Einsatz des Java Dynamic Management Kit für flexible, dynamische Managementsysteme

Helmut Reiser
Münchner Netzmanagement Team
Institut für Informatik, Ludwig–Maximilians–Universität München
Oettingenstr. 67, D-80538 München
Telefon: +49-89-2178-2163, Fax: -2262
E-Mail: reiser@informatik.uni-muenchen.de

Dipl.–Inform. Helmut Reiser, Studium der Informatik an der Technischen Universität München, Diplom 1997. Seit 1997 am Lehrstuhl Kommunikationssysteme und Systemprogrammierung (Prof. H.-G. Hegering) der Ludwig–Maximilians–Universität München beschäftigt und Mitglied des Münchner Netzmanagement Teams. Forschungsgebiete: Integriertes Netz– und Systemmanagement, Sicherheitsaspekte im IT–Management und in Mobilen Agentensystemen.

Zusammenfassung

Klassische, zentralisierte IT–Managementplattformen sind relativ unflexibel. Es kann daher nicht schnell auf Änderungen – in der Infrastruktur oder auf Seiten der Anforderungen – reagiert werden. In den letzten Jahren wurden deshalb neue Managementarchitekturen entwickelt, die sich dadurch charakterisieren lassen, daß sie flexibel anpaßbar sind, die Möglichkeit der dynamischen Funktionsdelegierung bieten und den Managementagenten eine größere Autonomie ermöglichen.

Vor dem Hintergrund eines Industrieprojekts, welches das Java Dynamic Management Kit (JDMK) zur Entwicklung einer Managementanwendung für einen Telephony Internet Server verwendet, wird untersucht, inwieweit JDMK geeignet ist, um große, flexible und dynamische Managementsysteme aufzubauen.

Schlüsselwörter: JDMK, JMX, flexibles und dynamisches Management, JavaBeans, Internet Telefonie Server

1 Einleitung

In einer globalisierten, arbeitsteiligen Wirtschaft sind vernetzte und verteilte IT–Systeme unabdingbare Infrastrukturen. Netze, Systeme und Anwendungen werden immer größer, eine manuelle Konfiguration und Überwachung ist wegen der Komplexität nicht mehr möglich. Das sichere Funktionieren der Systeme und die schnelle Anpaßbarkeit an geänderte Anforderungen bekommen zunehmend unternehmenskritische Bedeutung. Es ist daher nötig und unvermeidbar, geeignete Managementsysteme für die Steuerung dieser IT–Infrastruktur einzusetzen.

Es hat sich jedoch gezeigt, daß traditionelle Managementsysteme zu unflexibel sind, um schnell auf geänderte Anforderungen oder Rahmenbedingungen reagieren zu können. Die dynamische Verteilung von Managementfunktionalität auf erweiterbare Agenten wird als vielversprechender Ansatz für flexible, skalierbare und integrierte Managementsysteme betrachtet. Das Java Dynamic Management Kit (JDMK) verspricht ein geeignetes Hilfsmittel zu sein, um derartige Managementanwendungen entwickeln zu können. Es wird deshalb im Rahmen dieses Papiers untersucht und seine Einsatzfähigkeit zur Realisierung flexibler, dynamischer Managementsysteme für sehr große IT–Infrastrukturen bewertet.

Im folgenden Abschnitt werden die Anforderungen an ein flexibles, dynamisches Managementsystem und dessen Agenten abgeleitet. Dazu werden zuerst traditionelle, zentralisierte und plattformbasierte Managementsysteme und deren Schwächen vorgestellt. Es wird erläutert, welche neuen Managementkonzepte und Techniken entwickelt wurden, um diese Schwächen zu beseitigen. Im dritten Abschnitt werden die Architektur sowie die Dienste von JDMK vorgestellt. JDMK wurde in einer Industriekooperation mit der Siemens AG eingesetzt, um eine Managementanwendung für den Telephony Internet Server (TIS) zu entwickeln. Im vierten Abschnitt wird dieses Projekt und die praktischen Erfahrungen, die beim Einsatz von JDMK gewonnen wurden, vorgestellt. Vor dem Hintergrund dieses Projekts und den Anforderungen an zukünftige Managementsysteme wird JDMK im fünften Abschnitt bewertet.

Im Juni 1999 wurde die neue *Java Management Extensions (JMX)* Spezifikation veröffentlicht und für einen öffentlichen Review Prozeß freigegeben. JMX soll die *Java Management–API (JMAPI)* ersetzen und verwendet zum großen Teil Konzepte von JDMK. Aus diesem Grund wird JMX im Abschnitt 6 vorgestellt. Der letzte Teil faßt die Ergebnisse zusammen und diskutiert offene Fragestellungen.

2 Klassische Managementarchitekturen

In den Organisationsmodellen der klassischen Managementarchitekturen, dem OSI–Management und dem Internet–Management — nach dem dort verwendeten Simple Network Management Protokoll (SNMP), auch als SNMP–Management bezeichnet — werden zwei Rollen definiert; Agenten und Manager:

- *OSI–* oder *SNMP–Agenten* stellen Managementinformation über eine sogenannte Management Information Base (MIB) zur Verfügung und melden managementrelevante Beobachtungen, die auch als Managementereignisse, Events oder Traps bezeichnet werden. Agenten werden auf bzw. für alle zu managenden Ressourcen, also Netzkomponenten, Endsysteme oder auch Anwendungen, installiert.

- *Manager* fragen die Managementinformation von den Agenten ab und verarbeiten sie. Sie empfangen Managementereignisse und reagieren auf diese. Managementplattformen wie HP OpenView, Cabletron Spectrum, Tivoli TME oder CA Unicenter agieren in der Managerrolle. Sie stellen eine Plattform für Managementanwendungen und die Managementstationen, also die Arbeitsplätze der Administratoren bereit, die einzelne Managementaufgaben, z.B. Inventarisierung oder Software–Verteilung, realisieren.

Im Internet–Management wird die Schnittstelle zwischen Agent und Manager durch die *Structure of Management Information (SMI)* [12, 13, 11] definiert, ein datentyporientiertes Informationsmodell. Der Manager fragt die Werte einfacher Variablen oder Tabellen ab. Im OSI–Management wird im Prinzip auch so gearbeitet, jedoch wird ein objektorientiertes Informationsmodell, beschrieben durch die *Guidelines for the Definition of Managed Objects (GDMO)* [7], verwendet.

Die klassischen Internet–Managementsysteme sind als zentralisierte, plattformbasierte Architekturen realisiert. Von einer zentralen Managementplattform aus wird eine sehr große Zahl von Agenten verwaltet. Die Menge der von einem Agenten verwalteten Managementobjekte wird als (Agenten–) MIB bezeichnet. Ein *Managementobjekt (MO)* stellt die Abstraktion der realen Ressource zu Managementzwecken dar, auf der der Agent operiert. Die Agenten besitzen nur sehr einfache, starre Funktionalität und Zugriffsschnittstellen. Sie sind gewöhnlich nicht in der Lage, Managementdaten zu verdichten. In der heute überwiegend in der Praxis eingesetzten Version 1 von SNMP gehen bspw. alle Traps direkt an den Manager, insofern kann ein SNMP–Agent hier keine Vorverarbeitung durchführen.

Der *Remote Network Monitoring (RMON)* Standard macht hier allerdings eine Ausnahme. RMON [25, 26, 1] beschreibt Managementobjekte eines standardisierten entfernten Netzüberwachungsgeräts und die RMON–Agenten können auch Vorverarbeitung (z.B. Berechnung von Statistiken) durchführen. Andere SNMP–Agenten bieten nur die Möglichkeit, Daten aus dem MO auszulesen oder Werte zu verändern. Sie sind aber nicht in der Lage, selbst aktiv bzw. reaktiv auf Veränderungen der MOs zu reagieren. Derartige zentralisierte, plattformbasierte Ansätze werden auch als statisch bezeichnet, da vordefinierte Managementfunktionen vorausgesetzt werden und eine Änderung dieser Funktionalität zur Laufzeit nicht möglich ist. Bei einer Änderung der Managementanforderungen oder der Managementinformation (Management Information Base, MIB) ist folglich eine Neukompilierung des Agenten oder die Installation eines neuen Agenten notwendig.

Aus dem zentralisierten Managementansatz ergeben sich gravierende Nachteile.

- Die zentrale Managementplattform stellt einen „single–point–of–failure“ dar.

- Durch die große Anzahl von Agenten und die fehlende Verdichtung der Managementdaten durch die Agenten kann es zu sehr hoher Netzlast kommen. Die zentrale Managementstation ist der Engpaß des Systems und wird häufig überlastet.

- Die gesamte Intelligenz ist auf die Managementplattform konzentriert. Obwohl die Anzahl der Applikationen, die auf einer zentralen Managementstation zur Ausführung gebracht werden können, begrenzt ist, sind Agenten nicht in der Lage, selbst Managemententscheidungen zu treffen, um damit den zentralen Manager zu entlasten.

Das Problem der hohen Last, der die zentrale Managementplattform ausgesetzt ist, versuchte man durch die Einführung sogenannter *Mid–Level–Manager* zu lösen. Bei dieser Architektur wird zwischen der zentralen Managementplattform und den einfachen Agenten eine Schicht mit „vereinfachten“ Managern eingezogen, die mit eigener statischer Intelligenz ausgestattet werden und eine gewisse Vorverarbeitung durchführen können.

- Aber auch diese Architektur ist relativ unflexibel, eine beliebige, schnelle Anpaßbarkeit von Managementanwendungen ist nur schwer möglich. Für das Management von Infrastrukturen und Komponenten, die häufigen Änderungen unterworfen sind, ist Flexibilität, auch bei der Funktionalität der Agenten, für einen effizienten Betrieb unabdingbar.

Um eine solche Flexibilität zu erreichen, wurde eine Möglichkeit gesucht, die dynamische Verlagerung von Funktionalität auf die Agenten zu ermöglichen. Die Idee des *Management by Delegation (MbD)* [5, 17] besteht darin, die Statik der Funktionszuweisung durch eine dynamische Funktionsdelegierung aufzuheben. In jüngster Zeit gibt es auch im Internet–Management Bestrebungen MbD bzw. dynamische und flexible Managementsysteme zu realisieren [9].

Da auch beim MbD–Ansatz nur sehr einfache, nicht autonome Agenten Verwendung finden, wurde mit den in [14] eingeführten *flexiblen Agenten* das Konzept der Mid–Level–Manager und das MbD–Paradigma erweitert. Flexible Agenten zeichnen sich dadurch aus, daß sie — neben der Möglichkeit der Funktionsdelegierung zur Laufzeit — bis zu einem gewissen Grad autonom handeln und selbst aktiv auf Ereignisse reagieren können. Innerhalb eines Managementsystems können flexible Agenten zu Gruppen zusammengefaßt werden, die kooperativ eine bestimmte Aufgabe erledigen.

Wird der flexible Agent um das Konzept der Mobilität erweitert, erhält man mobile Agenten (MA) [16, 15], die sich in einem Netzwerk bewegen können, um im Auftrag eines Nutzers oder einer Organisation bestimmte Aufgaben zu erledigen. Auch im IT–Management wird über den Einsatz mobiler Agenten diskutiert und es existieren bereits erste Anwendungen [4]. Die Vorteile, die mobile Agenten für das IT–Management bieten, werden bspw. in [2] vorgestellt.

Aus den Defiziten bzw. Schwächen der vorgestellten Managementkonzepte lassen sich die Anforderungen an zukünftige Managementsysteme ableiten.

Daneben führen aber auch organisatorische Randbedingungen zu neuen Anforderungen an das Management. Netze sind in zunehmendem Maße mission–critical für die Unternehmen, d.h. an das Management dieser Netze und IT–Infrastrukturen werden immer höhere Anforderungen bezüglich Verfügbarkeit, Sicherheit, Zuverlässigkeit und Skalierbarkeit gestellt. In den letzten Jahren hat die Zahl der Managementsysteme stetig zugenommen [6]. Um in einem großen Unternehmen oder bei einem Provider die verschiedenen gewachsenen Managementsysteme, die i.d.R. nicht in der Lage sind mehrere Managementprotokolle zu unterstützen, integrieren zu können, ist ein *Umbrella–Management* notwendig, das die verschiedenen Informations– und Funktionsmodelle sowie unterschiedlichste Protokolle unterstützt. Bestehende Anwendungen oder Agenten müssen einfach in dieses Umbrella–Managementsystem integrierbar sein.

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

Durch die Größe vieler Unternehmen und ihrer IT–Systeme besteht der Bedarf, auch dezentral Management betreiben zu können. Mit den heutigen Managementsystemen ist eine Abbildung organisatorischer Strukturen in Domänen oder eine Gruppenbildung (funktionale Gliederung) jedoch nur schwer zu realisieren. Um organisatorisch selbständigen Einheiten die Möglichkeit zu geben, ihre Infrastruktur auch selbst und eigenverantwortlich verwalten zu können, muß ein Managementsystem einfache und effiziente Möglichkeiten zur Bildung von Gruppen und Domänen bieten.

3 Java Dynamic Management Kit (JDMK)

Das Java Dynamic Management Kit (JDMK)[1], entwickelt von SUN Microsystems [20, 21], verspricht viele der oben genannten Probleme zu beheben und eine Reihe von neuen Anforderungen zu unterstützen. Im

[1] http://www.sun.com/software/java-dynamic/

nächsten Abschnitt wird kurz auf die Architektur und auf die Dienste von JDMK eingegangen.

3.1 Architektur

JDMK stellt ein Framework und entsprechende Entwicklungswerkzeuge zur Verfügung, um eine auf dem JavaBeans Konzept [19] basierende Managementanwendung bzw. Managementagenten zu entwickeln. Die Basiskomponenten der Architektur sind in Abb. 1 zusammengefaßt. Die zentralen Bestandteile sind dabei das Core Management Framework, M– und C–Beans sowie verschiedene Adapter und Services.

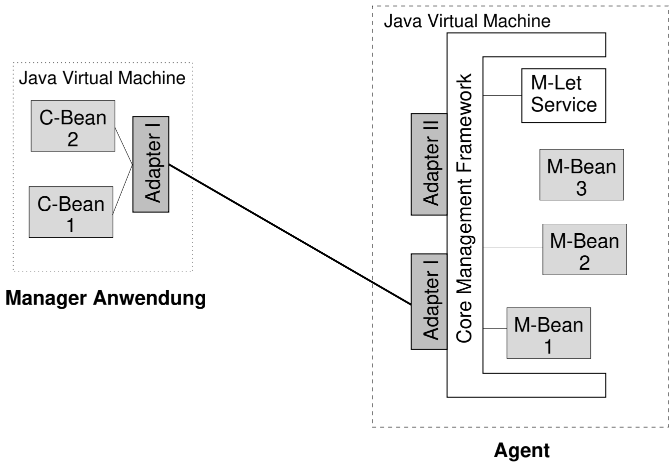

Abbildung 1: JDMK–Architektur

M–Beans (Managed Beans) sind Java Objekte, die die Intelligenz und die Funktionalität des Agenten implementieren. Ein M–Bean steht stellvertretend für ein oder mehrere *Managed Objects (MOs)* oder implementiert gewisse Funktionen eines Managers, z.B. die Vorverarbeitung von Daten. M–Beans werden nach dem JavaBeans Komponentenmodell entwickelt. Sie werden über ihre Objektnamen adressiert und eindeutig identifiziert. Ein solcher Objektname hat folgende Form: *domainPart*:*classPart*[.*attribute*=*value*[,*attribute*=*value*]*]. Die Namensteile können frei vergeben werden, insbesondere muß der *classPart* nicht mit den Namen der Implementierungsklasse übereinstimmen. Die optionalen *attribute*=*value* Paare können dazu verwendet werden, ein Bean näher zu klassifizieren. Für alle Teile des Namens können Filterregeln definiert werden, um bestimmte M–Beans zu selektieren (vgl. Filtering Service in Abschnitt 3.2).

Das *Core Management Framework (CMF)* stellt die BeanBox, d.h. eine Rahmenstruktur dar, in der die M–Beans registriert werden müssen, um auf JDMK–Dienste oder Kommunikationsmechanismen zugreifen bzw. von außen angesprochen werden zu können. Zur Registrierung wird der Objektname des M–Beans verwendet. Falls kein Name gesetzt wurde, generiert das CMF einen Namen für das M–Bean. Das CMF mit seinen M–Beans repräsentiert einen Management Agenten. Das CMF stellt also eine zentrale Schnittstelle bereit und die Objekte (M–Beans, Services, u.a.) können entweder vom Agenten selbst oder durch einen Manager im CMF registriert werden.

Aus den M–Beans (bzw. aus den `.class`–Files, die die M–Beans implementieren) lassen sich durch einen Compiler (`mogen`) C–Beans generieren. Die *C–Beans (Client Beans)* sind Proxy–Objekte für M–Beans. Die Daten und Funktionen eines entfernten M–Beans können durch den Aufruf von Operationen des lokalen C–Beans genutzt werden. C–Beans verwenden Adapter, um mit ihrem entsprechenden M–Bean zu kommunizieren. C–Beans und Adapter bilden, zusammen mit Objekten, die spezielle Managementfunktionen implementieren, die Manager–Seite der Anwendung. Auch ein Agent kann C–Beans in seinem CMF registrieren und wird dadurch zum Manager für einen anderen Agenten, der die entsprechenden M–Beans implementiert. Mit JDMK wird also die strenge Trennung zwischen Manager– und Agentenrolle aufgehoben.

Ein *Adapter* implementiert ein bestimmtes Protokoll und stellt eine Schnittstelle zum CMF dar. Damit lassen sich Agenten und Manager untereinander oder mit externen Anwendungen verbinden. Adapter sind ebenfalls als Beans realisiert und können sehr einfach in einem CMF registriert werden. Im Moment werden Adapter für die Protokolle RMI, HTTP, HTTP über SSL (HTTPS), IIOP und SNMP zur Verfügung gestellt. Daneben wird auch ein sogenannter HTML–Adapter angeboten. Dabei handelt es sich um einen Web–Server, der automatisch HTML–Seiten, als Zugriffsschnittstelle für die M–Beans im entsprechenden Agenten, erzeugt. Durch das Adapter Konzept kann ein JDMK–Agent über ein beliebiges Protokoll angesprochen werden. Dazu ist es nicht nötig, die Funktionalität oder den Code des Agenten zu ändern, sondern es muß lediglich ein entsprechender Adapter registriert werden. Soll bspw. über einen Standard Web–Browser auf einen Agenten zugegriffen werden, so muß im CMF des Agenten nur der HTML–Adapter registriert werden, der dann entsprechende HTML–Seiten für die registrierten M–Beans generiert. Natürlich ist es auch möglich, mehrere verschiedene Adapter gleichzeitig zu verwenden.

3.2 Dienste und Entwicklungswerkzeuge

Neben den beschriebenen Basiskomponenten, bietet JDMK eine Reihe von Diensten und Werkzeugen, um die Entwicklung von Managementanwendungen zu vereinfachen (vgl. Abb. 2).

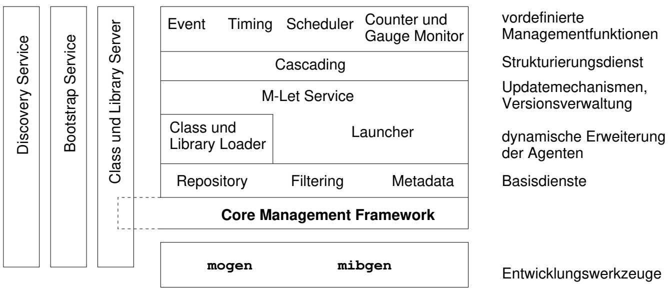

Abbildung 2: JDMK–Dienste und Entwicklungswerkzeuge

Der *Repository Service* wird im Zusammenhang mit der Registrierung von Beans im CMF verwendet. Er dient dazu, Beans persistent oder flüchtig im CMF zu registrieren.

Mit Hilfe des *Filtering Service* können für die bei einem CMF registrierten M–Beans Filterregeln angegeben werden, um alle M–Beans, die der jeweiligen Filterregel entsprechen, auszuwählen. Regeln können über Methoden, Attribute und deren konkrete Belegung sowie über Objektnamen oder Bestandteile davon definiert werden.

Um die Eigenschaften und Methoden, die ein M–Bean unterstützt, zu ermitteln, wird der *Metadata Service* verwendet, der sich auf die Java Reflection API abstützt. Dieser Dienst ist neben dem Repository– und dem Filtering Service ein sogenannter Basisdienst des CMF.

Um alle aktiven CMFs ermitteln zu können, kann der *Discovery Service* verwendet werden. Dieser Dienst sendet einen IP–Broadcast, auf den alle CMFs antworten, die einen sogenannten Discovery Responder registriert haben.

Agent und Manager können verschiedene *Class Loader* — sogar parallel — verwenden, um Implementierungsklassen, lokal oder von entfernten Quellen, zu laden. Der *Library Loader Service* ermöglicht es, plattformspezifische, nicht in Java implementierte Bibliotheken (dynamisch gebundene Libraries) innerhalb eines M–Beans zu verwenden. Als Repository für Class–Files und Bibliotheken kann ein lokaler oder entfernter *Class and Library Server* verwendet werden. Dieser Server kann sowohl als eigenständige Anwendung laufen, da er jedoch als M–Bean realisiert ist, kann er auch innerhalb eines CMF registriert werden.

Für das Nachladen von Code, für Update–Mechanismen und das Bootstrapping, werden M–Let und Launcher Service sowie der Bootstrap Service zur Verfügung gestellt.
Der *M–Let Service (Management Applet Service)* ist ein Dienst, um dynamisch M–Beans zu laden und zu konfigurieren. Dazu wurde ein neues HTML–Tag (`<MLET>`) definiert. Der M–Let Service arbeitet wie folgt (vgl. Abb. 3):

1. Der M–Let Service lädt über eine URL eine HTML–Datei, der er alle Informationen über zu ladende M–Beans (z.B. Namen, Objekte, Code Repository) entnehmen kann.

2. Mit Hilfe dieser Informationen kann er die Implementierungen der entsprechenden M–Beans laden und instantiieren.

3. Danach müssen die M–Beans noch im CMF registriert werden.

Im (`<MLET>`)–Tag können auch Versionsnummern angegeben werden. Damit kann der M–Let Service dann auch zur Versionsverwaltung verwendet werden.

Der *Bootstrap Service* ist eine standalone Java Anwendung, die die Verteilung und Instantiierung von JDMK–Agenten vereinfacht. Dieser Dienst kann Agenten von einem entfernten oder lokalen Server laden. Dazu initialisiert er das CMF, startet den M–Let Service, lädt die notwendigen Klassen und initialisiert, registriert und startet alle notwendigen M–Beans und Services des Agenten.

Der *Cascading Service* ermöglicht die Entwicklung von hierarchischen Master– und Subagentenstrukturen. Der Master Agent stellt dabei die zentrale Zugriffsschnittstelle für den Manager dar und macht die tatsächliche Lokation der Subagenten transparent.

Daneben stellt JDMK auch vordefinierte Managementbasisdienste zur Verfügung. Der *Event Service* ermöglicht eine asynchrone Kommunikation über Unicast– und Multicast–Events. Ein *Timer* und *Scheduler Service* ermöglichen es, Events, Alarme oder Aktionen zu bestimmten Zeitpunkten oder in bestimmten Intervallen auszulösen. Daneben werden Monitore zur Überwachung von Zählern (*Counter–Monitor*) und Schwellwerten (*Gauge–Monitor*) zur Verfügung gestellt.

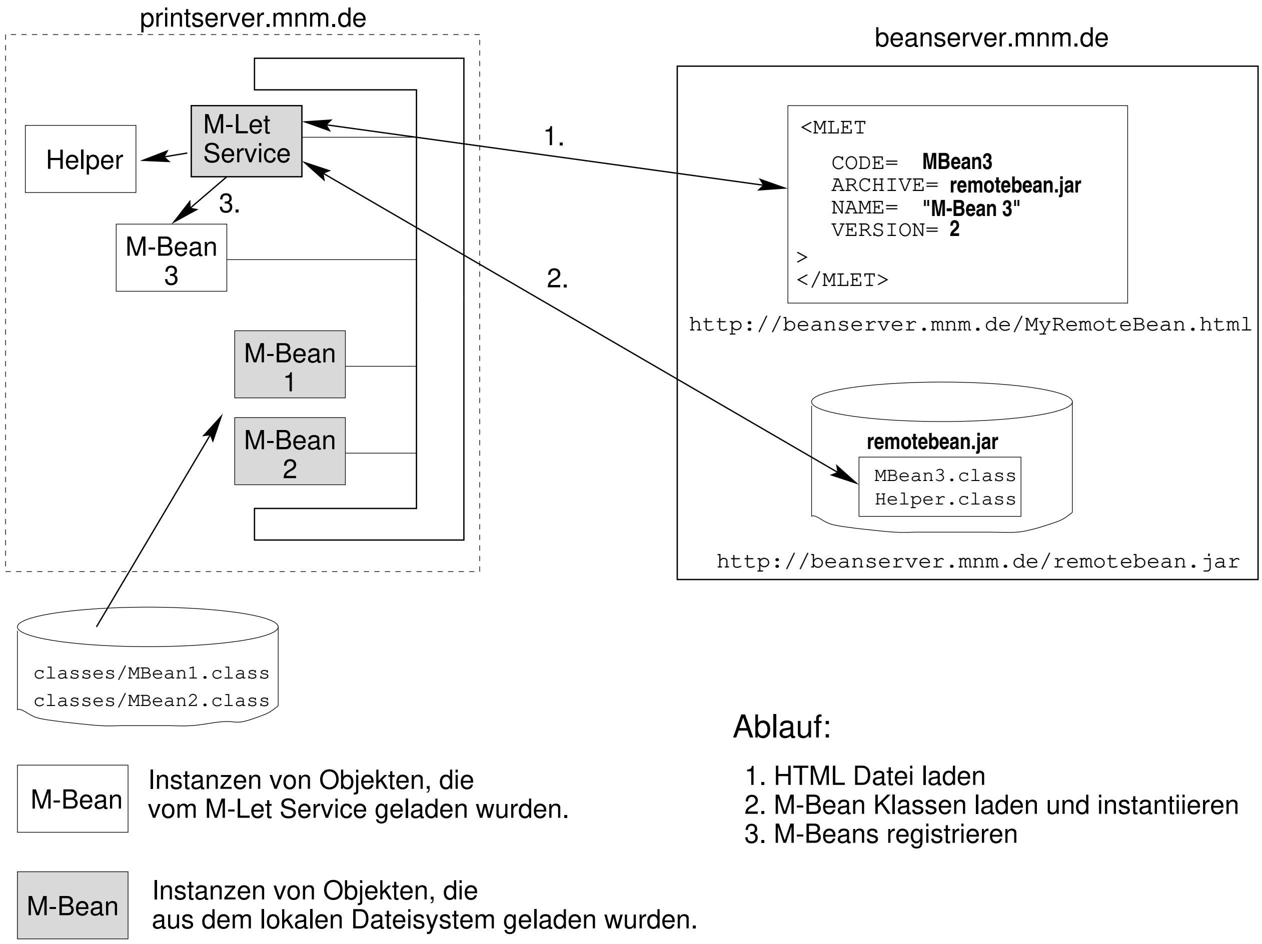

Abbildung 3: Funktionsweise des M-Let Service

Bei den Entwicklungswerkzeugen sind die beiden Compiler `mogen` und `mibgen` zu erwähnen; `mogen` wird verwendet, um aus M–Bean `.class` Dateien C–Beans zu generieren.
Sind für die zu managende Komponente bereits SNMP–MIB Beschreibungen vorhanden, so können mit dem `mibgen` Compiler daraus M–Beans erzeugt werden, die eine Rahmenstruktur implementieren, die die MIB repräsentiert. Diese M–Beans müssen um Funktionen, die den Zugriff auf das zu managende System implementieren, erweitert werden. Mit Hilfe von JDMK lassen sich standalone SNMP–Agenten sowie integrierte SNMP–Agenten, die innerhalb des CMF laufen, implementieren. Integrierte Agenten können über die Adapter des CMF, neben SNMP auch über andere Protokolle (z.B. RMI, IIOP, usw.) angesprochen werden. Standalone Agenten sind nur über SNMP zugreifbar, sie sind von außen nicht von einem gewöhnlichen SNMP–Agenten zu unterscheiden.

4 Management des Telephony Internet Servers mittels JDMK

Die Integration von Sprach– und Datenkommunikation wird immer wichtiger, um die Kosten für den Erwerb, den Betrieb und die Wartung von Kommunikationsdiensten und Geräten zu reduzieren. Es ist möglich, die hohen Kapazitäten bestehender breitbandiger Datennetze für die Übertragung von Sprache zu nutzen. Unternehmen haben damit die Möglichkeit, Anrufe zu entfernten Standorten lokal in Datenströme zu konvertieren und über das Internet oder bestehende eigene Datennetze zu übertragen. An der entfernten Lokation werden die eingehenden Datenströme wieder in Sprache umgesetzt und an den gewünschten Gesprächspartner vermittelt. Mit dieser Technik können teure Verbindungen über öffentliche Telefonnetze (public switched telephone networks (PSTN)) vermieden werden. Dies ist besonders für global agierende Unternehmen attraktiv, für die sich dadurch erhebliche Einsparpotentiale ergeben. Dazu müssen an den verschiedenen Standorten Gateways zwischen den Vermittlungsanlagen (private branch exchange (PBX)) und den Datennetzen installiert werden. Diese Gateways werden als *Internet Telefonie Server* bezeichnet.

Traditionell wurde die Administration und das Management von Telefonsystemen und den entsprechenden Netzen von Carriern mit speziell angepaßten und oftmals proprietären Managementsystemen, Protokollen und Werkzeugen durchgeführt. Die zunehmende Sprach/Daten Integration führt dazu, daß auch Telekommunikationssysteme mit Technologien verwaltet werden, die aus dem Management von Datennetzen kommen. Das bedeutet, daß offene und standardisierte Middleware Infrastrukturen ebenso wie bereits bestehende IT–Managementsysteme unterstützt werden müssen.

Die Telefonanlage Hicom 300 der Firma Siemens mit ihren zugehörigen Gateways ermöglicht die Vermittlung von Telefongesprächen über ATM oder über Internetprotokolle (TCP/IP) [18]. Das Gateway, das die Schnittstelle zwischen analogen sowie digitalen Daten (Audio, Video) und den Internetprotokollen realisiert, wird als *Telephony Internet Server (TIS)* bezeichnet. Außerdem ist der TIS in der Lage, verschiedene Protokolle für Videokonferenzen (H.323 und H.320) zu unterstützen und er integriert verschiedene Endgeräte (z.B. Telefone und Computersysteme), von denen aus telefoniert werden kann oder die an einer Videokonferenz teilnehmen können (vgl. Abb. 4).

Im Rahmen einer Industriekooperation mit der Siemens AG [8] wurden mit JDMK (Version 3.0 Beta 2) eine Managementanwendung und ein Agent für das Management des TIS entwickelt.

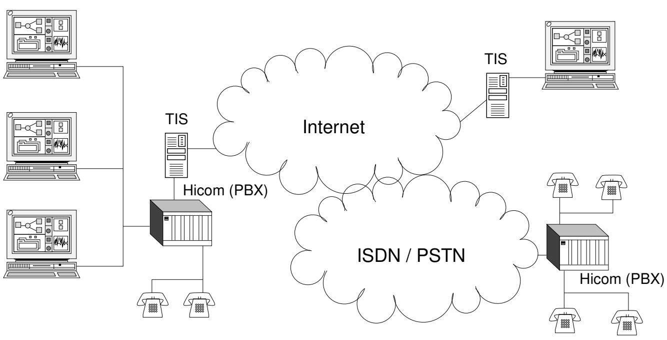

Abbildung 4: Telephony Internet Server (TIS), Anwendungsszenario

Der TIS selbst mit allen seinen Komponenten ist als Windows NT Dienst implementiert. Die bisherige Anwendung zum Management des TIS basiert auf SNMP. Der TIS–Agent ist ebenfalls als Windows NT Dienst implementiert und nutzt die Windows NT SNMP Extension Agent Facility. Damit kann ein neues Managementmodul (implementiert als dynamische Bibliothek) beim Agent registriert werden, ohne den Agenten neu compilieren zu müssen. Allerdings muß bei jeder neuen Registrierung der Agent gestoppt und neu gestartet werden. Der für den TIS verantwortliche Teil des SNMP–Agenten ist rund 250 kBytes groß, die TIS–MIB besteht aus rund 150 Variablen, 20 Tabellen und sechs Typen von asynchronen Events.

Es ist klar, daß — besonders in großen Unternehmensnetzen — Zugriffskontrollen auf die MIB–Variablen realisiert werden müssen. Für einige der Variablen soll der lokale Administrator das Recht zum Lesen und Setzen besitzen, auf die anderen soll nur der für das unternehmensweite TIS–Netzwerk zuständige Administrator zugreifen dürfen. Obwohl dies mit dem SNMP *View–based Access Control Model (VACM)* [27] möglich wäre, ist VACM bisher nicht im TIS–Agenten implementiert. Es ist bislang nicht möglich, abgestufte Zugriffsrechte auf Ebene der MIB–Variablen zu vergeben.

Die Anforderungen, die an ein Managementsystem für den TIS gestellt werden, lassen sich wie folgt zusammenfassen:

- Hoher Grad an Flexibilität, Anpaßbarkeit und Skalierbarkeit
- Möglichkeit die Funktionalität zur Laufzeit der Anwendung zu erweitern oder zu modifizieren
- Einfache Update–Mechanismen
- Fein abgestuftes Sicherheitskonzept (vergleichbar mit SNMP VACM)
- Persistente Speicherung des Agenten und dessen Zustandes und damit ein sicheres Wiederanfahren nach einem Ausfall
- Fähigkeit den Agenten in höherem Maße mit Autonomie in Form von Reaktions– und Aktionsfähigkeit auszustatten
- Zugriff auf standardisierte Middleware, Namens– und Verzeichnisdienste
- Einfache Integration in bestehende Managementsysteme sowie Unterstützung verschiedener Protokolle

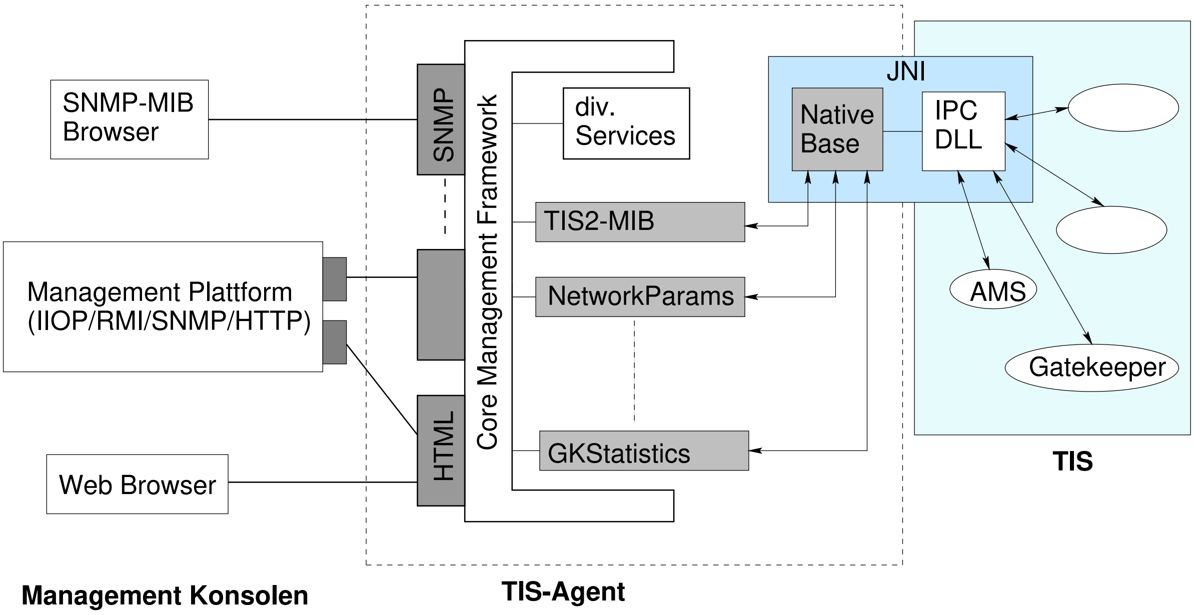

Abbildung 5: Management von TIS mit JDMK

- Unterstützung von Java basiertem Management, um einen höheren Grad an Plattformunabhängigkeit und Portabilität zu erreichen.
- Unterstützung des Web–based Management Paradigmas

JDMK wurde in diesem Projekt eingesetzt, weil es viele interessante Möglichkeiten bietet und viele der gestellten Anforderungen an eine Entwicklungsumgebung bzw. ein Framework erfüllt. Da sich der TIS noch in der Entwicklung befindet und sich deshalb die MIB und die Implementierung noch ändern können, sollte bei der Realisierung eines TIS–Agenten möglichst viel generiert werden können, um den Anpassungsaufwand nach einer Änderung gering zu halten. Aus der TIS–MIB wurden deshalb mit Hilfe des `mibgen` Compilers M–Beans erzeugt, d.h. die Umsetzung der MIB in eine M–Bean Struktur erfolgt automatisch. In den generierten M–Beans müssen vom Entwickler die spezifischen Algorithmen für den Zugriff auf den TIS implementiert werden. Eine Anpassung nach einer Änderung kann dennoch relativ schnell erfolgen, da nur die Implementierung der geänderten M–Beans anzupassen ist. Beim Agenten können die geänderten M–Beans zur Laufzeit durch die neuen ersetzt oder um zusätzliche M–Beans oder Objekte erweitert werden.

Bei einer Änderung der M–Bean Implementierung oder bei einem Versionswechsel des TIS–Agenten können der M–Let und der Launcher Service zum dynamischen Update verwendet werden. Dazu müssen lediglich im `<MLET>`–Tag eines HTML–Files die entsprechenden Klassen eingetragen werden. Der Launcher Service sorgt unter Zuhilfenahme des M–Let Service dafür, daß noch nicht installierte Klassen nachgeladen werden, oder er tauscht den kompletten TIS–Agenten aus, falls sich dessen Versionsnummer geändert hat.

Intern ist der TIS aus mehreren Komponenten (z.B. Gatekeeper, Administration Maintenance Server (AMS), u.a.) aufgebaut, die über ein eigenes Nachrichtenformat über Interprozeßkommunikation (IPC) miteinander kommunizieren. Die dafür notwendigen Methodenaufrufe sind in einer dynamischen Bibliothek zusammengefaßt. Um auf die Managementinformationen der Komponenten zugreifen zu können, mußte der TIS–Agent das Nachrichtenformat und die IPC–Mechanismen implementieren. Dazu wurde die entsprechende dynamische Bibliothek mittels Java Native Interface (JNI) eingebunden und in einer Java Klasse (`NativeBase`, vgl. Abb. 5) gekapselt. Dies bedeutet, daß der JDMK–Agent und dessen M–Beans über Java Methodenaufrufe den TIS managen können. Bei einer Änderung der MIB ändert sich diese Schnittstelle nicht.

Der TIS–Agent sollte Web–basiertes Management ermöglichen, d.h. ein Web–Browser sollte als Managementkonsole verwendet werden können. Diese Anforderung läßt sich sehr einfach durch die Registrierung des HTML–Adapters im CMF des TIS–Agenten realisieren. Dieser Adapter, der wie ein gewöhnlicher WWW–

Server angesprochen wird, erzeugt HTML–Seiten mit allen Informationen und Methoden, die der TIS–Agent zur Verfügung stellt. Die generierten Seiten entsprechen allerdings nicht immer den Anforderungen an ein modernes User Interface. Sie können jedoch als Basis für eigene Erweiterungen verwendet werden. Prinzipiell reicht die Registrierung des HTML–Adapters im TIS–Agenten aber aus, um den TIS über einen Web–Browser managen zu können.

Daneben sollte aber auch der traditionelle Zugriff über SNMP erhalten bleiben und zusätzlich die Möglichkeit eröffnet werden, über andere Protokolle (z.B. RMI) auf den TIS–Agenten zuzugreifen. Auch dies läßt sich durch eine einfache Registrierung des entsprechenden Protokolladapters realisieren.

Eine weitere Forderung war, daß der Agent relativ einfach in eine Corba basierte Architektur zu integrieren sein sollte. Von JDMK wird zwar auch ein IIOP–Adapter angeboten, aber um den TIS–Agenten in eine Corba Infrastruktur zu integrieren, ist dieser Adapter nicht ausreichend. Er implementiert nur das IIOP–Protokoll und verwendet keine Corba Services, d.h. insbesondere werden der TIS–Agent oder dessen M–Beans nicht in den Naming Service oder das Interface Repository eingetragen. Um dies realisieren zu können, wären größere Anpassungen am TIS–Agenten erforderlich.

Die im Rahmen des Projekts realisierte Architektur für das Management von TIS ist in Abb. 5 dargestellt. `TIS2-MIB`, `NetworkParams` und `GKStatistics` sind Beispiele für M–Beans, die Teile der TIS–MIB implementieren.

5 Bewertung von JDMK für das Management

Im folgenden soll JDMK im Hinblick auf die Schwächen traditioneller Managementsysteme, neuer Managementkonzepte und im Kontext der bei Siemens entwickelten Managementanwendungen für den TIS bewertet werden.

5.1 Rapid Prototyping für kleine flexible und dynamische Managementsysteme

Mit JDMK lassen sich gut dezentrale Managementarchitekturen aufbauen. Das Web–based Management Paradigma wird von JDMK sehr gut unterstützt, einfache Web–Interfaces können mit Hilfe des HTML–Adapters dynamisch erzeugt werden. Auch eine Integration in bestehende Managementarchitekturen ist durch die große Anzahl von unterstützten Protokollen möglich. Sollte hierfür ein Kommunikationsprotokoll benötigt werden, für das es keinen Adapter gibt, so kann ein eigener Adapter dafür entwickelt werden.

Mit Hilfe des Adapter Konzeptes und der Implementierung in Java lassen sich einfach multiarchitekturelle Agenten realisieren. Falls ein Agent verschiedene Managementprotokolle simultan unterstützen soll, so ist dies durch die Registrierung mehrerer Adapter sehr einfach möglich. Agenten können dynamisch zur Laufzeit um Funktionalität erweitert werden, wobei die spezifische Funktionalität nur dann geladen wird, wenn sie auch benötigt wird. Für die dynamische Erweiterung werden sowohl Pull– als auch Push–Mechanismen unterstützt. Mit JDMK können auch Methoden realisiert werden, mit deren Hilfe der Agent Aufgaben des Managers übernimmt und eine Vorverarbeitung und Verdichtung der Daten durchführt. Durch das JavaBeans Konzept und die einfache Registrierung der Beans im CMF kann schnell auf Veränderungen bei den Anforderungen an den Managementagenten oder bei den zu managenden Endsystemen reagiert werden.

Durch die zur Verfügung gestellten Dienste lassen sich Bootstrap– und Update–Mechanismen sowie zentrale oder verteilte Code–Repositories und entsprechende Verteilmechanismen etablieren.

Spezifische Endsystem– oder Managementfunktionalität, die in maschinenspezifischen Sprachen geschrieben wurden, läßt sich über dynamische Libraries relativ leicht einbinden. Sollte auf einem Endsystem keine Java Virtual Machine installierbar sein, kann ein JDMK–Agent auch als Proxy auf einem anderen Endsystem implementiert werden, der mit der zu managenden Komponente über SNMP kommuniziert.

Durch die mitgelieferten Entwicklungswerkzeuge ist eine einfache und schnelle Prototypentwicklung möglich.

JDMK unterstützt die persistente Speicherung von M–Beans. Damit können Agenten endsystemnah gespeichert werden und müssen nicht von einem entfernten Server geladen werden. Allerdings werden dabei nur die Java Serialisierungsmechanismen genutzt. Um einen Schutz bei Systemausfällen oder beim Ausfall des Agenten zu gewährleisten, müßte der Entwickler den Agenten und das CMF entsprechend erweitern (z.B. um eine zyklische Sicherung von Agenten und deren Daten).

Zusammenfassend läßt sich sagen, daß JDMK sehr gut für schnelle Prototypentwicklungen von verteilten Managementanwendungen geeignet ist. Das Adapterkonzept kann als Basis für die Integration in heterogene Managementumgebungen verwendet werden.

5.2 Fehlende globale Sicht auf große IT–Infrastrukturen

Neben diesen Vorteilen zeigen sich bei JDMK aber auch teils gravierende Nachteile und konzeptionelle Schwächen.
JDMK bietet keine standardisierten Lokalisierungs– oder Namensdienste an. Es werden auch keine Mechanismen zur Verfügung gestellt, um Dienste wie den Corba Naming– oder einen Trading Service zu nutzen. Daher muß der Entwickler selbst ein Namensschema etablieren und durchsetzen. Eine Ortstransparenz beim Zugriff, wie sie z.B. bei Corba mit Interoperable Object References (IORs) gegeben ist, läßt sich nur ausgesprochen schwer realisieren. Der Zugriff auf M–Beans erfolgt immer über einen Protokollidentifikator, den Host, einen bestimmten Port und den Objektnamen des M–Beans. Alle diese Informationen und ein bestimmtes Grundwissen über die beim jeweiligen CMF registrierten Beans müssen den Anwendern bekannt sein. Da JDMK kein eigenes Informationsmodell definiert und auch keine Gateways zwischen den verschiedenen Protokollen zur Verfügung gestellt werden, muß ein Agent von Hand angepaßt werden, um wirklich multiprotokollfähig zu sein. Um bspw. einen unter JDMK entwickelten SNMP–Agenten auch in der gewohnten Weise aus einer Corba basierten Managementanwendung ansprechen zu können, reicht es nicht, einfach den IIOP–Adapter zu registrieren.

Der Metadata Service, der dazu verwendet werden kann, Informationen über registrierte M–Beans abzufragen, ist immer nur lokal zu *einem* CMF definiert, d.h. es gibt keinen globalen Metadata–Service oder ein globales Interface Repository. Die einzige Möglichkeit, alle gerade aktiven CMFs zu finden, bietet der Discovery Service, der eine Broadcast–Nachricht verschickt, auf die die Agenten antworten. Um dann bspw. alle Agenten zu finden, die eine bestimmte Methode implementieren, muß jeder Agent bzw. jedes CMF, z.B. über den Filtering Service, abgefragt werden.

Auch Konzepte, um eine organisatorische Unterteilung in Domänen bzw. eine funktionale Gliederung in Gruppen zu ermöglichen, sind kaum vorhanden und müssen vom Entwickler selbst realisiert werden.

Eine Interagentenkommunikation ist nur zwischen Master– und Subagenten vorgesehen. Es sind auch keine bzw. nur schwache Ansätze vorhanden, um eine Kooperationsfähigkeit bei den Agenten möglich zu machen.

JDMK–Agenten besitzen nicht die Möglichkeiten, in einem Netz zu migrieren. Das CMF stellt keine Plattform für mobile Agenten, im Sinn der OMG Mobile Agent Systems Interoperability Facility (MASIF) [10] dar, sondern einen Rahmen und Infrastrukturdienste für M–Beans. JDMK–Agenten werden auf einem System instantiiert und sind dann nicht mehr in der Lage, ein anderes System „zu besuchen".

Aufgrund der angeführten konzeptionellen Schwächen ist es relativ schwierig, mit Hilfe von JDMK–Managementanwendungen für große, heterogene IT–Infrastrukturen, in denen eine große Anzahl an verschiedenen Agenten benötigt wird, zu entwickeln. JDMK ist, wenn die Konzeption der mitgelieferten Dienste betrachtet wird, nur für kleinere, lokale IT–Infrastrukturen, in denen die Anzahl und der Grad der Diversifikation der Agenten sehr beschränkt bleibt, geeignet. Da der globale Fokus auf ein großes System fehlt, wird JDMK in seinem momentanen Zustand in solchen Umgebungen auch nicht skalieren.

5.3 Fehlendes Sicherheitskonzept

Ein einheitliches Sicherheitskonzept oder eine Sicherheitsarchitektur für JDMK wird nicht spezifiziert. Es werden lediglich für bestimmte Adapter einfache Sicherheitsmechanismen zur Verfügung gestellt. Der Entwickler muß die verwendeten Konzepte sehr genau kennen, um eine sichere Anwendung implementieren zu können.

Beim SNMP–Adapter wird die Zugriffskontrolle über Access Control Lists (ACL) festgelegt. In einem sogenannten ACL–File kann definiert werden, welche Endsysteme, nur lesend oder auch schreibend, auf welche MIB–Bereiche zugreifen können. Dieses Konzept ist zwar schon eine Verbesserung gegenüber früheren (Paßwort basierten) Konzepten von SNMP, aber trotzdem noch zu grobgranular. Feingranulare Sicherheitsmechanismen wie das SNMP VACM werden bisher nicht unterstützt. Da die Authentisierung der Endsysteme über deren IP–Adresse erfolgt, bleibt das Sicherheitskonzept inhärent unsicher (IP–Spoofing).

Die HTTP/HTML–Adapter bieten als Authentisierungsverfahren eine Login/Paßwort Kombination. Da alle Daten im Klartext übertragen werden, kann damit keine sichere Anwendung entwickelt werden.

Bei den anderen Adaptern (RMI, IIOP) sind keinerlei Authentisierungs– oder Zugriffskontrollverfahren vorhanden.

Die einzige Möglichkeit, eine sichere Authentisierung durchzuführen, bietet der HTTPS (HTTP über SSL) Adapter, bei dem sich kryptographisch sichere Zertifikate zur Authentisierung verwenden lassen. Ein Zugriffskontrollsystem muß aber vom Entwickler selbst implementiert werden.

Verfahren zur Sicherung der übertragenen Daten (Integrität und Vertraulichkeit) sind nicht vorhanden. Insgesamt muß das Sicherheitskonzept von JDMK als ungenügend bezeichnet werden. Große Teile eines Sicherheitskonzepts müssen vom Entwickler selbst implementiert werden. Die großen Unterschiede der Sicherheitsmechanismen einzelner Adapter werden nicht in eine umfassende Sicherheitsarchitektur integriert und dadurch verschattet. Agenten, die verschiedene Managementprotokolle unterstützen, müssen deshalb auch verschiedene Sicherheitsmodelle implementieren.

6 Java Management Extensions

Mitte Juni 1999 wurde die Draft Version der neuen *Java Management Extensions (JMX)* Spezifikation [23] veröffentlicht und für einen öffentlichen Review Prozeß freigegeben. JMX integriert die früheren Entwicklungen im Bereich der Java Management–API (JMAPI) und von JDMK in ein Java basiertes Management Framework. Im Gegensatz zu JDMK liegt der Fokus von JMX nicht nur auf dem Agenten, sondern es soll auch die Manager–Seite betrachtet werden. In der vorliegenden Spezifikation ist der Manager jedoch noch nicht spezifiziert.

Da JDMK als integraler Bestandteil von JMX betrachtet werden kann, soll hier ein kurzer Überblick über die neuen Entwicklungen gegeben werden. Daneben wird die Frage untersucht, inwieweit die, im vorigen Abschnitt betrachteten, für JDMK kritischen Punkte auch für JMX gelten. Dabei muß natürlich beachtet werden, daß die JMX–Spezifikation bisher noch keinen endgültigen Status erreicht hat. Informationen über den aktuellen Stand der Spezifikation sind auf der Web–Seite von Sun Microsystems[2] zu finden.

Die Architektur von JMX ist der in Abb. 1 dargestellten Architektur von JDMK sehr ähnlich. JMX unterscheidet zwischen *Manager–*, *Agenten–* und *Instrumentierungsebene*: Die für die Verwaltung einer Ressource verantwortlichen M–Beans innerhalb eines Agenten bilden die Instrumentierungsebene. Eine zu verwaltende Ressource kann in JMX eine Anwendung, ein Dienst oder eine (Hardware–) Komponente sein. Die Instrumentierungsebene wird der Managementanwendung, die die Managerebene darstellt, durch die Agentenebene zugänglich gemacht. Die Agentenebene stellt eine Kommunikationsschnittstelle, eine Menge von Standarddiensten und die Laufzeitumgebung für M–Beans bereit. Das Core Management Framework von JDMK wurde in *MBean Server* umbenannt.

Eine der größeren Änderungen in JMX betrifft die MBeans[3], die in vier verschiedene Klassen eingeteilt wurden: Das *Standard MBean* repräsentiert eine Teilmenge der aus JDMK bekannten M–Beans. An der Schnittstelle dieses Beans werden Methodenaufrufe zugänglich gemacht. Im Gegensatz dazu werden bei *dynamischen MBeans* Attribute und Signaturen von Operationen durch fest definierte Zugriffsfunktionen bekanntgegeben. Damit wird es einfach, bestehende (Legacy) Anwendungen durch JMX–Agenten zu kapseln. Ein *offenes MBean* ist eine Unterart des dynamischen MBeans, das nur bestimmte, fest vordefinierte Datentypen und Funktionen zur Verfügung stellt. Zusätzlich wird mit dem Interface `MBeanInfo` detailierte Meta–Information zum entsprechenden MBean geliefert. Das Ziel hierbei ist es, sich „selbst beschreibende“ MBeans zu erhalten, die sehr einfach benutzt werden können. Das *Modell MBean* — ein weiteres dynamisches MBean — stellt ein generisches und konfigurierbares Management Template für Managementobjekte dar. Es kann entweder von einem JMX–Agenten, von anderen MBeans oder sogar von der zu verwaltenden Ressource selbst instantiiert werden. Zum Zeitpunkt der Erzeugung können die Signaturen der Operationen und die Menge der zur Verfügung gestellten Attribute durch XML, OMG IDL oder Java beschrieben werden. Außerdem bietet das Modell MBean die Möglichkeit der persistenten Speicherung.

Das Konzept des Adapters in JDMK wird nun in *Protokoll Adapter* und *Konnektoren* aufgeteilt. Protokoll Adapter werden verwendet, um JMX–Agenten mit Anwendungen, die nicht der JMX–Spezifikation entsprechen (z.B. SNMP, Common Information Model (CIM) [3] oder andere) zu verbinden. Im Gegensatz dazu werden Konnektoren eingesetzt, um eine entfernte JMX–Management Anwendung mit einem JMX–Agenten zu verbinden.

JMX erweitert auch den Event Mechanismus von JDMK zu einem *Notification Model*. Damit muß sich ein Event–Listener nur einmal registrieren, um alle Events eines Agenten zu empfangen. Mit einem *Notification Filter* können dann bestimmte Typen von Events selektiert werden. Wenn sich der Wert eines bestimmten MBean Attributs ändert oder wenn ein

[2] `http://java.sun.com/products/JavaManagement/`

[3] Der Trennstrich im Wort „M–Bean“ wurde in JMX entfernt.

MBean erzeugt oder gelöscht wird, können Events erzeugt werden.

Die Dienste, die im Abschnitt 3.2 beschrieben wurden, sind auch in JMX enthalten. Die vordefinierten Management Dienste `Timer` sowie `Counter` und `Gauge Monitor` wurden entsprechend dem Notification Model erweitert.

Um JMX in bestehende Management Lösungen integrieren zu können, wurde eine zusätzliche *Management Protokoll–API* definiert, welche eine offene Schnittstelle spezifiziert, die auch von anderen Herstellern, für eigene Entwicklungen genutzt werden kann. Gegenwärtig sind eine *SNMP–API* [24] und eine *CIM/WBEM–API* [22] definiert.

Die endgültige JMX–Spezifikation wird sowohl eine Referenzimplementierung als auch einen Kompatibilitätstest festlegen. Die Referenzimplementierung soll es Entwicklern erleichtern, JMX konforme Anwendungen zu entwickeln, wohingegen der Test die Kompatibilität mit der Spezifikation überprüfen soll. Allerdings ist noch keines der beiden spezifiziert oder gar als Implementierung erhältlich.

Basierend auf der gegenwärtigen Spezifikation muß festgestellt werden, daß die in Abschnitt 5 identifizierten Schwächen auch in JMX noch nicht behoben wurden. Bis zu welchem Grad der Review Prozeß diese Schwächen eliminieren wird bleibt eine offene Frage. Dies betrifft insbesondere die Managerebene, die im Moment noch völlig unspezifiziert ist.

7 Zusammenfassung und Ausblick

JDMK ist keine Managementarchitektur im eigentlichen Sinne, da kein Informationsmodell definiert wird. Es stellt ein Toolkit mit interessantem Konzept dar, um flexible Managementagenten zu implementieren. Die Entwicklungswerkzeuge ermöglichen eine schnelle Prototypentwicklung.

JDMK ist ein mächtiges Werkzeug, um Managementagenten zu entwickeln, die über verschiedene Kommunikationsmechanismen (IIOP, RMI, HTTP, SNMP, HTTPS) abgefragt und modifiziert werden können. Trotzdem werden die Unterschiede und Besonderheiten dieser Mechanismen nicht durch die Entwicklungsumgebung verschattet und müssen deshalb vom Entwickler beachtet werden.

Die meisten Dienste von JDMK beziehen sich in ihrem Kontext auf ein CMF. Es gibt aber kaum Hilfsmittel, um Informationen über alle Agenten hinweg zu erlangen. Beispielsweise gibt es kein globales Interface Repository oder einen Topology Service, mit dem automatisch Topologien ermittelt und dargestellt werden könnten. Da konzeptionell keine „globale“ Sicht auf große IT–Infrastrukturen besteht, ist es schwierig, eine skalierbare Lösung zu entwickeln.

Die Nutzbarkeit von Managementsystemen — besonders in einem unternehmensweiten Kontext — hängt zu einem hohen Grad von dessen Sicherheitseigenschaften ab. Die Sicherheitsmechanismen von JDMK sind jedoch nicht zufriedenstellend. Die verschiedenen, unter den einzelnen Kommunikationsprotokollen liegenden Sicherheitsmechanismen sind bisher nicht in eine umfassende Sicherheitsarchitektur integriert worden. Es hängt daher von den verwendeten Protokolladaptern ab, ob z.B. Verschlüsselung verwendet werden kann oder wie die Authentisierung durchgeführt wird.

JDMK wurde weder als Managementarchitektur noch als alleinstehendes Management Framework positioniert sondern als integraler Bestandteil der *Java Management Extensions (JMX)* des neuen Java basierten Management Frameworks, das im Moment von Sun und anderen Firmen aus dem Management Umfeld (IBM, Computer Associates, u.a.) innerhalb des „Java Community Process“ entwickelt wird. Auch wenn die JMX–Spezifikation im Moment noch nicht vollständig ist, kann angenommen werden, daß die zukünftige Entwicklung von JMX die Schwächen, die in diesem Beitrag identifiziert wurden, beseitigen wird.

Danksagung

Der Autor dankt dem Münchner Netzmanagement Team für intensive Diskussionen zu früheren Versionen dieses Beitrages. Das MNM–Team, das von Prof. Hegering geleitet wird, ist ein Gruppe von Wissenschaftlern beider Münchner Universitäten und des Leibniz–Rechenzentrums der Bayerischen Akademie der Wissenschaften. Der Webserver des Teams ist unter `http://wwwmnmteam.informatik.uni-muenchen.de` zu finden.

Literatur

[1] A. Bierman and R. Iddon. Remote Network Monitoring MIB Protocol Identifiers. RFC 2074, IETF, January 1997.

[2] Andrzej Bieszczad, Bernard Pagurek, and Tony White. Mobile Agents for Network Management. *IEEE Com-*

munication Surveys, 1(1), 1998. http://www.comsoc.org/pubs/surveys/4q98issue/bies.html.

[3] Common Information Model (CIM) Version 2.2. Specification, Distributed Management Task Force, June 1999.

[4] M. Feridun, W. Kasteleijn, and J. Krause. Distributed Management with Mobile Components. In M. Sloman, S. Mazumdar, and E. Lupo, editors, *Integrated Network Management VI (IM'99)*, pages 857–870, Boston, MA, May 1999. IEEE Publishing.

[5] G. Goldszmit and Y. Yemini. Distributed Managment by Delegation. In *Proceedings of the 15th International Conference on Distributed Computing Systems*, June 1995.

[6] H.-G. Hegering, S. Abeck, and B. Neumair. *Integriertes Management vernetzter Systeme — Konzepte, Architekturen und deren betrieblicher Einsatz*. dpunkt-Verlag, ISBN 3-932588-16-9, 1999. 607 S.

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

[8] H. Knöchlein. Management eines Internet Telefonie Servers mittels JDMK. Diplomarbeit, Februar 1999.

[9] D. Levi and J. Schoenwaelder. Definitions of Managed Objects for the Delegation of Management Scripts. RFC 2592, IETF, May 1999.

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

[11] K. McCloghrie, D. Perkins, and J. Schoenwaelder. Conformance Statements for SMIv2. RFC 2580, IETF, April 1999.

[12] K. McCloghrie, D. Perkins, and J. Schoenwaelder. Structure of Management Information Version 2 (SMIv2). RFC 2578, IETF, April 1999.

[13] K. McCloghrie, D. Perkins, and J. Schoenwaelder. Textual Conventions for SMIv2. RFC 2579, IETF, April 1999.

[14] M.-A. Mountzia. *Flexible Agents in Integrated Network and Systems Management*. PhD thesis, December 1997.

[15] K. Rothermel and F. Hohl, editors. *Mobile Agents (MA '98)*, volume 1477 of *LNCS*, Berlin; Heidelberg, 1998. Springer.

[16] K. Rothermel and R. Popescu-Zeletin, editors. *Mobile Agents (MA '97)*, volume 1219 of *LNCS*, Berlin, April 1997. Springer.

[17] Jürgen Schönwälder. Network Management by Delegation — From Research Prototypes Towards Standards. In *8th Joint European Networking Conference (JENC8)*, Edinburgh, May 1997.

[18] Siemens AG. The convergence of voice and data. White Paper, 1998. http://www.siemens.se/siemensab/communications/itnet/papers.html.

[19] Sun Microsystems, Inc. JavaBeans, Version 1.01. Technical Specification, Sun Microsystems, Inc., Palo Alto, CA, July 1997. http://www.javasoft.com/beans/docs/spec.html.

[20] Sun Microsystems, Inc. Java Dynamic Management Kit. White Paper, Sun Microsystems, Inc., Palo Alto, CA, 1998. http://www.sun.com/software/java-dynamic/wp-jdmk/.

[21] Sun Microsystems, Inc. Java Dynamic Management Kit 3.0 (beta). Programming Guide, Sun Microsystems, Inc., Palo Alto, CA, August 1998.

[22] Sun Microsystems, Inc. Java Mangement Extensions CIM/WBEM APIs. Preliminary Specification Draft 1.9, Sun Microsystems, Inc., Palo Alto, CA, June 1999.

[23] Sun Microsystems, Inc. Java Mangement Extensions (JMX). Preliminary Specification Draft 1.9, Sun Microsystems, Inc., Palo Alto, CA, June 1999.

[24] Sun Microsystems, Inc. Java Mangement Extensions SNMP Manager API. Preliminary Specification Draft 1.9, Sun Microsystems, Inc., Palo Alto, CA, June 1999.

[25] S. Waldbusser. Remote Network Monitoring Management Information Base. RFC 1757, IETF, February 1995.

[26] S. Waldbusser. Remote Network Monitoring Management Information Base Version 2 using SMIv2. RFC 2021, IETF, January 1997.

[27] B. Wijnen, R. Presuhn, and K. McCloghrie. View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP). RFC 2275, January 1998.