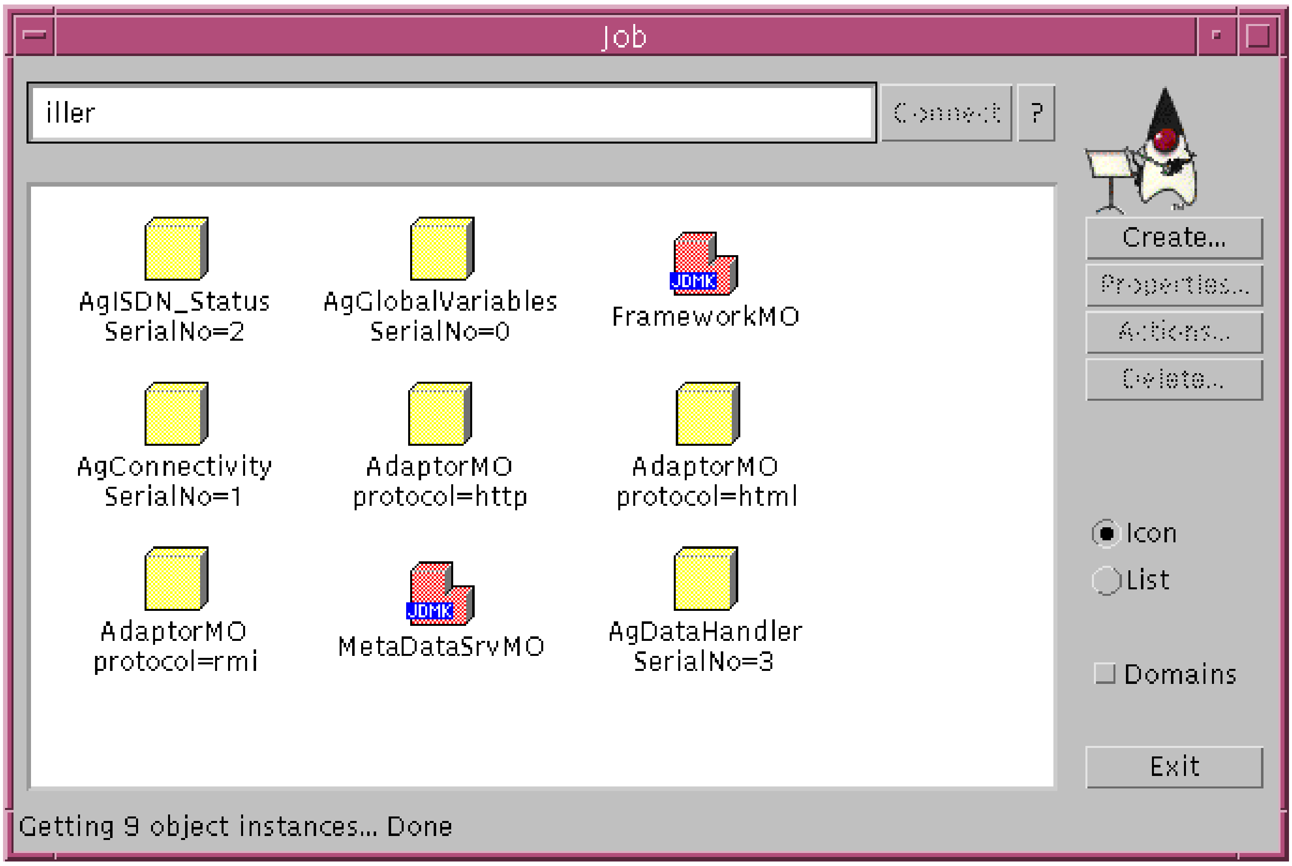

Abbildung 5.12: Das JDMK–Tool JOB

die in registrierten M–Beans zur Verfügung gestellt werden.

Kapitel 6

Zusammenfassung und Ausblick

Die rasende Entwicklung in Bereichen der Netzwerktechnologie erfordert ein leistungsfähiges, zuverläßiges und aussagekräftiges IT–Management. Zudem fordert der ständige Wandel in der Topologie dieser Netze Dynamik und Flexibilität.
In dieser Arbeit wird gezeigt, wie ein Konzept für ein solches Netzmanagement unter den speziellen Vorgaben der DeTeSystem und den begrenzten Anforderungen eines Teilbereiches aussehen kann. Die prototypische Implementierung eines Agentensystem zeigt, daß das auf der Theorie der flexiblen Management Agenten basierende Konzept mit den vorgegebenen Werkzeugen realisierbar ist.

6.1 Zusammenfassung der Ergebnisse

Die DeTeSystem ist ein Anbieter von Komplettlösungen für IT–Systeme, der diese kundenspezifisch entwickelt, realisiert und betreibt.
Anhand des Netzes eines Fahrzeugherstellers, der durch ein Extranet Online–Dienste fuer Händler bereitstellt, wurde beispielhaft dargestellt, wie sich Kundennetze in das Infrastrukturkonzept der DTS eingliedern. Besondere Beachtung fand hierbei die bedarfsabhängige Anbindung (Dail on Demand) der geographisch weit verteilten Händlerlokationen mittels einer Wählverbindung.
Um eine vertragliche Basis zwischen dem Fahrzeughersteller und der DeTeSystem zu definieren werden sogenannte Service-Level vereinbart, die die geforderte Qualität der IT–Services aus Sicht der Kunden dokumentieren.
Die Gegenüberstellung der bestehenden Management–Architektur und der aus diesen Service–Level resultierenden Anforderungen verdeutlicht die Problematik, vor denen viele Service Provider heute stehen. Die neugearteten Vertragsparameter, die Service–basiert die Sicherstellung eines Qualitätsgrades beim Endbenutzer definieren, stehen einer Architektur mit zentralisierter Management–Intelligenz gegenüber.
Einen vielversprechenden Ansatz zur Lösung dieser Problematik stellen verteilte Management Systeme auf der Basis flexibler Management–Agenten dar. Hierbei wird die Management–Intelligenz in der Nähe der zu verwaltenden Komponente platziert und führt von hier aus im Auftrage des Managers unterschiedliche Aufgaben aus.
Unter besonderer Berücksichtigung des Managements von Lokationen, die nur zeitweise über eine ISDN–Verbindung mit dem Extranet verbunden sind, leitet sich so ein Anforderungkatalog mit folgenden zentralen Elementen her:

- Dynamische Veränderbarkeit der Agenten
- Geringer Verteilungsaufwand
- Kein selbständiger ISDN–Verbindungsaufbau durch den Agenten

- Messung von Antwortzeiten auf Netzwerk– und Applikationsebene

Um ein solches Agentensystem zu entwickeln wird das Java–Dynamic–Management–Kit (JDMK) benutzt, das einen Satz von Klassen und Tools anbietet, die die Entwicklung von flexiblen und intelligenten Management–Agenten ermöglichen.
Die Antwortzeitmessung auf Applikationsebene erfordert die Instrumentierung der Anwendungen. Unter den aktuellen Ansätzen bietet sich für die Benutzung im Anwendungsszenario besonders die Application Response Measurement (ARM) –API an, da diese eine genaue Abgrenzung der zu messenden Transaktionen ermöglicht. Zudem ist bei entsprechender Implementierung die Korrelation von Sub–Transaktionen möglich, um so aussagekräftige Informationen über eventuelle Fehlerquellen zu erhalten.
Die Entwicklung des Konzeptes für ein Agentensystem zur autonomen Antwortzeitmessung stellt die Grundlage für die anschließende prototypische Implementierung dar. Kern dieses Konzeptes ist Entwicklung einer Architektur und deren Einordnung in das Anwendungsszenario. Dabei werden die für die Kommunikation zu verwendenden Protokolle festgelegt, ein Organisationsmodell für die Agenten entwickelt und Verfahren zur Messung der Antwortzeiten entworfen.
Die Umsetzbarkeit dieses Konzeptes wird durch die Realisierung eines Prototypen bewiesen. Dieser besteht aus einer Manager–Applikation und einem FMA, die über eine ISDN–Verbindung entsprechend der definierten Anforderungen miteinander kommunizieren.

6.2 Ausblick

6.2.1 Prototyp

Der Prototyp beweist die Realisierbarkeit des in dieser Arbeit entwickelten Konzeptes, setzt allerdings nicht alle durch das Anwendungsszenario gestellten Anforderungen um. In diesem Abschnitt wird ansatzweise dargestellt, welche Weiterentwicklungen vorgenommen werden können und welche erweiterten Einsatzgebiete eines solchen Systems vorstellbar sind.

6.2.1.1 Discovery Service

Der Discovery Service dient wie in Abschnitt 3.2.3 dargestellt der Ermittlung von Agenten im Netzwerk. Dieser Dienst ist im Rahmen des gegebenen Szenarios von größter Nützlichkeit, da mit ihm folgende Fragen einfach gelöst werden können:

- *Welche Lokationen sind online?*
  Um festzustellen, welche Lokationen gerade über eine ISDN–Verbindung mit dem jeweiligen PoP und somit mit dem DTS–CN verbunden sind, kann mit diesem Dienst eine einfache Verfügbarkeitsprüfung der Agenten durchgeführt werden.

- *Welche Agenten können gerade neu konfiguriert werden?*
  Ein solcher Update Mechanismus ist dann sinnvoll, wenn ein „Abholen“ der neuen Konfiguration durch den Agenten nicht möglich ist, z.B. wenn eine Funktionalität dazu nicht (mehr) vorhanden ist.

- *Wann war die Lokation zuletzt am Netz?*
  Sollte ein Agent längere Zeit keine Daten übertragen haben, ist es u.U. hilfreich zu wissen, wann er das letzte Mal erreichbar war. So kann sichergestellt werden, daß es sich hierbei nicht um einen Ausfall des Agenten handelt.

Diese und ähnliche Fragen können mit dem Discovery Service gelöst werden. Um diesen Dienst nutzen zu können, müssen die Agenten mit einem Discovery Responder ausgestattet werden. Dieser kann sich als M–Bean im CMF registriert in der spezifizierten Multicast–Gruppe anmelden und auf einen

Manager–Request reagieren.
Grundsätzlich muß allerdings für diesen Dienst sichergestellt sein, daß die PoPs, mit denen der Manager verbunden ist, für solche Anfragen keine ISDN–Verbindung aufbauen.

6.2.1.2 Cascading Service

Bezugnehmend auf die in Abschnitt 4.2.2 dargestellten organisatorische Merkmale eines Agentensystems kann der Cascading Service dazu benutzt werden, dieses Konzept zu implementieren. Da im Anwendungsszenario (Abb. 2.2) eine einstufige Hierarchie vorgegeben ist, wurde dieser Service im Prototypen nicht genutzt. Sobald sich allerdings eine mehrstufige Hierarchie ergibt, muß diese mit dem Cascading Service implementiert werden.

6.2.1.3 Gauge Monitor

Um die Forderung zu Erfüllen, mehr Intelligenz auf den Agenten zu verlagern, kann der Gauge Monitor des JDMK benutzt werden. Mit Hilfe dieses Dienstes hat der Agent die Möglichkeit, die ihm vorliegenden Daten zu analysieren und in Abhängigkeit von dem Ergebnis zu reagieren. So kann der er z.B. beim Über– bzw. Unterschreiten eines Grenzwertes den Manager alarmieren.
Die Funktionsweise des Gauge Monitor ist in Abbildung 6.1 dargestellt. Anhand des Beispieles der im

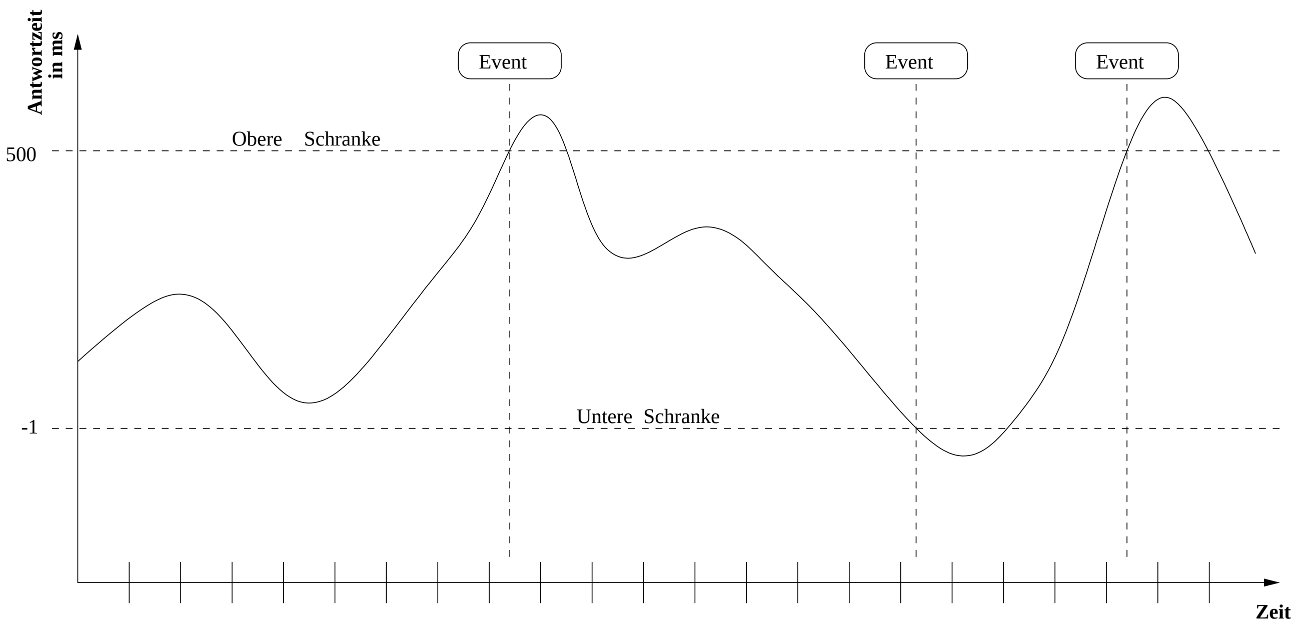

Abbildung 6.1: Die Funktionsweise des Gauge Monitors

Prototypen implementierten Meßmethode soll der Einsatz dieses Services dargestellt werden.
Der Gauge Monitor erzeugt bei Über– bzw. Unterschreiten einer Schranke ein Event. Diese Schranken sind nun so zu definieren, daß sie den im SLA vereinbarten Werten entsprechen. Liegen die Schranken unterhalb dieser Marke und signalieren sie das Überschreiten eines kritischen Punktes, so ermöglichen sie ein proaktives Management, d.h. es können Maßnahmen eingeleitet werden, die das Abweichen von dem vertraglich vereinbarten Service–Level verhindern.
Tritt nun der alarmierende „Ausnahmefall“ ein, daß so eine Schranke verletzt wird, so ist es sinnvoll, wenn der Agent auf jeden Fall eine Verbindung zum Manager herstellen darf, auch wenn dies über ein ISDN–Wählleitung geschieht.

6.2.1.4 Einbindung in eine bestehende Management–Architektur

Das Management eines Netzwerkes wird selten durch eine Applikation betrieben. Vielmehr sammeln sich unterschiedlichste Programme verschiedener Hersteller, die jeweils für ihren Teilaspekt der Aufgabe die beste Funktionalität beinhalten. Als Dach einer solchen Sammlung dienen Management–Plattformen wie z.B. HP OpenView.
Die Weiterentwicklung des prototypisch implementierten Managers wird dahin weiterzuführen sein, daß eine Eingliederung in eine solche Plattform ermöglicht wird. Die Vielzahl der durch das JDMK zur Verfügung gestellten Adapter erleichtert dies, insbesondere der HTML–Adapter, der ein Web–basiertes Management realisiert.
Der SNMP–Adapter erlaubt die Entwicklung von SNMP–Agenten. So kann eine Architektur entwickelt werden, in der der FMA in kritischen Situationen Traps an andere Management–Applikationen schickt.
Die Auswertung der gesammelten Daten sollte über eine externe Applikation geschehen, unter Umständen durch ein an eine Datenbank gekoppelte Funktionalität.

6.2.2 JDMK

Die Kombination einer plattformunabhängigen Programmiersprache wie Java mit einem vielseitigen Werkzeug zur Implementierung von Agentensystemen wie JDMK ist eine erfolgsversprechende Kombination zur Lösung aktueller Anforderungen an verteilte Systeme.
Allerdings befindet sich das JDMK derzeit noch in der Betaphase, was anhand einer nicht unwesentlichen Anzahl von Bugs auch sinnvoll erscheint. So zeigten sich auch in der Entwicklungsphase des Prototypen grobe Fehler in der Implementierung des MIBGEN–Tools und einiger Services wie z.B. dem Discovery Service.
Allerdings ist trotzdem bei der Verfolgung der JDMK–News–Foren deutlich zu erkennen, daß ein breites Interesse an diesem Entwicklungswerkzeug besteht. Auch größere Firmen wie Siemens und AT&T prüfen derzeit das JDMK auf mögliche Einsatzfelder.
Viel Arbeit wird derzeit, wenn man den Aussagen der Entwicklern trauen kann, in die Entwicklung einer neuen Version des JDMK investiert, die dann mit dem JDK1.2 kompatibel sein soll. Neben dem Beheben bestehender Fehler soll schon in dieser Version die Möglichkeit der Kombination mit JINI gegeben sein.

Anhang A

Sourcecode des flexiblen Management–Agenten

A.1 Die Klasse AgBasis

```
import java.lang.String;
import java.util.Vector;
import com.sun.jaw.reference.common.*;
import com.sun.jaw.reference.agent.cmf.*;
import com.sun.jaw.reference.agent.services.*;
import com.sun.jaw.impl.agent.services.light.*;
import com.sun.jaw.impl.agent.services.alarm.*;
import com.sun.jaw.impl.adaptor.security.*;

/**
 * Diese Klasse implementiert den Basisagenten:
 * Start der benoetigten Dienste, Instanziierung der M-Beans,
 * Registrieren im Framework
 *
 * @version 20.3.99
 * @author Michael Hojnacki
 *
 */
public class AgBasis {

    static AgConnectivity     connectivity;
    static AgISDN_Status      isdnStatus;
    static AgDataHandler      dataHandler;
    static AgGUIStatus        guiStatus;
    static AgGlobalVariables  globalVariables;
    static IoHandler          ioHandler;
    static AlarmClock         alarmClock;

    private Framework cmf;
    private ObjectName c, i, d, gv;


    /**
     * EmptyKonstruktor
     * @param Keiner
     */
    public AgBasis(){

ioHandler = new IoHandler();
guiStatus = new AgGUIStatus();
// Basisdienste des Agenten starten
try {

    String domain = "defaultDomain";

    // Framework starten
    cmf = new Framework();
```

```
    // MetaDataService starten
    String mtdSrvClass = "com.sun.jaw.impl.agent.services.light.MetaDataSrv" ;
    String mtdSrvName = domain + ":" + ServiceName.META ;
    cmf.newObject(mtdSrvClass, mtdSrvName, null) ;

    // RMI Adapter starten
    String rmiSrvClass = "com.sun.jaw.impl.adaptor.rmi.AdaptorServerImpl" ;
    String rmiSrvName = domain + ":" + ServiceName.ADAPTOR + ".protocol=rmi" ;
    cmf.newObject(rmiSrvClass, rmiSrvName, null) ;

    // HTTP Adapter starten
    String httpSrvClass = "com.sun.jaw.impl.adaptor.http.AdaptorServerImpl" ;
    String httpSrvName = domain + ":" + ServiceName.ADAPTOR + ".protocol=http" ;
    cmf.newObject(httpSrvClass, httpSrvName, null) ;

    // HTML Adapter starten
    String htmlSrvClass = "com.sun.jaw.impl.adaptor.html.AdaptorServerImpl" ;
    String htmlSrvName = domain + ":" + ServiceName.ADAPTOR + ".protocol=html" ;
    com.sun.jaw.impl.adaptor.html.AdaptorServerImpl html =
(com.sun.jaw.impl.adaptor.html.AdaptorServerImpl)cmf.newObject(htmlSrvClass, htmlSrvName,null) ;
    // HTML Adapter mit Authentifizierung
    AuthInfo ai1 = new AuthInfo("root","root");
    html.addUserAuthenticationInfo(ai1);
    AuthInfo ai2 = new AuthInfo("hojnacki","hojnacki");
    html.addUserAuthenticationInfo(ai2);

    guiStatus.performInitGUIStatus("Framework und Adapter bereit !") ;

}
catch (Exception e){
    System.out.println("BasisAgent.BasisAgent(): Exception !!");
    e.printStackTrace();
    System.exit(1);
}
    }


    /**
     * Objekte erzeugen und im Framework anmelden
     */
    public void initBasisAgent(){


// toConsole("BasisAgent.initBasisAgent(): new AlarmClock()");
// Variabeln
globalVariables = new AgGlobalVariables();
// toConsole("BasisAgent.initBasisAgent():new AgGlobalVariables()");
// Verwaltung des Isdnstatus
isdnStatus   = new AgISDN_Status();
// toConsole("BasisAgent.initBasisAgent(): new AgISDN_Status()");
// Messklasse
connectivity = new AgConnectivity();
// toConsole("BasisAgent.initBasisAgent(): new AgConnectivity()");
// Verwaltung der Daten
dataHandler   = new AgDataHandler();
// toConsole("BasisAgent.initBasisAgent(): new AgDataHandler()");


// M-Beans registrieren !!
try {
    String domain = cmf.getDomain();

    gv = new ObjectName(domain + ":" + "AgGlobalVariables.SerialNo=0");
    cmf.addObject(globalVariables, gv);
    c = new ObjectName(domain + ":" + "AgConnectivity.SerialNo=1");
    cmf.addObject(connectivity, c);
    i = new ObjectName(domain + ":" + "AgISDN_Status.SerialNo=2");
    cmf.addObject(isdnStatus, i);
    d = new ObjectName(domain + ":" + "AgDataHandler.SerialNo=3");
    cmf.addObject(dataHandler, d);
```

```
    toConsole("BasisAgent.initBasisAgent(): Objekte registriert!");
    toConsole("Agent gestartet ...");
}

catch (InstanceAlreadyExistException e){
    System.out.println("BasisAgent.initBasisAgent():" +
       " InstanceAlreadyExistException !!");
    e.printStackTrace();
    System.exit(1);
}

    }

    /**
     * zur Ausgabe auf eine Konsole
     */
    public static void toConsole(String text){
guiStatus.performInitGUIStatus(text);
globalVariables.setlogInfos(text);
    }

    /**
     * starten des Agenten !
     */
    public static void main(String argv[]){
// neuen Agenten erstellen
AgBasis agent = new AgBasis();
agent.initBasisAgent();
    }

}
```

A.2 Die Klasse AgConnectivity

```
import java.lang.String;
import java.util.Vector;
import java.net.*;
import java.util.Date;
import java.beans.*;

/**
 * Implementiert eine Pingroutine zur Ueberpruefung der Verbindung zu bestimmten
 * Netzkomponenten (Server, Gateways)
 *
 * @version 22.3.99
 * @author Michael Hojnacki
 */
public class AgConnectivity extends AgMeasurement implements PropertyChangeListener{


    private DataObject cdo;

    public AgConnectivity(){
super();
// auf die Aenderung des ISDN Status reagieren !
AgBasis.isdnStatus.addPropertyChangeListener(this);
    }

    /**
     * Ueberpruefung der Verbindung via ping
     */
    public void performMeasure(){

// ueberpruefen ob die ISDN - Verbindung noch steht !??!
if(AgBasis.isdnStatus.getISDNStatus()==1){
    AgBasis.toConsole("Messung moeglich");
    Date now = new Date();

    Vector adressen = AgBasis.globalVariables.gethostsToBeChecked();
```

```
    DataObject cdo =
new DataObject(now , true, getownIPAdress(), false,
       "Connectivity&ResponseTimeWithPing", adressen,getownIPAdress() );

    int anzahl = adressen.size();
    Vector results = new Vector();
    Process pingproc;
    Runtime runtime = Runtime.getRuntime();
    String parameter = "";

    if (anzahl > 0) {

// erster Ping verfaelscht !!
try{
    pingproc = runtime.exec("ping "+ InetAddress.getLocalHost().getHostName());
    pingproc.waitFor();
}
catch(Exception e){
    System.out.println("Connectivity.performMeasurement(): Exception ");
    e.printStackTrace();
}
for (int i=0; i<anzahl; i++){
    String adresse = (String)adressen.elementAt(i);
    try{
long begin = System.currentTimeMillis();
pingproc = runtime.exec("ping "+ adresse);
pingproc.waitFor();
long end=System.currentTimeMillis();
Integer result = new Integer((int)end-(int)begin);
if (pingproc.exitValue()==0){
    results.addElement(result);
    System.out.println("ping "+ adresse);
}
else
    results.addElement(new Integer(-1));
    }
    catch(Exception e){
System.out.println("Connectivity.performMeasurement(): Exception ");
e.printStackTrace();
cdo.setresults(results);
    }
}
    }
    cdo.setresults(results);
    AgBasis.dataHandler.addDataObject(cdo);
    alarmClock.setTimeoutAsLong(AgBasis.globalVariables.getcheckFrequence());
    alarmClock.performStart();
}

else {
    AgBasis.toConsole("Messung nicht moeglich");
    alarmClock.setTimeoutAsLong(new Long(10));
    alarmClock.performStop();
}
    }

    /**
     * Auf geaenderten ISDN Status reagieren
     */
    public void propertyChange(PropertyChangeEvent e){

alarmClock.performStart();
    }
}
```

A.3 Die Klasse AgDataHandler

```
import java.beans.*;
import java.util.Date;
import java.util.Vector;
```

```
import java.lang.String;
import com.sun.jaw.reference.common.*;
import com.sun.jaw.impl.adaptor.rmi.*;

/**
 * Klasse zur Verwaltung der Daten auf der Agentenseite
 *
 * @version 2.3.99
 * @author Michael Hojnacki
 *
 */
public class AgDataHandler implements PropertyChangeListener{


    private Vector dataObjects;        // die Datenobjecte

    private Date    lastTransfer;      // letzte Uebertragung
    private boolean newData;           // Uebertragung notwendig

    // Empty Konstruktor
    public AgDataHandler(){
dataObjects       = AgBasis.ioHandler.readDataObject();
lastTransfer      = new Date(0);    // ewig her ...
newData           = false;          // noch keine ...
    }

    public void addDataObject(DataObject d){

dataObjects.addElement(d);
performHandleData();
    }


    /**
     * Methode zur Uebergabe der Daten an den DataHandler
     * @param (DataObject)
     * @return -1 fuer Transfer fehlgeschlagen, 1 OK
     */
    public int performTransferData(DataObject da){

// ISDN Connection noch da ??
// => Daten uebertragen
if ((AgBasis.isdnStatus.getISDNStatus())==1){

    String managerHost     = AgBasis.globalVariables.getmanagerHostName();
    String receiverClass   = "MgDataReceiver";
    ObjectName name
= new ObjectName("defaultDomain:MgDataReceiver.SerialNo=1");
    AdaptorClient adaptor  = new AdaptorClient();

    try{
adaptor.connect(null, managerHost, 1099, ServiceName.APT_RMI);
AgBasis.toConsole("connected to Host >> " + managerHost);
MgDataReceiverMO dataReceiver
    = (MgDataReceiverMO)((Vector)adaptor.getObject(name, null)).firstElement();
AgBasis.toConsole("DataReceiverMO contacted");

boolean transferOK = dataReceiver.performReceiveData(da);

if (transferOK){
    AgBasis.toConsole("Daten wurden uebertragen");
    return 1;
}
else{
    AgBasis.toConsole("Uebertragung Fehlgeschlagen !!");
    return -1;
}

    }
    catch (Exception e){
AgBasis.toConsole("AgDataHandler.performReceiveData(): Exception !!");
```

```
AgBasis.toConsole("Uebertragung Fehlgeschlagen !!");
e.printStackTrace();
return -1;
    }
}

return -1;
    }


    /**
     * Diese Methode versucht, die Daten zu uebertragen.
     * Ist dies nicht moeglich, werden diese gespeichert.
     */
    private void performHandleData(){

// bei weiteren Bedingungen fuer den Datentransfer
// (z.B. nur alle 24 Std, wenn 20 DataObjects gesammelt sind,
// naechster Transfer nach, ISDN Conn muss mind 10min bestehen ...)
// diese hier ueberpruefen ...

// ISDN Status ueberpruefen und alles transferieren
if (AgBasis.isdnStatus.performCheckISDNStatus() == 1) {
    int ok = 1;
    while ((dataObjects.size() > 0) && (ok == 1)){
ok = performTransferData((DataObject)dataObjects.elementAt(0));
if (ok == 1 )
    dataObjects.removeElementAt(0);
    }
}

// Datei aktualisieren
AgBasis.ioHandler.writeDataObject(dataObjects);
    }

    public void propertyChange(PropertyChangeEvent e){
//int status = ((Integer)(e.getNewValue())).intValue();
performHandleData();
    }

    public Date getlastTransfer(){
return lastTransfer;
    }

    public void setlastTransfer(Date nDate){
lastTransfer = nDate;
    }

    public void setnewData(boolean nd){
newData = nd;
    }

    public boolean getnewData(){
return newData;
    }
}
```

A.4 Die Klasse AgGUIIsdnStatus

```
import java.awt.*;

public class AgGUIIsdnStatus extends Frame{

    Font font;
    Panel panel;
    Graphics light;
    String statusText;
    Color c1;
    Color c2;
    Color c3;
```

```
    public AgGUIIsdnStatus(){

super("ISDN");
font = new Font("Helvetia",Font.BOLD, 14);
setLayout(new FlowLayout());
setSize(110,240);
setLocation(300,300);

c1 = Color.black;
c2 = Color.yellow;
c3 = Color.black;

statusText = "Initialiere ...";

panel = new Panel();
light = panel.getGraphics();
add (panel);

setVisible(true);
    }

    public void paint(Graphics g){
g.setColor(c1);
g.fillOval(30,40,50,50);
g.setColor(c2);
g.fillOval(30,90,50,50);
g.setColor(c3);
g.fillOval(30,140,50,50);
g.setColor(Color.black);
g.drawString(statusText, 20, 210);
    }

    /**
     * Setzten der Farbe
     * @param String red, yellow, green
     */
    public void initColor(String farbe){

if (farbe == "green"){
    c1 = Color.black;
    c2 = Color.black;
    c3 = Color.green;
}
else if (farbe == "red"){
    c1 = Color.red;
    c2 = Color.black;
    c3 = Color.black;
}
else if (farbe == "yellow"){
    c1 = Color.black;
    c2 = Color.yellow;
    c3 = Color.black;
}
else{
    c1 = Color.black;
    c2 = Color.black;
    c3 = Color.black;
}
repaint();
    }

    /**
     * setzt den StatusText der Ampel
     * @param String text
     */
    public void initText(String text){
statusText = text;
repaint();
    }
}
```

A.5 Die Klasse AgGUIStatus

```
import java.awt.*;
import java.awt.event.*;
import java.util.Date;
/**
 * Fenster zur Darstellung von Informationen ueber
 * den Status z.B. eines Initialisierungsvorganges
 * @version 10.3.99
 * @author Michael Hojnacki
 */
public class AgGUIStatus extends Frame implements ActionListener{

    Label label;
    TextArea textArea;
    Font font;
    Button clearButton;
    Button closeButton;
    Button endButton;

    public AgGUIStatus(){
super("Status Agent");
font = new Font("Helvetia",Font.BOLD, 14);
label = new Label("Agent Console", Label.CENTER);
textArea = new TextArea();
clearButton = new Button(" Clear ");
closeButton = new Button(" Close ");
endButton = new Button("Stop Agent");


endButton.setFont(font);
clearButton.setFont(font);
closeButton.setFont(font);

label.setFont(font);
textArea.setFont(font);
textArea.setEditable(false);
setLayout(new FlowLayout());
setSize(550,300);
setLocation(300,300);

clearButton.addActionListener(this);
closeButton.addActionListener(this);
endButton.addActionListener(this);

add(label);
add(textArea);
add(clearButton);
add(closeButton);
add(endButton);

setVisible(true);

    }

    public void performInitGUIStatus(String text){
textArea.append((new Date()).toString()+" : ");
textArea.append(text);
textArea.append("\n");
    }

    public void performClearTextArea(){
textArea.setText("");
    }

    public void actionPerformed(ActionEvent e){
if (e.getSource() instanceof Button){
    if (e.getSource() == clearButton)
this.performClearTextArea();
    if (e.getSource() == closeButton)
this.setVisible(false);
```

```
    if (e.getSource() == endButton)
java.lang.System.exit(0);
}

    }

    public Insets getInsets(){
return new Insets(30,30,30,30);
    }

}
```

A.6 Die Klasse AgGlobalVariables

```
import java.lang.String;
import java.util.Vector;
import java.util.Date;
/**
 * Lokale Parameter des Agenten
 **/
public class AgGlobalVariables{

    String routerIpName;      // RouterIPname des lokalen Routers
    String routerIpAdress;    // RouterIPAdresse des lokalen Routers
    int    routerSnmpPort;    // SNMP Part des Routers (default 161)
    String agentHostIpName;   // IPName des AgentHosts
    String agentHostIpAdress; // IPAdresse des AgentHosts
    String managerHostName;   // Host des Managers (fuer Datenuebertragung)


    boolean guiVisible;       // Soll das GUI angezeigt werden

    Vector hostsToBeChecked;  // Informationen fuer die ConnectivityPruefung
    Long   checkFrequence;    // Messintervalle in Millisekunden
    String logInfos;          // LogInformationen koennen hier leicht ausgelesen werden



    public AgGlobalVariables(){
routerIpName      = "cisco1";
routerIpAdress    = "10.61.100.1";
routerSnmpPort    = 161;
agentHostIpName   = "iller";
agentHostIpAdress = "10.61.100.11";
managerHostName   = "isar";
// managerHostName   = "iller";
guiVisible        = true;
hostsToBeChecked  = new Vector();
checkFrequence    = new Long(30000);
logInfos          = new String();

// ein paar defaultAdressen
// /etc/hosts Auslesen ??
hostsToBeChecked.addElement("isar");
hostsToBeChecked.addElement("iller");
hostsToBeChecked.addElement("drucker");
hostsToBeChecked.addElement("cisco1");
hostsToBeChecked.addElement("cisco2");
hostsToBeChecked.addElement("bla");
hostsToBeChecked.addElement("isar");


    }


    public void setrouterIpName(String value){
routerIpName = value;
    }

    public void setrouterIpAdress(String value){
```

```
routerIpAdress = value;
    }

    public void setrouterSnmpPort(int value){
routerSnmpPort = value;
    }

    public void setagentHostIpName(String value){
agentHostIpName = value;
    }

    public void setagentHostIpAdress(String value){
agentHostIpAdress = value;
    }
    public void setmanagerHostName(String value){
managerHostName = value;
    }

    public void setguiVisible(boolean value){
guiVisible = value;
if (guiVisible){
    AgBasis.guiStatus.setVisible(true);
    AgBasis.isdnStatus.ampel.setVisible(true);
}
else{
    AgBasis.guiStatus.setVisible(false);
    AgBasis.isdnStatus.ampel.setVisible(false);
}

    }

    public void sethostsToBeChecked (Vector value){
hostsToBeChecked = value;
    }

    public void setcheckFrequence(Long value){
checkFrequence = value;
    }

    public void setlogInfos(String value){
logInfos += ((new Date()).toString()+" : ");
logInfos += (value + "\n");
AgBasis.ioHandler. writeLogfile(logInfos);
    }


    // ==============================================


    public String getrouterIpName(){
return routerIpName;
    }

    public String getrouterIpAdress(){
return routerIpAdress;
    }

    public int getrouterSnmpPort(){
return routerSnmpPort;
    }

    public String getagentHostIpName(){
return agentHostIpName;
    }

    public String getagentHostIpAdress(){
return agentHostIpAdress;
    }

    public String getmanagerHostName(){
return managerHostName;
```

```
    }

    public boolean  getguiVisible(){
return guiVisible;
    }

    public Vector gethostsToBeChecked (){
return hostsToBeChecked;
    }

    public Long getcheckFrequence(){
return checkFrequence;
    }

    public String getlogInfos(){
return logInfos;
    }


}
```

A.7 Die Klasse AgISDNStatus

```
import com.sun.jaw.impl.agent.services.alarm.*;
import com.sun.jaw.snmp.common.*;
import com.sun.jaw.snmp.manager.*;
import com.sun.jaw.reference.common.*;

import java.beans.*;

/**
 * Klasse zur Pruefung und Verwaltung des ISDN-Status.
 *
 * @version 22.3.99
 * @author Michael Hojnacki
 *
 */
public class AgISDN_Status implements AlarmClockListener{

    private    int                  ISDNStatus;
    private    AlarmClock           alarmClock;
    public     AgGUIIsdnStatus      ampel;
    protected PropertyChangeSupport propertyChangeSupport;

    /**
     * Empty Kontruktor
     */
    public AgISDN_Status(){
ISDNStatus = 0;

ampel = new AgGUIIsdnStatus();

alarmClock = new AlarmClock();
alarmClock.setTimeoutAsLong(new Long(10));
alarmClock.performStart();
alarmClock.addAlarmClockListener(this);

// Aus JavaBeans, soll andere Klassen von der Aenderung des
// ISDN Status benachrichtigen.
propertyChangeSupport = new PropertyChangeSupport(this);
    }

    public void addPropertyChangeListener(PropertyChangeListener listener){
propertyChangeSupport.addPropertyChangeListener(listener);
    }

    public void removePropertyChangeListener(PropertyChangeListener listener){
propertyChangeSupport.removePropertyChangeListener(listener);
    }
```

```
    /**
     * GetterMethoden
     */
    public int getISDNStatus(){
return ISDNStatus;
    }

    /**
     * SetterMethoden
     */
    private void setISDNStatus (int nISDNStatus){
int oldISDNStatus = ISDNStatus;
this.ISDNStatus = nISDNStatus;

propertyChangeSupport.firePropertyChange("ISDNStatus", new Integer(oldISDNStatus),
 new Integer(ISDNStatus));
    }

    public void handleAlarmClock (AlarmClockEvent e){

int status = performCheckISDNStatus();

if (status == 1){
    ampel.initColor("green");
    setISDNStatus(1);
}
else if (status == 2){
    ampel.initColor("red");
    setISDNStatus(2);
}
else if (status == 3){
    ampel.initColor("yellow");
    setISDNStatus(3);
}
else {
    ampel.initColor("yellow");
    setISDNStatus(-1);
}

alarmClock.setTimeoutAsLong(new Long(10000));
        alarmClock.performStart();
    }

    /**
     * Methode zur Abfrage des IfStatus des Routers !
     * u.U. um Abfrage der Interfacenummer des Routers ergaenzen !!
     * @return int 1 fuer up, 2 fuer down, 3 fuer test mode
     */
    public int performCheckISDNStatus(){

ampel.initText("Request ...");

int status = -1;

String host = AgBasis.globalVariables.getrouterIpName();
int    port = AgBasis.globalVariables.getrouterSnmpPort();

try {

    SnmpMain.initializeSNMP(new RFC1213_MIBStore());
    SnmpPeer agent= new SnmpPeer(host, port);

    SnmpParameters params= new SnmpParameters("public", "private");

    agent.setSnmpParam(params);

    SnmpSession session= new SnmpSession("performCheckISDNStatusSession");

    session.setDefaultPeer(agent);
```

```
    SnmpVarbindList list = new SnmpVarbindList("performCheckISDNStatusVarbindList");

    list.addVariable("ifOperStatus.1");
    list.addVariable("ifOperStatus.2");
    list.addVariable("ifOperStatus.3");

    SnmpRequest request = session.snmpGet(null, list);

    boolean completed = request.waitForCompletion(10000);
    if (completed == false) {
// AgBasis.toConsole("AgISDNStatus. performCheckISDNStatus(): Request timed out." +
//   "Check reachability of agent");
ampel.initText("No Agent ...");
return status;
    }

    int errorStatus= request.getErrorStatus();
    if (errorStatus != SnmpDefinitions.snmpRspNoError) {
AgBasis.toConsole(SnmpRequest.snmpErrorToString(errorStatus));
ampel.initText("SNMP-Error...");
return status;
    }

    SnmpVarbindList result= request.getResponseVbList();
    SnmpVar ifOperBRI  = result.getSnmpVarAt(0);
    SnmpVar ifOperB0   = result.getSnmpVarAt(1);
    SnmpVar ifOperB1   = result.getSnmpVarAt(2);

    int ifOperBRIint  = Integer.parseInt(ifOperBRI.getStringValue());
    int ifOperB0int  = Integer.parseInt(ifOperB0.getStringValue());
    int ifOperB1int  = Integer.parseInt(ifOperB1.getStringValue());

    //AgBasis.toConsole("BRI Status: " + ifOperBRIint);
    //AgBasis.toConsole("B0  Status: " + ifOperB0int);
    //AgBasis.toConsole("B1  Status: " + ifOperB1int);

    //ISDN Verbindung besteht, wenn mind. ein ISDN Kanal benutzt wird !
    if ((ifOperB0int==1)||(ifOperB1int==1))
status = 1;
    //ISDN Verbindung besteht nicht, wenn beide Kanaele getestet werden !
    else if ((ifOperB0int==3)&&(ifOperB1int==3))
status = 3;
    //ISDN Verbindung besteht nicht, wenn beide Kanaele getestet werden !
    else
status = 2;
    ampel.initText("Sleeping ...");
    return status;
}
catch (Exception e){
    ampel.initText("Exception ...");
    return -1;
}

    }


}
```

A.8 Die Klasse AgMeasurement

```
import java.lang.String;
import java.util.Vector;
import java.net.*;
import com.sun.jaw.impl.agent.services.alarm.*;


/**
 * Abstrakte Klasse !!
 * Definiert nur Grundlage der fuer die Messungen notwendigen Methoden und Variablen
 *
```

```
 * @version 30.3.99
 * @author Michael Hojnacki
 *
 **/
public abstract class AgMeasurement implements AlarmClockListener{

    private   String method;
    private   String ownIPAdress;
    protected AlarmClock alarmClock;

    // Empty Konstruktor
    public AgMeasurement(){

method      = new String("empty");

// AlarmClockService starten
alarmClock = new AlarmClock();
alarmClock.setTimeoutAsLong(new Long(10));

ownIPAdress = AgBasis.globalVariables.getagentHostIpName();

alarmClock.addAlarmClockListener(this);
    }


    /**
     * Diese Methode muss in der jeweiligen Klasse implemeniert werden
     */
    public abstract void performMeasure();




    /**
     * AlarmClockEvent ist Ausloeser fuer Messungen
     */
    public void handleAlarmClock (AlarmClockEvent e){
        AgBasis.toConsole("AgMeasurement: AlarmclockEvent erhalten");
performMeasure();
    }

    public void setmethod(String m){
method = m;
    }

    public void setownIPAdress(String name){
ownIPAdress = name;
    }

    public String getownIPAdress(){
return ownIPAdress;
    }

    public String getmethod(){
return method;
    }

}
```

A.9 Die Klasse DataObject

```
import java.util.Date;
import java.lang.String;
import java.util.Vector;
import java.io.*;

/**
 * Ist zwar komplett, soll aber fuer die einzelnen Messmethoden abgeleitet und angepasst werden !
 * @version 10.3.99
```

```
 * @author Michael Hojnacki
 */
public class DataObject implements Serializable{

    private Date    measurementDate;
    private boolean measurementResultOK;
    private Vector  ipAdresses;
    private Vector  results;
    private boolean transferred;
    private String  method;
    private String  ownIPAdress;
    private String  hostName;

    /**
     * Leerer Kontructor
     */
    public DataObject(){

ipAdresses          = new Vector();           // zu messende IP-Adressen
results             = new Vector();           // Ergebnisse
measurementDate     = new Date();             // Tag der Messung
measurementResultOK = false;                  // Wurde die Messung korrekt durchgefuehrt
transferred         = false;                  // Ergebnisse an Manager uebertragen
method              = new String("empty");    // Art der Messung
ownIPAdress         = new String("empty");    // eigene IPAdresse
hostName            = new String("empty");    // eigener Hostname
    }

    /**
     * Kompletter Konstruktor
     */
    public DataObject(Date nDate, boolean nOK, String nIPAdress, boolean nTransferred,
      String nMethod, Vector nipAdresses, String host){
measurementDate     = nDate;                // Tag der Messung
measurementResultOK = nOK;                  // Wurde die Messung korrekt durchgefuehrt
ownIPAdress         = nIPAdress;            // IPAdresse des Agenten
transferred         = nTransferred;         // Ergebnisse an Manager uebertragen
method              = nMethod;              // Art der Messung
ipAdresses          = nipAdresses;
results             = new Vector();         // Ergebnisse
hostName            = host;
    }

    // GetterMethoden
    public Date getmeasurementDate(){
return measurementDate;
    }

    public boolean getmeasurementResultOK(){
return measurementResultOK;
    }

    public String getownIPAdress(){
return ownIPAdress;
    }

    public boolean gettransferred(){
return transferred;
    }
    public String getmethod(){
return method;
    }

    public Vector getipAdresses(){
return ipAdresses;
    }

    public Vector getresults(){
return results;
    }
```

```
    public String gethostName(){
return hostName;
    }

    // SetterMethoden
    public void  setmeasurementDate(Date newDate){
measurementDate = newDate;
    }

    public void setmeasurementResultOK(boolean newMeasurementResultOK){
measurementResultOK = newMeasurementResultOK;
    }

    public void setownIPAdress(String newIpAdress){
ownIPAdress = newIpAdress;
    }

    public void settransferred(boolean newTransferred){
transferred = newTransferred;
    }
    public void setmethod(String newMethod){
method = newMethod;
    }

    public void setipAdresses(Vector nipAdresses){
ipAdresses = nipAdresses;
    }

    public void setresults(Vector r){
results = r;
    }

    public void sethostName(String name){
hostName = name;
    }
    // einfache ASCII - Ausgabe
    public String performPrint2ASCII(){
return new String(
    measurementDate      +"/t"+
    measurementResultOK  +"/t"+
    ownIPAdress             +"/t"+
    transferred          +"/t"+
    method               +"/n");
    }
}
```

A.10 Die Klasse IoHandler

```
import java.io.*;
import java.util.Vector;
import java.util.Date;

/**
 * Diese Klasse ermöglicht des Speichern und Wiederherstellen
 * von LogFiles und DataObjects
 */
public class IoHandler {

    public IoHandler(){
    }

    /**
     * Schreibt LogString in ein File
     * @param String
     * @return (int) 0 fuer OK, 1 fuer Fehler
     */
    public int writeLogfile(String value){
try{
    Date now = new Date();
```

```
    String logText
= new String("  ===================================================\n"
     + "<< LogFile erstellt am : " + now.toString()
     + " >>\n"
     + "  ===================================================\n\n"
     + value);
    File outputFile = new File("Agent.log");
    FileOutputStream out = new FileOutputStream(outputFile);
    byte[] c = logText.getBytes();
    out.write(c);
    out.close();
    return 0;
}
catch(Exception e){
    System.out.println("Schreibversuch des LogFiles gescheitert");
    return 1;
}
    }

    /**
     * Schreibt ein Objekt in ein File
     * @param (Object)
     * @return (int) 0 fuer OK, 1 fuer Fehler
     */
    public int writeDataObject(Object value){

try {
    FileOutputStream   s   = new FileOutputStream("Agent.dat");
    ObjectOutputStream oos = new ObjectOutputStream(s);
    oos.writeObject(value);
    oos.flush();
    s.close();
    return 1;
}
catch(Exception e){
    System.out.println("Datensicherung gescheitert");
    return 0;
}
    }

     /**
     * Liest ein Objekte aus einem File.
     * Im Falle einer Exception ist das RueckgabeObject == null.
     * @return (Vector) leerer Vector, wenn kein Agent.dat existiert
     */
    public Vector readDataObject(){
Vector data;
try {
    FileInputStream   s   = new FileInputStream("Agent.dat");
    ObjectInputStream ois = new ObjectInputStream(s);
    data = (Vector) ois.readObject();
    s.close();
    return data;
}
catch(Exception e){
    System.out.println("Datenlesen gescheitert");
    data = new Vector();
    return data;
}
    }
}
```

Anhang B

Sourcecode des Managers

B.1 Die Klasse MgBasis

```
import java.util.Date;

/**
 * Diese Klasse startet den BasisManager.
 * @author Michael Hojnacki
 * @version 20.3.99
 */
public class MgBasis{

    static MgGUIZentrale    guiStart;
    static MgDataHandler    dataHandler;
    static IoHandler        ioHandler;

    static MgCMF mgCMF;

    public MgBasis(){

guiStart  = new MgGUIZentrale();
ioHandler = new IoHandler();
guiStart.setlogInfoString((new Date()).toString()+" : Zentrale gestartet ...");
guiStart.setlogInfoString((new Date()).toString()+" : IoHandler gestartet ...");
dataHandler = new MgDataHandler();
guiStart.setlogInfoString((new Date()).toString()+" : DataHandler gestartet ...");
mgCMF       = new MgCMF();
mgCMF.initCMF();
guiStart.setlogInfoString((new Date()).toString()+" : Framework gestartet ...");
guiStart.setlogInfoString((new Date()).toString()+" : Manager bereit ...");
    }

    public static void logInfo(String value){
String logInfos = ((new Date()).toString()+" : " + value);
guiStart.setlogInfoString(logInfos);
    }

    public static void main(String argv[]){
MgBasis basis = new MgBasis();
    }

}
```

B.2 Die Klasse DataObject

```
import java.util.Date;
import java.lang.String;
import java.util.Vector;
import java.io.*;

/**
 * Ist zwar komplett, soll aber fuer die einzelnen Messmethoden abgeleitet und angepasst werden !
 * @version 10.3.99
 * @author Michael Hojnacki
 */
public class DataObject implements Serializable{

    private Date    measurementDate;
    private boolean measurementResultOK;
    private Vector  ipAdresses;
    private Vector  results;
    private boolean transferred;
    private String  method;
    private String  ownIPAdress;
    private String  hostName;

    /**
     * Leerer Kontructor
     */
    public DataObject(){

ipAdresses          = new Vector();           // zu messende IP-Adressen
results             = new Vector();           // Ergebnisse
measurementDate     = new Date();             // Tag der Messung
measurementResultOK = false;                  // Wurde die Messung korrekt durchgefuehrt
transferred         = false;                  // Ergebnisse an Manager uebertragen
method              = new String("empty");    // Art der Messung
ownIPAdress         = new String("empty");    // eigene IPAdresse
hostName            = new String("empty");    // eigener Hostname
    }

    /**
     * Kompletter Konstruktor
     */
    public DataObject(Date nDate, boolean nOK, String nIPAdress, boolean nTransferred,
      String nMethod, Vector nipAdresses, String host){
measurementDate     = nDate;                // Tag der Messung
measurementResultOK = nOK;                  // Wurde die Messung korrekt durchgefuehrt
ownIPAdress         = nIPAdress;            // IPAdresse des Agenten
transferred         = nTransferred;         // Ergebnisse an Manager uebertragen
method              = nMethod;              // Art der Messung
ipAdresses          = nipAdresses;
results             = new Vector();         // Ergebnisse
hostName            = host;
    }

    // GetterMethoden
    public Date getmeasurementDate(){
return measurementDate;
    }

    public boolean getmeasurementResultOK(){
return measurementResultOK;
    }

    public String getownIPAdress(){
return ownIPAdress;
    }

    public boolean gettransferred(){
return transferred;
    }
    public String getmethod(){
return method;
```

```
    }

    public Vector getipAdresses(){
return ipAdresses;
    }

    public Vector getresults(){
return results;
    }

    public String gethostName(){
return hostName;
    }

    // SetterMethoden
    public void  setmeasurementDate(Date newDate){
measurementDate = newDate;
    }

    public void setmeasurementResultOK(boolean newMeasurementResultOK){
measurementResultOK = newMeasurementResultOK;
    }

    public void setownIPAdress(String newIpAdress){
ownIPAdress = newIpAdress;
    }

    public void settransferred(boolean newTransferred){
transferred = newTransferred;
    }
    public void setmethod(String newMethod){
method = newMethod;
    }

    public void setipAdresses(Vector nipAdresses){
ipAdresses = nipAdresses;
    }

    public void setresults(Vector r){
results = r;
    }

    public void sethostName(String name){
hostName = name;
    }
    // einfache ASCII - Ausgabe
    public String performPrint2ASCII(){
return new String(
    measurementDate      +"/t"+
    measurementResultOK  +"/t"+
    ownIPAdress             +"/t"+
    transferred          +"/t"+
    method               +"/n");
    }
}
```

B.3 Die Klasse MgCMF

```
import java.lang.String;

import com.sun.jaw.reference.common.*;
import com.sun.jaw.reference.agent.cmf.*;
import com.sun.jaw.reference.agent.services.*;
import com.sun.jaw.impl.agent.services.light.*;

/**
 * Diese Klasse stellt das CMF fuer den Datentransfer bereit.
 * Start der benoetigten Dienste, Instanziierung des M-Beans,
 * Registrieren im Framework.
 * @version 20.3.99
 * @author Michael Hojnacki
 */
public class MgCMF{

    private MgDataReceiver dataReceiver;
    private Framework cmf;
    private ObjectName d;

    /**
     * Konstruktor
     */
    public MgCMF(){
// Basisdienste des Agenten starten
        try {

            String domain = "defaultDomain";

            // Framework starten
            cmf = new Framework();

            // MetaDataService starten
            String mtdSrvClass = "com.sun.jaw.impl.agent.services.light.MetaDataSrv" ;
            String mtdSrvName = domain + ":" + ServiceName.META ;
            cmf.newObject(mtdSrvClass, mtdSrvName, null) ;

            // RMI Adapter starten
            String rmiSrvClass = "com.sun.jaw.impl.adaptor.rmi.AdaptorServerImpl" ;
            String rmiSrvName = domain + ":" + ServiceName.ADAPTOR + ".protocol=rmi" ;
            cmf.newObject(rmiSrvClass, rmiSrvName, null) ;

      // HTTP Adapter starten
            String httpSrvClass = "com.sun.jaw.impl.adaptor.http.AdaptorServerImpl" ;
            String httpSrvName = domain + ":" + ServiceName.ADAPTOR + ".protocol=http" ;
            cmf.newObject(httpSrvClass, httpSrvName, null) ;

            // HTML Adapter starten
            String htmlSrvClass = "com.sun.jaw.impl.adaptor.html.AdaptorServerImpl" ;
            String htmlSrvName = domain + ":" + ServiceName.ADAPTOR + ".protocol=html" ;
            cmf.newObject(htmlSrvClass, htmlSrvName, null) ;


    //debug
            MgBasis.logInfo("Framework und Adapter des Managers bereit !") ;
}

        catch (Exception e){
            MgBasis.logInfo("BasisManager.BasisManager(): Exception !!");
            e.printStackTrace();
            System.exit(1);
        }
    }


   /**
     * Objekte erzeugen und im Framework anmelden
     */
    public void initCMF(){
```

```
        // Verwaltung der Daten
        dataReceiver   = new MgDataReceiver();
        MgBasis.logInfo("BasisManager.initBasisManager(): new MgDataReceiver()");

        // M-Beans registrieren !!
        try {
            String domain = cmf.getDomain();

            d = new ObjectName(domain + ":" + "MgDataReceiver.SerialNo=1");
            cmf.addObject(dataReceiver, d);
        }

        catch (InstanceAlreadyExistException e){
            MgBasis.logInfo("BasisManager.initBasisManager(): InstanceAlreadyExistException !!");
            e.printStackTrace();
            System.exit(1);
        }
    }

}
```

B.4 Die Klasse IoHandler

```
import java.io.*;
import java.util.Vector;
import java.util.Date;

/**
 * Diese Klasse ermöglicht des Speichern und Wiederherstellen
 * von LogFiles und DataObjects
 */
public class IoHandler {

    public IoHandler(){
    }


    /**
     * Schreibt LogString in ein File
     * @param String
     * @return (int) 0 fuer OK, 1 fuer Fehler
     */
    public int writeLogfile(String value){
try{
    Date now = new Date();
    String logText
= new String("  ==================================================\n"
     + "<< LogFile erstellt am : " + now.toString()
     + " >>\n"
     + "  ==================================================\n\n"
     + value);
    File outputFile = new File("Manager.log");
    FileOutputStream out = new FileOutputStream(outputFile);
    byte[] c = logText.getBytes();
    out.write(c);
    out.close();
    return 0;
}
catch(Exception e){
    System.out.println("Schreibversuch des LogFiles gescheitert");
    return 1;
}
    }

    /**
     * Schreibt ein Objekt in ein File
     * @param (Object)
     * @return (int) 0 fuer OK, 1 fuer Fehler
     */
    public int writeDataObject(Object value){

try {
    FileOutputStream   s   = new FileOutputStream("Manager.dat");
    ObjectOutputStream oos = new ObjectOutputStream(s);
    oos.writeObject(value);
    oos.flush();
    s.close();
    return 1;
}
catch(Exception e){
    System.out.println("Datensicherung gescheitert");
    return 0;
}
    }

     /**
     * Liest ein Objekte aus einem File.
     * Im Falle einer Exception ist das RueckgabeObject == null.
     * @return (Object) casten nicht vergessen!
     */
    public Vector readDataObject(){
Vector data;
```

```
try {
    FileInputStream   s   = new FileInputStream("Manager.dat");
    ObjectInputStream ois = new ObjectInputStream(s);
    data = (Vector) ois.readObject();
    s.close();
    return data;
}
catch(Exception e){
    System.out.println("Datenlesen gescheitert");
    data = new Vector();
    return data;
}
    }
}
```

B.5 Die Klasse MgConnectionHandler

```
import java.net.*;
import java.util.*;
import com.sun.jaw.reference.common.*;
import com.sun.jaw.impl.adaptor.rmi.*;

/**
 * Wird benoetigt, um Verbindung zu einem Agenten aufzubauen.
 * Muss dann fuer jeden Agenten MIT Parameter instanziiert werden !!
 *
 * @version 2.3.99
 * @author Michael Hojnacki
 **/


public class MgConnectionHandler {

    private boolean connectionEstablished;
    private AdaptorClient rmiAdaptor;

    public MgConnectionHandler(){
connectionEstablished = false;
System.out.println("ConnectionHandler.ConnectionHandler(): Objekt erstellt");
    }

    /**
     * @param keiner, es wird der gleiche Host genutzt
     */
    public void performConnectToAgentHost(){
try{
    //String agentHost = InetAddress.getLocalHost().getHostName();
    String agentHost = "iller";
    System.out.println(" >> Connecting to " + agentHost );
    // RMI - Adapter einrichten
    rmiAdaptor = new AdaptorClient();
    rmiAdaptor.connect(null, agentHost, 1099, ServiceName.APT_RMI);
    System.out.println("BasisManager.BasisManager(): Verbindung steht");
    setconnectionEstablished(true);
}
catch(Exception e) {
    System.out.println("BasisManager.BasisManager(): Exception, Verbindungsaufbau nicht moeglich!");
    setconnectionEstablished(false);
    //e.printStackTrace();
    //System.exit(1);
}
    }


    /**
     * @param  Bezeichnung des Hosts, auf dem der Agent laeuft
     */
    public void performConnectToAgentHost(String agentHost){
try{
    System.out.println(" >> Connecting to " + agentHost );
    // RMI - Adapter einrichten
    rmiAdaptor = new AdaptorClient();
    rmiAdaptor.connect(null, agentHost, 1099, ServiceName.APT_RMI);
    System.out.println("BasisManager.BasisManager(): Verbindung steht");
    setconnectionEstablished(true);
}
catch(Exception e) {
    System.out.println("BasisManager.BasisManager(): Exception, Verbindungsaufbau nicht moeglich!");
    setconnectionEstablished(false);
}
    }

    /**
     * Trennt Verbindung zum Agenten
     */
    public void performDisconnectFromAgentHost(){
```

```
rmiAdaptor.disconnect();
setconnectionEstablished(false);
    }


    public void setconnectionEstablished(boolean newValue){
connectionEstablished = newValue;
    }

    public boolean getconnectionEstablished(){
return connectionEstablished;
    }


    public static void main(String argv[]){
    MgConnectionHandler ch = new MgConnectionHandler();
    ch.performConnectToAgentHost("isar");
    }
}
```

B.6 Die Klasse MgDataHandler

```
import java.util.Vector;
import java.util.Date;

/**
 * MgDataHandler verwaltet die Hosts, auf denen JDMK Agenten laufen.
 * Sie basiert auf der Festlegung, dass auf jedem Host nur ein Agent laeuft.
 * @version 18.3.99
 * @author Michael Hojnacki
 */

public class MgDataHandler{

    // Vector aus Vectoren,
    // Erstes Object ist ein String mit dem HostsNamen,
    // restlichen Objekte beinhalten Daten der einzelnen Agenten.
    Vector dataObjects;

    /**
     * Konstruktor
     */
    public MgDataHandler(){
dataObjects = new Vector();
    }

    /**
     * Fuegt neuen AgentHost ein
     * @param String: name des Hosts
     * @return int: Index des einzufuegenden Hosts
     */
    public int performAddNewAgentHost(String name){

//sicherstellen, dass name nicht schon vorhanden
int index = performFindAgentHost(name);

if (index == -1){
    Vector newAgent = new Vector();
    newAgent.addElement(name);
    dataObjects.addElement(newAgent);
    index = performFindAgentHost(name);
}
return index;
    }


    /**
     * Findet den Index des AgentHosts in agentHosts
     * @param String: Name des Hosts
     * @return int: Index des Hosts in dataObjects oder -1, falls nicht vorhanden
     */
    public int performFindAgentHost(String name){
int searchedIndex = -1;
int noe = dataObjects.size();
if (noe > 0)
    for (int i=0; i<noe; i++){
Vector tempVector = (Vector)(dataObjects.elementAt(i));
String tempName   = (String)(tempVector.elementAt(0));
if (name.equalsIgnoreCase(tempName))
    searchedIndex = i;
    }
return searchedIndex;
    }

    /**
     * loescht ohne Tests oder Rueckfragen die spez. Agentendaten
     * @param hostName
     */
    public void performRmvAgentHost(String name){
//Index suchen
int index = performFindAgentHost(name);
```

```
dataObjects.removeElementAt(index);
    }

    /**
     * fuegt DatenObject zu dem jeweiligen AgentenVector,
     * ist dieser noch nicht vorhanden, wird ein neuer erzeugt.
     * @param DataObject
     */
    public void performAddDataObject(DataObject dataObject){

MgBasis.logInfo("performAddDataObject: Beginn ...");
String host = dataObject.gethostName();
int index = performFindAgentHost(host);
// host noch nicht vorhanden
if (index < 0){
    MgBasis.logInfo("performAddDataObject:  Neuer Host ...");
    performAddNewAgentHost(host);
    index = performFindAgentHost(host);
    MgBasis.logInfo("performAddDataObject:  Neuer Host gesetzt");
}
// Daten anfuegen
Date d               = dataObject.getmeasurementDate();
Vector results       = dataObject.getresults();
Vector ipAdresses    = dataObject.getipAdresses();
String ipAdress      = new String();
String method        = dataObject.getmethod();
int numberOfMeasures = results.size();

for (int i = 0; i<numberOfMeasures; i++){
    MgBasis.logInfo("performAddDataObject: Measure " + i + "angefuegt");
    String ipName = (String)(ipAdresses.elementAt(i));
    Integer result = (Integer)(results.elementAt(i));
    MgMeasure m = new MgMeasure(d, ipAdress, ipName, result, method);
    ((Vector)(dataObjects.elementAt(index))).addElement(m);
}
MgBasis.logInfo("performAddDataObject: Ende");
    }

    /**
     * Gibt einen Vector mit allen registrierten AgentHostNamen zurueck
     */
    public Vector getAgentHostsVector(){
Vector result = new Vector();
int noe = dataObjects.size();
if (noe > 0)
    for (int i=0; i<noe; i++){
Vector tempVector = (Vector)(dataObjects.elementAt(i));
String tempName   = (String)(tempVector.elementAt(0));
result.addElement(tempName);
    }
return result;
    }


    /**
     * Gibt die MeasureObjekte des Hosts in einem Vector zurueck
     */
    public Vector getAgentHostMeasures(String name){
Vector result = new Vector();
int index =  performFindAgentHost(name);
if (index >= 0){
    result = (Vector)(dataObjects.elementAt(index));
    // result.removeElementAt(0);  fatal !!!! pass by reference !!!
}
return result;
    }
}
```

B.7 Die Klasse MgDataReceiver

```
import java.util.Date;
import java.lang.String;

/**
 * Klasse zum Empfang der Daten auf ManagerSeite
 *
 * @version 10.3.99
 * @author Michael Hojnacki
 *
 */
public class MgDataReceiver {

    String testParam;

    // Empty Konstruktor
    public MgDataReceiver(){
MgBasis.logInfo("DataReceiver gestartet ...");
    }

    /**
     * Methode zum Empfang der Daten
     */
    public boolean performReceiveData(DataObject dataObject){
MgBasis.logInfo("Datenempfang ...");
MgBasis.dataHandler.performAddDataObject(dataObject);
MgBasis.logInfo("Daten wurden an den Datahandler uebergeben ...");
return true;
    }
}
```

B.8 Die Klasse MgGUIAgParam

```
import java.awt.*;
import java.awt.event.*;
import java.util.Vector;
import java.util.Date;
import com.sun.jaw.reference.common.*;
import com.sun.jaw.impl.adaptor.rmi.*;


public class MgGUIAgParam extends Frame implements ActionListener{


    private Button connectButton;
    private Button hostsToBeCheckedButton;
    private Button logInfosButton;
    private Button exitToPingListButton;
    private Button applyButton;
    private Button refreshButton;
    private Button exitButton;
    private Button exitLogInfoButton;
    private Frame logInfoFrame;
    private Frame toPingListFrame;

    private List checkedList;
    private TextField connectField;
    private TextField valueFields[];
    private Choice guiChoice;
    private TextArea logInfoTextArea;
    private String agLogInfos;
    private Vector agHostsToBeChecked;


    public MgGUIAgParam(){

super("Parameter des Agenten");

agLogInfos = new String(" ... noch keine Agenteninformationen geladen !");

GridBagLayout gridBag = new GridBagLayout();
GridBagConstraints constraints = new GridBagConstraints();
setLayout(gridBag);

Font font = new Font("Monospaced", Font.BOLD, 14);

// TitelPanel
Panel titlePanel = new Panel();
Font fontTitle = new Font("Monospaced", Font.BOLD, 30);
Label titleLabel = new Label("Agentenkonfiguration", Label.CENTER);
titleLabel.setBackground(Color.yellow);
titleLabel.setFont(fontTitle);
titlePanel.add(titleLabel);

buildConstraints(constraints,0,0,2,1,1,1);
gridBag.setConstraints(titlePanel, constraints);
this.add(titlePanel);


connectField = new TextField("iller",20);
connectField.setFont(font);
buildConstraints(constraints,0,1,1,1,1,1);
gridBag.setConstraints(connectField, constraints);
this.add(connectField);

connectButton = new Button("   Verbinden   ");
connectButton.setFont(font);
connectButton.addActionListener(this);
buildConstraints(constraints,1,1,1,1,1,1);
gridBag.setConstraints(connectButton, constraints);
this.add(connectButton);
```

```
String labelTexts[] = {"routerIpName","routerIpAdress","routerSnmpPort",
       "agentHostIpName","agentHostIpAdress","managerHostName",
       "checkFrequence", "guiVisible","hostsToBeChecked","logInfos"};
for (int i = 0; i<10; i++){
    Label l = new Label(labelTexts[i], Label.RIGHT);
    l.setFont(font);
    buildConstraints(constraints,0,i+2,1,1,1,1);
    constraints.anchor = GridBagConstraints.EAST;
    gridBag.setConstraints(l, constraints);
    this.add(l);
}

valueFields = new TextField[7];
for (int i = 0; i<7; i++){
    valueFields[i] = new TextField(20);
    valueFields[i].setFont(font);
    buildConstraints(constraints,1,i+2,1,1,1,1);
    gridBag.setConstraints(valueFields[i], constraints);
    this.add(valueFields[i]);
}
valueFields[3].setEditable(false);
valueFields[4].setEditable(false);

constraints.anchor = GridBagConstraints.CENTER;

guiChoice = new Choice();
guiChoice.add("true");
guiChoice.add("false");
guiChoice.setFont(font);
buildConstraints(constraints,1,9,1,1,1,1);
gridBag.setConstraints(guiChoice, constraints);
this.add(guiChoice);

hostsToBeCheckedButton = new Button(" Anzeigen/ Aendern ");
hostsToBeCheckedButton.setFont(font);
hostsToBeCheckedButton.addActionListener(this);
buildConstraints(constraints,1,10,1,1,1,1);
gridBag.setConstraints(hostsToBeCheckedButton, constraints);
this.add(hostsToBeCheckedButton);

logInfosButton  = new Button("    Anzeigen    ");
logInfosButton.setFont(font);
logInfosButton.addActionListener(this);
buildConstraints(constraints,1,11,1,1,1,1);
gridBag.setConstraints(logInfosButton, constraints);
this.add(logInfosButton);


Panel bottomPanel = new Panel();
exitButton = new Button(" Schliessen ");
exitButton.setFont(font);
exitButton.addActionListener(this);
applyButton = new Button(" Uebernehmen ");
applyButton.setFont(font);
applyButton.addActionListener(this);
refreshButton = new Button(" Aktualisieren ");
refreshButton.setFont(font);
refreshButton.addActionListener(this);
bottomPanel.add(applyButton);
bottomPanel.add(refreshButton);
bottomPanel.add(exitButton);

buildConstraints(constraints,0,12,2,1,1,1);
constraints.anchor = GridBagConstraints.CENTER;
gridBag.setConstraints(bottomPanel, constraints);
this.add(bottomPanel);


//valueFields[3].setText("blabla");

this.setSize(500, 700);
```

```
this.setLocation(100,100);
this.setVisible(true);
this.setResizable(false);


// LogInfoFrame !
logInfoFrame = new Frame("LogInfo");
logInfoFrame.setLayout(new FlowLayout());
logInfoTextArea = new TextArea(agLogInfos);
logInfoTextArea.setFont(font);
logInfoTextArea.setEditable(false);
exitLogInfoButton = new Button("schliessen");
exitLogInfoButton.setFont(font);
exitLogInfoButton.addActionListener(this);
logInfoFrame.add(logInfoTextArea);
logInfoFrame.add(exitLogInfoButton);
logInfoFrame.setSize(600, 300);
logInfoFrame.setVisible(false);
logInfoFrame.setResizable(false);


// toPingListFrame
toPingListFrame = new Frame("toBeCheckedList");
toPingListFrame.setLayout(new FlowLayout());
checkedList = new List(10);
checkedList.setFont(font);
exitToPingListButton = new Button("schliessen");
exitToPingListButton.setFont(font);
exitToPingListButton.addActionListener(this);
toPingListFrame.add(checkedList);
toPingListFrame.add(exitToPingListButton);
toPingListFrame.setSize(300, 300);
toPingListFrame.setVisible(false);
toPingListFrame.setResizable(false);


    }

    /**
     * EventHandling
     */
    public void actionPerformed(ActionEvent e){
if (e.getSource() instanceof Button){

    Object eventSource = e.getSource();
    int eventInt = -1;

    if (eventSource == connectButton)                eventInt = 1;
    if (eventSource == exitButton)                   eventInt = 2;
    if (eventSource == hostsToBeCheckedButton)       eventInt = 3;
    if (eventSource == logInfosButton)               eventInt = 4;
    if (eventSource == applyButton)                  eventInt = 5;
    if (eventSource == refreshButton)                eventInt = 6;
    if (eventSource == exitLogInfoButton)            eventInt = 7;
    if (eventSource == exitToPingListButton)         eventInt = 8;

    switch (eventInt){
    case 1 :
System.out.println("Verbinde mit : "+ connectField.getText());
initElements();
break;
    case 2 :
logInfoFrame.setVisible(false);
toPingListFrame.setVisible(false);
this.dispose();
System.out.println("exitButton");
break;
    case 3 :
System.out.println("hostsToBeCheckedButton");
initAgHostsToBeCheckedList();
toPingListFrame.setVisible(true);
```

```
break;
    case 4 :
System.out.println("logInfosButton");
logInfoTextArea.setText(agLogInfos);
logInfoFrame.setVisible(true);

break;
    case 5 :
System.out.println("applyButton");
initAgHostsToBeCheckedVector();
applyElements();
break;
    case 6 :
System.out.println("refreshButton");
initElements();
break;
    case 7 :
logInfoFrame.setVisible(false);
break;
    case 8 :
 toPingListFrame.setVisible(false);

    }
}
    }

    private void initAgHostsToBeCheckedList(){
if (agHostsToBeChecked.size() > 0){
    checkedList.removeAll();
    for (int i = 0; i<agHostsToBeChecked.size(); i++)
checkedList.add((String)(agHostsToBeChecked.elementAt(i)));
}
    }
    private void initAgHostsToBeCheckedVector(){
if (checkedList.getItemCount() > 0){
    agHostsToBeChecked.removeAllElements();
    for (int i = 0; i<checkedList.getItemCount(); i++)
agHostsToBeChecked.addElement(checkedList.getItem(i));

}


    }


    private void initElements(){

String agHost        = connectField.getText();
String agDataClass   = "AgGlobalVariables";
ObjectName name
    = new ObjectName("defaultDomain:AgGlobalVariables.SerialNo=0");
AdaptorClient adaptor  = new AdaptorClient();

try{
    adaptor.connect(null, agHost, 1099, ServiceName.APT_RMI);
    MgBasis.logInfo("connected to Host >> " + agHost);
    AgGlobalVariablesMO agGlobalVariables
= (AgGlobalVariablesMO)((Vector)adaptor.getObject(name, null)).firstElement();
    MgBasis.logInfo("AgGlobalVariables contacted");


    valueFields[0].setText(agGlobalVariables.getrouterIpName());
    valueFields[1].setText(agGlobalVariables.getrouterIpAdress());
    String temp = ((new Integer(agGlobalVariables.getrouterSnmpPort()).toString()));
    valueFields[2].setText(temp);
    valueFields[3].setText(agGlobalVariables.getagentHostIpName());
    valueFields[4].setText(agGlobalVariables.getagentHostIpAdress());
    valueFields[5].setText(agGlobalVariables.getmanagerHostName());
    temp = (agGlobalVariables.getcheckFrequence()).toString();
    valueFields[6].setText(temp);
    boolean tf = agGlobalVariables.getguiVisible();
```

```
    if (tf) guiChoice.select("true");
    else  guiChoice.select("false");
    agHostsToBeChecked = agGlobalVariables.gethostsToBeChecked();
    agLogInfos         = agGlobalVariables.getlogInfos();



}
catch (Exception e){
    MgBasis.logInfo("MgGUIAgParam.initElements(): Exception !!");
    MgBasis.logInfo("Uebertragung fehlgeschlagen !!");
    e.printStackTrace();
}
    }


    /**
     * Uebernehmen aller Aenderungen
     */
    private boolean applyElements(){
String agHost        = connectField.getText();
String agDataClass   = "AgGlobalVariables";
ObjectName name
    = new ObjectName("defaultDomain:AgGlobalVariables.SerialNo=0");
AdaptorClient adaptor  = new AdaptorClient();

try{
    adaptor.connect(null, agHost, 1099, ServiceName.APT_RMI);
    MgBasis.logInfo("connected to Host >> " + agHost);
    AgGlobalVariablesMO agGlobalVariables
= (AgGlobalVariablesMO)((Vector)adaptor.getObject(name, null)).firstElement();
    MgBasis.logInfo("AgGlobalVariables contacted");

    agGlobalVariables.setrouterIpName(valueFields[0].getText());
    agGlobalVariables.setrouterIpAdress(valueFields[1].getText());
    Integer temp = Integer.valueOf(valueFields[2].getText());
    agGlobalVariables.setrouterSnmpPort(temp.intValue());
    agGlobalVariables.setmanagerHostName(valueFields[5].getText());
    Long temp1 = Long.valueOf(valueFields[6].getText());
    agGlobalVariables.setcheckFrequence(temp1);
    String tf = guiChoice.getSelectedItem();
    if (tf.equals("true")) agGlobalVariables.setguiVisible(true);
    else agGlobalVariables.setguiVisible(false);

    agGlobalVariables.sethostsToBeChecked(agHostsToBeChecked);
    agGlobalVariables.setlogInfos(agLogInfos);
    return true;

}
catch (NumberFormatException e){
    System.out.println("Eingaben ueberpruefen ! ");
    return false;
}

catch (Exception e){
    MgBasis.logInfo("MgGUIAgParam.initElements(): Exception !!");
    MgBasis.logInfo("Uebertragung Fehlgeschlagen !!");
    e.printStackTrace();
    return false;
}


    }

    public Insets getInsets(){
return new Insets(30,30,30,30);
    }

    /**
     * Methode zum Setzen der Constraints
     * @param GridBagConstraint, gridx, gridy, gridwidth, gridheight, weightx, weighty
```

```
     */
    private void buildConstraints(GridBagConstraints gbc, int gx, int gy, int gw,
  int gh, int wx, int wy){
gbc.gridx      = gx;
gbc.gridy      = gy;
gbc.gridwidth  = gw;
gbc.gridheight = gh;
gbc.weightx    = wx;
gbc.weighty    = wy;
    }


    public static void main(String argv[]){
 MgGUIAgParam ag = new  MgGUIAgParam();
    }
}
```

B.9 Die Klasse MgGUIDataBrowser

```
import java.awt.*;
import java.awt.event.*;
import java.util.Vector;
import java.util.Date;

/**
 * Diese Klasse stellt einen Datenbrowser zur Verfuegung
 * @author Michael Hojnacki
 * @version 25.3.99
 **/
public class MgGUIDataBrowser extends Frame implements ActionListener{

    Button showButton;
    Button exitButton;
    Choice vonChoice;
    TextField zuChoice;
    Label listCountLabel;
    List list;
    int listCount;

    public MgGUIDataBrowser(){

super("Messdaten - Browser");

listCount = 0;

GridBagLayout gridBag = new GridBagLayout();
GridBagConstraints constraints = new GridBagConstraints();
setLayout(gridBag);

Font font = new Font("Monospaced", Font.BOLD, 15);

// TitelPanel
Panel titlePanel = new Panel();
Font fontTitle= new Font("Monospaced", Font.BOLD, 40);
Label titleLabel = new Label("Messdaten - Browser", Label.CENTER);
titleLabel.setBackground(Color.yellow);
titleLabel.setFont(fontTitle);
titlePanel.add(titleLabel);

// AuswahlPanel
Panel auswahlPanel = new Panel();
Label vonLabel     = new Label("Von : ", Label.RIGHT);
Label zuLabel      = new Label(" Zu : ", Label.RIGHT);
vonChoice          = new Choice();
zuChoice           = new TextField();
showButton         = new Button(" Anzeigen ");
GridLayout gl      = new GridLayout(1,5,10,10);


Vector agentHosts = MgBasis.dataHandler.getAgentHostsVector();
vonChoice.add("Alle");
for (int i = 0; i<agentHosts.size(); i++){
    String temp = (String)(agentHosts.elementAt(i));
    vonChoice.add(temp);
}
zuChoice.setText("Alle");
vonLabel.setFont(font);
zuLabel.setFont(font);
vonChoice.setFont(font);
zuChoice.setFont(font);
showButton.addActionListener(this);
showButton.setFont(font);
auswahlPanel.setLayout(gl);
auswahlPanel.add(vonLabel);
auswahlPanel.add(vonChoice);
auswahlPanel.add(zuLabel);
auswahlPanel.add(zuChoice);
auswahlPanel.add(showButton);
```

```
// List
list = new List(20, false);

// BottomPanel
Panel bottomPanel = new Panel();
bottomPanel.setLayout(new FlowLayout());
exitButton = new Button("  Schliessen  ");
exitButton.addActionListener(this);
listCountLabel= new Label("12341234");
listCountLabel.setFont(font);
Label listCountLabelLabel = new Label("Anzahl der Elemente:");
listCountLabelLabel.setFont(font);
exitButton.setFont(font);
bottomPanel.add(listCountLabelLabel);
bottomPanel.add(listCountLabel);
bottomPanel.add(exitButton);


// TitelPanel einfuegen
buildConstraints(constraints,0,0,3,1,1,1);
gridBag.setConstraints(titlePanel, constraints);
this.add(titlePanel);

// AuswahlPanel einfuegen
buildConstraints(constraints,0,1,3,1,1,1);
gridBag.setConstraints(auswahlPanel, constraints);
this.add(auswahlPanel);

// ListPanel einfuegen
buildConstraints(constraints,0,2,3,3,1,1);
constraints.fill = GridBagConstraints.BOTH;
gridBag.setConstraints(list, constraints);
this.add(list);

// BottomPanel einfuegen
buildConstraints(constraints,2,5,1,1,1,1);
gridBag.setConstraints(bottomPanel, constraints);
this.add(bottomPanel);

this.setSize(600, 600);
this.setLocation(300,300);
this.setVisible(true);
this.setResizable(false);
listCountLabelLabel.setText("0");
    }

    public Insets getInsets(){
return new Insets(30,30,30,30);
    }

    /**
     * EventHandling
     */
    public void actionPerformed(ActionEvent e){
if (e.getSource() instanceof Button){

    Object eventSource = e.getSource();
    int eventInt = -1;

    if (eventSource == showButton)    eventInt = 1;
    if (eventSource == exitButton)    eventInt = 2;

    switch (eventInt){
    case 1 :
System.out.println("showButton");
listCount = 0;
initList();
break;
    case 2 :
this.dispose();
```

```
System.out.println("exitButton");
break;
    }
}
    }


    private void initList(){

        String selection = vonChoice.getSelectedItem();
        String zuSelection = zuChoice.getText();

        if (list.getItemCount() > 0)
    list.removeAll();

        // von Alle zu Alle
        if ((selection.equals("Alle"))&&(zuSelection.equals("Alle"))){
    Vector agentHosts = MgBasis.dataHandler.getAgentHostsVector();
    if (agentHosts.size()>0)
        for (int i = 0; i<agentHosts.size(); i++){
    String temp = (String)(agentHosts.elementAt(i));
    Vector measures = MgBasis.dataHandler.getAgentHostMeasures(temp);
    int noe = measures.size();
    if (noe > 0){
        // in der ersten Zeile wird der Hostname nochmal uebergeben !
        String listItem = new String();
        for (int n = 1 ; n < noe; n++){
    MgMeasure m = (MgMeasure)(measures.elementAt(n));
    listItem = ((String)(measures.elementAt(0))) + "\t | ";
    String tmp = m.gettoIpName();
    if (tmp.length() < 15)
        for (int t = tmp.length(); t<15; t++)
    tmp+=" ";
    listItem = listItem + m.getmeasurementDate() + " | "   + tmp + "\t | " ;
    listItem = listItem + m.getmethod()          + "\t | " + m.getresultInMillis();
    list.add(listItem);
    listCount++;
    listCountLabel.setText(Integer.toString(listCount));
        }
    }
        }
        }
        // von Hostname zu Alle
        else if ((!(selection.equals("Alle")))&&(zuSelection.equals("Alle"))){
    Vector measures = MgBasis.dataHandler.getAgentHostMeasures(selection);
    int noe = measures.size();
    if (noe > 0)
        // in der ersten Zeile wird der Hostname nochmal uebergeben !
        for (int i = 1 ; i < noe; i++){
    MgMeasure m = (MgMeasure)(measures.elementAt(i));
    String listItem = new String();
    listItem = ((String)(measures.elementAt(0))) + "\t | ";
    listItem = m.getmeasurementDate()    + " | " + m.gettoIpName() + "\t | " ;
    listItem = listItem + m.getmethod() + "\t | " + m.getresultInMillis();
    list.add(listItem);
    listCount++;
    listCountLabel.setText(Integer.toString(listCount));
        }
        }
        // von Hostname zu Hostname
        else if ((!(selection.equals("Alle")))&&(!(zuSelection.equals("Alle")))){
    Vector measures = MgBasis.dataHandler.getAgentHostMeasures(selection);
    int noe = measures.size();
    if (noe > 0)
        // in der ersten Zeile wird der Hostname nochmal uebergeben !
        for (int i = 1 ; i < noe; i++){
    MgMeasure m = (MgMeasure)(measures.elementAt(i));
    if ((m.gettoIpName()).equalsIgnoreCase(zuSelection.trim())){
        String listItem = new String();
        listItem = ((String)(measures.elementAt(0))) + "\t | ";
        listItem = m.getmeasurementDate()    + " | " + m.gettoIpName() + "\t | " ;
```

```
        listItem = listItem + m.getmethod() + "\t | " + m.getresultInMillis();
        list.add(listItem);
        listCount++;
        listCountLabel.setText(Integer.toString(listCount));
    }
        }
        }
        // Alle zu Hostname
        else if ((selection.equals("Alle"))&&(!(zuSelection.equals("Alle")))){
    Vector agentHosts = MgBasis.dataHandler.getAgentHostsVector();
    if (agentHosts.size()>0)
        for (int i = 0; i<agentHosts.size(); i++){
    String temp = (String)(agentHosts.elementAt(i));
    Vector measures = MgBasis.dataHandler.getAgentHostMeasures(temp);
    int noe = measures.size();
    if (noe > 0){
        // in der ersten Zeile wird der Hostname nochmal uebergeben !
        String listItem = new String();
        for (int n = 1 ; n < noe; n++){
    MgMeasure m = (MgMeasure)(measures.elementAt(n));
    if ((m.gettoIpName()).equalsIgnoreCase(zuSelection.trim())){
        listItem = ((String)(measures.elementAt(0))) + "\t | ";
        String tmp = m.gettoIpName();
        if (tmp.length() < 15)
    for (int t = tmp.length(); t<15; t++)
        tmp+=" ";
        listItem = listItem + m.getmeasurementDate() + " | "   + tmp + "\t | " ;
        listItem = listItem + m.getmethod()          + "\t | " + m.getresultInMillis();
        list.add(listItem);
        listCount++;
        listCountLabel.setText(Integer.toString(listCount));
    }
        }
    }
        }
        }



    }

    /**
     * Methode zum Setzen der Constraints
     * @param GridBagConstraint, gridx, gridy, gridwidth, gridheight, weightx, weighty
     */
    private void buildConstraints(GridBagConstraints gbc, int gx, int gy, int gw,
  int gh, int wx, int wy){
gbc.gridx      = gx;
gbc.gridy      = gy;
gbc.gridwidth  = gw;
gbc.gridheight = gh;
gbc.weightx    = wx;
gbc.weighty    = wy;
    }
    public static void main(String argv[]){
MgGUIDataBrowser db = new MgGUIDataBrowser();
    }

}
```

B.10 Die Klasse MgGUIZentrale

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

/**
 * Startfenster fuer den Manager
 */
public class MgGUIZentrale extends Frame implements ActionListener{

    // Prozesskontrolle
    Button startFrameworkButton;
    Button agentConfigButton;
    String startFrameworkButtonText;
    // DatenBank
    Button saveButton;
    Button restoreButton;
    Button browseButton;
    //Auswertung
    Button auswertung1Button;
    Button auswertung2Button;
    Button auswertung3Button;
    // StatusText
    TextArea textArea;
    String logInfoString;
    // Bottom
    Button exitButton;
    Button clearButton;
    Label statusLabel;
    String statusLabelText;

    public MgGUIZentrale(){
super("Manager Zentrale");

GridBagLayout gridBag = new GridBagLayout();
GridBagConstraints constraints = new GridBagConstraints();
setLayout(gridBag);

// TitelPanel
Panel titlePanel = new Panel();
Font font = new Font("Monospaced", Font.BOLD, 40);
Label titleLabel = new Label("Manager Zentrale", Label.CENTER);
titleLabel.setBackground(Color.red);
titleLabel.setFont(font);
titlePanel.add(titleLabel);

Font fontPanelLabel = new Font("Monospaced", Font.BOLD, 20);

// buttonPanelProzesskontrolle
Panel buttonPanelProzesskontrolle      = new Panel();
startFrameworkButtonText               = new String("CMF stoppen");
Label buttonPanelProzesskontrolleLabel = new Label("Prozesskontrolle", Label.CENTER);
startFrameworkButton                   = new Button (startFrameworkButtonText);
agentConfigButton                      = new Button ("Agentenkonfiguration");
startFrameworkButton.addActionListener(this);
agentConfigButton.addActionListener(this);
buttonPanelProzesskontrolleLabel.setFont(fontPanelLabel);
GridLayout gl1 = new GridLayout(4,1,10,10);
buttonPanelProzesskontrolle.setLayout(gl1);
buttonPanelProzesskontrolle.add(buttonPanelProzesskontrolleLabel);
buttonPanelProzesskontrolle.add(startFrameworkButton);
buttonPanelProzesskontrolle.add(agentConfigButton);

// buttonPanelDatenbank
Panel buttonPanelDatenbank      = new Panel();
Label buttonPanelDatenbankLabel = new Label("Datenbank", Label.CENTER);
saveButton                      = new Button (" Speichern ");
restoreButton                   = new Button (" Wiederherstellen");
browseButton                    = new Button (" Anzeigen ");
saveButton.addActionListener(this);
```

```
restoreButton.addActionListener(this);
browseButton.addActionListener(this);
buttonPanelDatenbankLabel.setFont(fontPanelLabel);
GridLayout gl2 = new GridLayout(4,1,10,10);
buttonPanelDatenbank.setLayout(gl2);
buttonPanelDatenbank.add(buttonPanelDatenbankLabel);
buttonPanelDatenbank.add(saveButton);
buttonPanelDatenbank.add(restoreButton);
buttonPanelDatenbank.add(browseButton);


// buttonPanelAuswertung
Panel buttonPanelAuswertung      = new Panel();
Label buttonPanelAuswertungLabel = new Label("Auswertung", Label.CENTER);
auswertung1Button                = new Button("Auswertung 1");
auswertung2Button                = new Button("Auswertung 2");
auswertung3Button                = new Button("Auswertung 3");
auswertung1Button.addActionListener(this);
auswertung2Button.addActionListener(this);
auswertung3Button.addActionListener(this);
buttonPanelAuswertungLabel.setFont(fontPanelLabel);
GridLayout gl3 = new GridLayout(4,1,10,10);
buttonPanelAuswertung.setLayout(gl3);
buttonPanelAuswertung.add(buttonPanelAuswertungLabel);
buttonPanelAuswertung.add(auswertung1Button);
buttonPanelAuswertung.add(auswertung2Button);
buttonPanelAuswertung.add(auswertung3Button);

// statusTextPanel
textArea = new TextArea();
logInfoString = new String();
textArea.setFont(new Font("Helvetia",Font.BOLD, 14));
        textArea.setEditable(false);

// bottomPanel
Panel bottomPanel = new Panel();
statusLabelText   = new String("inaktiv");
exitButton        = new Button("     Beenden     ");
clearButton       = new Button("Clear");
Label status      = new Label("Status : ", Label.RIGHT);
statusLabel       = new Label(statusLabelText, Label.CENTER);
Font fontStatus = new Font("Monospaced", Font.BOLD, 14);
status.setFont(fontStatus);
statusLabel.setFont(fontStatus);
statusLabel.setBackground(Color.yellow);
GridLayout gl5    = new GridLayout(1,4,10,10);
bottomPanel.setLayout(gl5);
exitButton.addActionListener(this);
clearButton.addActionListener(this);
bottomPanel.add(status);
bottomPanel.add(statusLabel);
bottomPanel.add(clearButton);
bottomPanel.add(exitButton);


buildConstraints(constraints,0,0,3,1,1,1);
gridBag.setConstraints(titlePanel, constraints);
this.add(titlePanel);


// ButtonPanel 2. Reihe
Panel buttonPanel = new Panel();
GridLayout gl4    = new GridLayout(1,3,10,10);
buttonPanel.setLayout(gl4);
buttonPanel.add(buttonPanelProzesskontrolle);
buttonPanel.add(buttonPanelDatenbank);
buttonPanel.add(buttonPanelAuswertung);
buildConstraints(constraints,0,1,3,1,1,1);
gridBag.setConstraints(buttonPanel, constraints);
this.add(buttonPanel);
```

```
buildConstraints(constraints,0,2,3,1,1,1);
constraints.fill = GridBagConstraints.BOTH;
gridBag.setConstraints(textArea, constraints);
this.add(textArea);
constraints.fill = GridBagConstraints.NONE;


buildConstraints(constraints,0,3,3,1,1,1);
gridBag.setConstraints(bottomPanel, constraints);
this.add(bottomPanel);

this.setSize(830, 600);
this.setLocation(300,300);
this.setVisible(true);
this.setResizable(false);
    }


    /**
     * EventHandling
     */
    public void actionPerformed(ActionEvent e){
if (e.getSource() instanceof Button){

    Object eventSource = e.getSource();
    int eventInt = -1;

    // Prozesskontrolle
    if (eventSource == startFrameworkButton) eventInt = 1;
    if (eventSource == agentConfigButton)    eventInt = 2;
    // DatenBank
    if (eventSource == saveButton)           eventInt = 3;
    if (eventSource == restoreButton)        eventInt = 4;
    if (eventSource == browseButton)         eventInt = 5;
    //Auswertung
    if (eventSource == auswertung1Button)    eventInt = 5;
    if (eventSource == auswertung2Button)    eventInt = 7;
    if (eventSource == auswertung3Button)    eventInt = 8;
    // bottom
    if (eventSource ==  clearButton)         eventInt = 9;
    if (eventSource ==  exitButton)          eventInt = 10;

    switch (eventInt){
    case 1 :
if (startFrameworkButtonText == "CMF stoppen")
    startFrameworkButtonText = "CMF starten";
else
    startFrameworkButtonText = "CMF stoppen";
startFrameworkButton.setLabel(startFrameworkButtonText);
System.out.println("startFrameworkButton");
break;
    case 2 :
// Hier ein Frame zum konfigurieren des Agenten erstellen
MgGUIAgParam mgGUIAgParam = new MgGUIAgParam();
System.out.println("agentConfigButton");
break;
    case 3 :
int meldung = MgBasis.ioHandler.writeDataObject(MgBasis.dataHandler.dataObjects);
if (meldung == 1)
    setlogInfoString("DatenObjekte gespeichert");
else

    setlogInfoString("Versuchte Datensicherung gescheitert");
System.out.println("saveButton");
break;
    case 4 :
System.out.println("restoreButton");
MgBasis.dataHandler.dataObjects = MgBasis.ioHandler.readDataObject();
break;
    case 5 :
MgGUIDataBrowser db = new MgGUIDataBrowser();
```

```
System.out.println("browseButton");
break;
    case 6 :
System.out.println("auswertung1Button");
break;
    case 7 :
System.out.println("auswertung2Button");
break;
    case 8 :
System.out.println("auswertung3Button");
break;
    case 9 :
System.out.println("clearButton");
performClearTextArea();
break;
    case 10 :
System.out.println("exitButton");
int        result;
result = JOptionPane.showConfirmDialog(this, "Wollen Sie wirklich beenden ?");
if(result == JOptionPane.YES_OPTION)
    java.lang.System.exit(0);
break;

    }
}
    }

    // ===================================
    /**
     * Getter, Setter und Performer der AWT-Komponenten
     */
    public String getstartFrameworkButtonText(){
return startFrameworkButtonText;
    }
    public String getstatusLabelText(){
return statusLabelText;
    }
    public String getlogInfoString(){
return logInfoString;
    }

    public void setstartFrameworkButtonText(String value){
startFrameworkButtonText = value;
startFrameworkButton.setLabel(value);
    }
    public void setstatusLabelText(String value){
 statusLabelText = value;
 statusLabel.setText(value);
    }
    public void setlogInfoString(String value){
logInfoString = logInfoString + "\n" + value;
textArea.append("\n" + value);
MgBasis.ioHandler.writeLogfile(logInfoString);
    }

    /**
     * der logInfoString wird nicht geloescht, da er die Informationen
     * fuer das LogFile enthaelt !!
     */
    public void performClearTextArea(){
        textArea.setText("");
    }


    // ===================================

    public Insets getInsets(){
return new Insets(30,30,30,30);
    }

    /**
```

```
     * Methode zum Setzen der Constraints
     * @param GridBagConstraint, gridx, gridy, gridwidth, gridheight, weightx, weighty
     */
    private void buildConstraints(GridBagConstraints gbc, int gx, int gy, int gw,
  int gh, int wx, int wy){
gbc.gridx      = gx;
gbc.gridy      = gy;
gbc.gridwidth  = gw;
gbc.gridheight = gh;
gbc.weightx    = wx;
gbc.weighty    = wy;
    }

    /**
     * Debug
     */
    public static void main(String argv[]){
MgGUIZentrale guiStart = new MgGUIZentrale();
guiStart.setlogInfoString("Hier eine wichtige Mitteilung");
guiStart.setlogInfoString("Hier noch eine wichtige Mitteilung");
    }
}
```

B.11 Die Klasse MgMeasure

```
import java.util.Vector;
import java.util.Date;
import java.io.*;

/**
 * Diese Klasse verwaltet die Daten einer Messung
 * @author Michael Hojnacki
 * @version 12.3.99
 */
public class MgMeasure implements Serializable{

    private Date    measurementDate;
    private String  toIpAdress;
    private String  toIpName;
    private Integer resultInMillis;
    private String  method;

    /**
     * Konstruktor
     */
    public MgMeasure(){
measurementDate = new Date();
toIpAdress      = new String();
toIpName        = new String();
resultInMillis  = new Integer(0);
method          = new String();
    }

    /**
     * Konstruktor
     * @param (Date), (String)toIpAdress, (Integer)resultInMillis, (String)method
     */
    public MgMeasure(Date d, String tia, String tin, Integer r, String m){
measurementDate = d;
toIpAdress      = tia;
toIpName        = tin;
resultInMillis  = r;
method          = m;
    }

    public void setmeasurementDate(Date value){
measurementDate = value;
    }
    public void settoIpAdress(String value){
toIpAdress = value;
    }
    public void settoIpName(String value){
toIpName = value;
    }
    public void setresultInMillis(Integer value){
resultInMillis = value;
    }
    public void setmethod(String value){
method = value;
    }

    public Date getmeasurementDate(){
return measurementDate;
    }
    public String gettoIpAdress(){
return toIpAdress;
    }
    public String gettoIpName(){
return toIpName;
    }
    public Integer getresultInMillis(){
return resultInMillis;
    }
    public String getmethod(){
```

```
return method;
    }

    public Vector getallValues(){
Vector value = new Vector();
value.addElement(measurementDate);
value.addElement(toIpAdress);
value.addElement(resultInMillis);
value.addElement(method);
return value;
    }

}
```

Anhang C

Abkürzungsverzeichnis

A

ACL	Access Control List
ALI	Alarmintegration (DTS)
API	Application Programming Interface
ARM	Application Response Measurement

C

CCTA	Central Computer and Telecommunication Agency
CHAP	Challenge Handshake Protocoll
CMF	Core Management Framework (JDMK)
CMG	Computer Measurement Group
CN	Coorperate Network
CORBA	Common Object Request Broker Architecture

D

DNS	Domain Name Server
DMI	Desktop Management Interface
DMTF	Desktop Management Task Force
DoD	Dail in Demand
DTS	DeTeSystem GmbH

E

EMS	Element Management System

F

FMA	Flexible Management Agents

G

GUI	Graphical User Interface

H

HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol

I

IEEE	Institute of Electrical and Electronic Engineers
IETF	Internet Engineering Task Force
IIOP	Internet Inter–Orb Protocol
IOS	Internet Operating System
IP	Internet Protocol
ISDN	Integrated Services Digital Network
IT	Information Technology
J	
JIT	Just in Time Compiler
JDBC	Java Database Conectivity
JDK	Java Development Kit
JDMK	Java Dynamic Management Kit
JNI	Java Native Interface
JRE	Java Runtime Environment
K	
KQML	Knowledge Query and Manipulation Protocol
L	
LAN	Local Area Network
M	
MbD	Management by Delegation
MIB	Management Information Base
MIBGEN	Managed Information Base Generator
MIF	Managed Information Format
MLET	Management Applet (Service)
MO	Managed Object
MOGEN	Managed Object Generator
O	
OID	Object Identifier
P	
PMA	Plattform für Management–Applikationen
PoP	Point of Presence
Q	
QoS	Quality of Service
R	
RFC	Request for Comments
RMI	Remote Method Invocation
RPC	Remote Procedure Call
S	
SLA	Service Level Agreements
SLM	Service Level Management
SMC	System Management Center

SNMP	Simple Network Management Protocol
SNMP-DPI	SNMP - Distributed Program Interface
SRZ	Service Rechenzentrum
SSL	Secure Sockets Layer Protocol

T

TCP	Transmission Control Protocol
TTS	Trouble Ticket System

U

UDP	User Datagram Protocol
URL	Unified Resource Locator

V

VLAN	Virtual Local Area Network
VPN	Virtual Private Network

W

WAN	Wide Area Network
WWW	World Wide Web

Anhang D

Literaturverzeichnis

Literaturverzeichnis

[ARM 97] HP und IBM: *Application Response Measurement 2.0 API*, 1997.

[DiAl 99] DIERKS, 0. T. und C. ALLEN: *RFC 2246: The TLS Protocol Version 1.* RFC, IETF, Januar 1999.

[Duerr1] HELMUT DÜRR: *Netz- und Systemmanagement der DeTeSystem*, Januar 1999. DeTeSystem.

[Duerr2] HELMUT DÜRR: *Betriebsparameter*, Januar 1999. DeTeSystem.

[Flan 98] FLANAGAN, DAVID: *Java in a Nutshell.* O'Reilly & Associates, 2nd Edition Auflage, 1998.

[FrKi 98] FREED, N. und S. KILLE: *RFC 2248: Network Services Monitoring MIB.* RFC, IETF, Januar 1998.

[HASS 1] HASSENMÜLLER, HARALD: *Ein Dach für Netze, Systeme und Services.* LANLine, 4/98:103 ff., 1998. AWi LANline Verlagsgesellschaft mbH.

[HeAb 93] HEGERING, H.-G. und S. ABECK: *Integriertes Netz– und Systemmanagement.* Addison-Wesley, 1993.

[HPW 98] HARRINGTON, D., R. PRESUHN und B. WIJNEN: *RFC 2271: An Architecture for Describing SNMP Management Frameworks.* RFC, IETF, Januar 1998.

[ITIL 2] (ITIL), INFORMATION TECHNOLOGY INFRASTRUCTURE LIBRARY: *Service Level Management - Costumer Focused.* The Stationery Office, 1997.

[JDMK 98] SUN MICROSYSTEMS, INC.: *JDMK3.0(beta) Programming Guide*, 1998.

[Kalb 98] ALIA, C. KALBFLEISCH ET: *Application Management MIB.* Internet, November 1998.

[Knoe 99] KNÖCHLEIN, H.: *Management eines Internet Telefonie Servers mittels JDMK* . Diplomarbeit, Technische Universität München, Februar 1999.

[KrSa 98] KRUPCZAK, C. und J. SAPERIA: *RFC 2287: Definitions of System-Level Managed Objects for Applications.* RFC, IETF, Februar 1998.

[Lang 98] DANNY B.LANGE, MITSURU OSHIMA: *Programming and Developing Java Mobile Agents with Aglets.* Addison Wesley Longman, 2nd Edition Auflage, 1998.

[McBr 98] MCBRIDE, DOUG: *The SLA - Cookbook: A Recipe for Understanding System an Network Resource Demands.* Technischer Bericht Hewlett Pacckard, 1998.

[MH 98] MICHAEL HOJNACKI, CAROLA SCHENKEL: *Wiederverwendung in der Softwareentwicklung, Ansatz einer betriebswirtschaftlichen Betrachtung.* Technischer Bericht Technische Universität München, 1998. Wissenschaftliche Arbeit im Rahmen eines interdisziplinären Projektstudiums.

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

[Reif 97] REIFER: *Practical Software Reuse, Strategies for Introducing Reuse Concepts in your Organization.* Addison Wesley, 1997.

[Rose 91] ROSE, M. T.: *RFC 1227: SNMP MUX protocol and MIB.* RFC, IETF, Mai 1991.

[Salm 89] SALMINEN, VELI-MATTI J. (Herausgeber): *Leveraging Service Level Agreements.* Computer Measurement Group, Dezember 1989.

[Sol] SOLSTICE: *Solstice Enterprise Agents User's Guide Release 1.0.* Part Number 804-0008-10.

[SSL] ALAN O. FREIER, PHILIP KARITON, PAUL C. KOCHER: *The SSL Protocol Version 3.0.* Netscape Communication Corporation. Internet Draft.

[Stal 93] STALLINGS, WILLIAM: *SNMP, SNMPv2, and CMIP: The Practical Guide to Network Management Standards.* Addison-Wesley, 1993.

[Tanen 98] TANENBAUM, ANDREW S.: *Computernetzwerke.* Prentice Hall, 1998.

[UML] RATIONAL SOFTWARE, MS, HP, ORACLE, IBM, ...: *UML Notation Guide version 1.1*, September 1997.

[Unter 98] UNTERREITMEIER, GERTRAUD: *Sicherheit von mobilen Agenten am Beispiel von MAF und CORBA.* Doktorarbeit, Technische Universität München, Mai 1998.

[WPM 98] WIJNEN, B., R. PRESUHN und K. MCCLOGHRIE: *RFC 2275: View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP).* RFC, IETF, Januar 1998.