TECHNISCHE UNIVERSITÄT MÜNCHEN
Institut für Informatik

Diplomarbeit

Einsatz von WWW und Java im integrierten Management.

Markus Claassen

TECHNISCHE UNIVERSITÄT MÜNCHEN
Institut für Informatik

Diplomarbeit

Einsatz von WWW und Java im integrierten Management.

Aufgabensteller:	Prof. H-G. Hegering
Betreuer:	Boris Gruschke Stephen Heilbronner
Bearbeitung:	Markus Claassen M.Nr. 15 39 868
Abgabe:	15.08.1997

Erklärung

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

Datum Unterschrift

1 Einleitung

Die Tatsache, daß mittlerweile fast jeder Arbeitsplatz mit einem eigenen Rechner ausgestattet ist, eröffnet für die Anwendung, insbesondere die Kommunikation, neue Möglichkeiten und Perspektiven, die zum Teil schon fester Bestandteil der heutigen Arbeitsumgebung geworden sind. Gemeinsame Datenhaltung, electronic mail und intranet, um nur einige zu nennen, ermöglichen eine wesentliche Verbesserung, bei der Verteilung und Speicherung von Informationen. Allerdings ergeben sich durch diese neuen Möglichkeiten auch neue Probleme. Eines davon ergibt sich durch die wachsende Anzahl von Rechnern in Betrieben. Außerdem wird oft eine gewisse Uniformität der Rechner gefordert um Benutzern einen Wechsel zwischen den Rechnern zu erleichtern. Die rasende Entwicklung fordert häufige Änderungen der Software auf den Systemen. Noch dazu kommt die Entwicklung innerhalb von Betrieben, wie Neustrukturierungen von Abteilungen, Neueinstellungen und auch Umzügen. Aus Kostengründen will man selbstverständlich mit möglichst wenig Personal möglichst viele Rechner warten und kontrollieren.

In der Vergangenheit wurde in zunehmendem Maße an Lösungen für diese Problematik gearbeitet. Die Notwendigkeit für ein sogenanntes Netz- und Systemmanagement wurde früh von den Telekommunikationsanbietern wegen ihrer großen Verbreitung erkannt. Mittlerweile nimmt das Management innerhalb der Entwicklungen im Bereich der Rechnernetze eine bedeutende Stellung ein. Der Umfang des Managements dehnt sich dabei immer weiter aus. Netz, Hardware, Software und Benutzer werden mehr und mehr in das Management miteinbezogen. Diese Entwicklung steht allerdings erst am Anfang und wird in den nächsten Jahren noch einige Novitäten liefern.

1.1 Aufgabenstellung und Motivation

Die momentanen Managementlösungen bieten noch viel Platz für Verbesserungen. So gibt es für einige Probleme keine oder noch unbefriedigende Lösungen. Heterogenität ist schon alleine deshalb schwierig, da einige der heute verwendeten Systeme gar kein Management unterstützen.

Um Nachteile in den Griff zu bekommen, wird ständig nach neuen Möglichkeiten gesucht. Dabei werden in letzter Zeit zwei Entwicklungen, die ursprünglich nichts mit dem Integrierten Management zu tun hatten, in dieses zunehmend einbezogen.

Hier hat sich vor allem das World Wide Web (WWW) in der Vergangenheit besonders hervor getan, da es sich von einem einfachem Mechanismus zum weltweiten Austausch von Dokumenten zu einer Plattform für Anwendungen aller Art entwickelt. Diese Mobilität der Anwendung und vor allem die Verbreitung der web browser machen es für das Management interessant. Zum zweiten ist Java zu nennen, das gerade diesen Fortschritt im WWW ohne Umwege über Skriptenprogrammierung ermöglicht. Java ist durch seine Plattformunabhängigkeit und Sicherheit ideal für die Anforderungen im WWW. Aber auch für sich alleine hat Java viele Möglichkeiten, aber im Zusammenspiel mit dem WWW werden diese besonders gut genutzt.

Für das Integrierte Management erwartet man sich hiervon einige Vorzüge. Zum einen wünscht man sich eine mobile Managementoberfläche. Weiterhin erhofft man sich einheitliche Managementlösungen für die verschiedenste Hardware, da Java verspricht, daß ein Einsatz auch auf einfachsten Geräten möglich ist.

Im Rahmen dieser Arbeit sollen die neuen Möglichkeiten, aber auch Probleme durch den Einsatz dieser beiden Technologien durchleuchtet werden und aktuelle Entwicklungen in dem Bereich untersucht werden.

1.2 Überblick

Bei dieser Arbeit soll folgendermaßen vorgegangen werden: Zuerst sollen einige Grundlagen zu Netz- und Systemmanagement, WWW und Java dargestellt werden, soweit diese für das Thema oder das Verständnis der Arbeit relevant sind. Zum WWW werden besonders das HyperText Transfer Protokol (HTTP) und die HyperText Markup Language (HTML) betrachtet. Zu Java werden zuerst einige Grundlagen dargestellt, um es als Programmiersprache einschätzen zu können, dann werden die für das Management interessanten Bereiche erklärt. Dazu zählen die Schnittstellen, die Java unterstützt und die Unterstützung von verteilten Anwendungen.

Im Kapitel 3 „Teilmodelle, APIs, Protokolle“ wird ein Managementmodell aufgestellt, daß speziell auf die neuen Möglichkeiten, die sich durch Java und das WWW ergeben, eingeht. In diesem Rahmen werden zuerst funktionelle Rollen und Schnittstellen beschrieben. Dann werden diese auf Systeme und Kommunikationsprotokolle bzw. Programmierschnittstellen abgebildet. Der Kern ist dabei die Auflösung der Schnittstellen. Die Möglichkeiten für Aufteilung auf die Systeme ergibt sich hier zum Großteil aus den verwendeten Protokollen und der Software.

Der nächste Teil der Arbeit befaßt sich mit neuen Architekturen und Standards, die mittlerweile verfügbar sind oder sich noch in Vorbereitung befinden. Dabei werden zuerst einige Produkte aufgezeigt, die eine Portierung bestehender Standards darstellen. Dann wird

auf Systeme eingegangen, die eine komplett neue Managementarchitektur versprechen. Diese werden dann im vorherigen Modell eingeordnet. Dabei muß jedoch beachtet werden, daß in diesem Bereich noch sehr viel Bewegung herrscht und von einigen Initiativen eventuell sogar noch grundlegende Konzeptänderungen, aber von fast allen noch Erweiterungen und Verfeinerungen zu erwarten sind.

Kapitel 6 „Szenarien“ beschreibt Einsatzmöglichkeiten der durch WWW und Java realisierten Managementkomponenten und bewertet diese.

2 Grundlagen

Dieses Kapitel beschreibt die Grundlagen der Technologien, die im weiteren Verlauf der Arbeit auf ihre Einsatzmöglichkeiten im Management untersucht werden sollen. Zunächst werden einige im WWW verwendete Begriffe und Techniken, wie URL, URI, HTTP, HTML und cgi-Skripten beschrieben, danach wird die Programmiersprache Java mit ihren Zusätzen erläutert, soweit das für die weitere Arbeit sinnvoll ist.

2.1 World Wide Web (WWW)

Mit der weltweiten Verbreitung des Internets, und da mittlerweile fast jeder Rechner über einen Zugang dazu verfügt, ergeben sich neue Möglichkeiten nicht nur für Kommunikation und Informationsverbreitung. Auch die Verteilung von Diensten und Anwendungsprogrammen wird sich in der nächsten Zeit ändern. Aber gerade die weltweite Verfügbarkeit birgt große Gefahren in Bezug auf Mißbrauch. Dadurch entstehen für das Management von System nicht nur neue Möglichkeiten, sondern auch neue Schwierigkeiten.

In diesem Abschnitt werden einige wichtige Elemente im WWW beschrieben, um darauf im folgenden Teil der Arbeit zurückgreifen zu können.

2.1.1 Uniform Resource Locator (URL) / Identificator (URI)

Uniform Resource Locator (URL) ([BMM93]) werden eingesetzt, um die Quelle zu beschreiben, an der eine bestimmte Ressource zu finden ist. Dabei besteht ein URL zuerst aus der Art des Zugriffs auf die Ressource (*Scheme*), die durch das benutzte Protokoll gegeben wird. Danach folgt eine Beschreibung der Quelle (*scheme specific part*) entsprechend dem Protokoll. Bei vielen (*Common Internet Scheme Syntax*) folgt als nächstes die Internetadresse der Quelle mit optionaler Angabe des Ports. Daran schließt die Beschreibung der Ressource innerhalb des Quellsystems an. Das kann ein Dateiname mit Verzeichnis, ein Dienstname oder ähnliches sein. Schließlich können noch Parameter zu der URL angegeben werden. So werden URLs bei der Angabe von Internetdokumenten (http), Dateiablagen (http, ftp), Anwendungen (telnet) und auch von Diensten (rmi) eingesetzt. Der *Uniform Resource Identificator* (URI) ([Ber94]) bezieht sich dabei auf die Quelle selber und nicht wie der URL auf die Methode, wie die Information erlangt wird.

```
<scheme>:<scheme-specific-part>

Einige Beispiele für schemes:
```

ftp	File Transfer protocol
http	Hypertext Transfer Protocol
gopher	The Gopher protocol
mailto	Electronic mail address
news	USENET news
nntp	USENET news using NNTP access
telnet	Reference to interactive sessions
wais	Wide Area Information Servers
file	Host-specific file names
prospero	Prospero Directory Service

```
Common Internet Scheme Syntax:
 //<user>:<password>@<host>:<port>/<url-path>

http URL:
 http://<host>:<port>/<path>?<searchpart>
```

2.1.2 Hypertext Transfer Protokol (HTTP)

Das *Hypertext Transfer Protokol* (HTTP)([BFF96], [FGMFB97]) ist ein Protokoll der Anwendungsschicht zur Übertragung von Hypermedia Informationen. Dabei wird auf Einfachheit und Geschwindigkeit geachtet.

Das HTTP Protokoll basiert auf einem asymmetrischem Anfrage/Antwort Paradigma. Dadurch ergibt sich für jede Verbindung die Rollenverteilung von Klient und Server. Es sitzt auf dem darunterliegenden verbindungsorientierten TCP. Der Klient baut eine Verbindung zu einem Server auf und setzt eine Anforderung, mit Angabe der Art der Anfrage, der gewünschten Information, der verwendeten Protokollversion und Angaben über den Klienten ab. Optional können noch weitere Informationen wie zum Beispiel die erwartete Dateiversion oder ein Nachrichteninhalt mitgesendet werden. Der Server antwortet mit einer Statuszeile, die wiederum Protokollversion und einen Code zur Angabe, ob die Anfrage bedient werden kann, enthält. Darauf können Meta-Informationen und Nachrichteninhalt folgen. Normalerweise wird die Verbindung nach dem Absetzen der confirmation vom Server beendet.

In HTTP gibt es vier mögliche Nachrichtentypen: Simple-Request, Simple-Response aus der Version 0.9 und seit der Version 1.0 Full-Request und Full-Response. Letztere sollten vorgezogen werden, da sie dem Klienten das Ablehnen eines bestimmten Inhalttyps und eine Versionsüberprüfung ermöglichen und werden daher hier näher behandelt.

Sie bestehen aus der Request- beziehungsweise Status-Zeile und Header-Felder, die sich semantisch in Gruppen einteilen lassen. Diese Header-Felder enthalten den jeweiligen Feldnamen und die dazugehörige Information.

```
HTTP-message    =       Simple-Request          ; HTTP/0.9 messages
        |       Simple-Response
        |       Full-Request    ; HTTP/1.0 messages
        |       Full-Response

Full-Request    =       Request-Line            ;
                *(      General-Header  ;
                |       Request-Header  ;
                |       Entity-Header ) ;
                CRLF
                [ Entity-Body ]         ;

Full-Response   =       Status-Line             ;
                *(      General-Header  ;
                |       Response-Header ;
                |       Entity-Header ) ;
                CRLF
                [ Entity-Body ]         ;

Simple-Request  =       "GET" SP Request-URI CRLF
Simple-Response =       [ Entity-Body ]
```


2.1.2.1 Request-Line

```
Request-Line    =       Method SP Request-URI SP HTTP-Version CRLF
```

Die Request-Zeile gibt die angefragte Methode, den URI an den die Anfrage geschickt wird und die verwendete Protokollversion. Die Methode wird durch Schlüsselwörter angegeben.

2.1.2.1.1 GET

Die GET-Methode fordert die, mit dem URI assoziierte Information oder, falls der URI einen Prozeß angibt, die durch diesen produzierten Daten an. Dabei kann das Abfragen der Information von dem Datum der Quelle abhängig gemacht werden.

2.1.2.1.2 HEAD

Mit der HEAD-Methode wird die gleiche Operation, wie bei einem GET ausgeführt, allerdings wird nur der Nachrichtenkopf, ohne den Nachrichteninhalt zurückgesandt. Ein konditionales Abfragen ist hier allerdings nicht möglich. Diese Methode wird oft zum Überprüfen der Existenz einer URI benutzt.

2.1.2.1.3 POST

Mit POST werden Daten vom Klienten zum Server übertragen. Das wird zur Aufnahme neuer Ressourcen, Nachrichten für newsgroups etc. durch den Server, Übertragung von Formulardaten oder zum Einfügen in eine Datenbank benutzt. Die durch POST tatsächlich ausgelöste Aktion, hängt in erster Linie vom Server ab und wird normalerweise von diesem anhand des URI entschieden. Das verschickt Objekt sollte dabei dem Request-URI untergeordnet sein. Der Klient kann dabei ein URI für die neue Ressource vorschlagen. Es ist aber nicht notwendig, daß eine neue Ressource angelegt oder verfügbar gemacht wird. Die Antwort des Servers gibt darüber Auskunft.

2.1.2.1.4 PUT

Auch mit der PUT-Methode werden Daten an den Server übertragen, aber anders als bei POST gibt die Request-URI nicht Ressource an, die die Daten entgegennehmen soll, sondern das Ziel an dem Ressource gespeichert werden sollen. Bei Erfolg gibt der Server mit seiner Antwort an, ob die Daten überschrieben oder neu angelegt wurden. Ist die entsprechende URI nicht verfügbar, so kann der Server noch eine alternative URI zurückgeben, um dem Klienten einen neuen Request zu ermöglichen.

2.1.2.1.5 DELETE

Entsprechend dem Schreiben von Daten mit PUT können mit DELETE Ressourcen gelöscht werden. Ob die Daten tatsächlich vernichtet, zum vernichten vorgesehen werden oder unerreichbar werden, hängt vom jeweiligen Server ab.

2.1.2.1.6 LINK

Die LINK-Methode setzt Beziehungen zwischen der URI-Ressource und anderen existierenden Ressourcen. Dabei enthält die Anfrage kein Nachrichtenkörper und veranlaßt keine Erstellung einer neuen Ressource.

2.1.2.1.7 UNLINK

Hiermit werden Verbindungen zwischen Ressourcen geändert, ohne diese sonst zu verändern, oder neu Übertragen zu müssen.

2.1.2.2 Status-Line

```
Status-Line     = HTTP-Version SP Status-Code SP Reason-Phrase CRLF
```

Die erste Zeile einer Antwort ist die Status-Zeile. Sie enthält Status-Code und Reason-Phrase, die Aufschluß über den Erfolg der Anfrage geben, und die Verwendete Protokollversion. Dabei ist Reason-Phrase eine Kurzbeschreibung des Status-Codes. Die Status-Codes sind in fünf Gruppen eingeteilt, die durch die erste Ziffer der dreistellige Zahl erkennbar ist.

```
Status-Code     =       "200"   ; OK
        |       "201"   ; Created
        |       "202"   ; Accepted
        |       "203"   ; Provisional Information
        |       "204"   ; No Content
        |       "300"   ; Multiple Choices
        |       "301"   ; Moved Permanently
        |       "302"   ; Moved Temporarily
        |       "303"   ; Method
        |       "304"   ; Not Modified
        |       "400"   ; Bad Request
        |       "401"   ; Unauthorized
        |       "402"   ; Payment Required
        |       "403"   ; Forbidden
        |       "404"   ; Not Found
        |       "405"   ; Method Not Allowed
        |       "406"   ; None Acceptable
        |       "407"   ; Proxy Authentication Required
        |       "408"   ; Request Timeout
        |       "409"   ; Conflict
        |       "410"   ; Gone
        |       "500"   ; Internal Server Error
        |       "501"   ; Not Implemented
        |       "502"   ; Bad Gateway
        |       "503"   ; Service Unavailable
        |       "504"   ; Gateway Timeout
        |       extension-code
```

2.1.2.3 1xx Informational

Für die Gruppe der Codes, die mit der Ziffer „1“ beginnen, gibt es noch keine konkrete Verwendung. Sie sind für die Einführung von rein Informellen Nachrichten gedacht.

2.1.2.3.1 2xx Success

Diese Codes geben an, daß die Operation vom Server erfolgreich empfangen, verstanden und ausgeführt wurde.

2.1.2.3.2 3xx Redirection

Der Klient muß noch weitere Aktionen ausführen um die Anfrage durchzuführen. Der Grund dafür kann sein, daß die Daten verschoben wurden oder sich nicht geändert haben (304; Not Modified).

2.1.2.3.3 4xx Client Error

Der Klient hat eine Fehlerhafte Syntax in seiner Anfrage verwendet, oder letztere kann oder will vom Server nicht ausgeführt werden.

2.1.2.3.4 5xx Server Error

Der Server kann eine zulässige Anfrage nicht bedienen.

2.1.2.4 Header-Felder

Im Folgenden sollen die Header-Felder und ihre Bedeutung erläutert werden. Allgemein gilt, daß die Felder nicht notwendig sind, aber soweit dieses Sinn hat, angegeben werden sollen. Außerdem können weitere Felder ergänzt werden.

```
HTTP-header     =       field-name ":" [ field-value ] CRLF
field-name      =       1*<any CHAR, excluding CTLs, SP, and ":">
field-value     =       *( field-content | comment | LWS )
field-content   =       <the OCTETs making up the field-value
                and consisting of either *text or combinations
                of token, tspecials, and quoted-string>
```

2.1.2.4.1 General-Header

Zum General-Header zählen die Felder mit Information über Entstehungsdatum der Nachricht, eventuell benutzten Proxies, einem Nachrichtenidentifikator aus dem FQDN und einer auf dem Ursprungssystem eindeutigen Zeichenkette und der verwendeten MIME-Version.

```
General-Header  =       Date
        |       Forwarded
        |       Message-ID
        |       MIME-Version
        |       extension-header
```

2.1.2.4.2 Request Header

```
Request-Header  =       Accept
        |       Accept-Charset
        |       Accept-Encoding
        |       Accept-Language
        |       Authorization
        |       From
        |       If-Modified-Since
        |       Pragma
        |       Referer
        |       User-Agent
        |       extension-header
```

Die Felder Accept, Accept-Charset, Accept-Encoding und Accept-Language geben dem Klienten die Möglichkeit, die Antwort nur in bestimmten oder bevorzugten Datenformaten, Zeichensätzen, Kodierungen und Sprachen zu erhalten. Dadurch wir sichergestellt, daß die Daten in einer vom Klienten bevorzugten Form zurückgegeben werden, oder Daten, die der Klient nicht verarbeiten kann, erst gar nicht übertragen werden.

Die Felder Authorization und From sind für Zugangsbeschränkungen und zur Angabe der e-mail-Adresse des Anfragenden Benutzers.

Mit If-Modified-Since lassen sich Datum und Zeit der aktuellen Information angeben, so daß der Server die Daten nur überträgt falls diese sich in der Zwischenzeit geändert haben. Damit können Caches sehr netzlastschonend verwaltet werden.

Das Pragma-Feld definiert Direktiven, die alle Server vom Ursprung der Anfrage bis zum eigentlichen Server, befolgen müssen. Momentan gibt es nur die “no-cache” Direktive, die Proxies mit Cache veranlaßt, die Anfrage auf jeden Fall weiterzuleiten.

Mit Referer wird dem Server die URI der Quelle angegeben, von der die geforderte URI angefragt wurde. Damit wird dem Server die Möglichkeit gegeben Statistiken für Caching und Fehlersuche zu geben.

User-Agent enthält Namen und Version des Programms, bei dem der Request seinen Ursprung hat.

2.1.2.4.3 Response Header

```
Response-Header =       Public
        |       Retry-After
        |       Server
        |       WWW-Authenticate
        |       extension-header
```

Hier werden Informationen über die Antwort und den Server als Erweiterung der Statuszeile übergeben. Das Feld Public gibt außer den HTTP Standard Methoden (siehe 2.1.2.1 Request-Line Seite 15), weitere an, die vom Server unterstützt werden. Geht dieses Feld über einen Proxy, so muß dieser das Feld an seine eigenen Fähigkeiten anpassen oder löschen.

Mit Retry-After wird bei einer “503 Service Unavailable” Antwort ein Zeitraum oder ein Zeitpunkt angegeben nach oder an dem der Dienst wieder verfügbar sein kann.

Das Server Feld gibt die verwendete Software des Servers an. Bei einem “401 Unauthorized” muß das Feld WWW-Authenticate die Angaben zur Authentifizierung enthalten.

2.1.2.4.4 Entity Header

```
Entity-Header   =       Allow
        |       Content-Encoding
        |       Content-Language
        |       Content-Length
        |       Content-Transfer-Encoding
        |       Content-Type
        |       Derived-From
        |       Expires
        |       Last-Modified
```



```
        |       Link
        |       Location
        |       Title
        |       URI-header
        |       Version
        |       extension-header
```

Der Entity Header enthält Informationen über das in der Nachricht enthaltene oder durch die URI refrenzierte Objekt.

Allow gibt an welche Methoden von der URI akzeptiert werden. Bei einer “405 Method Not Allowed” Meldung muß es enthalten sein.

Es besteht die Möglichkeit, die übertragenen Daten zusätzlich zu ihrem orginären Format zu kodieren bzw. zu komprimieren. Die verwendete Methode steht dabei in Content-Encoding.

Mit Content-Language werden die Sprachen angegeben in denen das Dokument verfaßt wurde.

Die Content-Length gibt die Länge des Objekts oder bei der HEAD-Methode (siehe 2.1.2.1.2 HEAD Seite 15) die Länge des bei einem GET übermittelten Objekts an.

Zusätzlich zum Content-Encoding kann zur sicheren Übertragung ein Content-Transfer-Encoding angewandt werden.

Der Typ des übermittelten Objektes steht in Content-Type und mit Derived-From kann die Version der Ursprungsressource der Daten angegeben werden.

Expires gibt einen Zeitpunkt an, an dem eine Ressource voraussichtlich durch eine neue Version abgelöst wird. Dabei ist nicht klar, ob die Quelle zu diesem Zeitpunkt noch existiert.

Last-Modified gibt den Stand der Ressource an. Die genaue Interpretation (z.B. Dateidatum) hängt vom jeweiligen Server ab.

Mit Link lassen sich weitere URIs angeben, die zur aktuellen in Beziehung stehen. Location ist ein Vorgänger vom URI-Header (siehe 2.1.2.4.4 Entity Header, Seite 18). Das Title-Feld gibt den Dokumententitel an und antspricht dem <TITLE>-Element in HTML (2.1.3 Hypertext Markup Language (HTML), Seite 19).

Das URI-Feld enthält einen absoluten oder relativen URI und wird speziell mit den 201, 302 und 302 Meldungen (2.1.2.2 Status-Line, Seite 16) eingesetzt. Das Version-Feld ermöglicht schließlich mit Hilfe einer Versionsnummer ermöglicht gemeinsame Entwicklung durch mehrere Benutzer.

2.1.2.5 Entity Body

Am Ende der Nachricht kann dann der Inhalt der Nachricht übersendet werden. Dabei gelten die im Entity Header angegebenen Daten über Typ und Größe des Inhalts.

2.1.3 Hypertext Markup Language (HTML)

Hypertext Markup Language (HTML) ([BeC95]) ist eine Sprache zur Erstellung von Hypertext Dokumenten, die plattformunabhängig sind. Dabei umfaßt HTML strukturierte und formatierte Dokumente, die Verwendung verschiedener Schriftarten, Farben und Größen. Es können Formulare mit Elementen grafischer Oberflächen erstellt werden. Dadurch lassen sich einfache Benutzerdialoge realisieren. Weiterhin können Grafiken und Applets in Dokumente eingebunden werden. Mit in den Dokumenten definierten Links kann innerhalb und zwischen Dokumenten und anderen URLs gewechselt werden. Das darstellende Programm ist dabei für die Korrekte Verwendung des URLs verantwortlich.

Ein HTML-Dokument besteht aus Kopf, der zumeist Titel und Metainformationen enthält, und dem Körper. Letzterer enthält das Dokument an sich. HTML basiert auf Tags, die zum Großteil paarweise als Begrenzer auftreten.

Dadurch, daß WWW-Browser, die in erster Linie zur Darstellung von HTML-Seiten benutzt werden, eine so große Verbreitung erlangt haben, hat auch dieses Format eine ebenso große Anwendung gefunden.

2.1.4 cgi-Skripten

Um eine dynamische Erstellung von Dokumenten und Grafiken zu ermöglichen, besitzen viele WWW-Server eine Skriptenunterstützung. Dabei wird der Server bei Requests auf bestimmte Ressourcen diese von einem durch die Request-URI referenzierten Programm erstellen lassen und die Ausgabe des Programms als Information zurückgeben. Dabei ist für den Klient aus dem HTTP-Protokoll nicht erkennbar, ob eine Seite dynamisch erzeugt wurde oder statisch ist. Allerdings ist dies, zumindest für den Benutzer, meist aus dem Kontext oder der URI erkennbar. In der letzteren können noch Parameter für das jeweilige Programm kodiert werden. Diese gibt der jeweilige Server dann an die Skriptprogramme weiter. Zumeist sind diese Skriptprogramme Batch-Dateien oder Shell-Programme, die als Eingabe vom Server die Aufrufparameter enthalten und die Ausgabe vom Server abfangen und als Hypermedia übermitteln lassen. Diese Skripten werden eingesetzt um Seiten an aktuelle Gegebenheiten anzupassen. Oft wird dies für die Anzeige einer Uhrzeit oder eine Besucherzähler eingesetzt. Eine andere typische Anwendung ist der Zugriff auf Datenbanken, wie ihn viele Suchmaschinen benutzen. Mit Skripten lassen sich unter Verwendung der Formularoptionen von HTML interaktive Dialoge mit dem Benutzer gestalten. Welche Rechte die Skriptenprogramme bei der Ausführung auf dem Server besitzen, hängt von der jeweiligen Implementierung des WWW-Servers ab.

2.2 Java

Seit einiger Zeit ist Java ([Fla96]) ein fester Bestandteil des Internets geworden. Das besondere an Java ist die Portabilität des Bytecodes zwischen Plattformen, die eine Implementierung des Bytecodeinterpreters, der Java Virtual Machine (JVM), zur Verfügung stellen. Momentan wird gerade aus diesem Grund Java eine schlechte Laufzeitperformance vorgeworfen. Bei anderen Programmiersprachen, wie etwa C++, wurden nach der Einführung ebenfalls solche Argumente hervorgebracht. Allerdings hat die Entwicklung der Compiler diese Nachteile weitgehend in den Griff bekommen. Eine ebensolche Optimierung ist auch von dem Java Compiler und der Java VM zu erwarten. Auch die Entwicklungsumgebung läßt noch viele Wünsche offen und muß sich im Laufe der nächsten Zeit noch verbessern um modernen Ansprüchen in der Softwareentwicklung gerecht zu werden.

In diesem Abschnitt werden einige Eigenschaften von Java erläutert.

2.2.1 Die Java Virtual Machine (JVM)

Die *Java VM* ist eine virtuelle Maschine, das heißt ein Computersystem, das bis heute noch nicht existiert, aber von den meisten Systemen emuliert werden kann. Zu dem Spezifikation der Maschine zählen Prozessor, Speicher und Ein-/Ausgabe. An Betriebssystemdiensten stellt die Java VM Funktionen im Dateisystem und der Oberflächenprogrammierung bereit. Diese Spezifikation ist so gehalten, daß sie auf jeder

heute üblichen Plattform angepaßt werden kann. Mit einer entsprechenden Implementierung der Java VM kann also ein Java-Programm auf einem konkreten System ausgeführt werden. Dabei erscheinen Benutzeroberfläche und Dateisystem in der jeweiligen Form der Plattform.

Eine andere Möglichkeit stellt die Herstellung von Prozessoren, die direkt Java verstehen da, die also die Java VM hardwareseitig unterstützen. Hierin liegt ein großes Potential um die Geschwindigkeit von Java zu steigern. Jedoch muß man berücksichtigen, daß in der Spezifikation der Java VM auch Dienste, die von einem Betriebssystem bedient werden, mit enthalten sind.

2.2.2 Syntaktisch

Java hat Syntaktisch sehr große Ähnlichkeit mit C/C++, trotzdem gibt es einige Grundlegende Unterschiede, die soweit sie für diese Arbeit relevant sind hier angesprochen werden sollen.

2.2.2.1 Typen

Die in Java benutzten Typen lassen sich in zwei Kategorien aufteilen, die unterschiedlich gehandhabt werden. Zu den “Einfachen Datentypen” gehören die Zahlentypen, einzelne Zeichen und Wahrheitswerte. Sie werden stets als Wert übergeben. Dagegen stehen die “Referenzdatentypen” nämlich Zeichenketten, Felder und Objekte. Diese werden stets über eine Referenz angesprochen. Dabei kann diese Referenz, anders als in C/C++, nicht in andere Typen umgewandelt werden und nicht unmittelbar modifiziert werden. Damit wird vermieden, daß unterschiedliche Speicherdarstellungen zu Inkonsistenzen bei der Portierung führen würden.

Diese Typen können die Grundlage für eine Java-eigene MIB bilden, wobei die Serialisierung zur Übertragung zwischen Systemen bereits von Java vorgegeben wird.

2.2.2.2 Objektorientierung

Java ist von Grund auf objektorientiert, daher gibt es keine Prozeduren mehr, sondern nur noch Methoden, die einer Klasse angehören.

Ein Templatekonstrukt wie in C++ entfällt ebenfalls bei Java. Dafür lassen sich Klassen für den Typ “Object”, der Ahn für alle anderen Klassen ist, erstellen. Anders als bei einem Template kann dann während der Laufzeit entschieden werden, welche Typen in diesem Konstrukt abgelegt werden. Allerdings ist dann ein zusätzlicher Cast notwendig um die ursprünglichen Typen wiederzuerhalten.

Es besitzt keine Mehrfachvererbung, hat aber dafür die Möglichkeit zu einer Klasse Interfaces zu implementieren. Ein Interface ist ähnlich einer Klassendefinition, die nur aus abstrakten (virtuellen) Deklarationen besteht. Damit wird einer Klasse die ein Interface implementiert ein Funktionsumfang vorgegeben, die Methoden selber müssen aber in dieser Klasse geschrieben werden. Das entspricht einer Einfachimplementierungsvererbung und einer Mehrfachschnittstellenvererbung.

2.2.3 Nebenläufigkeit

Java ermöglicht es Nebenläufigkeit sehr einfach zu implementieren. Dazu existiert die Klasse „Thread“, die den Rahmen für einen einfachen Thread abgibt. Zusätzlich gibt es die Möglichkeit automatisch synchronisierte Variablen zu erstellen um einen wechselseitigen Ausschluß zu gewährleisten. Aber auch Methoden zum Warten und zur Notifikation sind verfügbar.

2.2.4 Softwareverteilung

In einem übersetzten Java-Programm wird jede Klasse in einer eigenen Datei mit entsprechendem Namen und Verzeichnis abgelegt. Zur Laufzeit liegen Klassendefinitionen und Kode in einem Objekt der Klasse „Class". Daraus können dann wiederum Objekte der jeweiligen Zielklasse erstellt oder statische Methoden und Werte angesprochen werden. Wird nun eine Klasse zum ersten mal benötigt, so lädt das Laufzeitsystem die entsprechende Datei und schreibt den Code in ein Objekt der Klasse „Class". Durch Erweiterung des „Classloaders" ließen sich Klassen von jeder beliebigen Quelle wie URLs oder Datenbanken laden. Hieraus ergeben sich neue Möglichkeiten der Softwareverteilung.

2.2.4.1 Klasse Class

Wie oben bereits erwähnt werden Klassen-, Interfacedefinitionen mit Code in Objekten der Klasse „Class" im Speicher gehalten. Aber auch die einfachen Datentypen und Felder sind als Objekte der Klasse „Class" verfügbar.

Mit den Methoden lassen sich Informationen über Art der Definition verwandte Klassen und Details der Implementierung abrufen. Es ist ebenfalls möglich die Instanzen einer Klasse zu erhalten und umgekehrt von diesen auf die Defnition zurückzugreifen.

2.2.4.2 Klasse Classloader

„Classloader" ist eine abstrakte Klasse, die soweit sie definiert und implementiert wird, den Mechanismus des Laufzeitsystems zum Laden von Klassen aus dem Dateisystem ersetzt. Ahnen von „Classloader" können Klassen aus beliebigen Quellen lesen und aus diesen dann ein Objekt von „Class" erstellen.

2.2.5 Oberflächenprogrammierung

Ähnlich dem Toolkit von TCL lassen sich auch in Java relativ einfach Benutzeroberflächen unabhängig von der grafischen Benutzeroberfläche und sogar dem Betriebssystem erstellen . Allerdings muß man bislang noch auf ein grafisches Werkzeug als Hilfe verzichten. Die Darstellung der jeweiligen Komponente findet dabei in der für die benutzte Plattform üblichen Form statt. Komplexere Elemente, wie Registerkarten und Hierarchische Browser sind allerdings noch nicht im Standardumfang enthalten, werden aber zum Teil mit dem Java Management API geliefert.

2.2.6 Netz

Java unterstützt TCP/IP Funktionen und Sockets auf sehr einfache Art und Weise, so daß man synchrone und auch asynchrone Kommunikation auf Schicht 4 direkt mit TCP/IP implementieren kann. Allerdings findet man im Standard-Java keine ISO/OSI Unterstützung. Für höhere Schichten über TCP/IP werden in zunehmendem Maße Pakete verfügbar.

2.2.7 Applets

Applets sind Java Programme, die in einer VM innerhalb eines Browsers oder eines Appletviewers ablaufen. Da der Programmcode von einem entfernten eventuell unsicherem System stammt, müssen hier besondere Vorkehrungen getroffen werden, um das eigene Gerät vor Schaden zu schützen.

Dabei sind Ressourcen wie Dateisystem für das Applet überhaupt nicht verfügbar. Dagegen hat das Applet sehr wohl Zugriff auf Rechenzeit, Benutzeroberfläche und Audioausgabe. Um ein Vortäuschen von Anwendungsfenstern (z.B. Passwortabfrage) zu vermeiden, sind alle Fenster, die ein Applet öffnet durch den Browser gekennzeichnet. Weiterhin hat ein Applet noch die Möglichkeit Netzwerkverbindungen zu dem Rechner aufzubauen, von dem es selber stammt. Server Sockets sind dabei überhaupt nicht möglich. Allerdings werden diese Sicherheitsmechanismen in den Browsern und Appletviewern sehr unterschiedlich gehandhabt und führen zu sehr unterschiedlichen und zum Teil undurchsichtigen Fehlern.

Um die Netzlast gering zu halten, besitzen die Browser bereits umfangreiche Standardklassen, die geladenen Applets zur Verfügung stehen. Weiterhin können die Grundklassen des Browsers auch nicht durch Applets überschrieben werden. Um Nebeneffekte zu vermeiden werden Applets vor der Ausführung geparsed um unzulässige und falsche Operationen zu erkennen.

Vor allem in Intranets hat sich diese äußerst restriktive Handhabung als übertrieben und auch sehr hinderlich erwiesen. Da man hier davon ausgehen kann, daß die Server innerhalb eines Unternehmens im Endeffekt von den selben Instanzen gemanagt werden wie die Klienten, kann man weitaus weniger Restriktionen ansetzen. Daher sollen sogenannte Trusted Server eingeführt werden, von denen Applets mit mehr Befugnissen geladen werden.

2.2.8 Ressourcenzugriff

Ein Zugriff auf Ressourcen ist in Java zunächst auf standardisierte Komponenten in einem allgemein verfügbaren Umfang beschränkt, um eine sichere Portierung zu gewährleisten. Dazu zählen Speicher, Rechenleistung, Konsole, Dateisystem, Netzwerk (TCP/UDP), Benutzeroberfläche, Grafik, Sound und Shell-Kommandos. Diese Ressourcen sind allerdings virtuell und werden erst von der Java VM auf reale abgebildet. Das stellt erst dann ein Problem dar, wenn eine Funktion einer Ressource nicht von der Java VM unterstützt wird, aber benutzt werden soll, oder tatsächlich Interesse an der Plattformspezifischen Ressource besteht. Letzteres ist bei der Überwachung durch Agenten interessant, da diese außer dem Zustand der Java VM auch den Zustand der darunterliegenden Hardware überwachen sollen. Hierzu ist dann eine Plattformabhängige Erweiterung zum Beispiel unter Einsatz von C/C++ notwendig. Sollte sich Java entsprechend durchsetzen, so kann man davon ausgehen, das solche Schnittstellen in Zukunft vom Hersteller einer Komponente geliefert werden. Sun entwickelte zu diesem Zweck das Java Native Interface (JNI) ([Sun96b]), das eine Programmierschnittstelle zu anderen Programmiersprachen darstellt. Dabei können zum Beispiel mit C/C++ Bibliotheken zusammengestellt werden, die ein Java Programm laden und benutzen kann. Zur Konvertierung der Java-Typen in native stellt JNI nativen Methoden Funktionen bereit. Außer den einfachen Datentypen werden hier auch Zeichenketten, Felder und Referenzen unterstützt. Weitere Funktionen gibt es für die Ausnahmebehandlung und Nebenläufigkeit. Auch können native Methoden über JNI eine Java VM starten. Das JNI fügt sich speziell in die Umgebung von C++ besonders gut ein.

2.2.9 Datenbankanbindung

Mit dem JDBC Package ([Sun96c]) stellt Sun für Java eine Schnittstelle für den Zugriff auf Datenbanksysteme zur Verfügung. Es wird in Java eine SQL-API implementiert, um

Anfragen zu formulieren und die SQL-Daten in Java-Programme zu übernehmen. Die API unterstützt dabei verschiedene Clients von Datenbanksystemen oder wird an ODBC adaptiert. Weiterhin gibt es die Möglichkeit spezielle Java-Internet-Datenbanken zu benutzen. Mit dieser API kann ein Java-Programm auf herkömmliche Datenbanksysteme zugreifen.

2.2.10 Remote Method Invocation (RMI)

RMI ([Sun96d]) ist eine Erweiterung von Java, die dem Remote Procedure Call (RPC) entspricht. Es kann hiermit über sogenannte Remote Referenzen auf entfernte Objekte zugegriffen und deren Methoden ausgeführt werden. Dabei ist diese Implementierung soweit Java-Konform, daß nur wenige konzeptionelle Zusätze und Erweiterungen und keine Änderungen an der Sprache notwendig sind. Auch die Abweichungen von der Java Semantik sind nur gering.

Der Zugriff auf ein entferntes Objekt oder die Ausführung einer entfernten Methode erfolgen synchron, bei einem Fehler in der Kommunikation wird ein Ausnahme erzeugt. Die Vorteile einer asynchronen Ausführung lassen sich nur über Umwege eines eigenen Threads innerhalb des aufrufenden Programms erreichen.

Damit eine Klasse als Server benutzt werden kann, muß sie ein spezielles Interface implementieren. Dazu wird ein Erweiterung des Interface “RemoteObject” erstellt, die alle entfernt verfügbaren Methoden definiert. Letztere müssen für den Fall eines Kommunikationsfehlers die Ausnahme “RemoteException” definieren. Diese wird im Fehlerfall automatisch durch RMI erzeugt. Um den Dienst zur Verfügung zu stellen, muß dem RMI-Dienst ein Port zugewiesen werden und die Klasse instanziiert werden und mit einem Namen versehen werden.

Der Klient erhält die Referenz über eine RMI-URL und kann diese von nun an wie eine lokale Referenz benutzen. Diese Referenz kann innerhalb oder übergreifend einer Java VM weitergegeben werden. Es ist zu beachten, das die Typidentifikation und ein Cast jeweils auf Seite des Servers stattfinden muß. RMI stellt entsprechende Methoden bereit.

Beim Aufruf werden Argumente und Parameter als Remote Referenz übergeben soweit es sich um entsprechende Remote Objekte handelt. Andere Objekte werden als Kopie übergeben. Zur Kodierung der Daten wird in Zukunft der Serialisierungsmechanismus von Java eingesetzt werden. Momentan lassen sich nur Klassen mit bestimmten Auflagen übergeben. Diese müssen vom Typ “final”(also eine Konstante) sein einen parameterlosen public Konstruktor haben und dürfen keine privaten, statischen oder nativen Eigenschaften haben. Ansonsten können über ein Interface eigene Kodierungen implementiert werden.

Bevor RMI entgültig zum Standardumfang von Java hinzugefügt wird, sind noch einige (vor allem syntaktische) Änderungen geplant. Asynchrone Elemente, wie sie etwa CORBA unterstützt, sind aber offensichtlich nicht geplant, was vor allem für die Verschickung von Ereignissen einen nicht unerheblichen Overhead bedeuten würde. Es spart Entwicklungsaufwand, daß eine Remote Object auch lokal genutzt werden kann, so daß die Implementierung eines Dienstes als entfernter oder lokaler, weitgehend einheitlich erfolgen kann, wobei natürlich der entfernte die härteren Vorgaben hat.

2.3 Zusammenfassung

In diesem Kapitel wurden mit HTTP, HTML und cgi-Skripten und mit dem java.awt in Applets zwei Alternativen zur Gestaltung von Benutzerdialogen dargestellt. Dabei ist die erste Variante auf die Steuerelemente von HTML-Formularen beschränkt. Die Aufbereitung

und Verarbeitung der Informationen geschieht dabei auf dem Server durch Links oder cgi-Skripten. Bei Applets können beliebige Oberflächen gestaltet werden und die Eingaben auch entsprechend verarbeitet werden, bevor sie zum Server oder auch an eine andere Stelle gelangen.

Um Managementszenarien, die nur auf Java basieren, in ihren Möglichkeiten einschätzen zu können wurden Typen und Klassen von Java und einige Pakete näher erläutert.

3 Teilmodelle, APIs, Protokolle

Um im Folgendem Funktionen und Schnittstellen beschreiben zu können, wird hier zunächst ein Modell für das integrierte Management aufgestellt. Die Nomenklatur ist zum Großteil an das Telecommunication Management Network (TMN) ([Bar97]) angelehnt, da diese am geeignetsten für dieses Modell erschien. Die Aufteilung der Funktionsblöcke, ihre weitere Granulierung in funktionale Elemente und die Lage der Referenzpunkte entsprechen dieser Arbeit.

Dazu werden zuerst die Funktionsblöcke wie auch in TMN festgelegt und beschrieben. Diese bestehen aus dem im Anschluß definierten funktionalen Elementen. Daraufhin werden die Schnittstellen (entsprechend den Referenzpunkten in TMN) kategorisiert und auf ihre Aufgabe hin beschrieben.

3.1 Die Funktionsblöcke im Managementmodell

Zunächst werden nun die Aufgaben der einzelnen Funktionen erklärt. Außerdem werden einige Anforderungen an diese Funktionen beschrieben, die aus dem heutigen Management hervorgehen und als bindend betrachtet werden. Einige Anforderungen in Bezug auf die Aufteilung der Funktionen auf System werden hier bereits erwähnt, soweit sie als Prämisse betrachtet werden können.

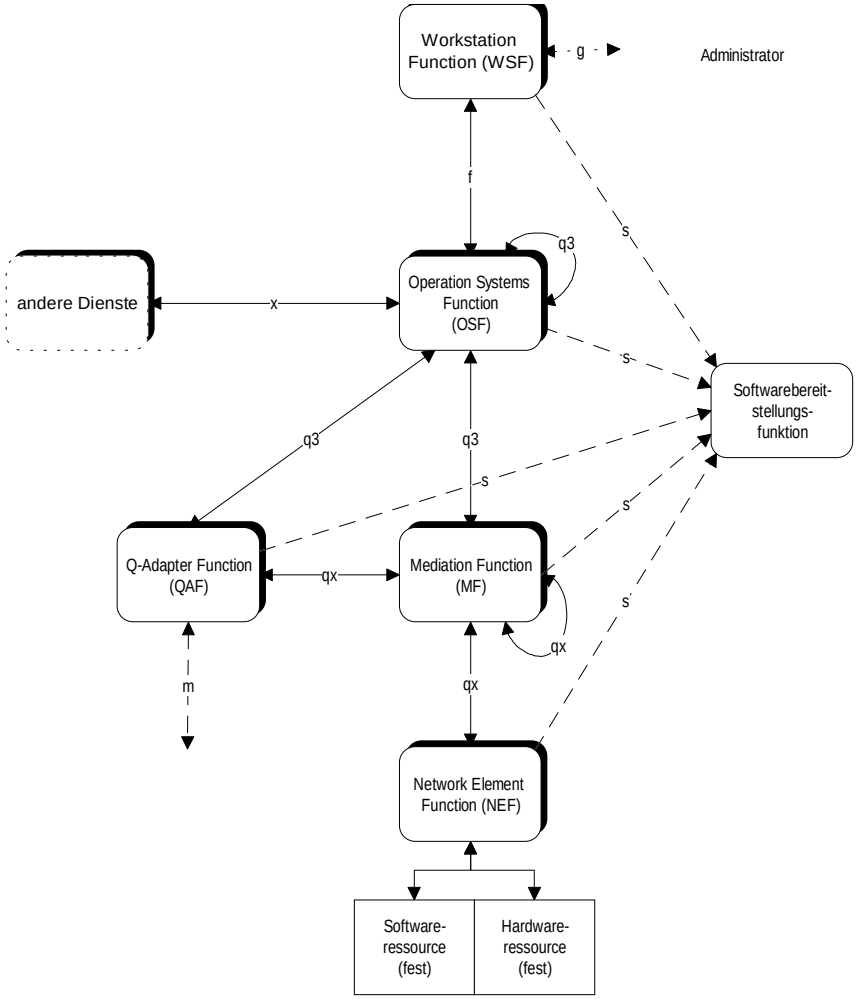

Abbildung 3.1-i: Funktionsblöcke und Referenzpunkte

Funktionsblock	Beschreibung
WSF Workstation Funktion	Darstellung und Eingabe der für den Nutzer relevanten Managementinformationen
OSF Operation System Function	Netzführung: Verarbeitung von Management-information, Beobachtung, Koordinierung, Kontrolle
MF Mediation Function	Manipulation von Informationen: Speicherung, Anpassung, Filterung; ermöglicht angepaßte Sicht für OSF
NEF Network Element Function	Kontrolle über Ressource
QAF Q Adaptor Function	Schnittstelle zu nicht konformen Management-komponenten

Tabelle 3.1-i: Übersicht Funktionsblöcke

3.1.1 Workstation Function block (WSF)

Die *Workstation Function* ist mit der Oberfläche, an der der Administrator arbeitet gleichzusetzen. Die Aufbereitung der Oberfläche und die Verarbeitung der Benutzereingaben ist hier noch nicht mit inbegriffen. Systeme, die eine WSF implementieren, werden Workstation (WS) genannt.

3.1.1.1 Anforderungen

Der Ort, an dem die WSF verfügbar ist, sollte variabel vom Administrator vorgegeben werden können, so daß dieser innerhalb eines Netzes an möglichst vielen Stellen diese Funktion benutzen kann. Ebenso sollte die WSF an mehreren Stellen gleichzeitig verfügbar sein, damit mehrere Administratoren parallel arbeiten können.

3.1.1.2 Aufgaben

Eine Aufgabe der Oberfläche ist die Überwachung von Managementinformationen durch den Administrator. Dazu sollen Daten in den verschiedensten Formen und aus diversen Quellen innerhalb des Szenarios dargestellt werden können. Die Darstellung sollte dabei aktuell und, wenn sinnvoll, grafisch aufbereitet sein. Der Benutzer soll eine einfache Möglichkeit haben, in den dargestellten Daten zu navigieren.

Außer der Anzeige von Daten sollen auch Daten von der Oberfläche aus eingegeben werden. Dies können zum Teil alphanumerische Informationen wie Parameterwerte oder eventuell Programmelemente sein. Allerdings werden bei topologischen Informationen evtl. auch grafische Eingabemöglichkeiten benötigt.

Für die Aufbereitung und die Weiterverarbeitung von den Informationen der WSF ist die im Managementanwendung der OSF zuständig.

3.1.2 Operating System Function block (OSF)

Die *Operating System Function* umfaßt die eigentliche Netzführungsfunktion. Hier wird das Managementsystem mit seinen Ressourcen kontrolliert, koordiniert und gesteuert. Auch die Aufbereitung der Daten zur Darstellung für den Benutzer kann eine Aufgabe der OSF sein.

Die OSF wird auf einem Opration System (OS) ausgeführt.

3.1.3 Softwarebereitstellungsfunktion

Abweichend von klassischen Managementszenarien, soll in diesem Modell ein einheitlicher Softwaredienst benutzt werden um die Funktionen oder später die Komponenten mit der erforderlichen Software zu bedienen. Dieser Dienst soll von allen Komponenten aus erreichbar sein, damit diese selbständig Programme oder Programmteile abrufen können. Damit soll erreicht werden, daß die Managementsoftware überall im System auf dem neusten Stand ist und die Software dynamisch an die jeweiligen Gegebenheiten angepaßt werden kann. So kann man damit auch Ressourcen auf den Systemen des Managements sparen indem nur die Software dort läuft, die momentan auch benötigt wird. Außerdem kann man die Software an Situationen anpassen und um Funktionen erweitern, die zur Entwicklung noch nicht berücksichtigt wurden.

3.1.4 Mediation Function block (MF)

Die Aufgabe der *Mediation Function* ist es Daten, die zwischen OSF und NEF oder QAF ausgetauscht werden zu manipulieren. Darunter fallen jegliche Arten der Filterung und der Zusammenfassung. Damit können Eigenschaften oder auch Gruppen von NEFs zusammen von der OSF kontrolliert werden.

Komponenten, die eine MF ausführen heißen Mediation Devices (MD).

3.1.5 Network Element Function block (NEF)

Die *Network Element Function* stellt die Einheit dar, die die direkte Kontrolle über die jeweiligen Soft- und Hardwareressourcen, das Network Element (NE) hat. Sie ist durch die verwalteten Ressourcen örtlich festgelegt und tritt im allgemeinen häufiger innerhalb eines Managementsystems auf. Die NEF entspricht einem minimalen Agenten ohne eigene Intelligenz.

3.1.6 Q Adaptor Function block (QAF)

Entsprechen der *Q Adaptor Function* in TMN wird auch in diesem Modell eine Schnittstelle zu anderen bzw. alten Managementstandard betrachtet. Speziell für die Einführung einer neuen Architektur oder von Teilen einer solchen ist die Kompatibilität zu existierenden Standards und Ressourcen wichtig.

Entsprechend den obigen Festlegungen wird diese Funktion auf einem Q Adaptor (QA) ausgeführt.

3.1.7 andere Dienste

Da das Managementsystem (wie schon bei der Automatisierung erwähnt) nicht in der Lage ist, alle für das Management relevanten Informationen aus sich selber zu beziehen, ist es notwendig weitere Datenquellen zu benutzen. Dabei ist die Eingabe durch die Administratoren nur eine unbefriedigende Lösung zumal fast alle Notwendigen Daten in der einen oder anderen Form bereits im Rechnernetz gespeichert sind. Wiederum verfügt das Managementsystem über Informationen, die in anderen Programmen benötigt werden. Ein sehr großer und zunehmend wichtiger Bereich ist die Abrechnung, da der Anteil der Kosten an der Arbeit in zunehmendem Maße von der EDV abhängt, oder sich durch diese detaillierter überwachen läßt. Daher muß eine Managementsystem in Zukunft in der Lage sein, mit anderen Anwendungen Daten auszutauschen. Die Zusammenarbeit der

verschiedenen Anwendungen sollte dabei weitgehend automatisch erfolgen, so daß zum Beispiel, bei der Eingabe eines neuen Mitarbeiters durch dir Personalabteilung, automatisch ein Account im Netz für diesen angelegt wird. Zu beachten ist dabei allerdings, daß nicht unbedingt alle erforderlichen Schritte durch die EDV automatisch ausgeführt werden können. So wird die Einrichtung eines Arbeitsplatzes noch auf menschliche Ressourcen zurückgreifen müssen.

Managementanwendung 3: Agent zu MA 2, Manager zu Agent2

3.2 Funktionale Elemente

Die Funktionsblöcke im vorangegangenem Abschnitt bauen sich aus den Funktionalen Elementen auf. Diese ermöglichen eine detailliertere Betrachtung der Modells, vor allem im Hinblick auf die später folgende Abbildung auf konkrete Systeme.

Management Application Function	(MAF)
Management Information Base	(MIB)
Information Conversion Function	(ICF)
Presentation Function	(PF)
Human Machine Adaption	(HMA)
Message Communication Function	(MCF)
Data Communication Function	(DCF)

Tabelle 3.2-i: Überischt Funktionale Elemente

3.2.1 Management Application Function (MAF)

Durch die Management Application Function werden die Managementdienste und Funktionen ausgeführt. Automatisierung, Überwachung und Fehlerkorrektur sollen von ihr übernommen werden. Abhängig von dem Funktionsblock, der sie enthält, bekommt sie dessen Namen als Präfix (OSF-MAF, MF-MAF, NEF-MAF)

- Automatisierung
- Überwachung
- Fehlerverhalten
- Anwendung in OSF, MF, NEF

3.2.2 Management Information Base (MIB)

Die *Management Information Base* bestimmt das Informationsmodell der Managementumgebung. Um die speziellen Eigenschaften von Java auszunutzen, ist hier ein objektorientiertes besser als ein hierarchisches Modell. Dazu besteht die MIB nicht nur aus hierarchisch organisierten Eigenschaften, entsprechend einem hierarchischen Modell, sondern es werden noch zusätzlich Vererbung und Methoden eingeführt. Damit kann in der

MIB definiert werden, welche Operationen auf ein Objekt angewendet werden können. Ebenfalls wird an dieser Stelle gleich definiert welche Ereignisse als Nebeneffekt einer Methode auftreten können.

Im folgenden wird nur oberflächlich auf die Definition einer MIB eingegangen, da dieser von den gemanagten Ressourcen und nicht dem Management abhängt.

Zum einen gibt es Informationen über die Soft- und Hardwareressourcen, die zur Kontrolle dieser und zum Teil zur Steuerung genutzt werden. Zu den über das System nicht unmittelbar steuerbaren Daten gehören Umgebungsinformationen, wie Stromversorgung oder Temperatur. Im weitesten Sinn strebt man danach alle Informationen, die über eine Ressource verfügbar gemacht werden können, und alle Steuerungen, die irgendwie daran ausgeführt werden können, in der MIB zu implementieren. Das ist die eigentliche Managementaufgabe.

Dann muß das Managementsystem selber wiederum gemanagt werden können. Hierzu muß der Zustand der Einheiten kontrolliert und gesteuert werden können. Auch die Verbindungen, die zwischen den Komponenten des Managements bestehen, und ihre Hierarchisierung soll mit eingehen. Auch ein Management, der Programme und Programmteile der Elemente kann hier miteingebracht werden.

Weiterhin soll eine Gruppierung von Ressourcen ermöglicht und unterstützt werden um eine einfachere Verwaltung zu ermöglichen. Dazu können Sammlungen von Ressourcen dienen auf die einheitliche Operationen ausgeführt werden können. Die Einteilung in Ressourcenklassen kann im Informationsmodell als Klassenhierarchie auftreten. Dabei bietet es sich an die MIB als Java-Klassen zu implementieren. Das ist vor allem bei einer reinen Java-Implementierung des Managements interessant.

Auch bei zunehmender Flexibilität der Software, wie sie später noch beschrieben wird, kann eine MIB zwar während des Betriebs erweitert werden, aber ein Wechsel des Informationsmodells oder eine Korrektur von Implementierungsfehlern wird nach wie vor mit Schwierigkeiten verbunden sein.

- Informationsmodell
- Modellierung des gemanagten Systems
- Hard-, Softwareressourcen
- andere Ressourcen (Service, Umfeld)
- Hierarchisches Modell
- Beziehungen zwischen Instanzen

3.2.3 Information Conversion Function (ICF)

Mit der *Information Conversion Function* werden Daten umgewandelt. Dieses Umwandeln kann einen Wechsel des Informationsmodell oder ein Zusammenfassen der Informationen darstellen. Daher ist die ICF ein fester Bestandteil der MF. Außerdem können MIBs von fremden Modellen angepaßt werden. Diese Aufgabe kommt der ICF innerhalb eines QAF zu einem anderen Standard zu.

- Zusammenfassung von Daten
- Adaption von Informationsmodellen

3.2.4 Presentation Function (PF)

Die *Presentation Function* ist auf die WSF festgelegt. Sie umfaßt ausschließlich die unmittelbare Umsetzung von bereits für Benutzer aufbereiteten alphanumerischen und grafischen Daten, in für diesen Erkennbare Signale. In heutigen Managementsystemen wird diese Funktion von der X-Window-System übernommen. Im Verlauf dieser Arbeit wird der Einsatz von WWW-Browse für diesen Zweck untersucht werden.

- Darstellung der bereits aufbereiteten Daten
- Eingabe von Informationen

3.2.5 Human Machine Adaption (HMA)

Die *Human Machine Adaption* ist für inhaltliche Aufbereitung der im Managementsystem befindlichen Informationen in für den Benutzer leicht verständliche Form. Darunter fällt die Umsetzung von Kodierungen in lesbare Formen oder das Hinzufügen von Hilfetexten. Außerdem findet hier die Authentifizierung und Autorisierung der Administratoren statt. Die eigentliche Darstellung, also die Schnittstelle zum Benutzer wird von der PF übernommen. Anders herum werden vom Administrator über die PF eingehende Informationen für das Managementsystem interpretiert überprüft und gegebenenfalls umgesetzt.

- Aufbereitung der Daten für die Darstellung
- Verarbeitung der Benutzereingaben

3.2.6 Message Communication Function (MCF)

Dieses funktionale Element ermöglicht die Kommunikation zwischen Funktionsblöcken und muß daher in jedem enthalten sein, der eine Schnittstelle besitzt. Im weiteren Verlauf werden hier zum Beispiel APIs für Kommunikationsmechanismen substituiert.

- Kommunikation zwischen Funktionsblöcken
- API oder Anpassung an Protokoll

3.2.7 Data Communication Function (DCF)

Der Transportmechanismus, der die Information zwischen den Funktionsblöcken austauscht, heißt *Data Communication Function*. Das ist der Dienst, der von der Message Communication Function in den Funktionsblöcken benutzt wird.

- Transportmechanismus
- zwischen den Funktionsblöcken

3.3 Verteilung der funktionalen Elemente auf Funktionsblöcke

Hier soll eine Übersicht über die möglichen Aufteilungen der funktionalen Elemente auf die Funktionsblöcke gegeben werden ([Bar97]). Die Message Communication Function muß dabei auf jeder Komponente enthalten sein und wird daher nicht weiter erwähnt. Die Data Communication Function ist der Transportmechanismus zwischen den Funktionsblöcken und daher in keinem enthalten.

Funktionsblock	Funktionale Komponente	
	notwendig	optional
OSF	OSF-MAF	MIB, HMA
OSF (untergeordnet)	MIB, OSF-MAF, ICF	HMA
WSF	PF	
NEF	MIB, NEF-MAF	
MF	ICF	MIB, MF-MAF, HMA
QAF	ICF	MIB, QAF-MAF

Tabelle 3.3-i: Funktionsblöcke und Funktionale Komponenten [Bar97]

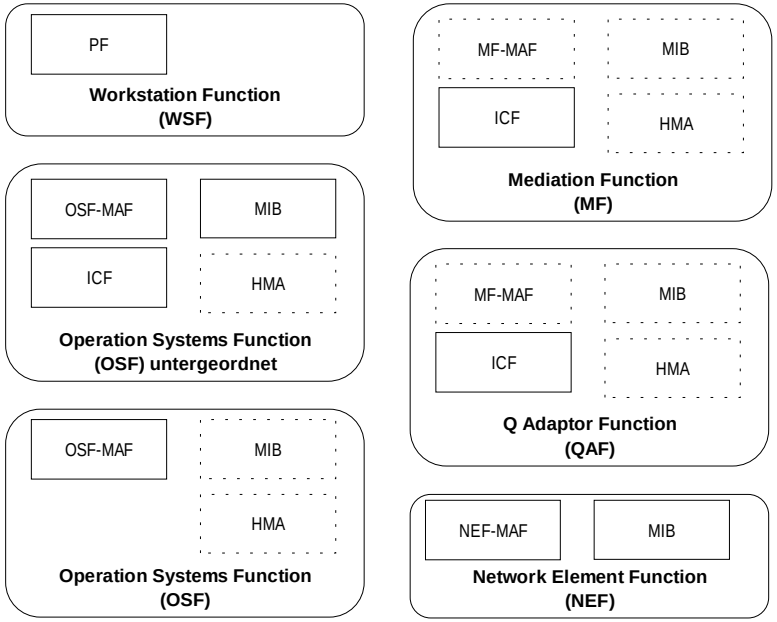

Abbildung 3.3-i: Verteilung der funktionalen Elemente auf Funktionsblöcke

3.4 Referenzpunkte

TMN kategorisiert im Kommunikationsmodell die Punkte des Informationsaustausches als Referenzpunkte. Da dies eine starke Vereinfachung der möglichen Schnittstellen ohne Beschränkung der Allgemeinheit darstellt, wird sie hier ebenfalls gemacht.

Referenzpunkt	Aufgabe
g	Benutzer - WSF
m	QAF - nicht konforme Ressourcen Integration anderer Standards
f	WSF - OSF Übertragung von Eingaben, zur Darstellung aufbereiteter Daten
x	OSF - andere Dienste Austausch von Daten mit anderen Systemen
q	OSF - NEF - MF - QAF Austausch von Managementdaten zwischen den Komponenten
s	SBF - Abruf von Softwarekomponenten des Managementsystems

Tabelle 3.4-i: Übersicht Referenzpunkte

3.4.1 g-Referenzpunkt

Ein *g-Referenzpunkt* ist die Schnittstelle zwischen Benutzer und der WSF. Diese wird hier nicht weiter behandelt.

3.4.2 m-Referenzpunkt

Zwischen anderen Ressourcen und einem QAF steht ein *m-Referenzpunkt*. Damit sollen andere Standards in das Modell integriert werden, was für die Entscheidung zur Einführung eines neuen Managementmodells ausschlaggebend sein kann. Dieser könnte zum Beispiel ein SNMP-Protokoll darstellen.

3.4.3 f-Referenzpunkt

F-Referenzpunkte liegen zwischen WSF und OSF oder MF. Hier werden Daten in einer Form, die unmittelbar zur die Darstellung dient oder die von der Eingabe entnommen wurden, übertragen.

3.4.4 x-Referenzpunkt

Der *x-Referenzpunkt* dient zum Austausch der Daten mit anderen Diensten oder Anwendungen außerhalb des Managementszenarios. Damit lassen sich Informationen mit Inventardatenbanken oder Abrechnungssystemen austauschen.

3.4.5 q-Referenzpunkte

Über die *q-Referenzpunkte* wird die Kommunikation innerhalb der Komponenten, die den Managementdienst und Ressourcenbereitstellung übernehmen, abgewickelt. Sie sind nochmals unterteilt in die q3-Referenzpunkte als Anschluß für komplexe Einrichtungen, die eine OSF mit einer MF, NEF, QAF oder wieder einer OSF verbinden, und die vereinfachten qx-Referenzpunkte zum Anschluß einfacher Übertragungs- und Vermittlungseinrichtungen, die zwischen einer MF und einer NEF, QAF oder anderen MF stehen.

Über q-Referenzpunkte werden Managementinformationen entsprechend der MIB abgefragt oder manipuliert. Weiterhin müssen Benachrichtigungen über Ereignisse weitergegeben werden können.

3.4.6 s-Referenzpunkt

Zusätzlich zu den Referenzpunkten in TMN wird hier noch der *s-Referenzpunkt* definiert. Er steht zwischen der Softwareverteilungsfunktion und theoretisch jedem anderen Funktionsblock, der sich selber über diesen Mechanismus anpassen kann. Damit wird die Möglichkeit eingeräumt, die in der MIB geforderte Kontrolle über die Software der Managementkomponenten über q-Referenzpunkte zu erhalten, die eigentlichen Programme über einen eigenen Referenzpunkt zu beziehen. Dabei wird über den s-Referenzpunkt Software nur auf Anfrage durch die Komponenten bezogen. Das heißt letztere wissen durch Informationen, die über einen q-Refrenzpunkt bezogen wurden, welche Programme oder Programmteile sie über den s-Referenzpunkt beziehen wollen.

Es wäre auch denkbar den q- und den s-Referenzpunkt in einer Schnittstelle zu integrieren.

3.5 Schnittstellen

Dieser Abschnitt betrachtet die einzelnen Schnittstellen innerhalb des Funktionenmodells. Die Schnittstellen werden aus den Referenzpunkten hergeleitet. Sie werden nun spezifiziert und auf Kommunikationsprotokolle und APIs abgebildet. Daraus ergeben sich auch sinnvolle und notwendige Einschränkungen für die Abbildung der Funktionsblöcke und funktionalen Elementen auf Systeme, soweit diese nicht schon im vorherigen Abschnitt angegeben wurden.

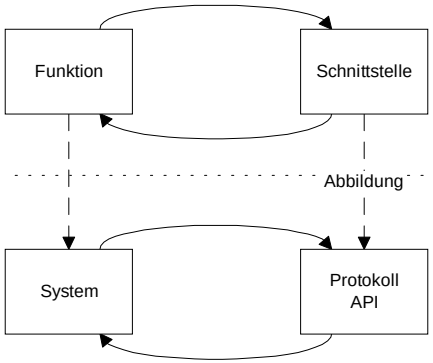

Abbildung 3.5-i: Abbildung von Funktionen und Schnittstellen auf Systeme und Protokolle

3.5.1 f-Schnittstelle

Für die f-Schnittstelle wird bei existierenden Managementanwendungen wie OpenView zumeist das X-Window-System benutzt. Das bedeutet beinahe schon eine Festlegung auf Unix-Systeme, oder aber es entsteht das Problem X-Windows auf anderen Systemen einzusetzen. Eine Lösung mit Komponenten, die in Zukunft auf jedem System verfügbar sein werden, wäre eher anzustreben, als X-Windows für nur wenige Einsatzbereiche auf den Systemen zu installieren.

Protokoll und API für die f-Schnittstelle:

- Hypertext Transfer Protocol
- Java Abstract Windowing Toolkit

3.5.1.1 Hypertext Transfer Protocol (HTTP)

Wird die WSF von einem WWW-Browser ohne den Einsatz von Applets wahrgenommen, so kann HTTP als Protokoll für die f-Schnittstelle benutzt werden. Auf Applets zu verzichten hat den Vorteil, daß sehr wenig Ressourcen der WS verbraucht werden. Außerdem wird eine Implementierung mit Applets im Allgemeinen ein zweites Protokoll über das Netz fahren, so daß dies bei Firewalls zusätzlich beachtet werden muß. Die Message Communication Function wird auf der einen Seite vom Browser, auf der anderen vom HTTP-Server übernommen. Die Human Machine Adaption kann entweder eine API oder eine Skriptenunterstützung des Servers sein. Letzteres könnte mit einem herkömmlichen WWW-Server mit cgi-Skripten realisiert werden. Die vorige Lösung würde einen WWW-Server mit Erweiterungsmöglichkeiten (zum Beispiel ein Java WWW-Server) oder aber eine Neuimplementierung des Servers erfordern.

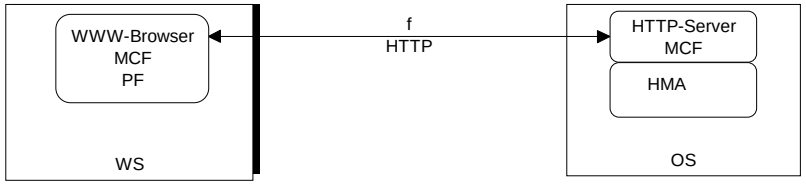

Abbildung 3.5-ii: HTTP als f-Schnittstelle

Bei der Verwendung von HTTP in Verbindung mit einem Browser und einem WWW-Server ergeben sich folgende Möglichkeiten für die WSF: Die Ein-, Ausgabemöglichkeiten werden zum einen durch die Fähigkeiten des Browsers beschränkt. Das, heißt dargestellt werden können Dokumente im HTML-Format, einfacher Text und diverse Grafikformate. Wobei bewegte Diagramme wie zum Beispiel Lastdiagramme nur über den Umweg eines regelmäßigen Reloads oder Erweiterungen für Animationen möglich sind, was einen große Overhead verursacht. Theoretisch besteht die Möglichkeit die meisten Browser durch sogenannte Plugins auf weitere Datenformate zu erweitern, allerdings würde man hier einen großen Teil der Plattformunabhängigkeit und der Einsparung an Installationsaufwand verlieren, die gerade die Vorzüge dieser Variante darstellen.

Bei der Eingabe muß man sich wiederum auf alphanumerische Daten und einfache grafische Auswahl beschränken. Eine grafische Manipulation, wie man sie etwa bei Maps einsetzen möchte, läßt sich hier nicht realisieren. Bei den heutigen Anforderungen ergibt sich daraus, daß diese Variante nur als Zusatz Möglichkeiten, aber nicht als alleinige Lösung, befriedigend ist.

Der Aufwand, der hier bei der WS gespart wird, muß nun in der OS bewältigt werden. Der HTTP-Server muß die Dokumente, die im Browser dargestellt werden, erstellen. Hierbei sind diese in drei Gruppen zu unterteilen: die erste Gruppe sind statische Dokumente, die sich außer bei konzeptionellen Änderungen am Managementsystem nie ändern. Dazu gehören die Menüstruktur und die MIB Ein-/Ausgabe.

Weiterhin gibt es Dokumente, die nur selten oder bei bestimmten Ereignissen angepaßt werden müssen. So muß eine am Netz neu angeschlossen Ressource in die Managementseiten eingetragen werden. Auch Maps werden sich eher vergleichsweise selten ändern.

Die dritte Gruppe stellt Daten, die sich beständig ändern, dar. Hierzu gehören die Zustände der NEs, Lastinformationen und Traps. Sie müssen im Allgemeinen bei jedem Zugriff auf den HTTP-Server erneuert werden.

Zur Erzeugung von dynamischen Seiten lassen sich WWW-Server mit Skriptenunterstützung einsetzen. Dabei würde ein Programm auf dem Server die notwendigen Informationen sammeln und daraus die entsprechenden Daten zur Übertragung an die WS zusammenstellen. Bei dieser Lösung kann man sich eines normalen WWW-Servers bedienen, der einem die Funktionalität für HTTP implementiert und somit eine MCF erfüllt. Die eigentliche Human Machine Adaption wird durch die Skripten erfüllt. Zusätzlich bietet diese Variante an die Softwareverteilungsfunktion ebenfalls von diesem WWW-Server erfüllen zu lassen.

Eine andere Möglichkeit ist die Implemtierung eines eigenen HTTP-Servers, wobei der Umweg über die Standard Ein-/Ausgabe vermieden wird, aber kein Gewinn bei der Nutzung

von HTTP zu erwarten ist. Diese Variante alleine wird wohl daher kaum den Mehraufwand rechtfertigen.

Die Abbildung des f-Referenzpunktes auf eine HTTP-Schnittstelle ist eine einfache Variante die aber die erwünschten Möglichkeiten einer WSF einschränkt. Weiterhin darf nicht außer acht gelassen werden, daß HTTP nur Dokumente als ganzes Überträgt und somit viel redundanter Verkehr entsteht.

3.5.1.2 Abstract Windowing Toolkit (java.awt)

Benutzt man eine Java Applikation oder ein Java Applet, so wird aus dem f-Referenzpunkt die API, die das Abstract Windowing Toolkit (java.awt) oder die erweiterten grafischen Funktionen die das JMAPI zur Verfügung stellt.

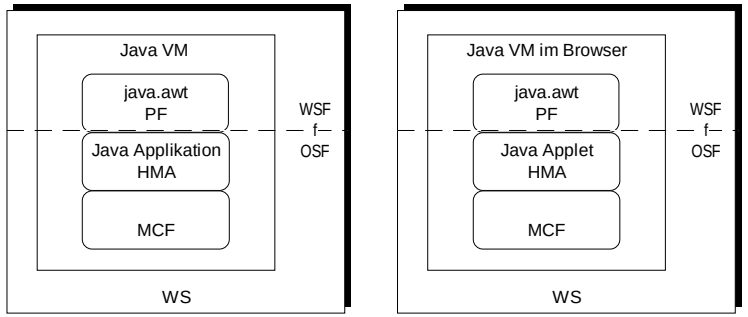

Abbildung 3.5-iii: java.awt API als f-Schnittstelle

Damit übernimmt das Abstract Windowing Toolkit die Presentation Function. Die Human Machine Adaption wird von der Java-Applikation oder dem Applet implementiert. Darunter befindet sich die Message Communication Function. Speziell im Fall eines Applets unterliegt die MCF Einschränkungen, da das Applet nur mit seinem Ursprungsserver kommunizieren kann, es sei denn es handelt sich um einen als sicher deklarierten Server.

Mit den Oberflächenfunktionen von Java lassen sich, anders als bei HTTP, beliebige grafische Benutzeroberflächen erstellen. Die Lösung mittels eines Applets läßt dabei das Problem der Softwareverteilung zum großen Teil vom WWW-Server und Browser lösen.

Bezüglich der Netzlast kann nach dem Laden der Anwendung ein besseres Ergebnis erzielt werden, da Informationen in einem kompakteren Format und selektiver als im folgendem Ansatz übertragen werden können. Allerdings wird durch das Laden der Software ein größerer Overhead zur Nutzung der WSF verursacht.

3.5.2 q-Schnittstellen

Bestehende q-Schnittstellen sind zum Beispiel SNMP oder DMI. Die hier betrachteten Schnittstellen sind asymmetrisch für eine Kommunikationsbeziehung. Das ist in ohne Einschränkung für die Allgemeinheit einsetzbar, insofern die Rollen zwischen den Kommunikationspartnern je nach Anforderung wechseln können.

Protokolle für die q-Schnittstellen:

- Remote Method Invocation
- Hypertext Transfer Protocol

3.5.2.1 Remote Method Invocation (RMI)

In Zusammenhang mit einer in Java implementierten MIB stellt RMI eine Möglichkeit als Schnittstelle zur Verfügung, die sowohl als Protokoll, wie auch als API ohne Effizienzverlust eingesetzt werden kann. Letzteres ist in sofern interessant, da die funktionalen Komponenten eine von den Aufgaben abhängige flexible Aufteilung auf Systeme ermöglichen.

Die Verwendung von RMI ermöglicht dem Entwickler eine weitgehende Unabhängigkeit von der Kodierung des Protokolls, da er sich in einer reinen Java-Umgebung befindet und keine Kodierung vornehmen muß. Der Aufwand zur Serialisierung wird jedoch im Allgemeinen bleiben, da der automatischen Serialisierung Einschränkungen auferlegt sind, die nicht in allen Fällen zu erfüllen sind. Außerdem müssen alle Managementkomponenten, die nicht über eine QAF angeschlossen werden ein Java VM besitzen und in Java implementiert sein oder ein Gateway bieten.

Die Operationen, die über diese Schnittstelle realisiert werden sollen, sind das Abrufen und Verändern von Werten der MIB und das Versenden beziehungsweise Empfangen von Ereignissen.

Dazu muß für eine in Java serialisierbar implementierte MIB ein Interface entwickelt werden, daß den Zugriff auf alle Werte der MIB realisiert. Das liegt daran, daß auf der Klientenseite nur ein Remote Interface aber kein Remote Object verfügbar ist, so daß die Eigenschaften des Remote Objects soweit sie keine Konstanten sind, nur über die Methoden des Interfaces erreichbar sind. An dieser Stelle wäre ein Remote Object für den Klient, daß automatisch den Zugriff auf die Eigenschaften umsetzt wünschenswert. Der Name Remote Method Invocation sagt jedoch schon aus, daß es sich um entfernte Dienste und nicht Objekte handelt. Um nicht für jede Eigenschaft zwei Methoden (get, set) schreiben zu müssen, wäre eine Erweiterung wünschenswert.

Bei der Kontrolle und Steuerung nimmt die im Management übergeordnete (also die kontrollierende) Instanz die Funktion als RMI Client wahr. Bei der Notifikation ruft die Ressource eine Methode einer übergeordneten, für das Eventhandling bestimmten, Instanz auf, die das Ereignis entgegennimmt und weiterleitet oder verarbeitet. Im Gegensatz zu Traps, wie sie in SNMP verwendet werden und die ohne Bestätigung an den Trap Server geschickt werden, entsteht hier ein höherer Aufwand bei der Kommunikation, vor allem aber wird das Netz stärker bealsted. Will man hier Verkehr verhindern, so muß man auf ein herkömmliches Protokoll für Ereignisversand zurückgreifen oder auf UDP aufbauen.

3.5.2.2 HTTP

Zu der asymmetrischen Kommunikation kommt bei HTTP noch das Request-Response-Paradigma hinzu. Das bedeutet das jede Anfrage neu aufsetzen muß außer man verwendet den Umweg über Cookies. Die Methoden GET und PUT entsprechen dabei semantisch den gewünschten Operationen zur Kontrolle und Steuerung. Die entsprechenden Parameter für die Anfrage (also der gewünschte Wert) müssen dabei im Header oder in dem URI kodiert werden. Diese beiden Alternativen sind semantisch noch mit dem HTTP Protokoll vereinbar und garantieren einen reibungslosen Transport über Proxies. Auch die Daten müssen kodiert werden. Werden die Daten als Text oder HTML Entity übertragen und ist die gewünschte

MIB-Eigenschaft im URI kodiert, so wäre es möglich Ressourcen über einen Browser direkt zu kontrollieren. Dadurch würden die q- und die f-Schnittstelle zusammenfallen. Bei der Kodierung der Daten ist zu beachten, daß sie dann sowohl im Browser für Menschen, als auch für andere Komponenten des Systems lesbar sein müssen und außerdem das Netz nicht stark belasten sollten. Da diese Ziele im Konflikt stehen, ist es sinnvoll, die Lesbarkeit durch den Benutzer auf ein Minimum zu reduzieren. Für die Fehlersuche wird diese Fähigkeit, auch wenn sie nicht komfortabel ist durchaus interessant sein. Während man beim Umsetzen der Daten in die HTTP Entity weitgehend freie Hand hat, ist es wünschenswert, daß der URI syntaktisch und semantisch konform und intuitiv benutzt wird. Eine Möglichkeit ist es, an den Hostnamen mit dem Managementport direkt die MIB-Struktur entsprechend einer Verzeichnisstruktur anzuhängen. Ein Nachteil ist, daß dabei immer nur eine Eigenschaft pro Dialog geändert werden kann. Eine andere Möglichkeit stellt die Kodierung in den Parametern des URIs dar.

Ein Einsatz von einem neuen Datenformat, das serialisierte Java-Objekte enthält, ist ebenfalls denkbar. Dabei würde man sich die Fähigkeiten von HTTP Operationen und die Transportmechanismen für HTTP innerhalb des Netzes (z.B. Firewalling) zu Nutze machen, jedoch ist das enthaltene Format, nur von entsprechenden Java-Programmen lesbar und nicht mehr durch Browser darstellbar.

Man kann HTTP durchaus für die q-Schnittstelle einsetzen, allerdings wird klar, das es nicht für solche Zwecke entwickelt wird. So wird man bei der Nutzung von HTTP immer im Konflikt zwischen semantisch korrektem Einsatz oder Erweiterung und einem effizienten Netzzugriff stehen.

3.5.3 s -Schnittstelle

Die Softwareschnittstelle soll Programme oder Teile davon transportieren. Weiterhin sind Informationen zur Versionsverwaltung wünschenswert. Die Realisierung dieser Schnittstelle wird hier in Bezug auf Java als Implementierungssprache betrachtet, da durch Java eine neue Situation entsteht. Erstmal kann ein Java Programm im Bytecode auf jeder Plattform mit Java VM eingesetzt werden und außerdem werden die Klassen in jeweils eigenen Dateien gespeichert, was den Austausch von Softwareteilen vereinfacht. Auch ein Austausch von Daten, die in Klassendefinitionen abgelegt sind, ist in dieser Schnittstelle impliziert. Ohne daß sich der Entwickler darüber Gedanken machen muß, wird eine Kapselung der einzelnen Programmteile erzeugt. Damit läßt sich der Zugriff auf Software auf einen Dateizugriff mit Versioninformation reduzieren.

Bei der Benutzung dieser Schnittstelle ist zu beachten, daß Java keine Mechanismen zum Austausch von Klassen während einem Programmlauf hat. Es wäre jedoch denkbar, daß ein Programm eine eigene Versionsüberwachung implementiert, die dann dafür sorgt, Inkarnationen von veralteten Klassen entfernt werden, daraufhin das entsprechende Objekt der Klasse Class zerstört und neu geladen wird. Das bedeutet allerdings, daß die Operationen, die von der erneuerten Klasse ausgeführt wurden, beendet und vorübergehend ausgesetzt werden.

Protokolle für die s-Schnittstelle:

- Network File System
- File Transfer Protocol

- Hypertext Transfer Protocol

3.5.3.1 Network File System (NFS)

Die Verwendung von NFS unterstützt die Softwareverteilung bei der Verwendung von Java Applikationen. Alle Komponenten, die diesen Dienst benutzen müssen am NFS angebunden sein und die Java VM muß das entsprechenden Basisverzeichnis im NFS kennen in dem die Packages und Klassen abgelegt sind. Die Klassenversion läßt sich aus dem Zeitstempel der Datei ableiten. Ein erneutes Laden der Klassen muß über den jeweiligen Classloader der Java VM veranlaßt werden. Diese Variante ist sehr einfach und wird von vielen Systemen erfüllt. Jedoch ist die Behandlung der Programmversionen nur rudimentär. Ob eine erweiterte Funktion notwendig ist, hängt dann von der Frequenz an, mit der System angepaßt werden soll.

3.5.3.2 File Transfer Protocol (FTP)

Bei der Verwendung von FTP muß zunächst eine Erweiterung der Klasse Classloader implementiert werden, um dieses Protokoll zu nutzen. Ansonsten bietet die Verwendung von FTP kaum Vorteile. Bei bestimmten Einsätzen mag FTP effizienter und fehlerstabiler als NFS arbeiten und es müssen die Quellen nicht erst “gemounted” werden.

3.5.3.3 Hypertext Transfer Protocol (HTTP)

HTTP wird bei Applets bereits eingesetzt, wobei hier noch ein Teil der Klassen lokal vom Browser zur Verfügung gestellt wird. Damit soll der Datentransfer für ein Applet reduziert werden und vor allem das System vor unzulässigen Zugriffen geschützt werden. Letzteres ließe sich in Zusammenhang mit einer sicheren Managementumgebung lockern, jedoch ist es sinnvoll einen Teil der oft benutzten und sich selten ändernden Java-Klassen auf den Komponenten zu lassen. Ein entsprechender WWW-Server könnte in HTTP noch Zusatzinformationen über die Version verschicken. So kann außer dem Dateidatum noch mit Expires das Datum der voraussichtlichen Nachfolgeversion angegeben werden. Damit könnte eine Managementkomponente selbständig dafür Sorge tragen sich zum gegebenen Zeitpunkt die neue Klasse zu laden, falls sie sich geändert hat, was wiederum mit dem If-Modified-Since Feld des Request erreicht wird. Auch eine Versionsnummer kann im Entity-Header von HTTP übertragen werden. Eine weitere Möglichkeit entsteht dadurch, daß ein Request eine Sprache wünschen kann. Auf diese Weise kann man jeden Anwender seine gewünschte Sprache für die Ein-/Ausgabe bieten, ohne dafür verschiedene URIs angeben zu müssen. Der Server muß in diesem Fall allerdings die Fähigkeit besitzen unter einer URI mehrere Klassen gleichen Namens in verschiedenen Sprachen zu speichern.

Will man diese Softwareverteilung für Java-Applikationen, die ja in fast allen Komponenten außer WSF notwendig sind, verfügbar machen, so muß man wiederum einen Classloader implementieren.

3.5.3.4 Zusammenfassung

Alle drei Möglichkeiten sind für die Realisierung der s-Schnittstelle denkbar und von der Leistung her in diesem Zusammenhang weitgehend äquivalent, jedoch wird HTTP in vielen Fällen bereits für die WSF eingesetzt. Damit ist es sinnvoll auch hier HTTP zu benutzen, da der Aufwand für die Installation weiterer Protokolle und damit verbundenem Firewalling etc. entfällt. Die Headerinformationen von HTTP sind hierfür ein weiteres Argument.

3.6 Systeme

Aus der Abbildung von Schnittstellen auf Protokolle und APIs ergeben sich zum Teil bereits Aufteilungen der Funktionen auf Systeme. So bleiben bei den Varianten für die f-Schnittstelle nur wenig Variationsmöglichkeiten für die Aufteilung der funktionalen Komponenten auf die Systeme. Soweit für die Workstation Function ein WWW-Browser verwendet wird, kann dieser nicht den proaktiven Teil der Management Application Function enthalten, der ständig im System verfügbar sein sollte. Damit können hier Java Applikationen, gegebenenfalls mit Zugriff auf Shell-Kommandos oder JNI zur Implementierung eingesetzt werden. Abhängig von Netzlast, Fehlersichertheit und den Fähigkeiten des jeweiligen Systems wird man diesen Teil möglichst nahe bei den Ressourcen halten wollen. Das heißt, daß ein Großteil der MAF direkt in den NEs implementiert werden sollte, soweit diese über die nötigen Fähigkeiten verfügen. Die Mediation Function, im besonderen die Information Conversion Function, soll auf einem System mit entsprechender Nähe zu den unterliegenden Komponenten und angemessener Zuverlässigkeit oder auf einer untergeordneten Komponente selber eingesetzt werden. Ersteres soll die Netzlast gering halten, und letzteres soll verhindern, daß Ressourcen aufgrund eines Ausfalls in einer darüberliegenden Instanz vom Management abgeschnitten werden. Gleiches gilt auch für den Einsatz einer ICF in einer Q-Adaptor Function, wobei sich bei einem einzelnen von der QAF verwalteten Network Element, auch dieses selber als System anbietet.

Eine eigenes Operation System einzusetzen ist dann nicht unbedingt notwendig, da alle funktionalen Komponenten bereits von anderen Funktionsblöcken übernommen werden, jedoch würde sich der WWW-Server zusammen mit der Softwareverteilungsfunktion hierfür anbieten.

4 Integration mit bestehenden Architekturen

Um Vorzüge wie Plattformunabhängigkeit und Applets auszunutzen, haben einige Hersteller Produkte zu Nutzung bestehender Architekturen und Standards entworfen. Hierbei wird Java als eine neue Programmiersprache eingesetzt. Die tatsächlichen Möglichkeiten von neuen Architekturen werden aber nicht, oder nur zum Teil, genutzt.

4.1 Advent Network Management Java SNMP Package

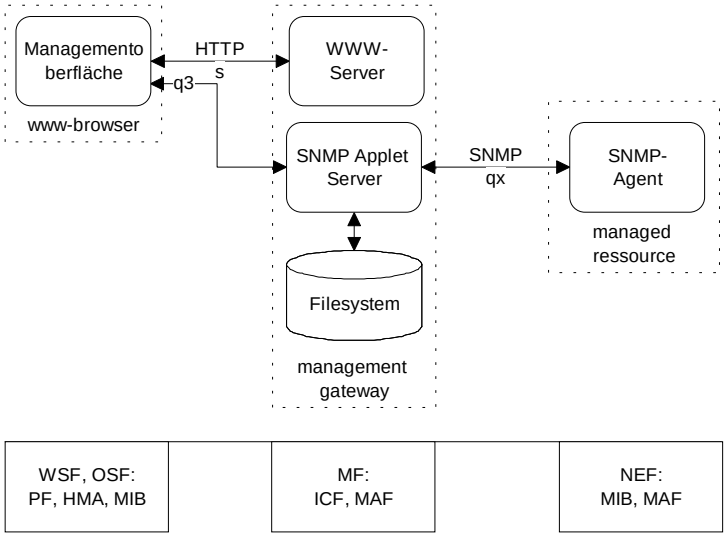

Abbildung 4.1-i: Advent Network Management Java SNMP Package

Advent bietet ein Entwicklungspaket aus mehreren Java Klassen um mit Java SNMP Anwendungen zu implementieren ([Adv96]). Dazu ist das SNMP-Protokoll in der Version 1 (CMU SNMP 1.2U) als q-Referenzpunkt implementiert. Die von Advent gelieferten Klassen erlauben das Schreiben von diversen Elementen einer Managementarchitektur. Da als Protokoll SNMP eingesetzt wird, lassen sich beliebige andere Komponenten bestehender SNMP Szenarien obiger Version einsetzen. Ein MIB-Browser-Applet ist als Beispiel verfügbar.

4.1.1 Funktionsblöcke

4.1.1.1 WSF

Für die WSF wird das Java Abstract Windowing Toolkit benutzt. Die Implemeniterung kann als Applikation oder als Applet erfolgen. Damit ist die Presentation Function in Ihren Fähigkeiten durch Java bestimmt.

4.1.1.2 OSF

Auf jeder Workstation (im Sinne von TMN), muß ein Teil der OSF enthalten sein, da der f-Referenzpunkt auf die Java.awt API abgebildet wird. Damit werden die Funktionalen Elemente Human Machine Adaption und der MIB als Minimum benötigt. Bei einem Browser, ohne Beschränkungen bezüglich des Netzzugriffes, oder einer Applikation ist auch der Einsatz einer Information Conversion Function denkbar. Die Management Applikation Function ist jedoch nur als Java-Applikation sinnvoll, da das System bei einem Browser keine Möglichkeit besitzt sicherzustellen, daß ein Thread ständig läuft.

4.1.1.3 Softwarebereitstellung

Zur Softwarebereitstellung kann ein einfacher WWW-Server benutzt werden.Dabei muß der SNMP Applet Server auf dem gleichen System installiert werden, soweit er eingesetzt werden soll.

4.1.1.4 SNMP Applet Server (SAS) - Mediation Function

Ein SNMP Applet Server ist ein Mediation Device und übernimmt damit den Funktionsblock der Mediation Funktion. Seine Aufgabe ist es Anfragen von Applets, die in einem Browser mit Zugriffsbeschränkungen laufen, an Agenten weiterzuleiten und einen Schreib/Lese-Dateizugriff auf dem Management Gateway. Letztere können zum Laden und Speichern von Benutzereinstellungen eingesetzt werden.

4.1.1.5 NEF

Auch die Network Element Function kann mit dem Advent Package implementiert werden soweit das Network Element über eine Java Virtual Machine verfügt. Dabei werden zur Kommunikation mit anderen Managementkomponenten die Klassen des Pakets eingesetzt. Der Zugriff auf Hardwarekomponenten muß aber mit dem JNI oder über Shell-Kommandos realisiert werden.

4.1.1.6 Q Adaptor Function, andere Dienste

Für eine Unterstützung für andere Managementkomponenten oder Dienste, die kein SNMP verstehen, ist eine Implementierung des x-Referenzpunktes auf TCP oder UDP Basis möglich, jedoch wird keine entsprechendes Paket mitgeliefert.

4.1.2 Funktionale Elemente

4.1.2.1 MIB

Die verwendete MIB entspricht der CMU SNMP 1.2U mit einem eigenen MIB-Parser, der mehrere MIB-Modul-Dateien verwalten kann. Diese können dynamisch in jede Java Applikation oder jedes Java Applet geladen werden. Ein Großteil des vorliegenden Packages besteht aus der Implementierung der MIB in Java.

4.1.2.1.1 SNMP variable classes

Das Paket enthält Klassen um SNMP-Variablen alr Java-Objekte zu präsentieren und Verbindungen zwischen Objekt-Referenzen und SNMP-Variablen herzustellen.

SNMP Variable class	SNMP variable type
SnmpInt	Integer
SnmpNull	Null
SnmpOID	Object Identifier
SnmpString	Octet Strings
SnmpOpaque	Opaque
SnmpIpAddress	IpAddress
SnmpUnsignedInt	(not used directly)
SnmpCounter	Counter
SnmpGauge	Gauge
SnmpTimeticks	Timeticks
SnmpVarBind	(variable binding)
ASNTypes	(handling of Abstrakt Syntax Notation)

Tabelle 4.1-i: Verwendung von SNMP-Variablen in Java

4.1.2.1.2 SNMP MIB related classes

Eine Instanz der Klasse MibModule wird aus einer MIB-Modul-Datei erstellt. Damit können , die Daten in der Datei benutzt und geparsed werden. Das Laden und Verwerfen von MIB-Modul-Dateien wird mit der Instanziierung und Dereferenzierung des Objekts erledigt. Die Instanz enthält die Knoten des MIB-Baums und die definierten Traps.

Von der Klasse MibModule werden die folgenden Klassen genutzt: MibMacro wird zum Parsen von Markofunktionen der MIB benutzt, wobei nur OBJECT-TYPE und TRAP-TYPE macros unterstützt werden. Mit MibTrap werden Daten und Typen von Traps gehalten. Einzelne Knoten in der -MIB werden durch die MibNode Klasse repräsentiert und in einer Liste in einer MibModule Instanz archiviert. Die Blätter der MIB werden mit Textkonventionen in der Klasse LeafSyntax gespeichert.

4.1.2.2 Message Communication Function

Neben der MIB besteht enthält das Advent SNMP Package Klassen zur Kommunikation mittels SNMP.

4.1.2.2.1 SNMP Communication classes

Mit der SnmpAPI Klasse werden von Benutzeranwendung erstellte Verbindungen verwaltet. Hier werden MIB-Module geladen und Schlüsselwerte für die Kommuniktion, wie Ports gespeichert. Die SnmpAPI Klasse hat eine Liste aller in Ihr erstellten SNMP-Verbindungen (auch zu verschiednen Peers) und überprüft diese jeweils mit einem eigenen Thread auf Zeitüberschreitungen und Nachrichten. Es werden einige Methoden unterstützt, die über alle diese Verbindungen laufen. Die SnmpAPI Klasse muß vor der Verwendung instanziiert und gestartet werden. Es können auch mehrere Instanzen angelegt werden.

Die SnmpSession Klasse repräsentiert eine Verbindung mit einem SNMP Peer. Dabei besitzt jede Instanz einen eigenen Thread, der synchrone und asynchrone Kommunikation abwickelt.

Die übertragene PDU wird von der Klasse SnmpPDU repräsentiert, in der Variablen und Methoden vermittelt werden.

4.2 Jump "Java-based Universal Management Platform"

Die Firma Outback Resource Group Inc. will mit Jump [OutBack Resource Group, Inc] nicht nur eine Managementumgebung, sondern auch gleich den Jump Workshop als Integrierte Entwicklungsumgebung (IDE) anbieten ([Out96]), der jedoch nur auf Windows 95 und NT einsatzfähig ist. Hier soll eine schnelle Entwicklungsmöglichkeit für Managementseiten für WWW-Browser, ohne das Schreiben von Kode ermöglicht werden.

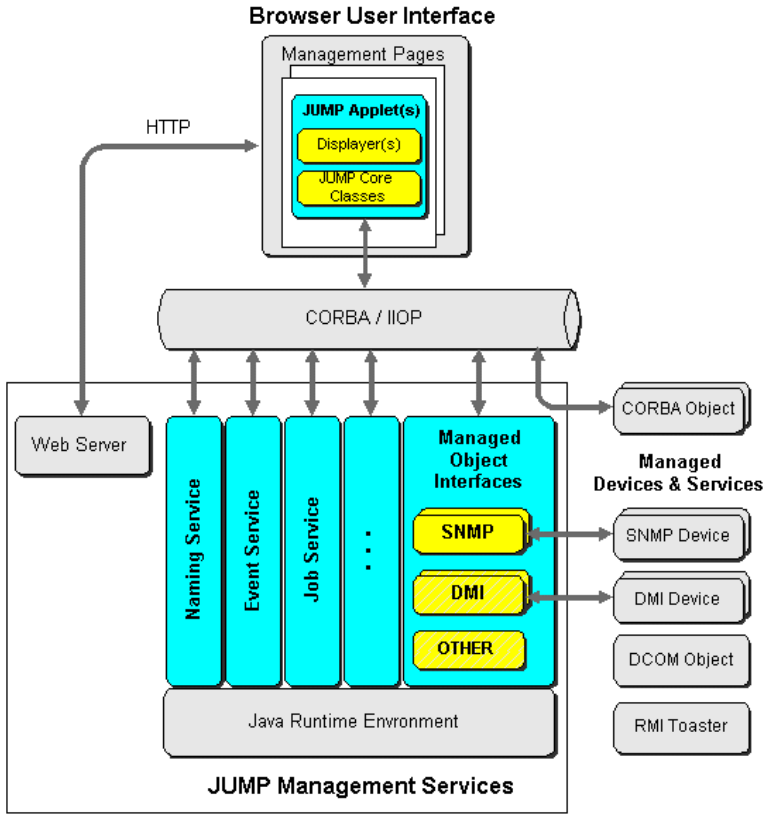

Abbildung 4.2-i: Jump Architektur [Out96]

4.2.1 Funktionsblöcke, funktionale Elemente

Beim Management benutzt Jump vorhandene Standards und neu auftretende Technologien, wobei die Protokollunabhängigkeit gewahrt werden soll. Jump versucht

dabei möglichst viele der Entwicklungsplattformen zu integrieren. So sollen HTML, CGI-Skripten, Java, ActiveX und CORBA unterstützt werden.

4.2.1.1 WSF, Browser User Interface

Die Presentation Function von Jump wird durch HTML-Seiten und Java Applets gestellt. Dabei stellen die *Jump Core Classes* grafische Elemente, die speziell im Management eingesetzt werden zur Verfügung um die Fähigkeiten von HTML zu erweitern.

4.2.1.2 OSF, QAF Jump Management Services (JMS)

Die *Jump Management Services* besitzen ein CORBA IDL und ein standard Java Interface und kann auf jedem System mit einer Java VM eingesetzt werden.

4.2.1.3 Softwareverteilungsfunktion

Die Softwareverteilungsfunktion, wird über einen WWW-Server auf dem Opertion System erfüllt.

4.2.2 Schnittstellen

4.2.2.1 f-Schnittstelle

Die f-Schnittstelle von Jump besteht aus HTTP zur Übertragung der HTML-Elemente und Applets und aus CORBA/IIOP über das die Jump Core Classes mit den Jump Management Services Kommunizieren.

4.2.2.2 q-Schnittstelle, m-Schnittstelle

Als q-Schnittstelle wird CORBA eingesetzt um mit Objekten zu Kommunizieren, die über CORBA verfügen. Weiterhin werden als m-Schnittstelle SNMP, DMI und zukünftig auch HMMP und RMI zur Verfügung stehen.

4.3 Java SNMP Control Applet (JaSCA)

Java SNMP Control Applet enthält alle notwendigen Komponenten, um mit einem WWW-Browser SNMP Attribute des Hosts, von dem das Applet geladen wurde zu überwachen und zu kontrollieren ([NiW96]). Damit entspricht JaSCA einem MIB-Browser Applet. Durch die Installation einer JaSCA Komponente auf dem Host, können beliebige SNMP-fähige Ressourcen kontrolliert und überwacht werden.

4.3.1 Funktionsblöcke, funktionale Elemente

Damit enthält JaSCA die Funktionsblöcke WSF und OSF. Diese implementieren selbstverständlich die Presentation Function, die Human Machine Adaption und die MIB. Die Komponente, die auf dem Host installiert werden kann, erfüllt dabei als Teil der Mediation Function die Information Conversion Function.

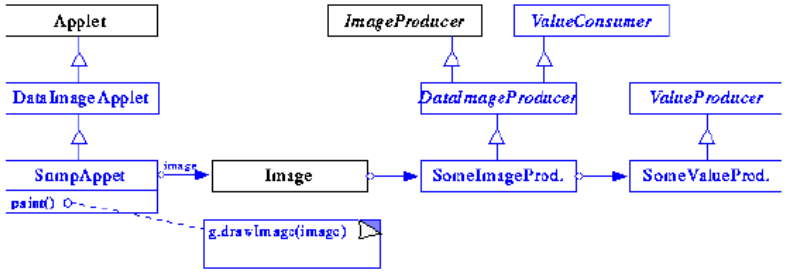

Abbildung 4.3-i: Architektur der WSF [NiW96]

4.3.2 Schnittstellen

Dabei benutzt JaSCA als f-Referenzpunkt die API des Java Abstract Windowing Toolkits. Die q-Referenzpunkte werden auf das SNMP-Protokoll abgebildet.

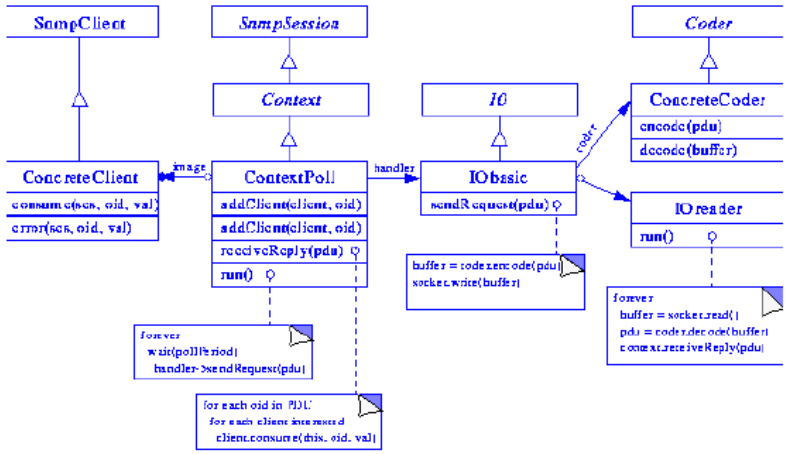

Abbildung 4.3-ii: Architektur der SNMP-Schnittstelle [NiW96]

4.3.3 Architektur

Beim Start des Applets wird mit einem Aufruf einer URL die Klasse SnmpStreamHandler als ihr Stream handler geladen. Durch seine Implementierung kennt dieser bereits, die unter ihm liegenden SNMP-Bibliothek. Bei Anfrage etabliert SnmpStreamHandler mit SnmpConnection eine Sitzung als eine Unterklasse von URLConnection. Diese Verbindung wird mit einem bestimmten Context verbunden. Die Java VM in einem Browser verhindert das Laden der SnmpStreamHandler Klasse. Wenn das Programm dies feststellt, wird der SnmpStreamHandler von Hand erzeugt.Er wird benötigt um eine SnmpConnection zu instanziieren.

Mit der Instanz der Verbindung wird nun ein Objekt von ImageProducer mit dem Inhalt erstellt. ImageProducer implementiert das DataImageProducer Interface und erstellt eine Abfolge von Seiten mit einer maximalen Rate 0,1 Hz.

Die Daten werden dabei über ein ContextPoll Objekt erhalten. Dieses erstellt in Abständen SNMP GetRequests, die an den IO handler weitergereicht werden. Dieser kodiert die Nachricht und schickt sie an den entsprechenden UDP socket. Die vom Agenten erzeugten Antworten, werden von einem eigenen IOreader Objekt empfangen, das in einem eigenen Thread läuft und die ganze Zeit auf eingehende PDUs wartet Trifft eine Nachticht ein, so wird diese in einen internen Puffer geschrieben und dekodiert und schließlich an das ContextPoll Objekt geschickt.Das ContextPoll Objekt findet nun alle DataImageProducer, die an SNMP Attributen der empfangenen PDU interessiert sind und schickt ihnen eine entstehende Nachricht. Ein benachrichtigter DataImageProducer erstellt mit dieser Nachricht eine neue Seite.

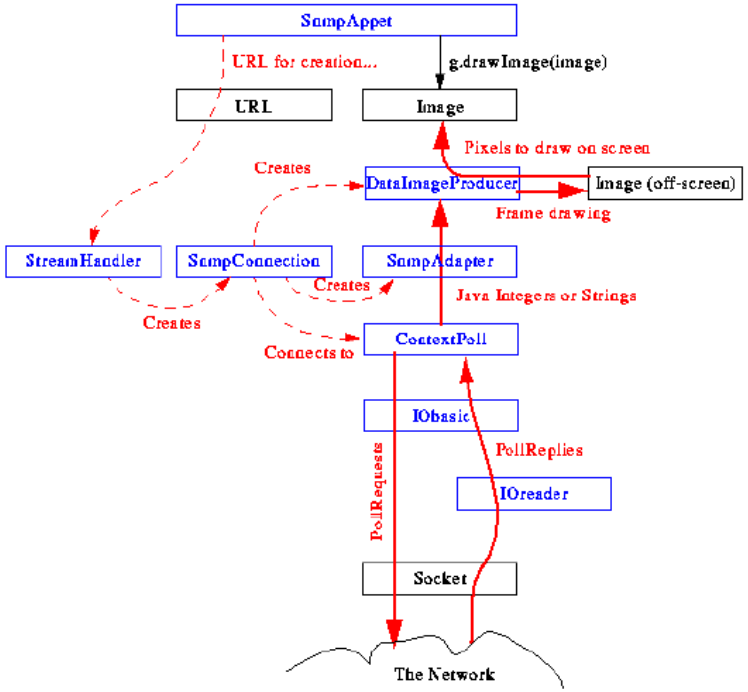

Abbildung 4.3-iii: JaSCA Architektur [NiW96]

5 Neue Architekturen und Standards

In diesem Abschnitt werden Ansätze behandelt, die auf den Fähigkeiten von WWW und Java aufbauend, komplett neue Managementsysteme beschreiben.

5.1 Sun Java Management API (JMAPI)

Die Java Management API [Sun96a] enthält eine eigene Managementarchitektur, eine Erweiterung der Benutzeroberfläche und Richtlinien zur Erstellung von Anwendungen und Hilfen. Dabei achtet Sun besonders auf die Integration üblicher Standards und die Portabilität.

Folgende Komponenten sind im JMAPI enthalten:

- User Interface Style Guide
- Admin View Module (AVM)
- Base Object Interfaces
- Managed Container Interfaces
- Managed Notification Interfaces
- Managed Data Interfaces
- Managed Protocol Interfaces
- SNMP Interfaces
- Applet Integration Interfaces

5.1.1 Funktionsblöcke, funktionale Elemente

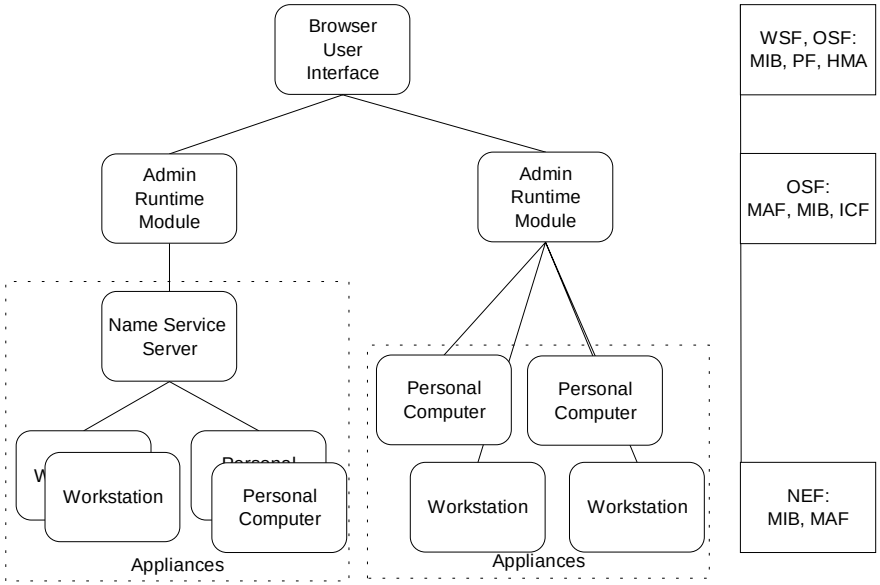

Abbildung 5.1-i: Funktionsblöcke in JMAPI [Sun96a]

5.1.1.1 WSF, Browser User Interface (BUI)

In dem Java Management API arbeitet der Administrator an einem WWW-Browser mit Javaunterstützung. Da der f-Referenzpunkt auf Java AWT und Admin View Module abgebildet wird, muß mit der Human Machine Adaption auch ein Teil der OSF auf dem gleichen System, wie die WSF liegen.

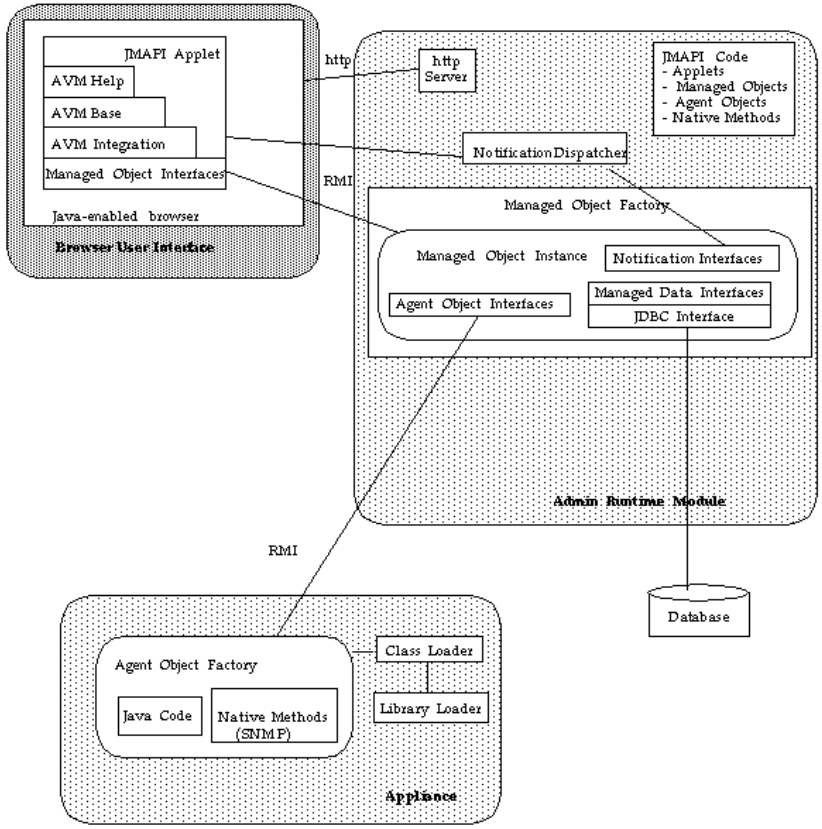

Abbildung 5.1-ii: WSF, Browser User Interface (BUI) [Sun96a]

5.1.1.1.1 Admin View Module (AVM)

Mit dem Admin View Module werden GUI Klassen für die Darstellung von Managementinformationen auf der Klientenseite zur Verfügung gestellt. Das AVM baut auf dem Java Abstract Windowing Toolkit auf. Dadurch ist auch das AVM portierbar und unabhängig vom der Benutzeroberfläche.

Die AVM Klassen sind in drei Pakete aufgeteilt: AVM Help, AVM Base und AVM Integration.

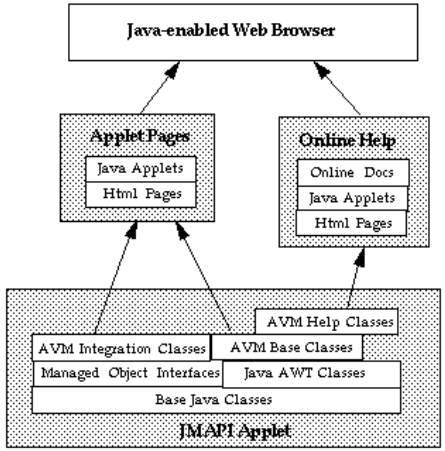

Abbildung 5.1-iii: Admin View Module (AVM) [Sun96a]

5.1.1.1.1.1 AVM Help

Mit dem AVM Help wird in erster Linie eine universell einsetzbare Hilfeumgebung um Programme mit einer Hilfefunktion zu versorgen.

Damit AVM Help universell ist, kommt es ohne eigene Tools, Skripten oder Programme aus um Hilfe-Dateien zu erzeugen. Die einzige Vorgabe ist, daß die Hilfe-Dateien im HTML-Format vorliegen und einen Satz JMAPI Tags enthalten, die während der Installation geprsed werden können um Navigationshilfen, Inhaltsverzeichnisse und Glossare zu erzeugen.

Der TOC/Navigator stellt ein Hierarchie von Hilfethemen dar, die in einem Programm verfügbar sind. Mit dem Doc Generator jmapidoc werden Verzeichnisse oder TOC/Navigator Dateien für ein Programm erstellt. Mit diesem Programm können Hilfe-Autoren ihre Arbeiten austesten. Weiterhin gibt es Standardisierte Hilfe-Dateien für Hilfe für Standardbedienung und die Hilfe selber. Schließlich sind noch Vorlagen und eine Suchmaschine enthalten.

5.1.1.1.1.2 AVM Base

Die AVM Base Klassen sind eine Erweiterung des Java AWTs, die speziell für Entwickler integrierter Managementanwendungen gedacht sind, um eine Benutzerschnittstelle im Hypertext-Stil auf einem Web-Browser zu erstellen. Das AVM Base Paket ist in weitere drei Bereiche unterteilt.

Die Base User Interface Klassen erweitern das Java AWT, um einen Satz von Komponenten zum Erstellen von reichhaltigen Benutzerschnittstellen zu erhalten. Darunter fallen Buttons mit Bildern, Fenster und Panele mit Bildlaufleisten, Toolbars und Komfortdialoge.

Das Power User Interface enthält Klassen um schnellere, einfachere Lösungen zu erstellen. Dazu gehören Tabellen, Hierarchische Browser, Charts und Pegelanzeigen.

Mit dem Management User Interface werden das Aussehen und die Benutzerinteraktion entsprechend dem API User Interface Style Guide unterstützt. Dazu gehören Verzeichnisse für Eigenschaften, Seitenköpfe und Konfigurationsbrowser.

5.1.1.1.1.3 AVM Integration

Die Hauptaufgabe der AVM Integration Klassen ist es eine Integration der AVm Base Klassen mit den Managed Object Interfaces zu liefern. Weiterhin können Managmentapplikationen mit den AVM Integration Klassen ihre Komponenten registrieren, damit bei neuen Lösungen die Integrität der Managementinformation im Netz sichergestellt wird. Damit lassen sich Management Applets, Seiten, Links und Erweiterungen des Managed Object Interfaces registrieren und unregistrieren. Die AVM Integration Klassen bauen auf den Manged Object Interfaces, den AVM Base Klassen, dem Jawa AWT und den Java Basis Klassen auf.

5.1.1.1.2 Managed Object Interfaces

Die Managed Object Interfaces benutzen RMI als q3-Referenzpunkt, um entfernte Management Methoden auszuführen. Mit den Managed Objects wir die Abstraktion der Ressourcen erreicht.

5.1.1.2 OSF, Admin Runtime Module (ARM)

Das Admin Runtime Module erfüllt einen Großteil der Operation System Function.

5.1.1.2.1