FACHHOCHSCHULE WEDEL

DIPLOMARBEIT

in der Fachrichtung

Wirtschaftsinformatik

Thema:

Konzeption und Implementierung
eines
Hypertext Informations- und Hilfesystems mit Client-Server Architektur
für eine VDM Klassenbibliothek unter Verwendung von Werkzeugen
des World Wide Web

Eingereicht von:	Marko Temme Hafenstraße 33 22880 Wedel Tel. 04103-97266 Matrikelnr. 8980
Erarbeitet im:	13. Semester
Abgegeben am:	15. September 1996
Referent (FH Wedel):	Prof. Dr. Uwe Schmidt

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

$\mathcal{N}_0$ .... Menge der natürlichen Zahlen inkl. 0
$\mathcal{N}_1$ .... Menge der natürlichen Zahlen exkl. 0
$\mathcal{Z}$ .... Menger der ganzen Zahlen
ADT .... Abstrakt(e) Datentyp(en)
BNF .... Backus-Naur-Form
BSD .... Berkeley Software Distributions
DSL .... Domain Specification Language
EPS .... Encapsulated Postscript
FAQ .... Frequenty Asked Questions
GIF .... Graphics Interchange Format
HTML .... Hypertext Markup Language
HTTP .... Hypertext Transfer Protocol
HTTPD ...... Hypertext Transfer Protocol Daemon
LAN .... Local Area Network
MIME .... Mulipurpose Internet Mail Extension Protocol
NCSA .... National Center for Supercomputing Applications
OSI .... Open System Interconnection
PPP .... Point To Point Protocol
RFC .... Request for Comments
SGML .... Standard Generalized Markup Language
SLIP .... Serial Line Protocol
TCL .... Tool Command Language
TCP/IP ...... Transmission Control Protocol / Internet Protocol
TK .... Toolkit
UDP .... User Datagram Protocol
URL .... Uniform Resource Locator
UUCP .... UNIX to UNIX Copy
VDL .... Vienna Definition Language
VDM .... Vienna Development Method
WAN .... World Area Network
WWW .... World Wide Web.
WYSIWYG ... What You See Is What You Get
YACC .... Yet Another Compiler-Compiler

Kapitel 1

Einleitung

1.1 Überblick

Diese Diplomarbeit ordnet sich ein in die Kette der Entwicklungen zu einer VDM Klassenbibliothek an der Fachhochschule Wedel seit 1990. Bereits mehrere Studenten zuvor, genannt seien Hans-Martin Hörcher, Michael Kuhn, Thorsten Tews, Jan Peter Mohr, Jens Wulf und andere nahmhafte als auch dem Autor unbekannte (Programmier-)Künstler haben mitgewirkt und erkennbare Spuren im Projekt selbst oder Dokumentation hinterlassen.

Aufgrund der erheblich komplexer werdenden Programmiersprachen und der damit verbundenen stetigen Zunahme von Fehlern sowie des Aufwandes diese zu entdecken und zu beheben, begann Mitte der 60er Jahre im Wiener IBM Labor die Suche nach einer Möglichkeit, Programmiersprachen formal zu definieren. Der Umfang der Programmiersprache PL/I gab Anlaß, einige der Ideen in einer formalen Sprachbeschreibung zu manifestieren. Man bediente sich dabei einer operationalen Semantik, welche als „Vienna Definition Language (VDL)“ bekannt wurde.

Eine Arbeitsgruppe, geleitet von Christopher Strachey an der Universität in Oxford entwickelte eine mathematische Semantik. Dieser neue Vorstoß wurde von den Wiener Entwicklern übernommen und „Vienna Development Method (VDM)“ getauft, was für Verwirrung sorgte, da die Urheber gleiche Abkürzung, jedoch anderen Sinns, verwendet hatten. Mittlerweile ist die Spezifikation aus Oxford als Z-Notation bekannt.

Der Begriff VDM umfaßt zwar die gesamte systematische Entwicklungsmethode, versteht sich jedoch auch für die mathematische Semantik der formalen Spezifikation, welche nicht maschinenlesbar und aus diesen Gründen für die Designphase ungeeignet ist. Daher wurde von der Wiener Arbeitsgruppe eine maschinenlesbare Zwischensprache (meta-language) verwendet, welcher der Name META-IV gegeben wurde. Ein Ergebnis dieser Arbeits ist die Version IV des PL/I D-Compilers für /360 Maschinen, welche in META-IV spezifiziert und entwickelt ist.

Die PL/I-Sprache wurde von der IBM in Zusammenarbeit mit Vertretern von SHARE, einer Organisation wissenschaftlicher Benutzer, und GUIDE, einer Organisation kommerzieller Benutzer elektronischer Datenverarbeitungsanlagen, definiert[1]. Als Besonderheit von PL/I als Vorbild vieler Programmiersprachen findet sich die in vielen späteren Sprachen verloren gegangene Möglichkeit, eine Art Invariante in Form der ON-Anweisung zu definieren, eine Idee, die der mathematischen Semantik entstammt. Das PL/I Programmierhandbuch [IB70, S. 7] stellt zu Recht fest:

> „PL/I ist viel weniger maschinenabhängig als die anderen gebräuchlichen Programmiersprachen."

Der Grund hierfür liegt in der Abstraktion durch den Einsatz formaler Entwicklungsmethoden.

Die VDM-Klassenbibliothek der FH-Wedel verwendet eine stark an META-IV angelehnte, jedoch im Sprachumfang verringerte, maschinenlesbare Spezifikation mit dem Namen „Domain Specification Language" (DSL), in die in das mathematische Modell übertragen werden kann. Aus dieser Beschreibungssprache entsteht mit Hilfe der Tools dieser Bibliothek Quellcode für die Applikation in der gewählten Zielsprache.

1.2 Aufgabenstellung

Die Arbeit mit der VDM-Klassenbibliothek erfordert allgemeine Kenntnisse über die Methoden der formalen Spezifikation zur Softwareentwicklung als auch das Wissen über die konkreten Inhalte der Bibliothek. Aufgabe dieser Diplomarbeit ist die Bereitstellung unterstützenden Materials zur Erleichterung des Verstehens und der Anwendung der Klassenbibliothek, konkret umgesetzt durch ein Hypertext Informations- und Hilfesystem, welches dem Anwender ermöglicht, gezielt Informationen zu bestimmten Sachverhalten abzurufen, elektronisch Dokumente einzusehen sowie Beispiele von Anwendungen zu der VDM Klassenbibliothek interaktiv auszuführen.

Unter Hypertext wird eine Dokument verstanden, welches über Verknüpfungen zu anderen Dokumenten verfügt. Die Anwahl einer solchen Verbindung schafft den sofortigen Zugang zu dem jeweiligen referenzierten Dokument. Es handelt sich damit um den klassischen Recherchevorgang, der elektronisch unterstützt wird. Eine Besonderheit dieser Arbeit ist die Nutzung von Werkzeugen des World Wide Web, das selbst ein Hypertext-basiertes System für die Suche nach und den Zugriff auf Resourcen ist. Das WWW ist fester Bestandteil des Internets, welches als weltgrößtes zu Recht als das „Netz der Netze" bezeichnet wird. Das World Wide Web nimmt neben Dateizugriff, Fernanmeldung, elektronischer Post, Nachrichtengruppen und weiteren Diensten unbestritten die wichtigste Rolle ein.

Die Verwendung der Techniken des World Wide Web gestattet den globalen Zugriff auf die VDM Klassenbibliothek sowohl im Intranet, wie ein firmeninternes TCP/IP Netzwerk bezeichnet wird, als auch aus dem Internet heraus.

1.2.1 Teilaufgaben

Das Thema der Diplomarbeit verbindet sich mit mehreren Teilaufgaben, die als Komponenten des Zieles zu verstehen sind.

[1] [IB69, S. 5] IBM: Einführung in die PL/I-Sprache, 1969

- Schaffung eines interaktiven Hypertextsystems
- Hypertext-Dokumentation der Diplomarbeit
- Einbetten der VDM Beispiele in das System

Neben der Konzeption und Realisierung eines Hypertextsystems für die VDM Klassenbibliothek, soll auch die Diplomarbeit selbst als Hypertext vorliegen, so daß sie über das Internet abrufbar ist. Gleichfalls kommt jedoch eine ausschließliche Erstellung im HTML-Format nicht in Frage, da hiermit die Richtlinien zur Anfertigung der Diplomarbeit der Fachochschule Wedel nicht erfüllt werden können. Zwangsläufig ist der Einsatz geeigneter Konverter erforderlich. Zu der Klassenbibliothek existieren zahlreiche Fallbeispiele für deren Anwendung. Als dritter Aufgabenabschnitt ist die Einbettung dieser Beipiele in das Hypertextsystem gefordert.

1.2.2 Allgemeine Anforderungen

Den Gestaltungsmöglichkeiten der Lösung werden mit der Aufgabenstellung allgemeine Anforderungen zugrundegelegt, die im Interesse des Gesamtprojektes und der Stabilität der Klassenbibliothek liegen. Die Erstellung des Hilfesystems soll weitgehend in der Tool Command Language erfolgen, auf eine Modifikation der Hochsprachenmodule der Klassenbibiliothek ist nach Möglichkeit zu verzichten.

Es ist damit auf den bestehenden Strukturen aufzubauen, was eine vorbereitende Analyse der vorhandenen Klassenbibliothek als auch der Tools zwingend erforderlich macht. Für die Implementation gilt damit, daß sie entsprechend vorsichtig vorgenommen werden muß, um die bestehende Funktionalität nicht durch Ergänzungen zu beschädigen.

Kapitel 2

Grundlagen der Arbeit

Zu den Grundlagen der Arbeit zählen neben der VDM Klassenbibliothek der FH-Wedel mit allen Tools, die Hochsprachen C und C++, die Interpretersprache TCL, die Techniken des World Wide Web und insbesondere die Kommunikation über Netzwerke, da eine Client-Server Lösung geschaffen werden mußte. Diese Kapitel gibt einen Überblick und wesentliche Informationen, die im Zusammenhang mit der späteren Realisierung stehen.

2.1 Tool Command Language - TCL

TCL ist eine zeichenkettenorientierte Interpretersprache, mit wenigen fundamentalen Konstrukten und einer kleinen Syntax. Eine grundlegende Verarbeitungsstruktur von TCL sind Substitutionen, also die Auflösung von Ausdrücken in ihre Grundstruktur, mit anschließender Ausführung. Diese Vorgehensweise schafft neben der Verarbeitung von regulären Ausdrücken eine außerordentliche Flexibilität und Gestaltungsfreiheit.

Der TCL-Interpreter ist in C erstellt und weitgehend systemunabhängig. Er ist neben UNIX auf verschiedensten Systemen verfügbar, und erlaubt die problemlose Portierung von in TCL erstelllten Programmen auf alle Systeme, für die es auch einen TCL-Interpreter gibt.

Da der geringe Befehlssatz von TCL in der Regel nicht ausreicht, komplexere Probleme lösen, können neue Befehle in die TCL-Shell, wie der Interpreter auch bezeichnet wird, eingebunden werden. Die Umsetzung neuer Befehle erfolgt in einer Hochsprache, dessen Objektcode zum Interpreter gebunden wird.

2.2 Techniken des World Wide Web

2.2.1 Historie

Das Word Wide Web (WWW), kurz auch „das Web“ genannt, ist der Informationsdienst im Internet, mit der größten Wachstumsrate, und bietet von allen Diensten im

Internet die größte graphische Orientierung. 1995 wurden ca. 24 Millionen Internetbenutzer geschätzt [1].

Das Web und die zugehörigen Techniken finden ihren Ursprung 1989 im Europäischen Zentrum für Hochenergiephysik CERN in Genf unter der Leitung von Tim Berners-Lee. Der Kern der Arbeiten war die Definition des Client/Server-Protokolls HTTP, welches bis heute die Basis der WWW-Dienste ist und nach wie vor in der Version 1.0 eingesetzt wird. Zu den Arbeiten zählt auch die Entwicklung eines Beispielservers sowie einer Programmbibliothek in C mit dem Namen *wwwlib* [2].

Der Grundstein, der in Genf gelegt wurde und die Verfügbarkeit der *wwwlib* löste eine Innovation bei unzähligen Unternehmen, Universitäten und Privatpersonen aus. Nunmehr stehen die unterschiedlichsten kommerziellen und nichtkommerziellen Web-Browser und -Server für alle kleinen und großen Rechnersysteme zur Verfügung. Mittlerweile gibt es kaum noch größere Unternehmen, Fernsehsender oder Parteien, die sich nicht mit einer „Homepage“ schmücken oder ihren Mitarbeitern einen Zugriff auf das Internet über das hauseigene Netzwerk ermöglichen.

Auch die Zahl der Anwender insbesondere im privaten Bereich steigt beständig an, was dazu geführt hat, das trotz des Ausbaus der Bandbreiten durch die Netzanbieter, die im Zusammenhang mit dem Internet stets „Provider“ genannt werden, von einem „Information-Highway“ zu bestimmten Tageszeiten kaum mehr die Rede sein kann.

Der ernorme Erfolg des World Wide Web überstieg die Möglichkeiten des CERN im Hinblick auf die Weiterentwicklung. Aus einer Kooperation des CERN mit dem Massachusetts Institute of Technology (MIT) entstand des World Wide Web Consortium (W3C), das bis heute die Weiterentwicklung, unterstützt durch Beiträge industrieller Mitglieder vorantreibt, und dessen Spezifkationen auch Basis dieser Diplomarbeit sind.

2.2.2 HTTP-Protokoll und Server

Das Hypertext Transfer Protocol

Zur Kommunikation zwischen Web-Browser und Web-Server wird das HyperText Transfer Protocol (HTTP) eingesetzt. Das HTTP-Protokoll ist eine sogenanntes „lightweight“-Protokoll. Dies bedeutet, jede Anfrage des Clients an den Server mit der Bitte um ein Dokument stellt eine Verbindung her, über welche das Dokument übertragen wird, und baut diese anschließend wieder ab. Diese Eigenschaft war eines der gößten Probleme bei der Lösung der Aufgabenstellung, wie in 4.2.1 dargestellt wird.

Common Gateway Interface - CGI

Üblicherweise sind alle Seiten, die durch den Web-Browser angezeigt werden als Dateien auf dem Host gespeichert, auf dem der Web-Server aktiv ist. Dieser übergibt

[1] [JN95, S. 1] Jones,Russ / Adrian, Nye: HTML und das World Wide Web, 1995

[2] Diese Bibliothek, ein HTTP-Server als auch ein einfacher Web-Browser, die Normen zum WWW sowie zahlreiche weitere Dokumente sind heute unter der URL des www.w3.org zu finden und waren zuvor unter www.cern.ch abgelegt

dem Browser auf Anfrage lediglich den Inhalt der Seite und dessen Format.

Diese sehr statische Vorgehensweise erlaubt in der Regel nicht die Abfrage von gezielten Informationen, und wird aufgebrochen durch das CGI-Gateway. Dies ist eine Schnittstelle zwischen dem Web-Server und externen Programmen, die anstelle vordefinierter Dateien, dynamisch eine Seite erzeugen und an den Browser übergeben. In der Regel werden diese Programme als kurze Shell-, Perl- oder TCL-Skripte realisiert und bearbeiten beispielsweise Suchanfragen oder Online-Bestellungen.

2.2.3 Hypertext Markup Language - HTML

HTML ist eine Seitenbeschreibungsprache, die vollständig durch SGML definiert ist. Es handelt sich in diesem Sinne um einer Untermenge von SGML, was jedoch nicht bedeutet, daß diese direkt zueinander kompatibel sind, sondern die Syntax von HTML wird lediglich mittels SGML festgelegt. Die HTML-Standards 2.0 und 3.0 wurden durch das W3-Consortium jeweils durch SGML-Dokumente herausgegeben.

HTML besteht aus Fließtext, der durch sogenannte „Tags“ formatiert wird. Ein Tag wird durch die Zeichen „¡“ und „¿“ eingeschlossen. Neben einfachen Tags zur Festlegung der Schriftgröße wie beispielsweise:

```
Normalschrift<H1>Grossschrift</H1>Normalschrift
```

lassen kompliziertere unter anderem auch das Einbinden von Bildern und anderen graphischen Elementen in den Text zu.

Der wichtigste Tag von HTML ist zweifelsohne derjenige, der erst die weltweite Verknüpfung von Dokumenten zuläßt. Die Adressierung von Dokumenten ist weltweit möglich und wird durch einen *Uniform Resource Locater (URL)* vorgenommen. Dieser Tag und der Aufbau des Tags mit einer URL ist wie folgt:

```
<A HREF=http://www.home.de/homepage>Welcome</A>
```

Die erste durch den Doppelpunkt getrennte Angabe spezifiziert das Protokoll, das verwendet werden soll. In obigem Beispiel ist dies das HTTP-Protokoll zur Verbindung mit einem Web-Server. Zu weiteren Protokollen, die anstelle von HTTP angegeben werden können, zählen *telnet*, *ftp* und andere. Dem Protokoll folgt die Internetadresse, oder der Name des Hosts[3] auf dem das Zieldokument, dessen Name dem Host folgt, abgelegt ist.

2.2.4 Multipurpose Internet Mail Extensions Protocol MIME

Ursprung von MIME

Als UNIX 1978 bei den Bell Laboratories in größerem Umfang eingesetzt wurde, war bald klar, daß eine Möglichkeit gesucht werden mußte, mit der Dateien von einem UNIX-Rechner auf einen anderen übertragen werden konnte. Zur Lösung die-

[3] Namen werden über das DNS-Protokoll in ihre IP-Adesse umgesetzt

ses Problems entstand uucp (UNIX to UNIX Copy), ein Programm zur Datenfernübertragung über mehrere Etappen (hops) von UNIX-Systemen mit automatischer Wähfunktion[4].

Da **uucp** zunächst lediglich in der Lage war reinen Text zu versenden, dieser durch das store-and-forward Übertragungsverfahren den Zeichensatzbedingungen verschiedenster Systeme unterlag und dadurch gelegentlich verfälscht wurde, entstand Bedarf an einer Kodierung, die eine Untermenge aller bekannten Zeichensätze ist. Auch die Übertragung von Binärdaten wie Programmen stellte ein Problem dar, da auch bei diesen eine verlustfreie Übertragung nicht gesichert war und beispielsweise durch Abschneiden des höchstwertigen Bits gestört werden konnte. Die Probleme wurden durch Erweiterung des uucp Paketes um die Programme *uuencode* und *uudecode* gelöst, dessen Kodierung Vorläufer des MIME-Protokolls ist.

Bedeutung von MIME für das Internet

Das MIME Format selbst wird im Internet nach wie vor neben dem Format von UUCP vorwiegend für die Übertragung elektronischer Nachrichten verwendet. Neben dem Inhalt eines Dokumentes wird dessen Art durch seinen MIME-Typen beschrieben, der aus Haupt- und Untertypen besteht. Der Haupttyp legt fest um welche Art von Informationen es sich handelt, wie Text, Bilder, Töne oder Filme, der Untertyp legt fest in welchem Format diese Informationen vorliegen. Der Mime-Typ *image/gif* bezeichnet ein Bild im GIF-Format.

Für das HTTP-Protokoll ist weniger die MIME-Kodierung von Bedeutung, sondern lediglich der MIME-Typ ist fester Bestandteil des Protokolls. Jeder Art von Daten, die über das HTTP-Protkoll übertragen werden, wird der Typ vorangestellt. Für HTML-Dokumente ist dies *text/html*. CGI-Gateways wird vom Web-Server vor dem Aufruf mitgeteilt, welche Formate akzeptiert und geliefert werden dürfen. Für die Realisierung des Hypertext Informations- und Hilfesystems wurde das MIME-Format verwendet, um binäre Daten in der Klassenbibliothek hinterlegen zu können.

2.3 Kommunikation im Internet

Das OSI-Referenzmodell (Open Systems Interconnection Model) ist ein gut dokumentierter Industriestandard für die Kommunikation offener Systeme, über den die meisten Computernetzte beschrieben werden. Das Modell definiert eine geschichtete Netwerkarchitektur, bei dem jede Schicht für einen getrennten Aspekt der Arbeit im Netz verantwortlich ist. Das OSI-Modell beschreibt den Datenfluß in einem Netzwerk von der niedrigsten Schicht der physikalisch vorhandenen Hardware bis hin zur Schicht der Anwendungsprogramme. Jede Schicht kann ausschließlich mit der direkt darüber- oder darunterliegenden Schicht kommunizieren. Die Schnittstellen zwischen den Schichten werden als Service Access Point (SAP) bezeichnet. Zwischen den SAPs werden PDUs (protocol data unit) als Protokolldateneinheiten übertragen, welche die

[4]Das auf uucp-basierende Netzwerk uunet ist heute mit ca. einer Million Anwender das zweitgrößte Netzwerk nach dem Internet und mit diesem verwoben.

Nutz- und Steuerinformationen in einem oder, wenn dies aufgrund der Länge praktikabler ist, mehreren PDUs einer Sequenz übertragen. Einer PDU wird zur Identifikation der Schicht der Anfangsbuchstabe vorangestellt.

Bei der Kommunikation zweier Instanzen im Netzwerk, kommunizieren jeweils die gleichen Schichten der Instanzen direkt miteinander. Dies erlaubt nicht nur einen hohen Abstraktionsgrad und schafft die nötige Transparenz zur Fokussierung, sondern gestattet auch den wahlfreien Austausch der unteren Schichten. Die Verwendung von Kommunikationsebenen ist die Vorraussetzung für die heutigen heterogenen Netzwerke.

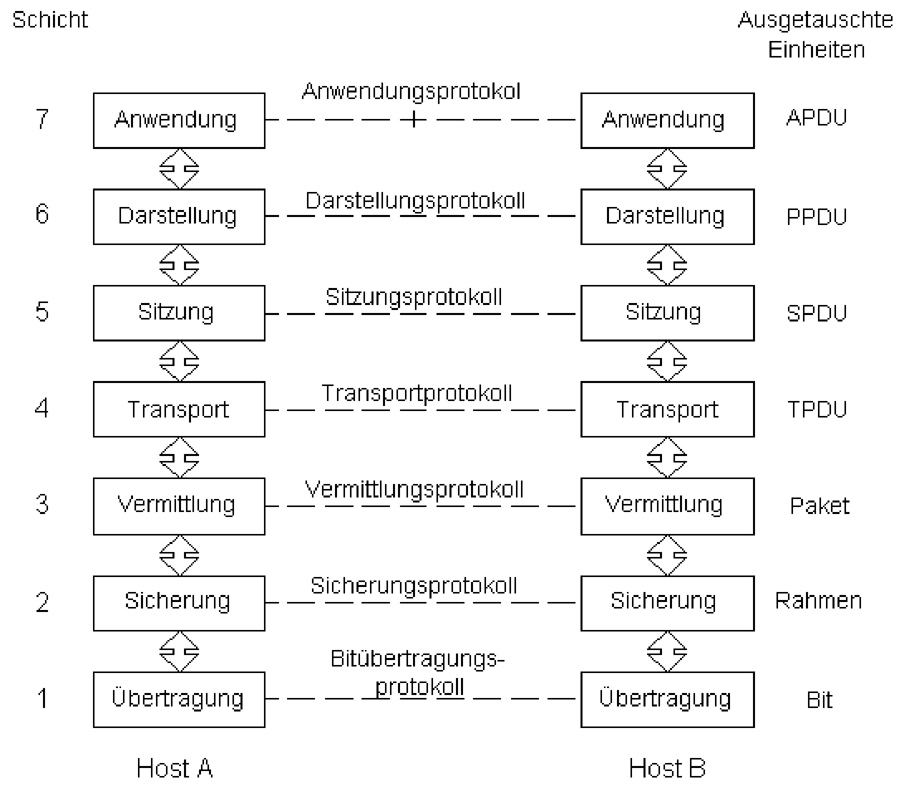


Abbildung 2.1: Das OSI-Referenzmodell

- Die *Anwendungsschicht* ist die Schicht, in der Anwendungen auf die Netzwerkdienste zugreifen. Diese Schicht enthält die Dienste, die Anwendungen direkt unterstützen wie zum Beispiel elektronische Post.

- Die *Darstellungsschicht* übersetzt die Daten der Anwendungsschicht in ein einheitliches Standardformat. Diese Schicht ist auch für die Datensicherheit zuständig. Sie kann Dienste wie Datenverschlüsselung oder -kompression anbieten. In dieser Schicht können auch landesspezifische Eigenschaften kodiert werden, zu nennen wäre hier die Übersetzung des Datums oder beispielsweise die Umwandlung eines boolschen Wertes in das jeweilige Wort in Abhänigkeit von der Landessprache. Die Darstellungsschicht kann ferner für die Auswahl eines gültigen Kontexts für die Anwendungsschicht Sorge tragen, wie dies für die vorliegende Diplomarbeit von Bedeutung ist.

- Die *Sitzungsschicht* erlaubt es zwei Anwendungen auf verschiedenen Instanzen im Netzwerk, eine Sitzung aufzubauen, zu nutzen und zu beenden. Eine Sitzung ist die Direktverbindung zwischen zwei Instanzen. Diese Schicht etabliert eine Dialogkontrolle zwischen den beiden Computern, reguliert die Länge der Datenübertragung und kümmert sich um die Empfangs- und Sendekontrolle.

- Die *Transportschicht* behandlt Fehlererkennung und Fehlerkorrektur. Sie verwandelt den Strom der Sendedaten in Datenpakete für die Übertragung und umgekehrt. Diese Schicht ist auch für das Versenden von Empfangsbestätigungen zuständig.

- Die *Netzwerkschicht* addressiert die Datenpakete und konvertiert logische Adressen und Namen in physikalische Adressen und Namen. Sie bestimmt den Übertragungsweg vom Quell- zum Zielrechner und andere Probleme im Netzwerkverkehr wie Leitungsumschaltung, Umleitungen oder Stockungen.

- Die *Verbindungssschicht* konvertiert die zu sendenden Datenpakete in logische, strukturierte Datenpakete die als „Rahmen“ (Frames) bezeichnet werden. Diese Schicht ist für das Verschicken und den Empfang der Rahmen von einem Rechner zum anderen verantwortlich. Die Verbindungsschicht ist mittlerweile in der Regel vollständig in der Hardware heutiger Netzwerkadaptern integriert[5]. Die für Fernverbindungen im Internet bedeutsamen Protokolle der Verbindungsschicht sind PPP als auch SLIP.

- Die *physikalische Schicht* überträgt Bits bzw. Informationen die Bits repräsentieren, wie Licht, Töne, oder elektrische Impulse über das Übertragungsmedium. Diese Schicht definiert die Art der Netzwerkverkabelung und die Übertragungstechnik. Als Übertragungsmedien dienen im LAN üblicherweise Twisted-Pair, Koaxial oder Lichtleiterkabel. Im WAN werden dagegen analoge und digitale Fernsprechleitungen verwendet.

Das Internet ist auf dem TCP/IP Protokollsatz aufgebaut, der weit mehr beinhaltet, als die Protokolle TCP und IP, die jedoch aufgrund ihrer Bedeutung als Synonym für den gesamten Protokollstapel dienen. Leider fügt sich das TCP/IP Protokoll nur grob in das OSI-Modell ein, anhand dessen es jedoch regelmäßig analysiert wird. Das Abbild 2.3 zeigt die Zuordnung von TCP/IP zu den einzelnen von ISO definierten Schichten.

Aus dem Abbild wird ersichtlich, daß die Mehrzahl der Anwendungen, wozu auch das WWW zählt, im Internet auf der Transportschicht aufsetzt. Diese und alle darunter liegenden Schichten werden in der Regel durch das Betriebsystem bereitgestellt. Applikationen umfaßen die obersten drei Schichten und trennen diese allenfalls intern.

Das Client-Server Konzept der vorliegenden Arbeit setzt ebenfalls auf der Transportschicht auf, aber implementiert die Sitzungsschicht nach den Regeln des OSI-Modells. Die Darstellungsschicht der Lösung wird dem Modell lediglich in groben Zügen nachempfunden, da diese auf der Fernausführung von Kommandos basiert.

[5]Im Detail sehr aufschlußreich sind hier [3C94] 3COM: Adapter Drivers Reference, 1994, [3CA92]3COM: Sample Driver Source Code, 1992 als auch [3C90] 3COM, Microsoft: NDIS Network Driver Interface Specification Version 2.01 FINAL, 1990.

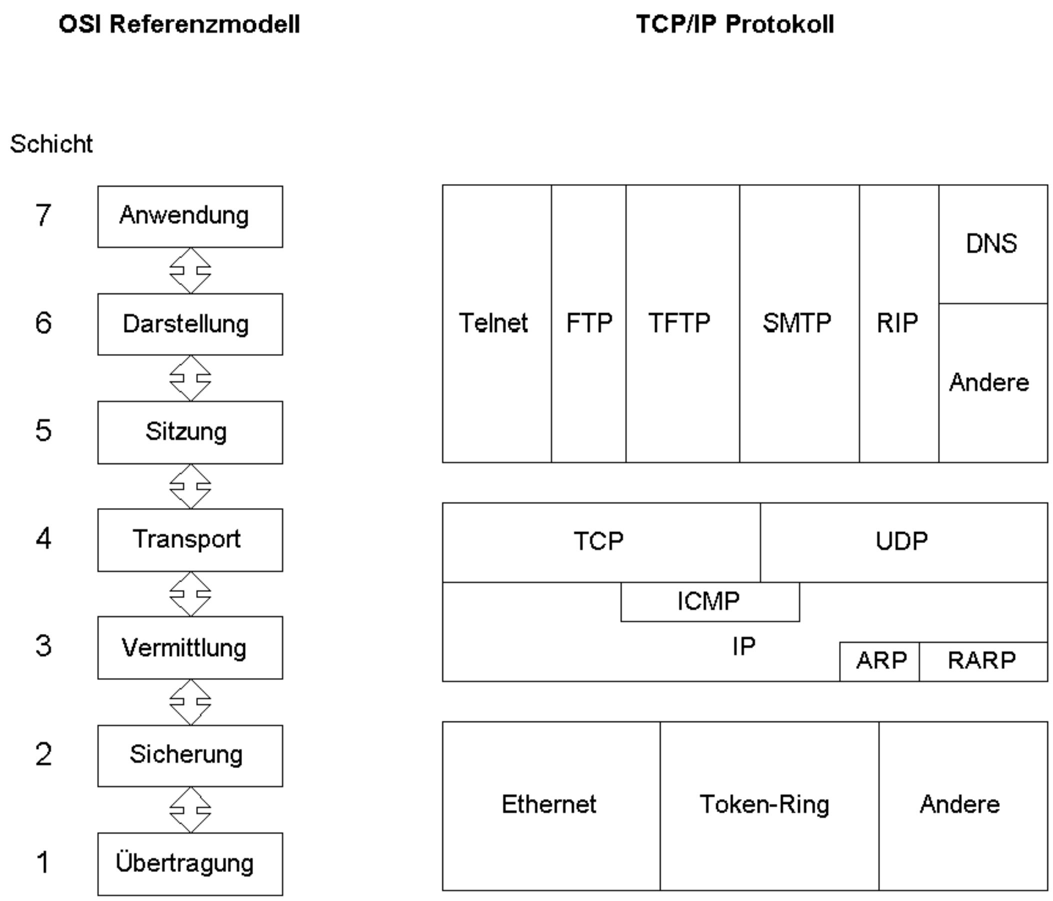


Abbildung 2.2: Einordnung von TCP/IP im OSI-Modell

Kapitel 3

Analyse der VDM Klassenbibliotek

Die Aufgabe, eine VDM Klassenbibliothek um ein weitere Komponente zu ergänzen, macht eine vorhergehende Analyse unvermeidlich, die daher keinesfalls nur theoretischen Wert hat, sondern überhaupt erst diejenigen Fakten aufdeckt, die eine erfolgreiche Implementierung erlauben.

3.1 Klassenbibliothek

Die Klassenbibliothek besteht aus einer vierstufigen Hierarchie, an dessen Ende sich jeweils ein Textfragment befindet. Zwischen den einzelnen Textfragmenten bestehen Zusammenhänge, die entweder per Definition hergestellt werden[1] oder sich aus der direkten Einbindung von Textfragmenten durch andere ergeben. Insgesamt ergibt sich ein außerordentlich komplexes Gefüge. Die Fragmente können jede Art von Daten beinhalten. Dazu zählen Parameter, Code einer Zielsprache, Makroanweisungen als auch Dokumentation.

Der Zugriff auf die Klassenbibliothek erfolgt zum einen über den Domaincompiler, der in der Lage ist, aus den Inhalten der Klassenbibliothek Quellcode für eine DSL-Spezifkation zu erzeugen und zum anderen durch den Datenbankmanager, der von weiteren Tools für den Zugriff auf die Bibliothek als Basis dient. Auch weitere Tools greifen direkt auf die Datenbank der Klassenbibliothek zu. Sie werden jedoch vernachlässigt, da ihnen keine Bedeutung für die vorliegende Arbeit zukommt, und sie teilweise bereits durch komfortablere ersetzt wuden[2].

Insgesamt ist festzustellen, daß die Bibliothek mit den Entwicklungen gewachsen ist, und die vierstufige Struktur mittlerweile keinesfalls mehr ausreichend ist, die Elemente angemessen zu ordnen. Nach Beurteilung des Autors sind zu viele Sonderfälle entstanden, die den Einstieg und den Umgang mit der Bibliothek erschweren, wenn es nicht nur die reine Anwendung betrifft, für welche Kenntnisse über die DSL ausreichen.

[1] Dies betrifft beispielsweise Signaturen, die von allen Tools der Klassenbibliothek explizit verwendet werden

[2] Wie beispielsweise vdminfo

Aus den ADTs der Bibliothek könnte in einer neuen Revision der Klassenbibliothek eine Struktur aufgebaut werden, die im Aufbau relationalen Datenbanksystemen nachempfunden wird, die über eine Systemtabelle alle weiteren Tabellen mit ihrem detailierten Aufbau verwalten. Die Definition der Datenbankstruktur über die Datenbank selbst, kann die Definitioen vieler Invarianten, die das Gefüge derzeit zusammenhalten, erübrigen. Auf die alternative Abbildung der Datenbank auf eine Verzeichnisstruktur, wie sie jetzt noch möglich ist, müßte dafür allerdings aller Wahrscheinlichkeit nach verzichtet werden.

3.2 Preprozessor

Der Preprozessor der VDM Klassenbibliothek stammt aus dem Jahre 1991. Der Programmname des Preprozessors lautet **vdmpp**. Als Ersteller ist ein Herr Schlotfeldt im Quellcode vermerkt. Der Preprozessor ist vollständig in der Hochsprache C erstellt, auf den Einsatz von Compilerbautools wie *Lex* oder *Yacc* wurde verzichtet. Es handelt sich bei **vdmpp** um einen vollständigen ANSI-C Preprozessor, der jedoch eine Übermenge (*superset*) der für ANSI-C definierten Strukturen verarbeiten kann.

Der **vdmpp** ist ebenso wie ein ANSI-C Preprozessor ein Makro-Prozessor, der vor der entgültigen Übersetzung des Quelltextes durch den Compiler eine Vorverarbeitung vornimt. Er wird als Makro-Prozessor bezeichnet, da er in der Lage ist, Makros, die eine Abkürzung für längere Konstrukte sind, zu substituieren. Das Zeichen „#“ spielt die entscheidene Rolle bei der Identifikation der Makros.

Die wesentlichen Funtionen des Preprozessors sind:

- Einbinden von Dateien
- Makro-Erweiterung
- Zustandsbedingte Übersetzung
- Zeilenkontrolle

Die Elemente, die durch einen ANSI-C Preprozessor verarbeitet werden, sollen in Kürze dargestellt werden, um die Funktionsweise zu analysieren. Im Anschluß erfolgt die Dartstellung der erweiterten Möglichkeiten durch den **vdmpp**.

1. **Globale Transformationen**

   Ersetzung von Kommentaren sowie Rückstrich-Seitenumbrüchen durch Leerzeichen als auch von vordefinierten Makros durch deren Erweiterung

2. **Preprozessorkommandos**

   Interpretation der Preprozessor-Kommandos, die mit „#“ beginnen. Der Kommandosatz des Preprozessors ist fest und kann nicht zur Laufzeit erweitert werden. Bekannte Kommandos in diesem Zusammenhang sind „#ifdef“, „#if“, „#else“, „#elseif“, „#endif“ als auch „#undef“.

3. **Dateieinbindung**

   Mit Hilfe des `#include` Preprozessorkommandos können weitere Dateien in den zu verarbeitenden Quelltext eingebunden werden.

4. **Einfache Makros**

   Einfache Beispiele hierfür sind

```
#define BUFFER 1024
#define BUFLEN BUFFER
```

5. **Makros mit Argumenten**

   Definition von Makros als Funktiondeklaration[3], wie folgende Beispiele zeigen

```
#define min(x,y) ((x) < (y) ? (x):(y))
#define foo(x) -1 / (x)
#define foo2 (x) -1 / (x)
```

   Beachtenswert ist, daß das Makro *foo* der obigen Beispiele durch *-1 / (x)* ersetzt wird, wogegen aufgrund des Leerzeichens bei *foo2* die Substitution durch *(x) -1 /x (x)* erfolgt.

6. **Stringumwandlung (stringification)**

   Definition von Makros in Form einer Funktionsdeklaration, wie folgende Beispiele demonstrieren

```
#define str(x) #x
#define NOTE(notification) \
        fprintf(stderr,"Please note: #notification ");
```

   Die Ersetzung mittels *stringification* führt mehr aus, als die bloße Umrandung des Textes mit Anführungszeichen. Damit ein gültiger String ensteht, werden die in Zeichenketten verwendeten Escape-Codes durch den Preprozessor eingesetzt.

7. **Verbindung (concatenation)** Verbindung von Elementen im Makrorumpf durch den operator „##“

```
#define NEWENTRY(name) { #name, name ## _entry }
```

   Der folgende Aufruf dieses Makros

[3] Auf die Gefahren durch Seiteneffekte wird hinreichend in der Literatur hingewiesen. Sie sind dem gewieften Programmierer bekannt, so daß hier nicht darauf eingegangen wird

```
struct {char *name, void(*function) () } list [] =
{
NEWENTRY(foo) ,
NEWENTRY(test)
}
```

ergibt substituiert

```
struct {char *name, void(*function) () } list [] =
{"foo", foo_entry },
{"test", test_entry }
};
```

Zu beachten ist bei der Verbindung durch den „##“-Operator, daß Leerzeichen und Tabulatoren (*whitespaces*) vor und nach dem Operator entfernt werden. Der Verbindung von Ausdrücken kommt bei der Anwendung der VDM Klassenbibliothek eine außerordentlich wichtige Rolle zu. Die Typengenerierung eines ADTs auf eine Domäne für C erfolgt bei der Umsetzung durch eine Zusammensetzung der Operationsbezeichner aus dem Namen der Operation sowie des Domänennamens. Eine Umsetzung in C++ basiert auf den für C generierten Funktionen, die von den Methoden der Klassen verwendet werden, dessen Klassenname dem Domnänennamen entspricht.

Der VDM Preprozessor von Schlotfeldt kennt zusätzlich zum ANSI-C Prozessor weitere Funktionen, die für die Umsetzung der derzeitigen Inhalte der VDM Klassenbibliothek von außerordentlicher Bedeutung sind. Schlotfeldt verweist im Quellcode hinsichtlich der Details auf die externe Dokumentation, die dem Autor dieser Diplomarbeit jedoch leider nicht vorliegt. Die Besonderheiten des VDM Preprozessors wurden daher anhand der Bibliotheksinhalte sowie des Quellcodes ermittelt, weshalb dem Anspruch nach Vollständigkeit an dieser Stelle nicht Rechnung getragen werden kann.

Die Ergänzungen des VDM Precompilers zum ANSI-C Preprozessors sind im wesentlichen:

1. **Globale Stringumwandlung**

   Textzeilen können an jeder Stelle, ohne daß es der Verwendung eines Makros bedarf, durch den „""#“-Operator (Zwei Doppelhochkommata gefolgt vom Doppelkreuz) am Zeilenbeginn, in eine Zeichenkette gewandelt werden. Wird das Zeichen als solches benötigt, wird diesem als Escape-Metazeichen ein Rückstrich (*backslash*) vorangestellt.

2. **Globale Stringverbindung**

   Bei diesem Feature handelt es sich um die wohl wichtigste Erweiterung zum ANSI Standard. Fast sämtliche Codefragmente der VDM Datenbank sind auf diese Möglichkeit angewiesen. Der Einsatz soll beispielhaft demonstriert werden:

```
ELEM2 checkread##_DOMAIN _FPH1(R2##_DOMAIN)
        { x1 = ovwrt_rd(x1,fread##_ELEM2(x2)); }
```

Der Leser wird gebeten, nicht zu versuchen, die Logik dieser Codierung zu verstehen, denn sie hat keine. Aufmerksamkeit ist jedoch den lexikalischen Verbindungen *checkread##_DOMAIN* , *R2##_DOMAIN* und *(fread##_ELEM2(x2))* zu schenken. Es wird an diesen Stellen eine Stringverbindung hergestellt, ohne daß es sich um eine Makrodefinition oder einen Makrorumpf handelt. Die Verbindung erfolgt inmitten des Textes. Das obige Beispiel demonstriert, wie anhand von Domänen gemeinsam mit Operationen Methoden und Funktionen für den Quelltext der Hochsprache erzeugt wird.

3. **Entfernen doppelter Leerzeilen und sonstige Features**

   Der Preprozessor ist ferner zu einer Entfernung von doppelten Leerzeichen, automatischer Stringverbindung und sonstigen Substitutionen in der Lage, dessen Nützlichkeit für die VDM Datenbank jedoch nicht mit der globalen Stringverbindung gleichzieht. Zu einer genaueren Beschreibung wird hier auf die Dokumentation des Erstellers Schlotfeldt verwiesen.

Kritik am Preprozessor

Abschließend soll (konstruktive) Kritik am Preprozessor nicht fehlen. Die Kritikpunkte betreffen genauso Feststellungen, die der Autor während der Bearbeitung der Diplomarbeit, treffen mußte als auch (für den Autor) wünschenswerte Erweiterungen.

Auffällig ist, daß der **vdmpp** alle Funktionen des ANSI-C Preprozessor unterstützt, jedoch nicht als vorzuziehender Preprozessor für den ANSI-C Prozessor in Durchführung zweier Übersetzungsläufe konzipiert ist, was den Umfang des Programmes erheblich reduziert hätte[4].

Die Methode der Zusammensetzung von Konstrukten ist zwar geeignet, die Umsetzung der DSL in Hochsprachen zu gestalten, sie schafft jedoch nur einen geringen Abstraktionsgrad, der das Verstehen und den Überblick über die Codefragmente erschwert und damit eine Fehleranfälligkeit mit sich bringt.

Der Autor hätte anstelle der Implementierung des ANSI-C Standards für den Preprozessor eine mehr problemorientierte an BNF angelehnte Formulierungsmöglichkeit bevorzugt, auch um Makros des VDM Preprozessors von den Makros der Zielsprachen C und C++ abzusetzen - selbst wenn es sich bei den Codefragmenten um Interna der Klassenbiliothek handelt. Transparenzprobleme wurden insbesondere darin gesehen, daß Makros, die keine Argumente erwarten, nicht von Text zu unterscheiden sind, wodurch zum einen unklar ist, ob es sich um ein Makro handelt und zum anderen die Gefahr besteht, daß versehentlich Text durch Makros ersetzt werden könnte.

Ein weiterer Kritikpunkt betrifft die Sonderzeichenverarbeitung des Preprozessors. Makrodefinitionen sind lediglich alphanumerisch vornehmbar, was den Unterstrich einschließt. Unverwechselbare, kryptische Makros sind hiermit nicht formulierbar. Ferner ist durch den Preprozessor die Verarbeitung von Makros mit einer variablen

[4] Wobei von einer generellen Faulheit aller Programmierer ausgegangen wird

Anzahl von Argumenten nicht möglich, was zu beispielsweise zu folgenden Konstrukten in der Datenbank führt:

```
#define FPAR1(t1) ()
#define FPAR2(t1,x1) (x2)
#define FPAR3(t1, x2  ,x3) (x2, x3)
usw.
```

Als Konsument des VDM Preprozessors, sind für den Autor Ansprüche dieser Art leicht formulierbar, er übt diese Kritik an dieser Stelle jedoch aus der Sicht als „Anwender“ unter Verdrängung des Realisierungsaufwandes.

Bei der Diagnose des Datenbankmanagers wurde festgestellt, daß das Einbinden von Dateien, beispielsweise durch `#include <stdlib/aux/setup/com>` für das ursprüngliche DIRECTORY-Modell ausgelegt ist, also für die Einbindung von physikalischen Dateien. Im FLAT-Modell führt die Verarbeitung des `#include` Kommandos zwangsweise zu Fehlern, da lediglich eine Bibilotheksdatei existiert. Als Umgehung (*workaround*) werden von allen Tools der Klassenbibliothek im FLAT-Modell alle Einbettungsanweisungen vor dem Aufruf des Precompilers ausgeführt.

Der Preprozessor ist ebenfalls im Domaincompiler, der im folgenden Abschnitt beschrieben wird, als Bibliothek eingebunden. Auch hier gilt, daß Einbettungsanweisungen vor der Vorverarbeitung aufgelöst werden müssen. In diesem Zusammenhang wird die Frage aufgeworfen, ob der Preprozessor als separates Programm überhaupt noch seine Existenzberechtigung hat, zumal der Quellcode des Preprozessors in der Entwicklungsinstallation der Klassenbibilothek auch doppelt vorgehalten wird[5].

3.3 Domaincompiler

Es ist insbesondere in der Softwarentwicklung nicht sinnvoll, das Rad neu zu erfinden. Der Entwickler greift zur Vermeidung von Ressourcenverschwendung in der Regel auf Bibliotheken (*librarys*) zurück. Ebensolche Ressourcen, die auch in der VDM Klassenbibliothek enthalten sind. Der Zugriff auf diese erfolgt durch den Domaincompiler mit dem Namen **vdm**. Der Domaincompiler wird in der Regel durch ein Shell-Skript mit dem Namen **VDM** aufgerufen, das die Parametrisierung vereinfacht.

Die Arbeitsweise des Domaincompilers ist durch das VDM-Handbuch von Jan Peter Mohr [JM95] hervorragen beschrieben. Es soll daher nur ein Abriß über den Domaincompiler wiedergegeben und auf diese Fundstelle verwiesen werden, um eine schlichte Wiederholung zu vermeiden.

Der Domaincompiler liest eine Datei ein, die DSL-Spezifikation enthält[JM95, S.143]. Der mit lex und yacc erzeugte Scanner und Parser des Domaincompilers verarbeitet die Syntax, und extrahiert aus der Klassenbibliothek die benötigen Fragmente. Da der Objektcode des VDM Preprozessors im Domaincompiler enthalten ist, kann eine

[5]Zum einen im Verzeichnis ./vdmpp, zum anderen unter ./domc/Vdmpp, ausgehend vom Basisverzeichnis der Installation

direkte Verarbeitung der Elemente noch durch den Compiler selbst erfolgen, der die vormals abstrakten Datentypen wie sets, tuples, maps, functions, trees oder bags konkretisiert und diese für die gewählte Zielsprache in Quellcode umsetzt.

Der Quellcode wird durch einen Compiler, der die Zielsprache verarbeitet, gemeinsam mit dem Quellcode der Applikation, welche die Bibliothek verwendet, übersetzt und gebunden.

3.4 Datenbankmanager

Der Datenbankmanager ermöglicht den einheitlichen Zugriff auf die Datenbasis der VDM Klassenbibliothek und schafft damit eine allgemeine Schnittstelle, die den Zugriff und die Verwaltung der Daten unabhängig vom Datenbankmodell gestattet. Der Datenbankmanager besteht beantwortet Anfragen über eine Textschnittstelle, die Kommandos erwartet.

3.4.1 Struktur des Datenbankmanagers

Der gelinkte VDM Datenbankmanagers trägt den Namen **dbm** und setzt sich aus folgenden Komponenten zusammen

1. **VDM**

   C- als auch hierauf aufbauender C++ -Quellcode der VDM-Klassenbibliothek, erzeugt anhand der in **dbm.vdm** hinterlegten DSL-Spezifikation, welche die Struktur der Datenbank, die verwendeten Implementationen als auch die Definition der auf die Datenbank anwendbaren Funktionen enthält.

2. **Hochsprachen**

   C++ Quellcode, bestehend aus den Dateien

   - **vdmdbm.cc**, welche die Implementationen der in dbm.vdm definierten Funktionen enthält, und, da auch Schnittstelle zum VDM-System, Kern des Datenbankmanagers ist,
   - **dbmtcl.cc**, welche TCL Kommandos als Schnittstelle zwischen Kern des Datenbankmanagers und der TCL-Shell einrichtet,
   - **vdmtcl.cc**, welche zur Erleichterung Makroerweiterungen für **dbmtcl.cc** herstellt und deshalb der Ordnung halber eigentlich dbmtcl.hpp heißen müßte.

3. **TCL**

   - Der Library **libtcl?.?.a**[6] welche den Objektcode der Tool Command Language enthält und aus dem verfügbaren C-Quellcode von TCL erzeugt wird.

[6]Die library von TCL trägt ab der Version 7.4 die Versionsnummer am Ende, also z.B. libtcl7.4.a, Die librarys der Vorgänger heißen regelmäßig libtcl.a. In den Makedateien wurde dies berücksichtigt.

- TCL Interpretercode mit den die Kommandos des Datenbankmanagers definiert bzw. erweitert werden. Der nötige Interpreter hierzu stammt aus der gebundenen TCL-Library und ist fester Bestandteil des Datenbankmanagers.

Das Resultat der Bindung aller Komponenten ist eine TCL-Shell mit dem Namen **dbm** und erweitertem Befehlsatz, die als Anwendungsschnittstelle den Zugriff auf die VDM-Datenbank ermöglicht, ansonsten jedoch mit der üblichen TCL-Shell identisch ist. Ebenso wie für die Standardshell **tcl** oder auch **tclsh** sind für den Datenbankmanager zwei Ausführungsmodi zulässig. Dies ist zum einen der interaktive Modus, der im Dialog mit dem Benutzer durchgeführt wird, und zum anderen die Interaktion als Filter. Die Shell ermittelt selbstätig anhand der Standardein- und ausgabeströme ob diese mit einem Terminal oder Pipes verbunden sind und verwendet den entsprechenden Modus.

3.4.2 Modelle des Datenbankmanagers

Als externe Repräsentationen sind zwei Modelle definiert, für die der Manager erzeugt werden kann. Dies ist zum einen das FLAT-Modell, in der die Bibliothek in einer einzigen Datenbankdatei verwaltet wird, das weitere Modell ist die Abbildung der Datenbank auf einen Verzeichnisstruktur und wird als DIRECTORY-Modell bezeichnet. Die DSL-Spezifikation, ist nachfolgend für beide Modelle angegeben.

Domänen

$$
\begin{array}{lcll}
VDMDatabase & = & ADTName & \xrightarrow{m} & ADT \\
ADT & = & ImplementationName & \xrightarrow{m} & Implementation \\
Implementation & = & OperationName & \xrightarrow{m} & Operation \\
Operation & = & ComponentName & \xrightarrow{m} & Component
\end{array}
$$

$$
\begin{array}{lcl}
ADTName & = & Str \\
ImplementationName & = & Str
\end{array}
$$

$$
\begin{array}{lcl}
OperationName & = & Str \\
ComponentName & = & Str \\
Component & = & Str
\end{array}
$$

Implementationen

A. Modell FLAT

$$
\begin{array}{lcl}
VDMDatabase & = & linklist + ... \\
ADT & = & linklist + ... \\
Implementation & = & linklist + ... \\
Operation & = & linklist + ...
\end{array}
$$
[7]

[7] An dieser Stelle folgen regelmäßig der Implementationsbeschreibung die „Enrichments“. Dies sind die als zulässig definierten Operationen, die hier der Übersicht halber regelmäßig durch die Auslassung angedeutet sind. Die Vollständige Definition ist in der DSL-Datei **dbm.vdm** des Datenbankmanagers hinterlegt.

$$Str \quad = \quad + ...$$ [8]

B. Modell DIRECTORY

$$\begin{aligned} VDMDatabase &= dir + ... \\ ADT &= dir + ... \\ Implementation &= dir + ... \\ Operation &= dir + ... \\ \\ Str &= + ... \end{aligned}$$

Im FLAT-Modell ist für den Datenbankmanager ein „Laden“ der Datenbank notwendig. Das bedeutet, die ADTs der Datenbankstruktur werden anhand der Datenbankdatei im Arbeitsspeicher wiederhergestellt. Für ein weiteres Modell für den Datenbankmanager würde sich, die Implementation „extern“ für die Struktur anbieten, die es erlaubt, die Datenbank auch direkt als Datei zu verwenden.

3.5 Bibliotheksmanager und -browser

Der Bibliotheksmanager **vdmdbm** und der Bibliotheksbrowser **vdmhelp** arbeiten kooperativ mit dem Datenbankmanager **dbm** zusammen. Bei beiden handelt es sich um ein TK-Skript, das mit der TK-Shell **wish** ausgeführt wird und eine graphische Oberfläche darbietet. Sie öffnen den Datenbankmanager als Filter und richten lassen alle Operationen, welche die Datenbnk betreffen, von diesem bearbeiten.

Der Bibliotheksmanager **vdmdbm** wird zur Pflege der Klassenbibliothek eingesetzt. Er erlaubt die Erweiterung und Reduzierung um ADTs, deren Implementationen, Operationen als auch Komponenten. Letztere können mit Hilfe eines externen Editors verändert werden, so daß die Pflege der gesamten Klassenbibliothek durch den Bibliothekbrowser möglich ist.

Der Datenbankmanager **dbm** wird über gleiche Kommunikationsmethode vom Bibliotheksbrowser **vdmhelp** für den Zugriff auf die Datenbank benutzt. Der Browser ist jedoch auf die Dokumentation, die in der Klassenbibliothek hinterlegt ist, spezialisiert. Er erlaubt die Anzeige der Beschreibungen zu ADTs, Implementationen und Operationen, die der Datenbankmanager auf Anfrage des Browsers erzeugt und liefert.

[8] Die Implementation Str zählt zu den vordefinierten atomaren Datentypen, weshalb die Implementation implizit gegeben ist. Die Operatoren sind dennoch wahlfrei.

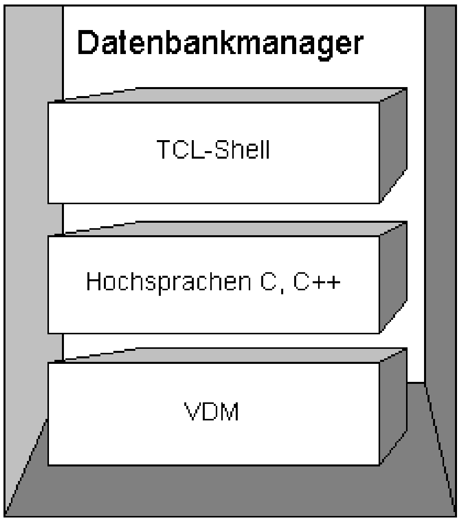


Abbildung 3.1: VDM Datenbankmanager

Kapitel 4

Projektmanagement und -planung

Es drängte sich bei der Planung der Lösung die Frage auf, inwieweit diese mit VDM modelliert werden kann, zumal es sich auch um eine Erweiterung einer VDM-Klassenbiliothek handelte. Aus den Anforderungen an die Lösung, Modifikationen bei Bewahrung des bestehenden Hochsprachencodes in der Tool Command Language (TCL) vorzunehmen, ergibt sich zum einen die Schwierigkeit, TCL aufgrund der besonderen Interpretereigenarten mit VDM, META-IV bzw. hier konkret mit DSL zu verbinden, zumal sich TCL mit Hilfe der Klassenbibliothek auch selbst nicht erzeugen läßt.

Zum anderen war bereits nach der Analyse der Klassenbibliothek erkennbar, daß erhebliches Prototyping notwendig sein würde, um die gestellte Aufgabe hinsichtlich der Realisierbarkeit zu überprüfen, da zahlreiche technische Aspekte zu berücksichtigen waren, die „Laborversuche" nötig machten, um auftretende Effekte nicht dem Zufall zu überlassen. In diesem Zusammenhang waren auch Einblicke in den Quellcode des Interpreters erforderlich, da weder Literatur noch Dokumentation von TCL Auskunft gaben.

TCL verfügt als Interpretersprache über eine besondere Form der späten Bindung, dessen treffenster Vergleich wohl die Veränderung von Maschinencode in einem laufenden Programm ist. Eine Unterscheidung von Code und Daten ist in TCL wohl möglich, eine Trennung nicht. Sowohl Code als auch Daten bestehen aus Text, Daten können jederzeit als Code ausgeführt werden. Weder META-IV noch DSL unterstützen Polymorphie, so daß sich hier weitere Gedankengänge erübrigt haben.

Gleichermaßen wird eine Modellierung von parallelen Prozessen durch VDM nicht unterstützt, welche jedoch unweigerlich zur Lösung benötigt werden. Dennoch läßt sich der Kontext einer Sitzung mit Hilfe von VDM spezifizieren und ist konkret durch Erweiterung der DSL-Bestandteile des VDM-Datenbankmanagers in die Praxis umgesetzt.

4.1 Anforderungen

Die Anforderungen an das System leiten sich zum einen aus dem Thema der Diplomarbeit als auch aus der bisherigen Funktionalität der Klassenbibliothek ab. Eine

Mindestanforderung für das Informations- und Hilfesystem ergibt sich bereits aus dem vorhandenen Bibliotheksbrowser, der in der Lage ist, in der Bibliothek hinterlegte Dokumentation anzuzeigen, aber auch mit keiner weiteren Funktionalität aufwartet. Es ist sehr schwierig, mit diesem System Zusammenhänge darzustellen, da keine Möglichkeit besteht, Verknüpfungen herzustellen. Es ist daher vorwiegend für den professionellen Benutzer geeignet, der genau weiß, an welcher Stelle die entscheidenen Informationen zu finden sind. Gefordert wird von dem neuen System, daß Verknüpfungen so weitreichend wie möglich gestaltet werden können.

Die Systemumgebung der Klassenbibliothek mit allen Tools ist UNIX. Diese Beschränkung wird durch die Verwendung von HTML für die Darstellung aufgehoben, da über heterogene Netzwerke gearbeitet werden kann, und HTML-Browser für jegliche Systeme verfügbar sind. Der Ausbau der Tools muß jedoch vor dem Hintergrund erfolgen, die derzeitige Portabilität der gesamten Klassenbibliothek an sich zu bewahren.

Als Schnittstelle ist der Datenbankmanager zu verwenden. Als Basis für die nötigen Datenbankzugriffe, sollen als Vorlage die Abläufe, die durch Bibliotheksbrowser und Bibliotheksmanager vorgegeben werden, dienen. Als Datenbasis für das neue Hypertextsystem dient ausschließlich die Datenbank der Bibliothek. Funktionale Beschreibungen des Systems werden, soweit es keinen Interpretercode betrifft, formal durchgeführt.

Anforderungen an die Effizienz des Systems bestehen nicht. Es wird von ausreichend mit Hauptspeicher und Rechenleistung versehenen Maschinen ausgegangen. Da die Benutzer des Systems ohnehin nur über das Netzwerk auf das Hypertextsystem zugreifen können, da für die Anzeige von HTML über Browser ohnehin das HTTP-Protokoll benötigt wird, und dieses eine Netzwerkverbindung zwischen Web-Server und Browser voraussetzt, muß Rechenleistung auf der Server-Maschine konzentriert werden[1].

Von der Wartung und Weiterentwicklung des Systems soll vorwiegend die Modifikation des Interpretercodes betroffen sein. Veränderungen am Hochsprachencode sollen nur dann erforderlich werden, wenn ein Strukturwandel der Datenbank diese ohnehin erfordert und ist ansonsten dauerhaft ausgelegt.

4.2 Ableitung und Definition der Teilziele aus grundlegenden Problemen

4.2.1 Zwingende Notwendigkeit einer Client-Server Lösung

Eine grundlegende Erkenntnis aus der Analyse des Datenbankmanagers unter 3.4 ist, daß die VDM-Datenbank im FLAT-Modell[2] im Arbeitsspeicher vorgehalten wird. Die Datenbankdatei wird vom VDM Datenbankmanager in den Arbeitsspeicher geladen

[1] Es wird hiebei davon ausgegangen, daß Web-Server und Datenbankmanager in der Form als VDM Server auf gleichem Host ausgeführt werden, was durchaus nicht zwingend ist

[2] „Flaches“ Modell bedeutet in diesem Zusammenhang, daß die konventionelle Verzeichnisstruktur der Inhalte auf eine Datenbankdatei abgebildet ist.

und hat derzeit eine Größe von ca. 1 MB. Dieser einmalige, zur Initialisierung des Datenbankmanagers notwendige Vorgang, benötigt nicht unerhebliche Rechenzeit, was auch mit durch den Zugriff auf Betriebsmittel bedingt ist.

Bei Maschinen mit virtuellem Adreßraum, dies ist bei den Portierungen der VDM-Klassenbibliothek regelmäßig der Fall, ist ferner nicht ausgeschlossen, daß Teile der Datenbank auf Festplattenspeicher ausgelagert sind. Versuche mit einem mit unzureichenden 4 MB Hauptspeicher ausgestatteten Linux System, bestückt mit einem 40-MHz getakteten AMD 80386 Prozessor, haben gezeigt, daß die Ladezeiten bis zu 6 Sekunden betragen können. Selbst ein mit 100-MHz Pentium Prozessor und 16 MB ausgestattetes System benötigt ca. 1 Sekunde zum Laden der Datenbank. Selbst die SUN Sparc20 der FH-Wedel mit 256 MB Hauptspeicher schneidet hier teilweise nicht besser ab[3].

Wie eingangs unter 2.2.2 beschrieben, handelt es sich bei dem verwendeten Zugriffsprotokoll HTTP für das Informations- und Hilfesystem um ein „lightweight“-Protokoll. Dies bedeutet, daß jeder Neuzugriff auf die Serverseite von allen vorhergehenden Zugriffen unabhängig ist und macht es unmöglich, eine dauerhafte Verbindung zwischen Browser und Web-Server herzustellen, die über die Übertragung einer Web-Seite hinausgeht.

Die direkte Verwendung des CGI-Gateways für den Zugriff auf den Datenbankmanager würde dazu führen, daß die VDM-Datenbank für jeden Client erneut in den Arbeitsspeicher geladen würde. Dies wäre ein nicht zu vertretender Aufwand. Selbst wenn das Ziel lediglich ein Single-User System sein würde, und festzustellen ist, daß sich durch den Festplattencache die Ladezeiten bei mehrmaligem Zugriff erheblich verbessern, sind die Reaktionszeiten jedoch keinerzeit ausreichend genug, um das erneute Laden der Datenbank als auch des nicht weniger umfangreichen Datenbankmanagers bei jedem Datenbankzugriff zu rechtfertigen.

Die zu ziehenden Konsequenzen aus den Beobachtungen sind

- entweder auf das FLAT-Modell der VDM-Klassenbibilothek zu verzichten, und auf die Verzeichnis-Implementation auszuweichen was die Inkaufnahme einer halben Lösung bedeutet hätte
- oder eine Client-Server Lösung zu schaffen, mit einem Server, der in der Lage ist, die VDM-Datenbank einmalig kontinuierlich im Arbeitsspeicher zu halten und allen Clients zur Verfügung zu stellen.

Die erstgenannte Alternative wurde zwar in Erwägung gezogen, aber verworfen, da die Entwicklung der Klassenbibliothek zweifelsohne vom Verzeichnis- auf das FLAT-Modell übergeht und dies eine Behinderung der Innovation bedeutet hätte. In der Analysephase wurde zunächst der VDM Bibliotheksbrowser betrachtet und als Basisobjekt für das Informations- und Hilfesystem herangezogen. Bei der Pflege der VDM Klassenbibliothek erwies es sich zunächst als umständlich und mühevoll, jede Änderung mit einer Speicherung der Bibliothek abzuschließen, um diese zum Test anschließend erneut zu laden.

[3] Sie steht jedoch auch nicht exklusiv zur Verfügung und bearbeitet auch zahlreiche Prozesse

Durch Prototyping zu Analysezwecken wurde zunächst der Bibliotheksbrowser in ein Client-Server Modell verwandelt. Anschließend folgte der Bibliotheksmanager, der die Pflege der VDM Datenbank unterstützt. Auch im Hinblick einer integrierten Gesamtlösung schien es erforderlich, Augenmerk auf eine gemeinsame Datenbasis für alle Tools der Klassenbibliothek zu richten. Die Realisierung der Lösung mit Client-Server Struktur beseitigt die Problematik, die aus dem HTTP-Protokoll erwächst, fördert die Integration der Tools, vereinfacht die Pflege der Klassenbibliothek und bietet darüber hinaus weitergehende Möglichkeiten im Hinblick auf Zugriffskontrolle und das Arbeiten im Netzwerkverbund.

4.2.2 Einführung von Multisessionfähigkeit

Da jede Verbindung eines HTML-Browsers zum Informations- und Hilfesystem unabhängig vom jedem vorausgegangenen Zugriff ist, läßt sich keine verbindliche Zuordnung zwischen Browser und Server vornehmen. Zwar erlaubt das CGI-Interface, die Internetadresse der Gegenstelle zu ermitteln, der einem Zugriff zugewiesene Port ist jedoch regelmäßig verschieden. Nur die Internetadresse mit dem Port gemeinsam bilden zusammen eine eindeutige Identifizierung einer Verbindung[4]

Eine *Verbindung* (*connection*) ist als eine Kommunikationsverbindung zwischen 2 Prozessen definiert. Als *Zuweisung* (*association*) wird ein Fünfertupel bezeichnet, das eine Verbindung ausmacht:

{*protocol, local-addr, local-process, foreign-addr, foreign-process*}

Durch die lokale und und auswärtige Adresse (*local-addr, foreign-addr*) werden die Netzwerk und Host-IDs des lokalen und fernen Hosts angegeben.[5] Durch *local-process* und *foreign-process* werden die miteinander verbundenen Prozesse spezifiziert. Das Format dieser Größen hängt vom Protokollsatz *protocol* ab. In der vorliegenden Arbeit wird lediglich das bedeutsamste Protokoll TCP verwendet. UDP und weitere hier nicht im Detail behandelte Protokolle wären SNA, IPX, XNS (ein Vorläufer von IPX), OSI, AppleTalk, NetBEUI für die ebenfalls gleiche Zuweisungen gelten.

Eine korrekte Zuweisung für das TCP Protokoll könnte lauten:

{*tcp, 192.43.235.2, 1500, 192.43.235.6 , 1117* }

Die lokale Adresse wäre 192.43.245.2, der lokale Prozeß 1500, wogegen die auswärtige Adresse 192.43.245.6 und der auswärtige Prozeß 1117 ist. Jeweils eine halbe Zuweisung (*half association*), {*protocol, local-addr,local-process*} und {*protocol, foreign-addr, foreign-process*} ist der Endpunkt der Kommunikation. Für das TCP Protokoll wird dieser Endpunkt als *Socket* (in BSD Terminologie) oder *Transportadresse* (System V) bezeichnet. Die Prozesse (*local-process* und *foreign-process*) werden im TCP (als auch im UDP) durch Kanäle (ports) spezifiziert.

Nun läßt sich über die Parameter des CGI-Gateways *REMOTE_ADDR* [6] die Internet-Adresse und *REMOTE_HOST*[7] der Internet Domänen Name des Hosts der anfragt

[4] Vergl. [3C90, S. 211] Stevens: Programmieren von UNIX-Netzen.

[5] Diese können auch identisch sein, wenn der Host mit sich selbst kommuniziert, der Klarheit halber werden hier verschiedene Hosts unterstellt.

[6] Beispielsweise 193.7.22.11

[7] Zum Beispiel diogenes.home.de

ermitteln. Der Identifikationsgehalt dieser Angaben ist jedoch identisch. Eine Übergabe des Prozesses der Gegenstelle, der mit dem HTTP-Server verbunden ist, an den CGI-Client ist in der derzeit gültigen Fassung des CGI in der Version 1.1 nicht definiert. Lediglich der Port des HTTP Servers, der Anfragen entgegennimmt, wird mit dem CGI Parameter *SERVER_PORT* mitgeteilt. Dieser ist jedoch auf Serverseite hinlänglich bekannt.

Der Fall, daß mehrere HTTP-Klienten des gleichen Hosts auf den VDM Datenbankmanager in der ursprünglichen Form zugreifen[8], bewirkt, daß vormals für eine Verbindung gültige Einstellungen des Datenbankmanagers, die eigens für bzw. durch diese Verbindung eingerichtet wurde, für den nächsten Klienten den Eintritt in einen undefinierten Zustand bedeutet. Gleiche Problematik gilt für Zugriffe aller Art auf den gleichen Datenbankmanager.

Zur Abhilfe des Problems sind zunächst zwei realisierbare Lösungsansätze denkbar:

A. Erzeugung eines Kindprozesses für jede Verbindung

B. Schaffen eines eigenständigen Kontexts für jede Verbindung

Erstgenannter Ansatz würde im FLAT-Modell[9] des VDM Datenbankmanagers bedeuten, daß die gesamte Datenbank im Arbeitsspeicher mit seiner Größe von mehr als 1 MB für jeden Klienten im Arbeitsspeicher vollständig kopiert würde. Selbst wenn dieser unvertretbare Aufwand in Kauf genommen würde, sind Änderungen an der kopierten Datenbasis durch den Klienten nach dem Beenden der Verbindung genauso verloren, wie der nicht mehr benötigte Kontext des Klienten. Ein Vorteil liegt jedoch in den definierten Anfangszuständen, die aus der Kopie des Elternprozesses erwachsen.

Durch Einsatz von mehrfach genutztem Speicher (*shared-memory*) in Verbindung mit Semaphoren zur Synchronisation der Zugriffe unter UNIX ([RS95] und [RS92] geben hier Aufschluß), ließe sich verhindern, daß die Datenbank des Datenbankmanagers für jeden Klienten kopiert wird. Der Einsatz dieser sehr systemnahen Techniken ist jedoch zumindest bedenklich, und hätte eine empfindliche Änderung bzw. Erweiterung der VDM Klassenbibliothek mit sich gebracht, da die Datenbank selbst durch ADTs der Klassenbibliothek beschrieben ist.

Im Vorgriff kann gesagt werden, daß letztendlich hauptsächlich der Lösungsansatz B weiterverfolgt, und Multisessionfähigkeit in den VDM Datenbankmanager integriert wurden. Es lassen sich nunmehr wahlfrei viele Sessions einrichten, verwenden und entfernen. Jede Session besitzt bei der Erzeugung einen definierten Anfangszustand, die Eigenschaften einer Session sind der *Kontext* dieser Session. Der Wechsel einer Session löst die Sicherung des aktuellen Kontexts der vormals aktiven Session und die Wiederherstellung des neuen gültigen Kontexts aus.

[8] Dies ist zumindest wahrscheinlich, und in der Fachhochschule Wedel beispielsweise dadurch gegeben, daß viele Studenten gleichzeitig den dort derzeit schnellsten UNIX-Host *sparc20* zum „Web-Surfen“ verwenden

[9] Auch im DIRECTORY-Modell wächst der Anteil der Datenbank im Arbeitspeicher insbesondere bei Veränderungen stetig an.

Dem Lösungsansatz A wurde in soweit Rechnung getragen, daß der VDM Datenbankmanager dahingehend konfiguriert werden kann, die Datenbank in einem eigenständigen zusätzlichen Kindprozeß zu verwalten, mit welchem über Kanäle (*pipes*) kommuniziert wird. Der Vaterprozeß kann in diesem Fall Kindprozesse erzeugen, ohne daß dies eine Kopie der Datenbank zur Foge hat. Für Multisessionfähigkeit des Datenbankmanagers auf alleiniger Basis des ersten Ansatzes hätte der jeweilige Kontext des Vaterprozesses auf den Datenbankprozeß übertragen werden müssen, weshalb auch hier Kontextwechsel nach Lösungs B verwendet wird.

Aus den vorgenannten Aspekten der Prozeßidentifizierung, wurde Multisessionfähigkeit geplant und mit Single-Thread Verarbeitung umgesetzt. Dies bedeutet, daß alle Anfragen von Klients in einer Warteschlange auflaufen, und seriell in der Reihenfolge ihres Eintreffens verarbeitet werden. Parallelverarbeitung von Anfragen ist aus den technischen Eigenschaften von TCL derzeit nicht möglich, und zur Lösung des Problems auch nicht erforderlich, wie unter 7.3.4 gemeinsam mit den Einzelheiten der Anfragenbearbeitung dargestellt wird.

4.2.3 Schaffung von Remote-Procedure Calls

Während des Prototypings wurde ein Derivat von TCL 7.3 mit der Bezeichnung TCP-DP [10] in der Version 3.2 eingesetzt. TCL-DP erlaubt als Besonderheit im wesentlichen die Definition von Objekten und dessen verteilten Ablauf im Netzwerk sowie die Ausführung von vollduplex RPCs.

Bedauerlicherweise hat die Mehrzahl der TCL-Derivate die Eigenschaft [11] , lediglich mit einer äquivalenten Version des Originals kompatibel zu sein. Während der Bearbeitung der Diplomarbeit, die zunächst auf TCP-DP basierte, erschien TCL in der Version 7.5, welche erstmals die einfache Kommunikation über TCP sockets ermöglichte und zumindest zur Lösung der Kommunikation zwischen CGI-Client und Datenbankmanager geeignet schien.

Da sich RPCs insbesondere im Einsatz mit TCL bereits beim Prototyping als sehr praktikabel und flexibel erwiesen, ist die Implementation von RPCs für TCL 7.5 als Teilziel definiert. Eine sehr starre Alternative hierzu wäre die Definition eines zusätzlichen Kommunikationsprotokolls auf Textbasis und der Austausch von entsprechenden Dateneinheiten zwischen CGI-Client und VDM Datenbankserver gewesen. Die RPC Aufrufe wurden bei der Realisierung denen von TCL-DP nachempfunden, der C-Quellcode als auch TCL-Code der *dp-library* wurden analysiert, um einen Einblick in die konkrete Funktionsweise zu gewinnen.

Als weiteres Ziel für die Implementierung der RPCs in TCL 7.5 wurde gesetzt, daß im Gegensatz zu TCL-DP keinerlei Hochsprachenerweiterungen verwendet werden und die Codierung ausschließlich in TCL erfolgt. Dies aus dem praktischen Grund, die unveränderte TCL-Shell als RPC-Client bzw. RPC-Server einsetzten und gegebenfalls auch lediglich den Binärcode von TCL verwenden und auch distributieren zu können. Die auf dieses Weise später implementierten Remote Procedure Calls sind von ihrer Funktionalität denen von TCL-DP gleich, basieren jedoch auf einer vollständig anderen Struktur und sind daher auch nicht mit diesen verträglich.

[10] DP steht für Distributed Processing

[11] Einige sind erwähnt durch [BW95, S. 385] Welch: Practical Programming in Tcl and Tk, 1995

4.2.4 Bewahrung der (Abwärts)-Kompatiblität zum bestehenden System

Der Zusammenhalt eines Systems hängt nicht zuletzt von der Anzahl seiner Komponenten ab. Wie Eingangs unter 1.2.2 erwähnt, zählte es daher zur Aufgabe des Diplomanden, auf den bestehenden Komponenten der VDM Klassenbibliothek aufzubauen, und, soweit möglich, Quellcode der Tool Command Language zu verwenden. Dennoch ist eine Erweiterungen der DSL-Spezifikation und des Hochsprachenteils des VDM Datenbankmanagers im Hinblick auf die erforderliche Multisessionfähigkeit unumgänglich.

Ein grundsätzliches Teilziel aller Erweiterungen ist es daher, daß keine Modifikationen eine Veränderung des ursprünglichen Verhaltens der Komponenten der Klassenbibliothek hervorruft, sondern in diesem Sinne jede Modifikation eine Erweiterung und keine Veränderung darstellt. Hinsichtlich der Hochsprachenerweiterungen, die bei der Realisierung jedoch einmalig am Datenbankmanager durchgeführt sind, und keine weiteren Bestandteile der Klassenbibliothek betreffen, ist festzustellen, daß ein Schaffen neuer Kommandos nicht gegen das Teilziel der Abwärtskompatibilität verstößt und die neu hinzugekommene Multisessionfähigkeit ebenfalls mit diesem Ziel verträglich ist, wenn ein immer definierter und voreingestellter Standardkontext eingerichtet wird.

4.2.5 Benutzerfreundlichkeit des Hilfesystems

Auch wenn sich das Informations- und Hilfesystem alleine aufgrund der Thematik an den professionellen Benutzer richtet, ist eine qualitativ hochwertige Unterstützung durch das System nur dann gegeben, wenn es jegliche Mühen bei der Benutzung vermeidet, da die Konzentration des Benutzers auf der eigentlichen Arbeit liegen soll. Die Verwendung des Informations- und Hilfesystems muß daher einfacher und insbesondere schneller sein, als der Griff zur Literatur.

Damit verbinden sich für eine maximale Benutzerfreundlichkeit folgende Teilziele:

- Vollständige Bedienung des Systems über das Zeigegerät, ergo die Maus. Unnötiger Wechsel zwischen Tastatur und Maus ist zu vermeiden
- Sofortiger Einstieg in den Kern des Systems, dies ist der Zugriff auf die Datenbank
- Aktualität der Informationen. Dies wird gewährleistet durch die Ablage des Hypertexts in der Klassenbibilothek selbst
- Nachverfolgung des Werdeganges (history) und Einfügen von Lesezeichen (bookmarks). Da die Mehrzahl der Web-Browser dies in der Regel bereits ermöglichen, sind diese Kriterien bereits erfüllt.
- Verknüpfungsmöglichkeit jeglicher zusammengehöriger Themenbereiche im Hypertext, hierzu zählen auch ausführbare VDM Anwendungsbeispiele.
- Die konventionelle Dokumentation von ADT in der Datenbank ist untrennbar mit dem ADT selbst verbunden. Daraus folgt als Zielstellung die Erweiterung der Informationsmöglichkeiten um allgemeine von ADT unabhängige Komponenten.

- Einführung von graphischen Elementen zur Verdeutlichung und Erleichterung des Lernprozesses.

Ein weiteres Element des Informations- und Hilfesystems, welches bei der Definition der Teilziele übersehen wurde, hat den Eingang in das Projekt erst nach der Fertigstellung gefunden, und soll daher an dieser Stelle nicht unerwähnt bleiben. Es handelt sich hierbei um die Suche nach Themen (*topics*) und Volltextsuche. Themen sind hierbei jeweils die Inhalte der Klassenbibliothek und damit die definierten ADTs mit ihren Implementationen und Operationen.

4.3 Gliederung der Projektphasen

Die üblichen Kriterien für das Projektmanagement und die Planung **Termine, Kosten und Qualität** beschränken sich bei der Bearbeitung einer Diplomarbeit auf die Faktoren Qualität und Termine. Die Einhaltung der Kriterien kann durch ein Phasenmodell aufgeteilt und verkleinert werden, daß an die Projektgröße, das Problem sowie Methoden und Werkzeuge angepaßt wird. In der Literatur finden sich zahlreiche Vorschläge für Phasenmodelle aller Art. Das Modell von Daly [SI92, 2.2.1] kommt der Realitiät vermutlich am nächsten.

Eine Software streng nach dem Wasserfallmodell zu entwickeln, ist selbst bei Bearbeitung eines Problems durch eine Person schlichtweg unmöglich. Es bietet jedoch eine grobe Richtung für die termingerechte Fertigstellung.Das dynamische *Spiralmodell* von Boehm [SI92, 2.23] gibt den Ablauf des Projektes vor. Es werden mit *rapid prototyping* einfache lauffähige Prototypen entwickelt, anhand derer das nächste Problem gelöst oder Alternativen gesucht werden können. Die konkreten Projektphasen gliedern sich in dieser Hinsicht wie folgt:

1. Analyse des der gesamten Klassenbibliothek
2. Testen des CGI-Interface
3. Aufruf des Datenbankmanagers über das CGI-Interface
4. Realisierung von verteilten Anwendungen (*distributed processing*) mit TCL
5. Integration von Serverfunktionalität in den Datenbankmanager
6. Schaffen eines Serverklienten
7. Anpassen des Bibliotheksbrowsers für die Benutzung des Klienten für den Zugriff Datenbankmanager, zum Test der Realisierungmöglichkeit
8. Schaffen eines einfachen Server-Klienten als CGI-Gateway
9. Beseitigen der Probleme des Mehfachzugriffs
10. Erweitern des Bibliotheksmanagers für den Zugriff auf den Server, zur Pflege der Datenbank, als auch um zusätzliche Funktionen

11. Integration von Hypertextfunktionen in die Datenbank und Auflösung dieser durch den VDM Server
12. Gestaltung der Benutzeroberfläche
13. Einbettung der Verarbeitung von MIME-Elementen in den VDM Server
14. Hinzufügen der VDM Anwendungsbeispiele zum Informations- und Hilfesystem
15. Erweiterung des Servers um Daemon-Funktionalität und Fernwartung
16. Portierung des Bibliotheksbrowsers und -managers auf 32-Bit Windows
17. Perfektionierung des Systems. Schaffen von Suchfunktionen mit Thema und Volltextsuche sowie Hilfen.
18. Test des Systems
19. Erstellung von Hochsprachenschnittstellen zum RPC-System von TCL

Die Projektphasen wurden in der Praxis nur teilweise in der dargestellten Reihenfolge bearbeitet. Teilweise mußte vorgegriffen werden, um das Risiko einer Fehlentwicklung zu vermeiden. Einige der Phasen mußten teilweise parallel bearbeitet. Asynchron zu der Vorgehensweise ist die regelmäßige Portierung vom Entwicklungssystem Linux auf SUN-OS vorgenommen worden, um systemspezifische Festlegungen zu vermeiden. Gleiches gilt für die Konvertierung der Diplomarbeit in HTML, die ebenfalls zwischenzeitlich vorgenommen wurde. Das Abbild 4.1 gibt einen Überblick über das gesamte konzpierte und realisierte Gesamtsystem, das der Autor bei der Entwicklung stetig vor Augen hatte.

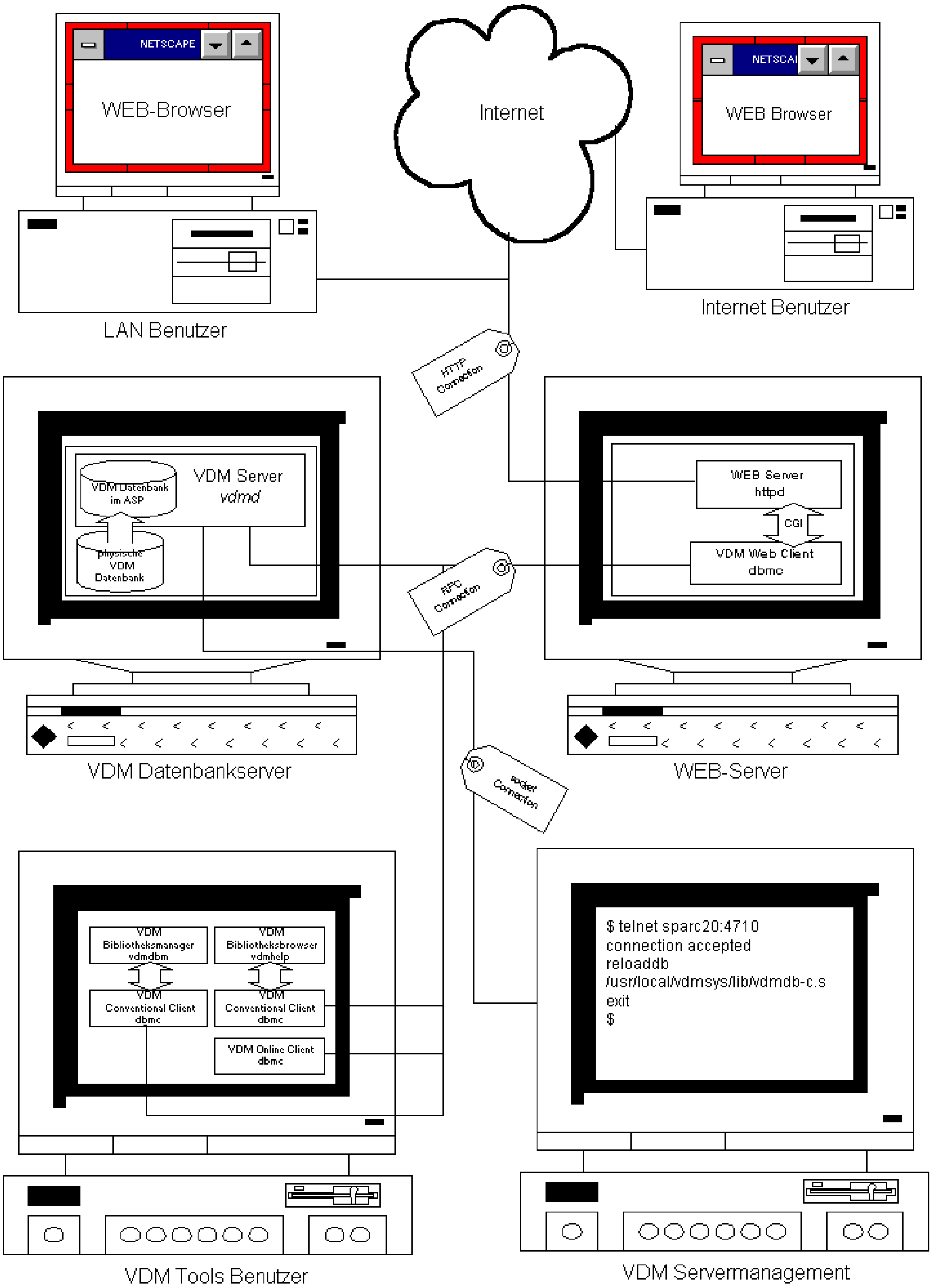


Abbildung 4.1: Überblick über das Gesamtsystem

Kapitel 5

Systemdefinition

5.1 Definition des Client / Server Modells

5.1.1 Entwicklung nach OSI

Für die Entwicklung eines Client-Server Systems bietet sich das OSI-Referenzmodell wie kein anderes an. Unter 2.3 ist bereits das OSI-Modell mit TCP/IP verglichen worden. Eine wesentliche Feststellung ist, daß Applikationen unter TCP/IP die oberen drei Schichten vermengen und nicht dem OSI-Modell folgen. Dies wäre selbstverständlich auch für das Informations- und Hilfesystem möglich gewesen, und wurde auch für schnelles Prototyping so gehandhabt.

Letztlich läßt aber eine Entwicklung unter Heranziehung des OSI-Modells eine planvollere und strukuriertere Lösung mit entsprechender Wiederverwendbarkeit zu. Aus diesem Grund wurde die Sitzungsschicht in Anlehnung an OSI programmiert. Sie kann damit auch Grundlage für andere Problemlösungen sein, und erlaubt den Vollduplexaustausch von Daten ohne den Zugriff auf auf die Transportschicht neu implementieren zu müssen.

5.1.2 Transportschicht TCP

Neben dem Protokoll TCP existieren auch zahlreiche weitere Protokolle der Transportschicht, auf die unter UNIX auch durchaus zugegriffen werden kann. Letztendlich wird in der Regel neben TCP jedoch nur UDP verwendet, das als verbindungsloses Protkoll nur den Austausch von Datagrammen ermöglicht, und daher für Lösung eher ungeeignet ist. Zudem wird durch TCL 7.5 derzeit nur TCP unterstützt, so daß dieses Protokoll als Transportschicht für das Hypertext Informations- und Hilfesystem verwendet werden muß.

5.1.3 Definition eines OSI-Dienstes

Ein Dienst wird formal durch eine Reihe von **Dienstelementen** (*primitives*) oder auch Operationen gekennzeichnet, über die es dem Anwender oder der Instanz möglichen, den Dienst in Anspruch zu nehmen. Diese Dienstelemente sorgen dafür, daß

Dienstelement	Bedeutung
Anforderung (request)	Eine Instanz möchte einen Dienst dazu veranlassen, etwas bestimmtes zu tun
Anzeige (indication)	Eine Instanz soll über ein Ereignis informiert werden
Antwort (response)	Eine Instanz möchte auf ein Ereignis reagieren
Bestätigung (confim)	Eine Instanz soll über eine bestätigte Anforderung informiert werden

Tabelle 5.1: Vier Klassen von Dienstelementen

der Dienst eine bestimmte Handlung durchführt, oder daß er auf eine Hnadlung der Partnerinstanz in einer bestimmten Weise reagiert. Innerhalb des OSI-Modells können Dienstelemente in vier Klassen aufgeteilt werden. Diese sind in Tabelle 5.1 dargestellt.

Die erste Klasse von Dienstelementen ist das Anforderungselement (*request*). Damit wird eine bestimmte Arbeit begonnen, zum Beispiel der Aufbau einer Verbindung oder die Versendung von Daten. Die Partnerinstanz bekommt über eine Anzeige (*indication* die Anforderung signalisiert. In OSI-Schreibweise erhält damit durch CONNECT.request die Partnerinstanz (*peer*) ein CONNECT.indication und beantwortet dies durch CONNECT.response, was beim Versender der Anforderung ein CONNECT.confirm auslöst[1].

Jeder bestätigende (*confirmed* Dienst verfügt über alle vier Dienstprimitive. Unbestätigte (*unconfirmed*) Dienste kennen lediglich eine Antwort und eine Anzeige. Weitere Dienste können im OSI-Modell auch über weniger Dienstprimitive verfügen, aber niemals über mehr. Die vier vorgestellten Primitive sind für das Modell erschöpfend. Ein Dienst besteht aus einem Satz von Operationen bzw. Dienstelementen.

Zur Transparenz des Dienstekonzeptes, soll ein Beispiel für einen verbindungsorientierten Dienst mit acht Dienstelementen angeführt werden, und zwar die telefonische Bestellung einer Pizza:

1. CONNECT.request - Eine Telefonnummer des Pizza-Service wählen
2. CONNECT.indication - Das Telefon läutet dort
3. CONNECT.response - Der Inhaber nimmt das Telefon ab
4. CONNECT.confirm - Es wird vernommen, daß jemand an das Telefon geht
5. DATA.request - Aufgeben der Pizzabestellung
6. DATA.indication - Pizzadienst nimmt Bestellung entgegen
7. DATA.request - Pizzadienst stellt Lieferung in 20 Minuten in Aussicht
8. DATA.indication - Aufnahme der Lieferinformation
9. DISCONNECT.request - Auflegen des Bestellers

[1]Grundsätzlich sind die Formulierungen „eine Verbindung öffnen“ und „eine Verbindung schließen“ im Zusammenhang mit dem OSI-Modell tabu und sollten auch ansonsten nicht verwendet werden. Richtiger und offiziell sind die Formulierungen „aufbauen“ und „abbauen“

10. DISCONNECT.indication - Auflegen des Pizza-Lieferanten

Dienste und Protokolle haben unterschiedliche Bedeutung und Konzepte. Währen ein Dienst die Sammlung der Dienstelemente einer Schicht ist, die der über ihr liegenden Schicht zur Verfügung gestellt werden, ist ein Protokoll das Regelgefüge, welches das Format und die Bedeutung der von den Partnerinstanzen innerhalb einer Schicht ausgetauschten Rahmen, Pakete oder Nachrichten festlegt. Mit Protokollen führen Schichten ihre Dienste aus, Dienste sind die Schnittstelle für den Dienstnutzer. Von außerordentlicher Wichtigkeit ist es, daß die Protokolle auf beiden Partnerinstanzen absolut identisch sind. Dies macht bei der Vielzahl von Betriebssystemen und Technologien die Wichtigkeit der einheitlichen Normung deutlich.

5.1.4 Definition der Sitzungsschicht

Implementationen nach OSI

Tabelle 5.1.4 stellt alle OSI-Sitzungsdienstelemente der Sitzungsschicht dar. Nicht alle Dienstelemente wurden implementiert, da sie für Lösung des Problems nicht benötigt werden und den TCL-Quellcode überfrachtet hätten. Folgende Elemente wurden mit allen Primitiven des OSI-Modells implementiert:

1. S-CONNECT
2. S-RELEASE
3. S-U-ABORT
4. S-P-ABORT
5. S-DATA
6. S-U-EXCEPTION-REPORT
7. S-P-EXCEPTION-REPORT

Der Bezeichner „S“ kennzeichnet, daß es sich um die Sitzungschicht handelt. „U“ und „P“ stehen für „User“ und „Provider“ und kennzeichnen den Ursprung von Aktionen eines Dienstelementes. Für die Sitzungsschicht wird damit ein Service Access Point (*SSAP*) definiert, der genau diese Dienstelemente beinhaltet.

Definition von Sitzungen

Eine Sitzung ist einer Transportverbindung sehr ähnlich, sie sind jedoch nicht identisch. Wenn in der Sitzungsschicht eine Anforderung für den Sitzungsaufbau eintrifft, muß eine Transportverbindung aufgebaut werden, die die Sitzungsverbindung trägt. Sobald die Sitzung beendet ist, wird auch die Transportverbindung abgebaut. Das bedeutet, daß die Sitzung eins zu eins auf eine Transportverbindung abgebildet wird. Es sind aber auch andere Abbildungen möglich. Eine Transportverbindung kann kontinuierlich bestehen, Sitzungen werden jedoch nur bei Bedarf aufgebaut. Es ist durchaus

OSI-Sitzungsdienstelemente	Dienstprimitive: REQUEST	INDICATION	RESPONSE	CONFIRM	SPDU, die bei der Anforderung gesendet wird	SPDU, die bei der Antwort gesendet wird
S-CONNECT	X	X	X	X	CONNECT	ACCEPT, REFUSE
S-RELEASE	X	X	X	X	FINISH	DISCONNECT, NOT FINISHED
S-U-ABORT	X	X			ABORT	(ABORT ACCEPT)
S-P-ABORT		X			(ABORT)	
S-DATA	X	X			DATA-TRANSFER	
S-EXPEDITED-DATA	X	X			EXPEDITED DATA	
S-TYPES-DATA	X	X			TYPED DATA	
S-CAPABILITY-DATA	X	X	X	X	CAPABILITY DATA	CAPABILITY DATA ACK
S-TOKEN-GIVE	X	X			GIVE TOKENS	
S-TOKEN-PLEASE	X	X			PLEASE TOKENS	
S-CONTROL-GIVE	X	X			GIVE TOKENS CONFIRM	(GIVE TOKENS ACK)
S-SYNC-MAJOR	X	X	X	X	MAJOR SYNC POINT	MAJOR SYNC POINT ACK
S-SYNC-MINOR	X	X	X	X	MINOR SYNC POINT	MINOR SYNC POINT ACK
S-RESYNCHRONIZE	X	X	X	X	RESYNCHRONIZE	RESYNCHRONIZE ACK
S-ACTIVITY-START	X	X			ACTIVITY START	
S-ACTIVITY-END	X	X	X	X	ACTIVITY END	ACTIVITY END ACK
S-ACTIVITY-DISCARD	X	X	X	X	ACTIVITY DISCARD	ACTIVITY DISCARD ACK
S-ACTIVITY-INTERUPT	X	X	X	X	ACTIVITY INTERRUPT	ACTIVITY INTERRUPT ACK
S-ACTIVITY-RESUME	X	X			ACTIVITY RESUME	
S-U-EXCEPTION-REPORT	X	X			EXCEPTION REPORT	
S-P-EXCEPTION-REPORT	X				(EXCEPTION REPORT)	

Tabelle 5.2: Dienstelemente der Sitzungsschicht mit ihren SPDUs

möglich und zulässig über die gleiche Transportverbindung mehrere Sitzungen zu führen. Als Sitzungen werden Verbindungen bezeichnet, die es ermöglichen, Verbindungen aufzubauen und mit diesen geordnet Daten für Benutzerprozesse zu übertragen.

Problematik von OSI mit RPC

Der rechnerferne Prozeduraufruf (*RPC*) fügt sich nicht besondes gut in das OSI-Referenzmodell ein, und ist daher auch nicht in mehreren Schichten aufgebaut. Er wird zwischen Sitzungsschicht und Anwendungsschicht eingeordnet. RPCs greifen in die Authorität der oberen drei Schichten ein. Eine genauere Einordnung hängt deshalb von der Art der über RPCs ausführbaren Funktionen ab.

5.1.5 Anlehnung an die Darstellungsschicht

Tabelle 5.3 stellt die Dienstelemente der Darstellungschicht dar, beinhaltet jedoch nicht die Primitive der verbindungsunabhängigen Datenübertragungen, da diese Art der Übertragung für die vorliegende Arbeit nicht von Bedeutung ist. Vollständige Informationen hierzu gibt Tanenbaum [AT92][S. 579].

Da für die Kommunikation zwischen Server und Client RPCs geplant sind, läßt dies keinen Raum für die Implementation der Darstellungsschicht. Dennoch bietet diese Schicht eine gute Vorlage, für eine Gestaltung von RPCs, die die Funktionalität nachempfinden. Tatsächlich entspricht die Anmeldung an einem Server, der Austausch von Kommandos und Funktionswerten in weiten Teilen auch der Darstellungsschicht.

Auch die Anpassung der Umgebung für einen Klienten mit Hilfe von RPCs steht funktional nicht im Widerspruch zu den Definitionen von ISO für diese Ebene, wie im folgenden Abschnitt dargestellt wird, denn RPCs können ebensogut die Aufgaben übernehmen, für die ansonsten die Dienstelemente der Darstellungschicht zuständig wären.

Definition des Kontext

Die Bedeutung des lat. Begriffes Kontext wird allgemein als (innerer) Zusammenhang verstanden. Eine genauere Interpretation eines Sachverhaltes allein anhand des Begriffes ist nicht möglich, da dessen Verwendung zu vielfältigen Themen, eine eindeutige Begriffsbestimmung nicht zuläßt. Zu nennen im Software-Zusammenhang wäre hier als typisches Beispiel der Kontext nach der Norm X.500 [2] , welcher die Position eines Objektes im Verzeichnisbaum angibt.

Hinsichtlich dieser Definitionsproblematik ist eine genaue Festlegung der Bedeutung des hier verwendeten Kontext-Begriffes erforderlich. Der in dieser Diplomarbeit verwendete Kontext ist ebenfalls nicht zu verwechseln mit dem Kontext eines Anwendungsprogrammes (*user context*) oder dem Kontext eines Kernels (*kernel context*)

[2] Größere Verbreitung hat diese Norm durch das Netzwerkbetriebssystem NetWare 4.x des Herstellers Novell gefunden, in welche diese Norm als „NetWare Directory Services“ integriert wurde. Es handelt sich um die Anordnung von Objekten wie Computer, Benutzer, Gruppen etc. in einer internationalen Struktur.

OSI-Sitzungsdienstelemente	REQUEST	INDICATION	RESPONSE	CONFIRM	Bedeutung
P-CONNECT	X	X	X	X	Einrichtung einer Darstellungsstruktur
P-RELEASE	X	X	X	X	Geordneter Abbau
P-U-ABORT	X	X			Abrupter Abbau durch Benutzer
P-P-ABORT		X			Abrupter Abbau durch Anbieter
P-DATA	X	X			Übertragung von normalen Daten
P-EXPEDITED-DATA	X	X			Übertragung von beschleunigten Daten
P-TYPES-DATA	X	X			Übertragung von Außerbanddaten
P-CAPABILITY-DATA	X	X	X	X	Übertragung von Steuerinformationen
P-TOKEN-GIVE	X	X			Ein Token an den Partner übergeben
P-TOKEN-PLEASE	X	X			Ein Token vom Partner anfordern
P-CONTROL-GIVE	X	X			Alle Token an den Partner übergeben
P-SYNC-MAJOR	X	X	X	X	Hauptsynchronisierungspunkt einfügen
P-SYNC-MINOR	X	X	X	X	Nebensynchronisierungspunkt einfügen
P-RESYNCHRONIZE	X	X	X	X	Zurückgehen an früheren Synchronisationspunkt
P-ACTIVITY-START	X	X			Aktivität beginnen
P-ACTIVITY-END	X	X	X	X	Aktivität beenden
P-ACTIVITY-DISCARD	X	X	X	X	Aktivität abbrechen
P-ACTIVITY-INTERUPT	X	X	X	X	Aktivität aussetzen
P-ACTIVITY-RESUME	X	X			Ausgesetzte Aktivität wiederaufnehmen
P-U-EXCEPTION-REPORT	X	X			Ausnahmebedingung des Benutzers berichten
P-P-EXCEPTION-REPORT	X				Ausnahmebedingung des Anbieters berichten
P-ALTER-CONTEXT	X	X	X	X	Wechsel des Kontext

Tabelle 5.3: Verbindungsorientierte Dienstelemente der Darstellungsschicht

der sich auf Speicherbereiche und deren Zugriffsmodi bezieht[3], sondern richtet sich einzig und allein auf den Kontext der Darstellungschicht (*presentation layer*) des OSI-Modells, beschrieben auch durch Tanenbaum [AT92, S. 575] und illustriert unter 2.3.

Die Darstellungsschicht ist seit der Einführung des OSI-Modells in großen Ausmaße weiterentwickelt worden, nachdem diese Schicht lange Zeit auf der Suche nach ihrer eigentlichen Aufgabe gewesen ist. Ursprünglich war sie nur als Stelle angesehen worden, an der Konvertierungen durchgeführt werden konnten, beispielsweise so, daß ASCII- mit EBCDIC-Rechnern kommunizieren konnten. Mittlerweile entschloß man sich, der Darstellungsschicht alle Aufgaben, einschließlich Konvertierung, Verschlüsselung und Komprimierung zu übertragen, so daß sie zutreffender als „Repräsentationsschicht“ bezeichnet werden könnte. Eine ISO-Norm zur Lösung dieser Aufgaben heißt abstrakte Sytnaxnotation1 (ASN.1).

Die vier Hauptaufgaben der Darstellungschicht sind damit:

1. Die Möglichkeit zur Ausführung der Sitzungsdienstelemte zu geben
2. Daten zwischen interner und externer Form zu konvertieren
3. Möglichkeiten zur Spezifikation komplexer Datenstrukturen zu bieten
4. Die benötigten Datenstrukturen zu verwalten

Diese Aufgaben machen deutlich, daß sich die Darstellungschicht des VDM Datenbankmanagers sowohl in 1. und 2. auf den TCL-Code als auch in 3. und 4. auf den Hochsprachen- bzw. VDM-Anteil erstreckt. Sitzungsdienstelemete betreffen die in TCL implementierte Sitzungschicht, als auch die implementierte Konvertierung von MIME-Elementen. Alle weiteren Tools zur Klassenbibilothek beruhen in der Darstellungschicht auf TCL.

Viele der Dienstelemente der OSI-Darstellungschicht sind mit denen der OSI-Sitzungsschicht identisch. Der Aufruf von P-CONNECT.request von einer Anwendungsinstanz bewirkt, daß von der Darstellungsschicht S-CONNECT.request ausgelöst wird. Tatsächlich werden fast alle Dienstelemente der Darstellungsschicht an die Sitzungschicht übergeben, und umgekehrt. Augenmerk ist jedoch auf die Primitive **P-ALTER-CONTEXT** der Darstellungsschicht zu legen, wie Tabelle 5.3 zeigt.Die für bestimmte Anwendungen eingesetzten Datenstrukturen können in Gruppen zusammengefaßt werden, welche als **Kontext** bezeichnet werden.

Der Kontext, der den Zustand des VDM Datenbankmanagers beschreibt, besteht aus nachfolgend dargestellter Gruppe[4], die als Zeichenketten bzw. Zeichenketten-Sets im Hochsprachencode des Datenbankmanagers enthalten sind. Zu den einzelnen Elementen dieser Gruppen sind diejenigen TCL-Kommandos, welche für die Einstellung dieses Kontexts verantwortlich sind, als auch die Kommandos, auf welche sich diese Einstellungen auswirken, benannt:

[3]Vergl. [RS92, S.25] Stevens: UNIX Network Programming, 1990

[4]Der jeweils gültige Kontext ist hinterlegt in der Klasse DBMANAGER, definiert in der Quelldatei vdmdbm.cc

1. **adttypes**

   Das Kommando

   select adttypes [adts]

   stellt die Selektion von ADTs der Klassenbibilothek ein bzw. hebt diese wieder auf, sofern keine *adts* angegeben werden. Auswirkungen treten auf den Befehl

   adtdocu

   auf, welcher in Abhängigkeit der selektierten ADTs Dokumentation der Bibliothek liefert.

2. **special_impl**

   Die Bedeutung von

   select specialimpl [impls]

   ist dem vorangegangenen Kommando ähnlich, die Auswirkungen beziehen sich jedoch auf

   impdocu adt

3. **special_ops**

   select specialops

   zählt in seiner Bedeutung zu den vorangegangenen Befehlen select adttypes und select specialimpl. Die Auswirkungen betreffen

   oprdocu adt impl

4. **filter_adts**

   Filterung von ausgewählten ADTs durch

   select filteradts adts

   beeinflußt die Ausgabe durch

   partialexport

5. **filter_impl**

   Implementationsfilter durch

   select filterimpl impls

   Auswirkung auf

   partialexport

6. **filter_oper**

   Filterung von Operationen durch

   select filteroper opers

   Auswirkung auf

   partialexport

7. **filter_comp**

   Filter von Komponenten durch

   select filtercomp comps

Auswirkung auf

partialexport

8. **filter_params**

   Filter für in der Klassenbibliothek hinterlegte Parameter

   select filterparam

   der in seiner Funktion identisch ist mit Befehl operparams des Datenbankmanagers. Die Auswirkungen betreffen

   partialexport

9. **operationtype**

   Der Operationtypus der Klassenbibliothek wird mittels

   operationtype type

   eingestellt und kann derzeit als Wertebereich C, C++ oder A++ (für die automatische Speicherverwaltung) annehmen. Es sind jedoch auch weitere Werte vorstellbar, die mit den Inhalten der Klassenbibliothek wachsen. Auswirkung treten auf für alle Kommandos, die mit den Elmenten **adttypes**, **special_impl**, und **special_ops** in Zusammenhang stehen.

Es ist ohne weiteres ersichtlich, daß im Multisessionbetrieb die Einstellung eines speziellen Operationstyps durch einen Benutzer des Datenbankmanagers diesen Status dauerhaft verändert. Würde dieser Benutzer, noch bevor er die nachfolgenden, in diesem Zusammenhang stehenden Folgebefehle abgesetzt hat, durch einen anderen Benutzer unterbrochen, der den Operationstyp anderweitig verändert, so wäre ein willkürliches Resultat das Ergebnis. Ebenso erginge es dem unterbrechenden Benutzer, der sich auf eine Initialisierung des Datenbankmanagers mit den Standardwerten verläßt.

VDM-Spezifikation Die Einstellung des Kontexts wird nachfolgend in VDM spezifiziert, wobei nicht auf eine vollständige Definition des Datenbankmanagers Wert gelegt, sondern der Schwerpunkt auf der Definition und den Methoden zur Kontexteinstellung des Datenbankmanagers liegt. Die konkrete Definition des Kontexts für das aktuelle Projekt selbst, ist in DSL spezifiziert und im Anschluß wiedergegeben.

Der Leser wird gebeten, den Tree *DBCONTEXT* wie folgt als C++ Klasse zu abstrahieren, in welcher sämtliche Methoden gekapselt sind. Wesentliche Inhalte dieser Klasse sind der *ContextCounter*, welcher für implizit erzeugte Sitzungen verwendet wird, *CurrentContext* als Index des aktuellen Kontexts, *DefaultContext* als Standardinitialisierung und Tree bestehend aus der gültigen Gruppe, sowie *ApplicationContext* als Synonym für die Einstellungen der Klasse *DBMANAGER* .

Die in der Klasse *DBMANAGER* für unterschiedliches Verhalten verantwortlichen ADTs, wurden in der ursprünglich vorliegenden Version nicht gruppiert. Gemäß der Anforderungen in 1.2.2 wurde die Klasse *DBMANAGER* lediglich um die Klasse *DBCONTEXT* erweitert, und der gesamte Quellcode ansonsten im wesentlichen unverändert belassen. Aus diesem Grund liegt eine Gruppierung erst in *DBCONTEXT* vor, welcher Referenzen dieser ADTs bei der Konstruktion mitgeteilt werden. Der

Kontext des Datenbankmanagers wird durch eine Manipulation der ADTS über diese Referenzen umgeschaltet. Da Referenzen dieser Art in VDM nicht modelliert werden können, ist *ApplicationContext* des Trees DBCONTEXT vom Leser mit jenen ungruppierten Variablen der Klassen DBMANAGER gleichzusetzen. Eine lückenlose und identische formale Definition des Kontextwechsels in VDM wäre themenfremd gewesen, hätte die Modellierung des gesamten Datenbankmanagers erfordert und den Text an dieser Stelle überfrachtet.

Domaingleichungen

$DBCONTEXT$	:: ContextCounter CurrentContext DefaultContext ApplicationContext ContextMap
$ContextCounter$	$= \quad INTG$
$CurrentContext$	$= \quad INTG$
$DefaultContext$	= Context
$ApplicationContext$	= Context
$ContextMap$	$= \quad INTG \quad \overrightarrow{m} \quad Context$
$Context$	$:: \quad adttypes\ special\_impl\ special\_ops$ $filter\_adts\ filter\_impl\ filter\_oper$ $filter\_comp\ filter\_params\ filter\_comps$ $filter\_comps\ operationtype$
$adttypes$	$: \quad Strings$
$special\_impl$	$: \quad Strings$
$special\_ops$	$: \quad Strings$
$filter\_adts$	$: \quad Strings$
$filter\_impl$	$: \quad Strings$
$filter\_oper$	$: \quad Strings$
$filter\_comp$	$: \quad Strings$
$filter\_params$	$: \quad Strings$
$filter\_comps$	$: \quad Strings$
$operationtype$	$: \quad Str$
$Strings$	$= \quad Str\ -set$
Str	$= \quad Char*$
$Char$	$= \quad 1..255$

Operationen

1 Konstruktion der Kontextverwaltung

- Kontextzähler initialisieren mit 0
- Aktueller Kontext ist 0
- Der Applikationskontext (die Referenz) wird registriert
- Der Standardkontext (ist aktueller Kontext) enthält Vorgabewerte

make-DBCONTEXT(appcontext) =
reload-DBCONTEXT(mk-DBCONTEXT(mk-INTG(0),mk-INTG(0),
emptycontext(),appcontext,[INTG(0) $\longrightarrow$ *emptycontext()]))*

type Context $\longrightarrow$ *DBCONTEXT*

2 Übernahme des Applikationskontext

- Die (möglicherweise geänderten) Einstellungen des Datenbankmanagers werden in den aktuellen Standardkontext übernommen
- Die Kontext-Tabelle (oder MAP) wird aktualisiert mit dem Standardkontext

update-DBCONTEXT(dbcontext) =
let mk-DBCONTEXT(,currcontext,defcontext,appcontext,contextmap) = dbcontext in
let defcontext = appcontext
let contextmap = contextmap + [currcontext $\longrightarrow$ *defcontext]*

type DBCONTEXT $\longrightarrow$ *DBCONTEXT*

3 Setzen des Applikationskontext

- Der Standardkontext ist (siehe Invariante) immer identisch mit dem durch *CurrentContext* selektiertem Eintrag der Kontext-MAP
- Der Applikationskontext wird durch den Standardkontext gesetzt
- Der Standardkontext ist nur bei der Initialisierung mit dem Kontext 0 identisch, und wird durch Wechsel des Kontexts mit Einträgen der Kontext-Tabelle getauscht

reload-DBCONTEXT(dbcontext) =
let mk-DBCONTEXT(,,defcontext,appcontext,) = DBCONTEXT in
let appcontext = defcontext

type DBCONTEXT $\longrightarrow$ *DBCONTEXT*

4 Erzeugen eines definierten Anfangszustandes

- Vorbestimmung des Operationstyps für die Hochsprache C

emptycontext() =
mk-Context($\{\ \}, \{\ \}, \{\ \}, \{\ \}, \{\ \}, \{\ \}, \{\ \}, \{\ \}, \{\ \}, \{\ \}$, *c)*

type $\longrightarrow$ *Context*

5 Kontext Existenzübrprüfung

exists(CID,dbcontext) =
CID ∈ dom s-ContextMap(dbcontext)

type INTG DBCONTEXT ⟶ BOOL

6 Wechsel des Kontexts

- Ein Wechsel zu einem bereits gewählten Kontext ist unnötig
- Der aktuelle Applikationskontext muß gesichert werden
- Die Identifikation des aktuellen Kontexts wird geändert
- Der in der Kontext-Tabelle gesicherte Kontext wird wiederhergestellt
- Setzen des Applikationskontext
- Prämisse: Der Zielkontext muß existieren

switch(CID,dbcontext) =
if s-CurrentContext(dbcontext)
then mk(dbcontext)
else
let reload-DBCONTEXT(,currcontext,defcontext,
contextmap) =
update-DBCONTEXT(dbcontext) in
let currcontext = CID
let defcontext = contextmap(currcontext)

type INTG DBCONTEXT ⟶ DBCONTEXT

pre-switch(CID,dbcontext) =
exists(CID,dbcontext)

type INTG DBCONTEXT ⟶ BOOL

7 Implizites erzeugen eines Kontext

- Implizit (ohne Kontext-Identifikation) erzeugte Kontext werden mit negativen Ganzzahlen identifiziert[5].
- Dekrementieren des Kontextzählers für implizite Erzeugung
- Hinzufügen eines neuen Kontext durch Vereinigung[6]

[5] Aus dem praktischen Grund, daß für positive Kontext-IDs die Portadresse der Verbindung zum Netzwerkklienten verwendet wird, um eine Beziehung zwischen Klienten und Kontext herzustellen

[6] Es wird hier davon ausgegangen, daß der Zähler bis ins Unendliche negativ anwächst. In der praktischen Implementation erfolgt bei Überlauf des definierten negativen Zahlenbereichs eine Rücksetzung auf den Startwert

create(dbcontext) =
let mk-DBCONTEXT(contextcnt,,,contextmap) = dbcontext in
let contexcnt = contextcnt -1
let contextmap = contextmap $\cup$ *[contextcnt* $\longrightarrow$ *emptycontext()]*

type DBCONTEXT $\longrightarrow$ *DBCONTEXT*

8 Explizites Erzeugen eines Kontext

- Erzeugen eines neuen Kontext mit Identifikation[7]
- Prämisse: Die Kontext-Identifikation ist nicht bereits vergeben

create(CID,dbcontext) =
let mk-DBCONTEXT(,,,,contextmap) = dbcontext in
contextmap = contextmap + [mk-INTG(CID) $\longrightarrow$ *emptycontext()]*

type NAT0 DBCONTEXT $\longrightarrow$ *DBCONTEXT*

pre-create(cid,dbcontext) =
˜exists(cid,dbcontext)

type NAT0 DBCONTEXT $\longrightarrow$ *BOOL*

9 Entfernen eines Kontext

- Wenn der zu löschender Kontext der aktuelle ist, zuvor Wechsel zum Standardkontext 0
- Entfernen des Kontext aus der Kontext-MAP

release(CID,dbcontext) =
if CID = s-CurrentContext(dbcontext)
then let dbcontext = switch(INTG(0),dbccontext)
let mk-DBCONTEXT(,,,,contextmap) = dbcontext in
contextmap = contextmap $\setminus$ *CID*

type INTG DBCONTEXT $\longrightarrow$ *DBCONTEXT*

pre-release(CID,dbcontext) =
˜exists(CID,dbcontext) OR CID $\neq$ *mk-INTG(0)*

type INTG DBCONTEXT $\longrightarrow$ *BOOL*

[7]Die ID des Kontext ist in der praktischen Realisierung mit dem Port, welcher der Klientenverbindung zugewiesen ist identisch

10 Ermitteln aller vergebenen Kontext-Identifikationen

list(dbcontext) =
  dcl cids := {} *type INTG -set*
  for all cid $\in$ *dom s-ContextMap(dbcontext) do*
    let cids = *cids* $\cup$ {*cid*}
  cids

type DBCONTEXT $\longrightarrow$ *INTG -set*

11 Ermitteln der aktuellen Kontext Identifikation

cid(dbcontext) = *s-CurrentContext(dbcontext)*

type DBCONTEXT $\longrightarrow$ *INTG*

12 Ermitteln des aktuellen Kontext

current(dbcontext) = *s-DefaultContext(dbcontext)*

type DBCONTEXT $\longrightarrow$ *Context*

13 Konsistenzbedingung

- Es existiert der Standardkontext 0
- Der Kontext-Zähler ist null oder negativ
- Der Standardkontext ist der aktuelle Kontext
- Der Kontext der Applikation entspricher dem Standardkontext

inv-DBCONTEXT((contextcnt,currcontext,defcontext,
appcontext,contextmap)) =
  (*0* $\in$ *dom contextmap*) *AND*
  (*contextcnt* $\leq$ *mk-INTG(0)*) *AND*
  (*defcontext* = *contextmap(currcontext)*) *AND*
  (*appcontext* = *defcontext*)

type inv-DBCONTEXT:DBCONTEXT $\longrightarrow$ *BOOL*

DSL-Spezifikation Abschließend ist der Anteil der formalen Definition, der für die vorliegende Arbeit verwendet wurde, als Auszug von Inhalten der Datei *dbm.vdm*, welche die Definitionen des Datenbankmanagers in Form der Domain Specification Language enthält, wiedergegeben. Als Metasprache ist dies die (aus erwähnten Gründen nur ungefähre) maschinenlesbare Umsetzung der VDM Spezifikation des vorangegangenen Abschnitts.

$$
\begin{array}{lll}
ContextMap & = & Intg\ -m->\ Context \\
\\
Context & :: & s\_adttypes : Strings \\
 & & s\_special\_impl : Strings \\
 & & s\_special\_ops : Strings \\
 & & s\_filter\_adts : Strings \\
 & & s\_filter\_impl : Strings \\
 & & s\_filter\_oper : Strings \\
 & & s\_filter\_comp : Strings \\
 & & s\_filter\_params : Strings \\
 & & s\_filter\_comps : Strings \\
 & & s\_operationtype : Str \\
\\
Strings & = & Str\ -set
\end{array}
$$

Implementation

$$
\begin{array}{lll}
ContextMap & = & linklist + default \\
Context & = & ptstruct + default \\
\\
Str & = & default + ...
\end{array}
$$

Definition von Benutzeragenten

Als Benutzeragent wird der Codeabschnitt des Servers definiert, der durch das Umschalten des Kontext eingegrenzt wird, über eine eigenständige Umgebung verfügt, und insbesondere die Bearbeitung von Fehlern auf diesen Bereich reduziert. Für jeden Anmeldevorgang wird ein Benutzeragent eingerichtet, der den Klienten für die gesamte Lebensdauer betreut. Er ist damit aus Sicht des Klienten der einzigste Ansprechpartner auf Serverseite.

5.2 Definition der HTML-Generierungsprozesse

Die Erzeugung von HTML-Seiten ist abhängig durch Parameter, die vom CGI über den Client an den Server übermittelt werden. Die Parametrisierung erfolgt über eine Zeichenfolge. Der generelle Ablauf, der durch eine Interaktion des HTML-Browser in Gang gesetzt wird, läßt sich wie folgt defniert:

1. Der Benutzer wählt zu Beginn den URL des VDM Datenbank Clients *dbmc* an, welcher im CGI-Gateways-Verzeichnis abgelegt ist. Ein üblicher Standardwert für dieses Verzeichnis lautet *„cgi-bin“* . Ein vollständiger URL könnte damit *http://www.fh-wedel.de/cgi-bin/dbmc* lauten.

2. Der Web-Server (der häufig den Namen *httpd* trägt), wird durch den URL adressiert und setzt zunächst mehrere Umgebungsvariablen des aufzurufenden

Gateways. Von Bedeutung sind hier insbesondere *SCRIPT_NAME* und *QUERY_STRING*, welche den Namen des aufzurufenden Scripts (also */cgi-bin/dbmc*) sowie Parameter übergibt und damit beim Erstaufruf für diesen einen Leerstring. Anschließend wird das Gateway aufgerufen. Dieses ist identisch mit *SCRIPT_NAME*. Pfade im Zusammenhang mit dem Web-Server sind immer relativ zu den Einstellungen des HTTP-Servers zu betrachten.

3. Der VDM Datenbank Client verbindet sich mit dem VDM Server *vdmd* an einer beliebigen voreingestellten Maschine. Er liest die gesetzten Umgebungsvariablen und leitet sie an den VDM Server weiter.

4. Der VDM Server bearbeitet die Anfrage anhand der Parameter und liefert dem Klienten in Abhängigkeit dieser Parameter ein Resultat als auch den Typen des Resultats. Treten bei der Bearbeitung Fehler auf, werden auch diese mit genauer Indentifikation dem Client übergeben. Bei der Erzeugung des Resultats bearbeitet der Server alle Hyperlinks anhand des bekannten Gatewaynamens in *SCRIPT_NAME* und substituiert diese durch einen URL auf sich selbst. Im URL werden die Parameter hinterlegt, welche die Wirkung dieses Hyperlinks einstellen. Beispielsweise löst der URL *http://www.fh-wedel.de/dbmc-/cgi-bin?hot=help* bei der Anwahl durch den Benutzer die Produktion einer Hilfeseite durch den Server aus.

5. Zu den gültigen Resultaten des VDM Servers zählen Bilder, HTML-Seiten als auch andere gültige MIME-Elemente. Einige Elemente liegen in der VDM Datenbank in MIME-Codierung vor, z.B. Bilder, da die Datenbank aufgrund Ihrer Struktur selbst ausschließlich aus Text besteht und eine Verarbeitung des gesamten ASCII-Codes von 0 bis 255 nicht möglich ist, und der ASCII-Code 0 ohnehin von TCL nicht verarbeitet werden kann. Der Server teilt dem Client mit, ob das Resultat in MIME-Codierung vorliegt.

6. Der VDM Datenbank Client erhält das Ergebnis vom VDM Server. Liegt eine MIME-Codierung des Ergebisses vor, so wird unmittelbar vor der Ausgabe eine Decodierung vorgenommen. Das Resultat wird ansonsten unverändert an den Web-Server übergeben und durch diesen an den WEB-Browser als Antwort (*reply*) übermittelt.

7. Der HTML-Browser baut die neue, mit Hyperlinks versehende Bildschirmseite auf, die Anwahl eines Hyperlinks durch den Benutzer löst den gesamten Ablauf wiederholt aus und damit auch den Generierungsprozess einer neuen Seite. Es handelt sich grundsätzlich und immerfort um ein und deselben URL und damit um die gleiche Seite, die jedoch in Abhängigkeit der Parameter immer neu gestaltet wird.

5.3 Definition der Ein- und Ausgaben

Der Ablauf der Generierung von HTML-Seiten durch den VDM Datenbank Server wird allein durch die Parameter bestimmt, die dem URL des VDM Clients hinzugefügt sind. Parameter, die in Form einer Zeichenkette über das CGI-Interface übermittelt werden, lassen sich durch vier Eigenschaften charakterisieren:

1. Einzelne Parameter werden in der Zeichenkette durch das Zeichen „&" getrennt
2. Parameter liegen in der Form „parameter=value" vor
3. Mehrere Parameter gleichen Namens sind möglich
4. Zeichen, die nicht Zahlen und Buchstaben sind, werden im Format „%xx" übergeben, wobei xx den hexadezimalen ASCII-Code angibt.

Ferner läßt sich zu CGI-Parametern die Warnung aussprechen, daß abhängig vom eingesetzten WEB-Server Leerzeichen Probleme bereiten können, jedoch nicht müssen. Auf Leerzeichen bei der Parameterisierung wurde im gesamten Projekt verzichtet. Hexadezimale Werte, die durch das Escape-Zeichen „%" angeführt werden, setzt der VDM Server in den gültigen ASCII-Code um.

Alle gültigen Parameter für den VDM Datenbank Klienten, sowie die entsprechenden weitere Subparameter werden nachfolgend definiert und beschrieben.

Ein grundsätzlicher Steuerungparameter ist

conttype=mime — raw — text

Fehlt dieser Parameter gänzlich, ist sein Wert undefiniert oder leer, so wird als Standardwert **text** angenommen. Der Wertbereich von **conttype** hat damit folgende Bewandnis:

mime Es wird ein MIME-Element der VDM Bibilothek referenziert. Zusätzlich wird erwartet, daß die Subparameter **adt**, **impl**, **oper** und **comp** einen gültigen Inhalt in der gesamten Datenbankstruktur referenzieren. In der Regel ist dies als ADT **mimelib**, als Implementation **aux**, sowie als Operation und Komponente ein Eintrag zu der Sparte mimelib, denn um einen „echten" wohlgeformten ADTs handelt es sich bei mimelib nicht. Die MIME Haupt- und Untergruppe als auch der MIME-kodierte Inhalt selbt werden zur Interpretation durch den VDM Datenbank Klienten zurückgeliefert.

raw Es wird ein von ADTs unabhängiger Hilfetext zurückgegeben, der auch Hypertext beinhalten kann und durch den VDM Preprozessor aufbereitet ist. Die Referenzierung des Hypertextes in der Datenbank erfolgt durch die Subparameter **adt**, **impl**, **oper** und **comp** . Das Ausgabeformat ist HTML.

text Der Wert **text** für **contenttype** ist wie erwähnt auch Standardwert, und wird regelmäßig dann angenommen, wenn nicht explizit ein anderer Typ gegeben ist. Die Subparameter für diesen Inhaltstypen sind so vielfältig, daß für sie eine gesonderte Beschreibung notwendig ist.

Die Eigenschaften des Inhaltstyps **text** werden durch den Subparameter **hot** untergliedert. Von dem Wert von **hot** ist die Gültigkeit weiterer Subparameter abhängig. Grunsätzlich ist **hot** nur dann gültig, wenn **contenttype** entweder auf **text**, überhaupt nicht oder auf einen undefinierten Wert gesetzt ist. Mit **hot** wird ein anderer, als der Standardeinsprungspunkt für die Seitengenerierung festgelegt. Ist **hot** nicht oder ungültig definiert, so wird die Seite für die Direktauswahl von ADTs in der Datenbank ausgewählt.

Gültige Werte für **hot** :

setup Es wird eine Seite produziert, welche die komfortable Einstellung der Tabelleneintragsgröße durch den Benutzer zuläßt.

help Verschiedene Hilfsmöglichkeiten werden angeboten. Wählbar sind hierdurch auch die direkt mit **hot** erreichbaren Einsprungsstellen **help_usage**, **help_examples**, und **help_search**.

help_usage Eine Seite mit einer Bedienungshilfe zum System wird produziert

help_examples Die Haupt-Seite der VDM Anwendungsbeispiele wird zurückgegeben, sofern die Subparameter **runtype** und **example** kein bestimmtes Beispiel und dessen Ausführungsart bestimmen. Die zuässigen Werte für die Subparamter werden unter Abschnitt B dieser Arbeit gemeinsam mit detailierten Informationen erläutert.

help_search Einsprung zur den Suchfunktionen

help_search_commit Suche nach Themen. Diese sind ein oder mehrere ADTs, dessen Implementationen oder Operationen. Die äquivalenten Subparameter sind entsprechend **adts**, **opers** und **impls**.

help_search_full Volltextsuche zu gefundenen Themen. Die gefundenen Themen werden anhand der Subparameter **adts**, **opers** und **impls** ermittelt.

enlarge Erweitern der aktuellen Tabelleneintragsbreite in **cols** um 1 und Ausgabe der Standardseite mit Auswahltabelle.

reduce Reduzieren der aktuellen Tabelleneintragsbreite in **cols** um 1 und Ausgabe der Standardseite mit Auswahltabelle.

Alle weiteren Werte führen führen neben **enlarge** und **reduce** ebenfalls zur Standardseite mit Auswahltabelle. Die Auswahltabelle kennt folgende Subparameter

adt Selektierter ADT. Zusätzlich zu den üblichen ADTs der Datenbank bewirkt eine Angabe von **adt=Predefined** die Ausgabe der vordefinierten Abstrakten Datentypen

impl Selektierte Implementation zum selektierten ADT

oper Operation der gewählen Implementation des angegebenen ADTs

general Allgemeine Informationen zum mit **adt** gewählten Abstrakten Datentypen werden ausgegeben

feature Besonderheiten und Merkmale einer mit **adt** und **implementation** gewählten Implementation werden ausgegeben

optype Der globale Operationstypus für jegliche Auswahl wird festgelegt. Gütige Werte sind **C**, **C++**, und **A++**. Der Standard- und Vorgabewert ist **C**.

Zusätzlich sind zwei „globale“ Parameter definiert, die unabhängig von allen weiteren Parametern gesetzt werden können und allein die Darstellung der Auswahl in der Standard- bzw. Heimatseite des Sytems betreffen. Die Namen dieser Parameter sind **cols** und **text**, dessen Definition ist wie folgt:

cols Zunächst soll der globale Parameter **cols** beschrieben werden, welcher grundsätzlich von (fast) allen Seiten „mitgeschleift“ wird. Das bedeutet, daß (fast) jeder Einsprungspunkt des Programms, diesen Parameter in allen produzierten URLs integriert, sofern dieser als Parameter übergeben wurde. Ist **cols** nicht spezifiziert, so wird ein Standardwert eingesetzt. Der Anwender, der in seinem HTML-Browser ein Lesezeichen (*bookmark*) für das VDM Informations- und Hilfesystem einrichtet, kann damit auch seine Konfiguration speichern. Der Wert für **cols** kann durch das Setup als auch durch die Enlarge-und Reduce-Funktionen verändert werden.

text Der Parameter **text** ist für die Ausrichtung des Hilfetextes in Bezug auf die Auswahltabelle verantwortlich. Vorgabe ist die Darstellung des Textes unterhalb der Tabelle. Seine gültigen Werte sind

- **bottom** Anzeige von Hilfstexten unterhalb der Tabelle
- **top** Anzeige von Hilfstexten überhalb der Tabelle mit Justierung der Seite auf den Textanfang
- **topcenter** Hilfstexte werden überhalb der Tabelle angezeigt, die Justierung erfolgt auf dem Beginn der Auswahltabelle zur Erleichterung der Folgeauswahl

Selbstverständlich werden alle Parameter, die nicht der gültigen Struktur entsprechen, stillschweigend ignoriert und vom VDM Server gefiltert. Der „hackende“ Benutzer hat damit nicht die Chance die Datensicherheit gefährdende Operationen auszulösen.

Kapitel 6

Spezifikation der Lösung

6.1 Erweiterung der VDM-Klassenbibliothek

Für die Verwaltung der Hypertextfunktionen sowie des Hyptertextes selbst ist eine Erweiterung der Klassenbibliothek notwendig. Zum ursprünglich vorliegenden Inhalt der Klassenbibliothek zählt auch die Dokumentation zu ADTs, Implementationen und Operationen, welche als Signatur hinterlegt ist und als Komponentennamen „docu" trägt.

Ziel ist es, sowohl die gesamten bisherigen Dokumentationsinhalte der Klassenbibiliothek hyptertexttauglich zu machen, als auch weitere von ADTs unabhängige Textelemente zu schaffen, die direkt oder über die Dokumentation der ADTs zugänglich sind. Die Hinterlegung von Hypertext soll dabei in der Klassenbibiliothek selbst erfolgen, da diese als eine portable und distributionsfähige Einheit zu betrachten ist.

6.1.1 Integration des Hypertext

Die Integration von Hyptertextelementen in den Fließtext, der in der VDM Bibliothek enthalten ist, soll über Makros erfolgen, welche durch den VDM Preprozessor in Referenzen aufgelöst werden, die ausreichend durch den HTML-Seitengenerator des VDM Servers interpretiert werden können. Der Vorteil des Einsatzes von Makros liegt in der Möglichkeit, neben HTML auch anderen Hyptertext erzeugen zu können. Beispielsweise Hypertextformate von Firmen wie Adobe oder auch RTF-Dateien, die mit Hilfe des Help-Compilers in Windows-Hilfedateien übersetzt werden können. Diese Formate sind jedoch nicht Thema der vorliegenden Arbeit, für welche als Zielstellung allein das HTML-Format von Bedeutung ist.

Die Erweiterung um Hyptertext betrifft drei Elemente der Klassenbibliothek, welche wie folgt im Punkt-Format „adt.impl.oper.comp" angegeben werden, wobei der Verzicht auf nachgeordnete Hierarchien bei der Angabe als die Gesamtheit dieser angesehen werden:

1. stdlib.sign.setup.docu
2. stdlib.sign.setup.hyper

3. mimelib.aux

Die Erweiterung des ersten Elementes **stdlib.sign.setup.docu** betrifft die Initialisierung von Dokumentation für die Vorverarbeitung durch den VDM Preprozessor, insbesondere durch Verwendung geeigneter include-Anweisungen. Der derzeitige Inhalt dieser Komponente soll aufgrund seiner Kürze hier wiedergegeben werden:

```
#include <stdlib/sign/setup/gsig>

#define OPERATOR
#define MEMBERS
#define UPPER

#include <stdlib/sign/setup/cpp>

#undef RUMPF
#undef inline
#define inline
#define RUMPF(x)

#include <stdlib/sign/setup/defp>

#include <stdlib/sign/setup/hyper>
```

Als neuer Eintrag ist die Include-Anweisung

```
#include <stdlib/sign/setup/hyper>
```

in der letzten Zeile hinzugefügt worden[1]. Dies ist die geeignetste Stelle in der Datenbank, da der Hypertext eine Übermenge der Dokumentation darstellt. Die Definitionen der Hypertextmakros in der Komponente **stdlib.sign.setup.hyper** die hierdurch eingebettet wird, soll nicht vorenthalten werden und lauten:

```
#ifdef GEN_HYPER
#define __HYPERLINK(l,w) <A VDMREF=MKSTRING(w)>l</A>
#define __IMAGELINK(alttxt,imgref) <IMG SRC=MKSTRING(imgref) ALT=MKSTRING(alttxt)>
#define __IMAGELINKALIGN(alttxt,imgref,align> <IMG SRC=MKSTRING(imgref) ALT=MKSTRING
#define HYPERBIG(txt) <H3>txt</H3>
#define HYPEREXAMPLE(l,r,e) <A EXAMPLE r=MKSTRING(e)>l</A>
#else
#define __HYPERLINK(l,w) l
#define __IMAGELINK(alttxt,imgref) imagine MKSTRING(alttxt) here
```

[1]Die `#include` Anweisungen werden nicht durch den Preprozessor aufgelöst, sondern durch die Hochsprachenerweiterung des Datenbankmanagers, welcher die Dokumentation liefert. Eine rekursive Definition der `#include`-Anweisungen in der Datenbank haben eine Rekursion innerhalb des Datenbankmanagers bei Abruf der Dokumentation zur Folge, welcher daraufhin mit einem Stacküberlauf terminiert.

```
#define __IMAGELINKALIGN(alttxt,imgref,align) imagine MKSTRING(alttxt) here
#define __HYPERBIG(txt) txt
#define __HYPEREXAMPLE(l,r,e) l
```

Es ist erkennbar, daß die Umsetzung der Hyptertext-Makros von der Defininition von **GEN_HYPER** abhängig ist. Der Grund hierfür ist der bereits bestehende VDM Bibliotheksbrowser, welcher die Anzeige einer Hilfe erlaubt, jedoch nur reinen Text akzeptiert. Für ein Bild, welches in einer HTML-Seite durch den Tag `<IMG SRC=pictureref ALT=alttxt>` eingebunden wird, kann und sollte neben einer Referenz auf das Bild auch ein alternativer Text hinterlegt werden, welcher für HTML-Textbrowser[2] anstelle des Bildes angezeigt wird. Wenn lediglich Dokumentation für den Bibliotheksbrowser erzeugt wird, so wird bei diesem anstelle des Bildes nach obiger Definition **imagine „bildname“ here** ausgegeben. Eine ähnliche Bearbeitung gilt für die weiteren Makros.

Die Dokumentation der Klassenbibliothek, die durch den Bibliotheksbrowser angezeigt wird, entstammt ebenfalls der Ausgabe des VDM Preprozessors, welcher mit dem vordefinierten Symbol **GEN_DOCU** gestartet wird. Hieraus folgt, daß der Preprozessor für die Erzeugung von Hypertext sowohl mit dem vordefinierten Symbol **GEN_DOCU** als auch mit **GEN_HYPER** gestartet werden muß. Weitere Informationen zu der Auflösung und der Verarbeitung der aufgelösten Makros durch den Server werden unter 6.4 gegeben.

Das letzte der Klassenbibliothek neu hinzugefügte Element ist **mimelib.aux**. Mimelib trägt diesen Namen, da hier jegliche hinterlegte Informationen gültige MIME-Typen sind. Da es sich auch bei HTML (mit dem Mime Haupt- und Untertyp text/html) um einen gültigen Typen handelt, ist auch über die übliche Dokumentation der ADTs hinausgehender Hypertext unter der ADT-Bezeichnung **mimelib.aux** abgelegt. Die Gruppierung unterhalb der Implementations-Bezeichnung *aux* (*auxiliary*) kann per Definition der Datenbank frei gestaltet werden. Bilder könnten nun unter der Operations-Bezeichung *pics* als Komponente mit ihrem Namen gespeichert werden[3] und weiterer Hypertext unter anderen Bezeichnungen.

Die Kreation des ADT-Eintrages **mimelib** ist jedoch nicht Bedingung, es können auch Einträge anderen Namens in der Datenbank erfolgen. Der Zugriff auf die Komponenten erfolgt direkt. Es ist lediglich darauf zu achten, daß die Integrität der Datenbank nicht durch Einträge, die ganz oder teilweise der Struktur von ADT-Definitionen entsprechen, verletzt wird.

6.1.2 Design der Hytertextreferenzen

Die durch Makros in den Fließtext eingebundenen Hypertextelemente, werden nach der Bearbeitung durch den Preprozessor in Referenzen aufgelöst. Die Spezifikation hierzu wurde unter 6.1.1 dargelegt. Die durch den Preprozessor erzeugten Referenzen bedürfen einer Weiterverarbeitung durch den VDM Server.

[2] Der wohl bekannteste HTML-Textbrowser ist Lynx

[3] Es ist empfehlenswert als Komponentenname den Namen des Bildes zu wählen, da beim Export mit Hilfe des Bibliotheksbrowsers der Komponentennames mit einer anhand des Mimetypes ermittelten Erweiterung als Dateiname vorgeschlagen wird

Bei den erzeugten Konstrukten handelt es sich um HTML-ähnliche Tags, die aber nicht Bestandteil der HTML-Definitionen sind. Eine direkte Wiedergabe durch den HTML-Browser würde ein schlichtes Ignorieren dieser unbekannten Tags zur Folge haben. Da der VDM Server jedoch alle unbekannten Tags als Text interpretiert, werden diese in der HTML-Kodierung für den Latin-1 Zeichensatz wiedergegeben, können daher vom Browser auch nicht als Tags erkannt werden und die Darstellung erfolgt äquivalent zum Ursprungstext.

Die Umsetzung des zwar durchaus gültigen Tags des HTML-Standards

```
<ADDRESS>Marko Temme</ADRESS>
```

hat nach einer Bearbeitung durch den VDM Server folgende Ausgabe zur Folge, wenn dieser den Tag nicht kennt:

```
&ltADDRESS&gtMarko Temme&lt/ADRESS&gt
```

Eine Erweiterung der Hypertextmakros ist jederzeit möglich. Verständlicherweise können neue Makros aber nur in solche Tags substituiert werden, zu dessen Interpretation der VDM Server in der Lage ist. Dem Server ist zunächst ein fester Satz von Tags bekannt, dessen Einsatz immer möglich ist. Nun bietet auch HTML selbst zahlreiche Tags, die den Gestaltungsmöglichkeiten des Hyptertextes in der Klassenbibliothek nicht vorenhalten sein sollen.

Diese Tags können über die Konfigurationsdatei des Servers explizit bekannt gemacht werden, wodurch eine Umsetzung in den Latin-1 Zeichensatz vermieden wird. Die Vornahme dieser Konfiguration ist unter C.3.2 erläutert. Das Hinzufügen von <ADDRESS> sowie </ADRESS> zu den bekannten Tags würde die Umsetzung im obigen Beispiel verhindern.

Der feste Kommandosatz des VDM Servers besteht aus folgenden drei Tags, welche eine Vielfalt von Einsatz und Kombinationsmöglichkeiten bieten:

1. <A VDMREF=*reference*>*description*</A>
2. <IMG SRC=*reference [ALIGN=align] ...*>
3. <A EXAMPLE *RUN* | *VIEW*| *REF* = *example*>*description*</A>

Der Tag **VDMREF** ist die klassische Möglichkeit, Verweise aus der ADT-Dokumentation oder anderen Dokumenten zu weiteren Dokumenten, ADT-Dokumentationen der Klassenbibliothek, Bildern oder anderen MIME-Elementen herzustellen. Auf was genau durch **VDMREF** verwiesen ist, wird durch *reference* festgelegt.

Syntax und Spezifikation von *reference* ist wie folgt:

1. **Referenzierung von ADT-Elementen der Klassenbibliothek**

   Durch *adt.impl.oper* wird die Dokumentation zur Operation *oper* der Implementation *impl* des ADTs *adt* referenziert. Lautet die Referenz *bag.automem.all* wird die Operation *all* der Implementation *automem* des ADTs *bag* referenziert. Die Dokumentation eines ADTs ist abhängig vom gewählten Operationstypus, der die Zielsprache festlegt. Als Operationstypus wird der im Informations- und Hilfesystem bereits durch den Anwender ausgewählte verwendet.

   Soll durch die Referenzierung ein Wechsel des Operationstypes zu einer anderen Zielsprache erfolgen, so wird dieser der Referenz vorangestellt, so daß diese damit lautet: *opertyp.adt.impl.oper*, wobei derzeit gültige Werte von opertyp „C", „C++" oder „A++", für die in C++ implementierte automatische Speicherverwaltung, sind. Die große oder kleine Schreibweise des Operationstyps ist nicht von Belang, die Angabe ist nicht „case-sensitive". Eine gültige Referenz auf die Operation *mk* der Implementation *linklist* des ADTs *map* für die Zielsprache C++ würde somit lauten *c++.map.linklist.mk*. Ist der momentan im Informations- und Hilfesystem gewählt Operationstypus mit der Angabe des neuen identisch, hat diese Art der Referenzierung verständlicheweise keine Wirkung.

   Die Dokumentationen der ADTs bzw. dessen Operationen und Implementationen, die durch diese Art der Referenzierung aus der Klassenbibliothek gewonnen werden, sind identisch mit denen, die mit dem Tool **vdmhelp** der VDM Klassenbibliothek angezeigt werden können. Sie werden nach den Regeln, welche die Wohlgeformtheit eines ADTs einschließlich seiner Dokumentation, festlegen, gebildet.

2. **Referenzierung von Elementen der gesamten Klassenbibliothek**

   Dem Zugriff durch das Hypertextsystem bleibt kein Inhalt der Datenbank verborgen. Zu Informationszwecken können selbst Codefragmente oder Parametrisierungen zur Anzeige gebracht werden, wobei hier selbstverständlich die Definition von Tags und Referenzen zwar möglich aber nicht ratsam ist[4].

   Die direkte Referenzierung erfolgt durch **raw:adt.impl.oper.comp** gemäß der vierstufigen Hierarchie der Datenbank. Das Schlüsselwort *raw* bezeichnet „puren" Text, der jedoch auch der Bearbeitung durch den Preprozessor unterliegt und demnach Verweise, Bilder sowie weitere Hypertextelemente enthalten kann.

   Diese Form der Referenzierung macht es möglich neben ADT-spezifischen Informationen auch solche allgemeiner Natur in der Bibliothek zu hinterlegen.

   Ein Beispiel für eine gültige Referenzierung ist **raw:mimelib.aux.help.common**, welche auch tatsächlich in der Distribution der VDM Klassenbibliothek existiert und allgemeine Hinweise zur Gestaltung von Hypertext enthält. Es obliegt dem Gestalter des Hypertextes die Pflicht, bei der Kreation neuer Hypertexte die Wohlgeformheit der Datenbank nicht zu verletzen. Sie werden daher in der Regel unter der Implementationsrubrik „aux" einzurichten sein.

3. **Referenzierung von MIME-Elementen der Klassenbibliothek**

[4] Tags, die in Codefragmenten definiert werden, könnten durch die Auflösung von Makros in Nichts vom Preprozessor beseitigt werden

HTML bietet als Möglichkeit neben der Darstellung formatierten als auch unformatierten Texts auch die Wiedergabe von Bildern, Filmen, Ton und anderen Medien an. Diese liegen in der Regel in einem binären Format vor, das die Aufnahme in die VDM Klassenbibliothek verhindert. Sie werden daher in der standardisierten MIME-Kodierung niedergelegt, die im wesentlichen aus Zeichen und Zahlen besteht und daher für die Aufname in die Datenbank geeignet ist. Die Art der Daten wird anhand der MIME Haupt- und Untertypen klassifiziert, welche per Definition in der ersten Zeile der MIME-Komponente in der Datenbank festgelegt werden.

Anhand der Referenz erkennt der VDM Server, ob es sich bei dem referenzierten Inhalt um Text oder eine MIME-Komponente handelt. Die Dekodierung des MIME-Formats erfolgt jedoch nicht durch den Server selbst, sondern er teilt dem VDM Server-Klienten mit, daß ein solches Format zu interpretieren ist.

Die Referenz auf eine MIME-Komponente unterscheidet sich von der vorstehend beschriebenen „puren" *raw* Referenz durch das Schlüsselwort *mime*. Die Referenzierung wird damit durch **mime:adt.impl.oper.comp** vorgenommen. Ein Beispiel für die Referenzierung eines Bildes wäre **mime:mimelib.aux.pics.vdmlogo**.

6.2 Trennende Zerlegung der Server- und Clientaufgaben

Dem VDM Server sollen grundsätzlich so viele Aufgaben wie möglich übertragen werden. Bei einigen ist dies unmöglich, dies betrifft die Kodierung und -Dekodierung von binären Daten, die vom VDM-Server im MIME-Format vorgehalten werden müssen, da TCL eine Verarbeitung des ASCII-Codes 0 nicht zuläßt und jede Zeichenkette mit diesem Zeichen terminiert. Daraus folgt, daß Kodierung und Dekodierung von binären Daten wie Bildern ausschließlich durch Klienten erfolgt, die sich zu diesem Zweck eines externen Konverters bedienen.

Die Trennung der Aufgaben soll zunächst hinsichtlich des Bibliotheksbrowsers und -managers erfolgen, wenn diese als Klient eingesetzt werden. Während der Server die Datenbank auf seinem Host lädt und speichert, wird der Import und Export von Datenbankinhalten als auch der Datenbank selbst auf dem Host der Klienten durchgeführt.

Da Browser und Manager den Datenbankmanager über eine Pipe aufrufen, muß sich der VDM Datenbank Klient, der auf dem Host des Klient anstelle des Datenbankmanagers aufgerufen wird, genauso verhalten, wie es der Datenbankmanager auf dem gleichen Host getan hätte. Er muß diesen also emulieren. Die definierten Funktionen werden vom VDM Klient als Stumpf (*stub*) definiert. Die Anfragen werden an den Server weitergeleitet, von diesem beantwortet und an den Klienten zurückgegeben, der die Antwort seinerseits an das aufrufende Programm zurückgibt.

Da der Bibliotheksbrowser teilweise davon ausgeht, daß insbesondere beim Export von Datenbankinhalten der Datenbankmanager eine Datei auf dem gleichen Host erstellt, muß in diesen Fällen der VDM Datenbank Klient diese Aufgabe übernehmen, und die Datei auf dem Host des Klienten erstellen. Die Abwicklung dieser Funktionen wird über die Protokolle definiert, die diesem Abschnitt folgen.

6.3 Festlegung und Definition der Kommunikationsprotokolle

Den gesamten Datenverkehr zwischen Tools, den VDM Klienten als auch dem VDM Server regeln Kommunikationsprotokolle, die verbindlich die Art der ausgetauschen Daten vorschreiben und auch die Flußkontrolle regeln.

6.3.1 VDMDBM und VDMHELP zu VDM Datenbankmanager

Die Kommunikation zwischen Bibliotheksbrowser und -manager war Gegenstand der ersten Analysearbeiten. Bereits hier war eine verbindliche Art des Datenaustausches definiert. Der Datenbankmanager arbeitet als Filter, akzeptiert Befehle über die Standardeingabe und gibt das Resultat über die Standardausgabe zurück. Die Flußsteuerung legt bei der Ausagbe die Größe der Daten fest, die vom Klienten gelesen werden müssen, da das System ansonsten „hängt“. Die Kommunikation ist in den TCL-Dateien *commdbm.tcl* und *communicate.tcl* programmiert, und wurde unverändert gelassen.

Das Protokoll besteht aus insgesamt drei Kommandos für das Absetzen von Befehlen an den Datenbankmanager. Diese sind:

```
evalCmd <arg1> <arg2> <...>

evalMessage <arg1> <arg2> <...>

evalMessageToFile <file> <arg1> <arg2> <...>

exit
```

Das Kommando `evalMessageToFile` erwartet als ersten Parameter einen Dateinamen, in welche die durch den Befehl erzeugte Ausgabe geleitet wird.Wärend für `evalCmd` keine Antwort und lediglich die stillsschweigende Ausführung des Kommandos erwartet wird, geben `evalMessage` und `evalMessageToFile` eine Antwort in folgendem Format zurück:

```
error | ok <nl>

length <nl>

message <nl>
```

Die erste Zeile gibt an, ob das Kommando erfolgreich ausgeführt wurde, und besteht entweder aus `error` oder `ok` aus. Es folgt in der nächsten Zeile die Länge der Nachricht, die übermittelt wird, einschließlich aller Zeilenumbrüche. Anschließend wird die Nachricht empfangen, die mit einem Zeilenumbruch abgeschlossen ist.

Wenn der **vdmdbm** oder **vdmhelp** beendet werden, wird das Kommando `exit` an den Datenbankmanager abgesetzt, um diesen zu beenden.

6.3.2 VDMDBM und VDMHELP zu VDM Client zu VDM Server

VDMDBM und VDMHELP zu VDM Client

Da sich der VDM Client im kompatiblen Modus genauso verhält wie der Datenbankmanager, versteht dieser das gleiche Protokoll, das in 6.3.1 dargestellt ist. Der

`evalMessageToFile`

Befehl wird durch den Client in das Kommando

`evalMessage`

verwandelt und an den Server weitergereicht. Das Resultat wird anschließend vom Client in die angegebene Datei geschrieben. Das Kommando

`exit`

wird vom VDM Client abgefangen und führt nicht zur Beendigung des VDM Servers. Vielmehr meldet sich der Client vom Server ab und terminiert anschließend.

VDM Client zu VDM Server

Der VDM Client verständigt sich mit dem VDM Server ausschließlich über Remote Procedure Calls, die auf der Sitzungschicht basieren. Da von der Sitzungsschicht abstrahiert werden kann, ist es letzendlich belanglos, wie das Protokoll der Sitzungsschicht im einzelnen implementiert sind. Die Funktionalität ändert sich durch die Implementierung des Protokolls nicht, da die Dienstelemente vom Modell vorgegeben werden. Dies gilt auch für die Implementierung der RPCs selbst, dessen einfaches Protokoll unter 7.2.2 dargestellt ist.

6.4 Erweiterungen für MIME

Zur Einbettung von Bildern und anderen binären Daten in die Klassenbibliothek wird ein externer Mime Coder als auch Decoder benötigt, der genauso portabel sein muß, wie die Klassenbibliothek. Das Paket Metamail von Bell enthält einen solches Programm, der dieses Kriterium erfüllt. Dieser Coder bzw. Decoder mit dem Namen **mmencode**, wird für die Klassenbibliothek übernommen.

Da MIME-kodierte Daten zum einen in die Datenbank übertragen werden müssen, als auch ausgelesen werden sollen ist eine Modifikation des Bibliotheksbrowsers **vdmdbm** erforderlich, damit dieser in die Lage versetzt wird, die Aufgabe übernehmen zu können. Ferner muß auch der VDM Client **dbmc** im Web-Betriebsmodus in der Lage sein MIME-Daten aus der Bibliothek decodieren und an den Web-Server übergeben können.

Eine MIME-Komponente in der Datenbank unterscheidet sich nach der Kodierung nicht von anderen Komponenten, es handelt sich lediglich um Text. Da der Web-Server vor der Ausgabe der in Binärdaten dekodierten MIME-Komponente vom VDM Client die Angabe des Mime-Typen erwartet, muß auch dieser in der Datenbank hinterlegt werden. Es wird daher spezifiziert, daß die erste Zeile einer Mime-Komponente in der Datenbank diesen Typen enthält und diesem die kodierten Daten folgen. Dies sähe damit für ein GIF-Bild wie folgt aus:

```
image/gif
AKJHGHDJLNfasdfhasdjkfhdsaf
JLKJFlfdksajflfsadfjLJKlkjLKJlkjf
.....
```

Um das Pflegen der MIME-Komponenten in der Datenbank durch den Bibiliotheksmanager zu ermöglichen, ist die Ergänzungen der Funktionalität um den Import und Export von MIME-Komponenten in bzw. aus der Datenbank nötig. Da ohne eine selbsttätige Erkennung des Mimetypen einer importierten Datei der Mimetyp manuell für die Datenbank-Komponente nachgetragen werden müßte, wird für die automatische Erkennung die Namenserweiterung (*extension*) der Quelldatei ausgewertet.

Umgekehrt muß daß System beim Export anhand des Mime-Typen der Komponente die Namenserweiterung vorschlagen[5].

Da Bilder für die nahtlose Einbindung in Web-Seiten über transparente Hintergrundfarben verfügen müssen, soll für das gebräuchlichste Bildformat GIF die Einbindung eines Tools in den Bibiliotheksmanager vorgenommen werden, das die Umwandlung von Bildern in dieser Hinsicht vor dem Import in die Datenbank zuläßt.

[5]Die Zuordnung zwischen Extension und Mime-Typ im lauffähigen System wird in einer Datei mit dem Namen *mime.types* vorgenommen, die auch Browsern und Web-Server regelmäßig beiliegt. Sie ist im Verzeichnis *./lib/vdm* der Klassenbibliothek zu finden.

Kapitel 7

Implementierung

7.1 Programmierung der Sitzungsschicht mit TCL

7.1.1 TCP-Sockets als Basis

Die vollständige Implementierung der Client-Server Lösung unter TCP basiert auf drei TCL Befehlen, die ein Neuzugang der Version 7.5 der Tool Command Language sind. Aufgrund der Bedeutung dieser Befehle für das Projekt, sind diese einer eingehenden Untersuchung unterzogen worden, und sollen Erwähnung finden.

Die Interpreterbefehle mit ihrer Syntax sind

1. (a) socket [-myaddr addr] [-myport port] [-async] host port
   (b) socket -server command [-myaddr addr] port
2. (a) fileevent channelId readable [script]
   (b) fileevent channelId writable [script]
3. fconfigure channelId [value = name value=name ...]

Der **socket**-Befehl erlaubt nicht nur die Erzeugung eines horchenden (*listening*) sockets für einen festgelegten Port, um einen Server zu konstruieren, sondern auch die Kreation eines passenden Kommunikationsendes für einen Client.

Das **socket**-Kommando verlangt als Parameter eine TCL-Anweisung (*command*), die ausgeführt wird, wenn ein Client beim Server „anklopft“ als auch den Port (*port*), über den sich jegliche Klienten anmelden dürfen. Die Anmeldung geschieht von der Clientseite aus durch Aufruf des **socket**-Befehls unter Angabe des Serverhosts und Listening-Ports.

Das Anmeldeprocedere, das zu einer eindeutigen Verbindung führt, wird vom TCL-Interpreter übernommen. Der Klient kontaktiert den Server am listening-port, der ihm bekannt ist. Der Server teilt dem Klienten nur eine neue exklusive Verbindung mit, über die zukünftig kommuniziert wird. Anschließend wird der listening-port wieder freigegeben, damit auch anderen Clients die Möglichkeit haben sich anzumelden.

Nachdem vom TCL-Interpreter eine neue Verbindung (mit einer neuen Portnummer) ausgehandelt wurde, wird die TCL-Anweisung *command*, die bei der Einrichtung des Serversockets als Parameter angegeben war, mit der Angabe des neuen Sockets sowie weiteren Informationen über den Client als Ereignis (*event*) aufgerufen.

Das aufgerufene Kommando *command* ist in der Regel der Name einer TCL-Prozedur, in der nun über die sockets ebenso wie mit einem Dateideskriptor durch Lesen und Schreiben Daten ausgetauscht werden können. Der Client erhält diesen Socketdeskriptor unmittelbar als Rückgabewert.

Der Server könnte noch in der aufgerufenen Prozedur mit der Kommunikation beginnen. Dies ist jedoch nicht in allen Fällen zweckmäßig, da ein unmittelbarer Datenaustausch nicht erforderlich sein muß, und zunächst mit dem TCL-Programm fortgefahren werden soll. In diesem Fall bietet das Kommando **fileevent** Abhilfe. Hiermit läßt sich festlegen, daß nur dann jeweils ein Skript aufgerufen werden soll, wenn Daten am socket anliegen oder dieser beschrieben werden kann. In diesen Fällen wird jeweils ein Ereignis ausgelöst, das die Kommunikation abwickelt.

Mit Hilfe des **fconfigure** Kommandos können Eigenschaften von Datei- und Socketdeskriptoren verändert werden. Wesentliche Eigenschaften sind die Art der Umsetzung von Neuzeilen, Puffer und die Festlegung von blockierendem oder unblockierendem Modus. Der VDM Server arbeitet auschließlich im voreingestellten blockierendem Modus, der sich dadurch auszeichnet, daß auf Eingaben gewartet wird. Der nichtblockierende Modus kehrt sofort ohne Resultat zurück, wenn keine Daten zu lesen sind. Ferner wird vom VDM Server die Pufferung ausgeschaltet, so daß sich ein Durchspülen der Leitung erübrigt.

7.1.2 Eventverarbeitung

Der TCL-Interpreter stellt kein Multitasking zur Verfügung, gestattet aber, daß zwischen der Abarbeitung von zwei Befehlen das laufende Programm unterbrochen wird, um vorübergehend an einer anderen Stelle weiterzuarbeiten. Das Programm wird jedoch nicht immer selbsttätig unterbrochen, sondern dazu Bedarf es der Ausführung des TCL-Kommandos **update** durch das laufende Programm, welches intern über die Funktion *Tcl_DoOneEvent* ein Ereignis aus der Warteschlange abarbeitet, in die alle Ereignisandorderungen gestellt werden.

Für einen Server, dessen Aufgabe es ist, auf die Wünsche der Clients zu reagieren, würde dies in der Zeit wo kein Wunsch vorliegt bedeuten, in einer Endlosschleife fortwährend den Befehl **update** auszuführen. Dies ist sehr rechenzeitintensiv.

Eine bessere Lösung ist es, den Server **update** nur dann ausführen zu lassen, wenn tatsächlich ein Ereignis vorliegt und ansonsten am besten stillzustehen. Dies kann mit dem TCL-Befehl **vwait** realisiert werden, der als Parameter eine globale Variable erwartet und solange an dieser Stelle ohne nennenswerten Verbrauch von Rechenzeit verbleibt, bis diese Variable durch eine Ereignisprozedur auf einen beliebigen Wert gesetzt wird, auch wenn sich der Inhalt der Variable dadurch nicht verändert. Entscheidend ist lediglich der Zugriff. **vwait** stoppt lediglich das aktuelle Programm, läßt aber weiterhin die Abarbeitung eines einzigen Events zu.

Die Endlosschleife des Servers muß zum sparsamen Umgang mit Rechenzeit daher folgendes Aussehen haben:

```
while {1} { update ; vwait var }
```

Um zu verhindern, daß die Abarbeitung von Ereignissen ins Stocken kommt, ist es erforderlich, daß nach jedem Ereignis **update** ausgeführt wird. Dies wird dadurch gesichert, daß jede Ereignisprozedur die globale Variable *var* am Ende der Bearbeitung setzt.

7.1.3 Zugriff auf die Sitzungschicht

Der Zugriff auf die Sitzungsschicht erfolgt durch einen SSAP (Session Service Access Point). Der SAP der in TCL implementierten Sitzungsschicht besteht aus einer Liste, in der allerdings nur die Rückruf (*Callback*) Funktionen enthalten sind, die selbstätig von der Sitzungsschicht aufgerufen werden, um die über ihr liegende Schicht mit Daten zu versorgen. Datenverkehr in der umgekehrten Richtung findet durch Aufruf der Dienstelemente der Sitzungschicht statt, dessen TCL-Prozedurname der OSI-Definition entspricht, wie beispielsweise CONNECT.request.

Als Prozeduren für die Liste, welche den SAP darstellt, wird erwartet:

1. Eine Login Funktion zur Anmeldung,
2. eine Logoff Funktion zur Abmeldung,
3. eine Funktion, die bei eintreffenden Daten aufgerufen wird und
4. eine Prozedur zur Ausnahmebehandlung (*exceptions*)

Den einzelnen Funktionen sind Argumente, die sie für die Kommunikation und andere Zwecke benötigen, vorbestimmt, die an dieser Stelle jedoch keine Erwähnung finden sollen. Zu beachten ist jedoch, daß das alle Funktionen dazu verpfllichtet sind, selbst **update** auszuführen oder ein Signal für eine Serverschleife zu setzen, die dies tut.

7.2 Distributed-Processing

7.2.1 Remote-Procedure Calls mit TCL

Basierend auf Sitzungsschicht, sind Vollduplex-RPCs implementiert. Das bedeutet, das nicht nur der Client am Server, sondern auch der Server am Client Kommandos ausführen kann. Dies ist beim VDM-Server jedoch selbst nicht möglich, da dieses Feature zugunsten einer Kommandozeile geopfert wurde.

Zu der RPC Implementation existieren drei Funktionen. Die Einrichtung eines Server, die eines Clients, und das Kommando zum Absetzen eines RPCs. Zunächt soll das Kommando zur Einrichtung eines RPC-Servers dargestellt werden:

```
proc rpc_MakeServer {port {loginFunc ""} {logoffFunc ""} {checkFunc ""} \
                         {errFunc ""}}
```

Es wird der Listening-Port festgelegt, sowie Callback-Funktionen zum Anmelden, Abmelden, zum Überprüfen von Kommandos und zur Behandlung von Ausnahmen.

Der Client meldet sich am RPC-Server mit folgender Funktion an, und übergibt ebenfalls eine Callback-Funktion für die Überprüfung von RPC-Kommandos die vom Server aufgegeben werden als auch eine für die Behandlung von Ausnahmebedingungen. Als Funktionswert erhält der Client, wenn er vom Server akzeptiert wurde, einen Socketdeskriptor als Resultat.

```
proc rpc_MakeClient {host port {checkFunc ""} {errFunc ""}}
```

Sowohl Server als auch Client setzen RPCs mit folgendem Kommando ab. Das Argument **server** bezeichnet den Socketdeskriptor der Gegenstelle, denn auch der Client kann als RPC-Server bedienen. Währen dem Client der Deskriptor durch **rpc_MakeClient** bekannt ist, muß der Server die globale Variable **rpcFile** verwenden, die durch das RPC-System vor Aufruf der Callback-Funktionen gesetzt wird.

```
proc rpc_Exec {server args}
```

7.2.2 Kommunikationsprotokoll

Die für RPCs nötige Kommunikation besteht aus dem Absetzen des Kommandos und dem Empfang des Resultats. Zunächst wird die Kommunikation durch das Dienstelement S-DATA.request eingeleitet, was auf der Gegenstelle die Sitzungsschicht veranlaßt, die Callback-Funktion aufzurufen, die für den Empfang von RPCs definiert wurde. Da das Packen von PDUs für die Kommunikation zu aufwendig wäre, wird anschließend direkt über die Socketverbindung kommuniziert. Dabei wird in beiden Richtungen das gleiche Protokoll verwendet.

```
len [type] <nl>
message
```

len gibt die Länge der Nachricht *message* an, optional kann ein Typ angegeben werden, der nach Ausführung des RPCs dazu verwendet wird, den Erfolg der Operation mitzuteilen.

7.2.3 Fehlerbehandlung

Die Ausführung von Fernkommandos kann auf der Gegenstelle zu Fehlern oder zur korrekten Bearbeitung führen. Da der Fehler selbstverständlich beim Aufrufer und nicht beim entfernten Host angezeigt werden soll, wird dieser bei der Ausführung mit dem TCL-Kommando **catch** abgefangen.

Die Fehlermeldung wird mitsamt der Information, daß ein Fehler aufgetreten ist, an den Aufrufer übertragen. Dort wird der Fehler mit dem TCL-Kommando **error** unter Angabe der Meldung neu erzeugt. Überwacht der Aufrufer seinerseits das RPC-Kommandos mit **catch**, so macht es für die Fehlerbehandlung keinen Unterschied, ob ein Fehler lokal oder entfernt auftritt.

7.3 VDM Datenbankserver

Der VDM Datenbank Server **vdmd** arbeitet auf Basis von RPCs. Grundsätzlich könnte jede Prozedur des Servers aufgerufen werden oder auch auf Ebene 0 Kommandos abgesetzt werden. Der VDM Server Client **dbmc** ruft lediglich beim Direktzugriff im Online-Modus Kommandos auf der untersten Ebene auf, und verwendet ansonsten nur Funktionen, die eigens zur Lösung seines Problems erstellt wurden.

Die konkreten Prozeduren des Servers, die vom VDM Client aufgerufen werden sind:

- **html_main**, wenn Web-Seiten produziert werden sollen.
- **agent_Connect**, **agent_Disconnect** und **agent_Call**, wenn der Client im Kompatibliätsmodus läuft, da der Bibliotheksbrowser oder -manager ihn benutzen. Gleiche Kommandos werden auch im Online-Modus des VDM Client benutzt, wenn explizit eine Kommunikation mit dem Benutzeragenten gewünscht ist.

7.3.1 Server-Kommandozeile

Eine Fernwartung mit TELNET als auch die direkte Eingabe von Kommandos für den VDM Server, wäre nicht möglich, wenn eine Serverschleife existieren würde, wie sie unter 7.1.2 abgebildet is. Es hatte sich insbesondere bei Test und Entwicklung als außerordentlich ärgerlich erwiesen, Kommandos nicht direkt am Server eingeben und damit auch nicht über die globale Variable *errorInfo* Fehler diagnostizieren zu können.

Daher wurde mit dem bereits beschriebenen TCL-Befehl **fconfigure** eine Ereignisprozedur für den Deskriptor der Standardeingabe **stdin** definiert, die Eingaben aufnimmt und mittels **uplevel** auf Ebene 0 ausführt. Auch nicht abgeschlossene Kommandos, oder Zeilen, die mit einem Backslash verlängert werden, werden verarbeitet. Damit präsentiert sich der Server wie eine interaktiv ausgeführte TCL-Shell.

Dies ist auch die Vorraussetzung für die Fernwartung, da Standardeingabe, -ausgabe und Fehlerausgabe sonst zwar an den fernen Host übertragen wären, dieser aber ebenfalls keine Kommandos ausführen könnte.

7.3.2 Steuerung der Mehrfachverarbeitung

Server müssen in der Lage sein eine Vielzahl von Klienten (*Clients*) zu bedienen. Wenn sich zwei Kunden des Server gleichzeitig dazu entschließen, auf gleiche Ressourcen zurückzugreifen, muß der Server in der Lage sein, auch solche Konfliktsituationen zu verarbeiten. Die Lösung dieser Probleme liegt in der Regel in der Verwendung von Sperren oder von Warteschlangen oder in einer Kombination aus beidem. Die Warteschlange ist eine Alternative zum bloßen Zurückweisung.

Die Verarbeitung von Mehrfachzugriffen bietet viele Möglichkeiten einer Optimierung. Die bekannteste ist das Multi-Thread Verfahren, in dem sämtliche Klient-Anforderungen in eine Warteschlange eingereiht, und asynchron mit der günstigsten

Bearbeitungsfolge ausgeführt werden[1]. Viele Datenbanksysteme beispielsweise, optimieren in diesem Zusammenhang mengenorientierte Anfragen und optimieren diese oder lösen diese vor der Bearbeitung auf.

Einzelne Aufgaben der Klienten werden so weit als möglich in weitere Abschnitte unterteilt. Der kleinste nicht mehr teilbare selbständige Aufgabenabschnitt mit der Eigenschaft, reentrant und beliebig oft aufgerufen zu werden, wird als **Thread** bezeichnet und läßt sich treffend mit Nadelöhr übersetzten. Auf Mehrprozessormaschinen werden **Threads** zur optimalen Auslastung durch den Dispatcher-Task verteilt.

Der Server des Informations- und Hilfesystems wurde mit Single-Thread Verarbeitung konzipiert, obwohl die Möglichkeit eines Multi-Thread Servers durchaus bestand. Die Gründe, die zu diesem Enschluß geführt haben, setzten sich zum einen aus dem Problem zusammen, daß die VDM Datenbank im Arbeitsspeicher vorgehalten wird und bei Erzeugung von Kindprozessen kopiert wird und zum anderen aus der Eigenschaft von TCL, Anfragen über das Netzwerk solange als Ereignisse (*events*) in einer Warteschlange einzureihen, bis ein Befehl zur Abarbeitung von Ereignissen den TCL-Interpreter erreicht.

Die Schaffung von pseudo-preemptiven Multitasking in TCL durch Einfügen dieses „update“-Befehls[2] an unterschiedlichen Stellen hätte einen Deadlock in der Ereignis-Verarbeitung mit sich gebracht, da die Bearbeitung eines Ereignisse, daß bearbeitet wird, nach der Ausführung von update in der Ereignisfunktion selbst dafür gesorgt hätte, daß das nächste Ereignis in der Warteschlange ausgeführt, und das aktuelle nie zuendegebracht würde.

Eine Hochprachenimplementierung und Einbettung einer solchen Funktionalität in die TCL-Shell wurde als nicht sinnvoll erachtet, da immerhin die Möglichkeit besteht, daß in Nachfolgeversion von TCL 7.5 die Fortsetzung von Ereignissen implementiert wird, und diese Schwachstelle damit beseitigt ist. Eine Funktion zum Neueinreihen von Ereignissen in die Warteschlange von TCL wäre hierzu völlig ausreichend.

Der Entschluß, Single-Thread Verarbeitung zu wählen, stellte sich insbesondere bei der Implementierung der VDM Anwendungsbeispiele als richtig heraus, denn sobald die Generierung eines VDM Beispieles gestartet wird, kann vor der Beendigung parallel hierzu keine erneute Erzeugung gleichen Beispiels gestartet wird. Ferner sind Deadlocks durch gegenseitige Blockierungen ausgeschlossen. Nachteilig zeigt sich diese Methode jedoch bei der Volltextsuche in der Datenbank, wo bei großem Suchumfang der Server für längere Zeit blockiert sein kann.

7.3.3 Das Transaktionskonzept

Transaktionen (*transactions*) sind eine Sammlung von Verarbeitungsschritten, die entweder vollständig, oder gar nicht durchgeführt werden. Die Ausführung einer Transaktion wird als Atomaktion bezeichnet. Üblicherweise wird vom Klienten eine Folge von Befehlen abgesetzt, und zum Schluß die Transaktion abgeschlossen. Anschließend ist es Sache des Servers diese Atomaktion ohne Unterbrechung des Clients auszuführen.

[1] Ein typisches Beispiel das patentierte „Elevator-Seeking“ des Netzwerkbetriebssystemherstellers Novell, wo Festplattenzugriffe der Clients so geordnet werden, daß der zeitaufwendige Spurwechsel der Festplatte reduziert und möglichst viele Blöcke in Folge gelesen werden können

[2] Es handelt sich um das TCL-Kommando„update“

Wenn der Client sich entschließt, die Operation abzubrechen, etwas schiefgeht, oder die Klientverbindung zwischenzeitlich abgebrochen ist, wird die Atomaktion nicht ausgeführt, und damit keine der enthaltenen Aktionen.

Bei der Implementierung des VDM Servers des Informations- und Hilfesystems werden Transaktionen zwar unterstützt, jedoch aufgrund der Eigenschaften von TCL nicht vollständig. Transaktionen werden in der Regel dadurch vollzogen, daß eine Kopie des veränderten Objektes gezogen, und anhand dieser der Originalzustand wiederhergestellt werden kann (*rollback*). Transaktionen wurden im VDM Server mit Hilfe des TCL-Befehls „catch“ implementiert, welcher den TCL-Interpreter rekursiv mit gleicher TCL-Umgebung aufruft und dort spezifizierte Kommandos ausführt, alle Änderungen bei Auftreten eines Fehler verwirft oder ansonsten die Inhalte des rekursiv aufgerufenen Interpreters an den Aufrufer-Interpreter überträgt.

Nicht verworfen werden können durch den „catch“-Befehl alle Objekte, die sich auf Betriebsmittel, beziehen, oder in Hochsprachenmodulen in der TCL-Shell eingebettet sind. So kann verständlicherweise das Anfügen oder Löschen einer Datei genausowendig rückgängig gemacht werden, wie Veränderungen, die in eingebetteten Hochsprachen durchgeführt werden, obwohl sich dies hier durchaus durch ausschließliche Verwendung von Variablen des TCL-Interpreters über die Schnittstellenfunktionen von TCL realisieren ließe.

Das Transaktionskonzept ist insofern im VDM Server integriert, als daß alle Anfragen über Remote-Procedure Calls (RPCs) ausgeführt, und diese vor der Ausführung vollständig in Textform übermittelt werden. Als RPC kann an den Datenbankserver nicht nur ein einzelner, sondern auch mehrere oder eine ganzer Verarbeitungsblock von Befehlen übergeben werden. TCL-Befehle, welche die VDM Datenbank betreffen und diese verändern sind entweder der letzte oder einzige Befehl, der als RPC vom Server bearbeitet wird. Die sichere Ausführung von Atomaktionen und damit korrekte Transaktionsbearbeitung ist damit durch den VDM Server gegeben.

7.3.4 Beschreibung der Verarbeitungs-Threads

Das implementierte RPC-System setzt, bevor es die zur Bearbeitung von RPCs definierte Callback-Funktion aufruft eine globale Variable mit dem Namen **rpcFile**, die von außerordentlicher Bedeutung ist. Da alle Clientzugriffe ereignisgesteuert ablaufen, und lediglich ein Ereignis zur Zeit bearbeitet wird, ist dies ohne weiteres möglich.

Die Variable **rpcFile** wird auf den TCL-Kanal gesetzt, über welchen der RPC empfangen wurde. Funktionen des Servers, die über RPCs ausgelöst werden, kennen durch Zugriff auf diese Variable ihren Klienten. Beide Partner könnten, was jedoch konzeptionell unerwünscht ist, direkt über dieses socket kommunizieren. **rpcFile** enthält Werte wie `sock4` oder `sock7`.

Für den Benutzeragenten, der die Verarbeitung der Threads durchführt, ist lediglich die Nummer die in **rpcFile** enthalten ist interessant, denn sie erlaubt die eindeutige Identifizierung der Verbindung, und wird zur Einstellung des Kontext verwendet.

Der Client meldet sich beim Benutzeragenten durch Ausführung des RPC-Kommandos

`agent_Connect`

an, und erhält seine Sitzungsnummer als Ergebnis. Gleichzeitigt wird auch der Kontext des Benutzers eingerichtet. Das Abmelden erfolgt über das RPC Kommando

`agent_Disconnect`

Wird die Verbindung nicht durch den Klienten ordnungsgemäß geschlossen, und bricht diese ab, meldet der Server den Klienten selbsttätig mit der Funktion `agent_DisconnectInternal` ab, wobei die Sitzungsnummer explizit übergeben und der Kontext gelöscht wird. Der Selbstätige Abmeldevorgang wird durch eine Umleitung des **close** Kommando eingeleitet[3].

Jeweils dann, wenn über die Funktion

`agent_Call <command;command;...>`

ein Befehl oder eine Befehlsfolge an den Benutzeragenten abgesetzt wird, erfolgt eine Auswertung von **rpcFile** und das Setzen des richtigen Kontext. Anschließend wird das Kommando auf Ebenen 0 des Datenbankmanagers ausgeführt, oder, wenn ein externer Datenprozeß definiert ist, an diesen weitergeleitet und dort bearbeitet. Lediglich die Fehler, die durch im Bereich (*skope*) des Benutzersagenten liegen, werden an den Klient mit dem Hinweis übermittelt, das der Benutzeragent fehlgeschlagen ist. Damit ist ferner eine genaue Eingrenzung für die Fehlersuche gegeben.

7.3.5 HTML als Benutzeroberfläche

Die Benutzeroberfläche des Hypertext Informations- und Hilfesystems wird durch HTML Datenströme definiert. Die tatsächliche optische Wiedergabe ist hiermit jedoch nicht festgelegt, es handelt sich vielmehr um eine ausgabeunabhängige Definition von Text und graphischen Elementen, dessen Wiedergabe vom jeweiligen HTML-Browser abhängig ist.

Während Textbrowser lediglich zur Wiedergabe von Zeichen in der Lage sind, lassen beispielsweise die Browser Navigator und Explorer der namhaften Hersteller Netscape und Microsoft unter Verwendung geeigneter Erweiterungen die Wiedergabe von Filmen und anderen multimedialen Ereignissen zu. HTML wird in dieser Hinsicht fast jeder Browserform gerecht, es läßt sich für Bilder bereits bei der Definition ein alternativer Text festlegen. Hier liegt ein wesentlicher Unterschied von HTML zu anderen Definitionssprachen wie Postscript, dessen Merkmal die ausgabeunabhängig identische Wiedergabe ist. Der entscheidende Vorteil von HTML liegt dagegen in der kompakten Form, welche mit wenigen Textelementen ansprechende Formatierungen zuläßt.

Design der Oberfläche

HTML läßt eine begrenzte und überschaubare Anzahl von Gestaltungsmöglichkeiten zu, hierzu zählen im wesentlichen Text, Schriftgrößen und Bilder, aus welchen durch

[3] Die Realisierung von *atclose* und *atexit* mit TCL ist im Verzeichnis *./tools/g++vdmdbm/dp* zu finden

eine Verknüpfung mit einem URL der Hypertext entsteht. Ferner sind „forms“ genannte Formatanweisungen definiert, welche blätterbare Listen, Wahlfehler und die Eingabe von Text durch den Anwender zulassen.

Ein entscheidener Nachteil der Verwendung von Formatanweisungen wird durch die Vorschrift bestimmt, durch Anwahl eines Knopfes, die Formate zu bestätigen um diese an den Web-Server zu versenden. Diese Restriktion verhindert bei interaktiven Eingaben eine benutzerfreundliche Gestaltung und macht derart gestaltete Oberflächen in der Regel immer dort zur Makulatur, wo mehrfach Daten vom Benutzer eingefordert werden, da es diesen zwingt, jede Eingabe zusätzlich zu bestätigen. Nach ersten Ansätzen der Gestaltung der Benutzerschnittstelle durch Formatanweisungen, wurde dies verworfen und nunmehr ausschließlich mit direkten Hypertext-Referenzen gearbeitet.

Platzoptimierte Tabelle Die Heimatseite (homepage) des Systems stellt eine HTML-Tabelle dar, welche für die geordnete Darstellung der Inhalte der Klassenbibliothek eine optimale Gestaltung ermöglicht. Über diesen Ausgangspunkt des Informations- und Hilfesystems sind zum einen die Inhalte der Bibliothek direkt erreichbar als auch Optionen einstellbar, welche die Steuerung des Systems ermöglichen als auch eine Hilfe. Über diese Hilfe können nicht nur allgemeine Fragen zur Bedienung geklärt werden, sondern es bietet sich auch die Möglichkeit an, direkt in den Hyptertext als auch zu den VDM Anwendungsbeispielen zu verzweigen.

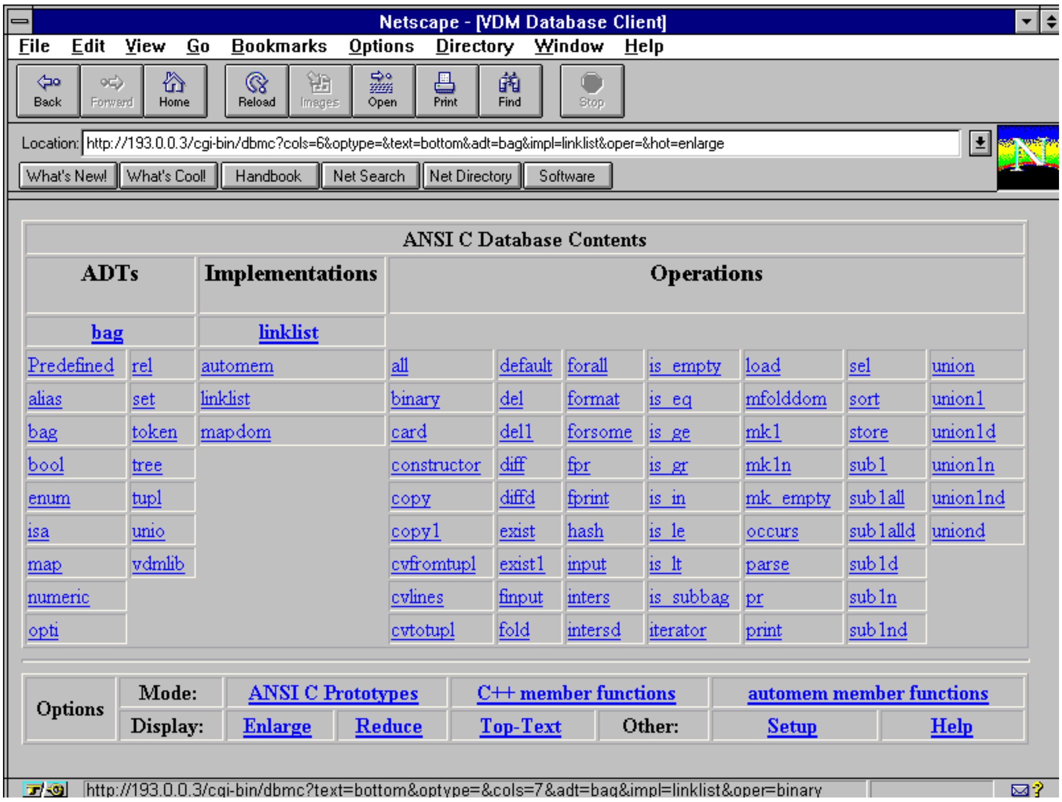


Abbildung 7.1: Auswahltabelle des Informations- und Hilfesystems

Anhand der Tabelleninhalte selektierter Hypertext zu einem ADT der Klassenbibliothek, wird, in Abhängigkeit von der jeweiligen Einstellung, unterhalb oder überhalb der Tabelle angezeigt. Sofern der Hyptertext über der Tabelle angezigt wird, läßt sich ferner festlegen, ob regelmäßig selbsttätig eine Positionierung auf der Tabelle selbst vorgenommen werden soll. Abbild 7.3.5 zeigt die Auswahltabelle, die gleichzeitig die Heimatseite des Informations- und Hilfesystems ist.

Die ADTs der Klassenbibliothek verfügen vielfach über eine große Anzahl von Operatoren. Da der Platz des Browserfensters naturgemäß beschränkt ist, muß dieser bei Darstellung der Auswahltabelle so optimal wie möglich genutzt werden, um ein Blättern des Benutzers nach Möglichkeit zu vermeiden. Für die Einstellung der Anzeigeoptionen ist der Benutzer jedoch selbst zuständig, da dem Server die Größe des Browserfensters nicht bekannt ist.

Der Benutzer kann mit Hilfe der Optionen **Enlarge**, **Reduce** oder dem **Setup** einen Wert für die optimale Darstellung festlegen, der als Tabellenspaltenbreite bezeichnet wird. Als CGI-Parameter **cols** wird dieser Wert beständig von Seite zu Seite beibehalten. Der Algorithmus, der die optimale Tabellenanordnung der dreispaltigen Tabelle vornimmt, ist wie folgt:

Gegeben:

cols: Tabellenanordnung

nAdt: Anzahl der Adts in der Liste

nImpl: Anzahl der Implementationen in der Liste

nOper: Anzahl der Operationen in der Liste

Gesucht:

cols1, cols2, cols3: Optimale Breiten der Listen

Algorithmus:

$fit(l,n) = (l+n-1)/n$

$n1 = fit(nAdt, cols)$

$n2 = fit(nImpl, cols)$

$n3 = fit(nOper, cols)$

$n_{max} = max(n1, n2, n3)$

$cols1 = fit(nAdt, n_{max})$

$cols2 = fit(nImpl, n_{max})$

$cols3 = fit(nOper, n_{max})$

Anhand der ermittelten Werte für *cols1*, *col2* und *cols3* werden Tags und Inhalt der HTML-Tabelle angeordnet.

Dynamische Erzeugung der Oberfläche

Die Benutzerschnittstelle für das Hypertext Informations- und Hilfesystem besteht aus einzelnen Seiten, die immerfort neu erzeugt und auf dem Browser, mit dem der

Anwender arbeitet, angezeigt werden. Jeder Hyperlink löst die Produktion einer neuen Seite aus. Die vollständige Funktionsweise dieses Vorgangs wird durch Abbild 7.3.5 wiedergegeben.

Der VDM Server kann bei der Produktion dieser Seiten als nichtdeterministischer Automat angesehen werden, dessen vielfältigen Zustände jedoch insbesondere von den Inhalten der Klassenbibliothek als auch dem Umfang der Beispiele vorgegeben werden.

Der Kreislauf der mit der Anwahl einer URL beginnt, und mit der neuerstellten HTML-Seite endet, zeigt auch, an welcher Stelle im Konzept der MIME-Decoder als auch das VDM Tester anzusiedeln sind. Dieser Rückkopplungsprozeß anhand der Parameter, die der URL angefügt sind, entspricht der Spezifikation die unter 5.3 vorgenommen wurde.

Die Komplexität zeigt, daß das HTTP-Protokoll, zu dem auch die CGI-Funktionalität zählt, zur Gestaltung von interaktiven Oberflächen trotz seiner großen Verbreitung eher ungeeignet ist. Auch das HTML-Format zeichnet sich mehr durch Schlichtheit als durch Funktionalität aus, auch wenn es aufgrund der außerordentlichen Zuwachsraten im WWW zum verbreitetsten Standard geworden ist.

Die Unzulänglichkeiten werden beständig durch neue standartisierte Tags beseitigt oder durch Firmenstandards behoben, wie beispielsweise *Netscape-Enhanced*, der auch vom erbittersten Konkurrenten übernommen worden ist. Die neuesten Entwicklungen in dieser Richtung sind derzeit Java, Java-Skript als auch neue und verbesserte Formate *forms*, die insbesondere auf den ausschließlichen Einsatz von Web-Browsern für die gesamte Datenverarbeitung von Unternehmen abzielen, und mit Hilfe neuer Wortschöpfungen wie „Intranet“ vermarket werden[4]

7.4 VDM Serverclient

7.4.1 Aufgabe und Funktion

Der VDM Server wird vom Informations- und Hilfesystem ausschließlich über Netzwerkverbindungen bedient. Die Serverkonsole ist lediglich für die Fernwartung, Verwaltung und Ausgabe von Meldungen des Systems vorgesehen. Anwendungen, die vom Server Gebrauch machen bedienen sich ausschließlich des VDM Server Clients **dbmc**, der aufgrund der Unterschiedlichkeit der Anwendungen multifunktional ist.

Zu den Applikationen, die den VDM Clienten nutzen können, zählen Web-Server wie der **httpd**, der diesen als CGI-Interface verwendet, der VDM Bibliotheksbrowser **vdmhelp** und -manager **vdmdbm** als auch Applikationen, die ursprünglich den VDM Datenbankmanager als Filter verwendet haben, und dies nun mit dem VDM Client tun können.

Der VDM Client verfügt über diverse Parameter, die seine Funktionsweise festlegen oder das Mitschreiben der Sitzungsschicht oder von RPCs erlauben. Der Aufruf des Clients, der vollständig in TCL erstellt ist, mit Angabe von `dbmc -h` gibt eine Hilfe

[4] Die Firma Scotch hat bereits beispielsweise das gesamte Informationssystem auf Web-Browser und -Server umgestellt

über die implementierten Funktionen, die auch in den VDM Manual-Seite zu *dbmc(1)* finden sind.

7.4.2 Anwendungsmodi

Kompatibilitätsmodus

Der Kompatibliliätsmodus von **dbmc** läßt lediglich die Angabe der Kommandos zu, die zwischen VDM Bibliotheksbrowser als auch -manager und dem Datenbankmanager vereinbart waren, also `evalCmd`, `evalMessage` und `evalMessageToFile`. Das Kommunikationsprotokoll, daß der Client hierbei unterstützt ist unter 6.3.2 definiert.

Fehler, die bei der Kommunikation auftreten, wie beispielsweise bei Beendigung des Servers, und andere lokale Fehler werden vom Klienten im gleichen Format, ausgegeben, wie Fehlermeldungen, die im Protokoll definiert sind. Diese werden damit von den Applikationen auch erfaßt und ausgegeben. Beispielsweise immer dann, wenn auf dem gewünschten Host unter der angegebenen Portnummer der VDM Server nicht erreichbar ist.

HTML-Modus

Im HTML-Modus spiegelt der VDM Client ein CGI-Gateway wieder, das HTML-Seiten ausgibt. Im *cgi-bin* Verzeichnis des Web-Servers befindet sich in der Regel ein Shell-Skript, welches *dbmc* unter Angabe des Host und der Portadresse im Web-Modus aufruft, wo der VDM Server abläuft.

Fehlermeldungen, die vom Benutzeragenten des Servers stammen, werden unter dem MIME-Typen *text/plain* ausgegeben, da die Fehlermeldung auch gültige HTML-Tags enthalten könnte, die jedoch nicht als solche zu interpretieren sind.

Die Anzeige von Fehlermeldungen direkt am Browser anstelle auf der Konsole des VDM-Servers wurde vorgezogen, da dies nicht nur für die Entwicklung des Systems komfortabler ist, sondern auch eine explizite Zuordnung von Fehlern zur Diagnose erlaubt[5].

Online-Modus

Der Online-Modus emuliert nicht nur das Prompt, das ursprünglich durch den Datenbankmanager ausgegeben wurde, sondern den ganzen Datenbankmanager. Es ist damit die Eingabe von einzelnen Kommandos oder auch ganzen TCL-Prozeduren möglich. Für den Anwender macht es (fast) keinen Unterschied, ob er am Datenbankmanager direkt oder mittels des VDM Klienten arbeitet. Da die Abwicklung der Kommandos jedoch über RPCs erfolgt, werden lediglich Rückgabewerte von TCL-Funktionen dargestellt. Die Standardausgabe und Fehlerausgabe bleibt verborgen. Damit führt die Ausführung von

[5]Denn es steht den Anwendern jederzeit frei, sich über Fehler zu beschweren, oder diese per Mail an den Systemverwalter zu senden

```
exec ls
```

zur keiner Ausgabe, während das TCL-Kommando

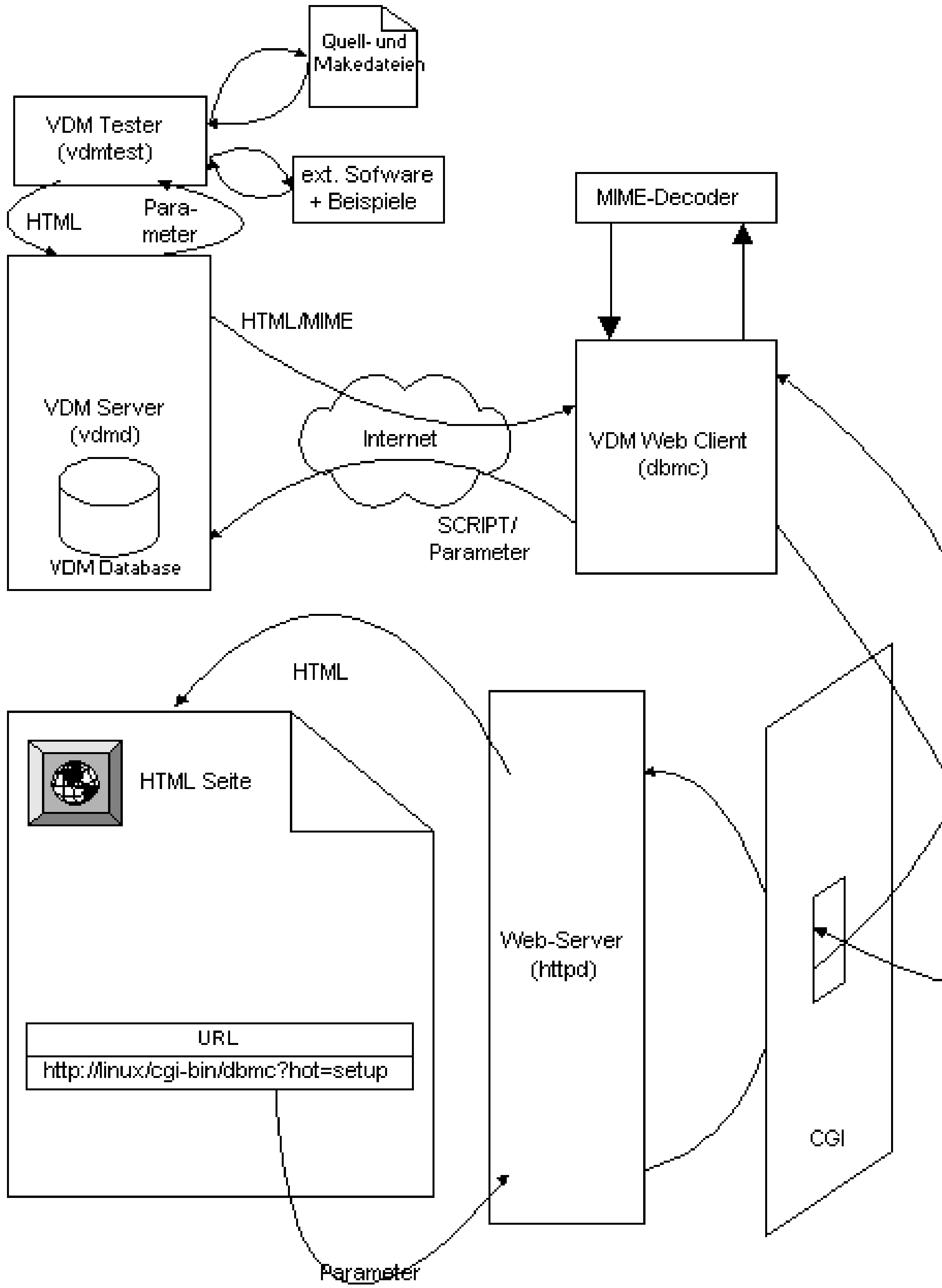
```
set x test
```

das Ergebnis anzeigt. Die Standardausgabe und Fehlerausgabe des VDM Servers läßt sich nur mit Hilfe der Fernwartung kontrollieren. Im Gegensatz zur Fernwartung, die nur einem Manager zur Zeit zuläßt, können jedoch mit **dbmc** beliebig viele Benutzer am Server arbeiten.

Der VDM Client kennt zwei Betriebsarten für den Online-Modus

1. Kommunikation mit dem VDM Server
2. Direkte Kommunikation mit einem Benutzeragenten

Während die erste Betriebsmodus direkt mit dem VDM Server verbindet, erfolgt über den zweiten Modus ein Verbinden mit einem eigens für diesen Klienten geschaffenen Benutzeragenten. Wenn ein externer Datenbankprozeß konfiguriert ist, stellt dies die einzigste Möglichkeit dar, auf diesen zuzugreifen. Der VDM Server leitet alle Anfragen weiter, und es wird ausschließlich mit diesem Kindprozeß kommuniziert.


Quell- und
Makedateien
VDM Tester
(vdmtest)
ext. Sofware
+ Beispiele
HTML
Para-
meter
MIME-Decoder
HTML/MIME
VDM Server
(vdmd)
VDM Database
Internet
VDM Web Client
(dbmc)
SCRIPT/
Parameter
HTML
HTML Seite
URL
http://linux/cgi-bin/dbmc?hot=setup
Web-Server
(httpd)
CGI
Parameter

Kapitel 8

Test des Systems

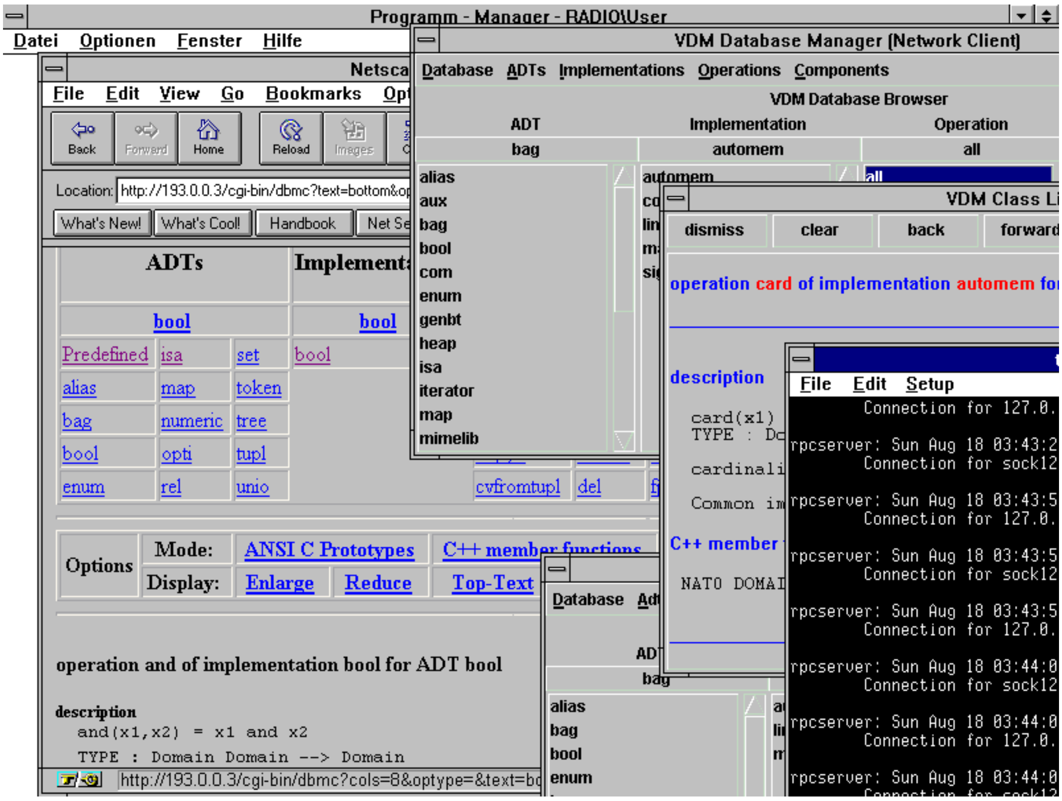


Abbildung 8.1: VDM Applikationen unter Windows NT

Der Server des Hypertextsystems wurde unter Linux und SUN-OS getestet und arbeitet in beiden Fällen einwandfrei. Da der Diplomand nicht über Supervisorechte an der Sparc20 der Fachhochschule Wedel verfügt, ist ungewiß, ob der Server auch dort als Daemon ausführbar ist. Auch die Installation des Paketes der Klassenbibliothek ist auf beiden Systemen nach einigen Anpassungen erfolgreich geglückt.

Zur Übersetzung des Beispiels wurde bei der Entwicklung der GNU Compiler g++ in der Version 2.6.3 eingesetzt. Als Phänomen trat bei dem Compilerlauf für die Datei *vdmdbm.cc* in der Methode VDMDB::sel_comp mit gleichem Compiler in der Version 2.7.2 folgender Fehler auf:

```
813: Internal Compiler Error
bug report to: bug-g++@prop.ai.mit.edu
```

Da der Compiler nur zeitweilig abbrach, wurde Argumente zusätzlich gekammert, was nicht gleichbedeutend mit der Behebung dieses Fehlers sein muß, auch wenn er anschließend nicht wieder auftrat.

Der Bibliotheksbrowser und -manager sind sowohl unter Windows NT als auch Windows 95 als Klienten uneingeschränkt funktionstüchtig. Sie funktionieren ebenso wie der Fernzugriff mit einem HTML-Browser über eine Fernverbindung. An HTML-Browsern wurden Lynx, Arena und Netscape mit dem Informations- und Hilfesystem getestet.

Die Ausführung des Beispieltesters **vdmtest** direkt als CGI-Gateway, bricht den Aufruf von Beispielen und Kommandos ab, da der Web-Server **httpd** als Daemon das signal **SIGCLD** (bzw.**SIGCHLD**) ignoriert, anstelle es auf der Standardbehandlung zu belassen. Die TCL-Shell kann daher nicht die ordnungsgemäße Beendigung von Kindprozessen behandeln und bricht mit einer Fehlermeldung ab. **vdmtest** muß daher zur Nutzung des vollständigen Funktionsumfang auf dem üblichen Weg über den VDM Server aufgerufen werden.

Eine Überprüfung der gesamten Funktionalität des Hyptertext-Systems, was den Zugriff auf einzelne ADTs, die VDM Anwendungsbeispiele, und die Darstellungsoptionen einschließt, hat lediglich Fehler in einigen wenigen Dokumentations-Komponenten der Datenbank, der durch den Precompilers verursacht wird, bei der Volltextsuche über alle Dokumente aufgedeckt. Der VDM Server ein Abbruch des Preprozessors auf der Console an, und damit auch diese Fehler.

Kapitel 9

Kritische Betrachtung des Systems

9.1 Vollständigkeit

Das Ergebnis der Arbeit entspricht in seinem Funktionsumfang den Planungen. Es konnten über die geplanten Minimalanforderungen hinaus noch einige zusätzliche Funktionen integriert werden, die im wesentlichen beim Analyseprozeß und dem Prototyping entstanden. Dies betrifft insbesondere den Bibliotheksbrowser und -manager, die ursprünglich nur zu Übungszwecken in ein Client-Server Modell verwandelt wurden, dann aber perfektioniert, portiert und in das Gesamtkonzept übernommen wurden.

Alle Elemente, die in vergleichbaren Hypertextsystemen, wie beispielweise Windows-Hilfen oder ElectroText, vorkommen sind ebenfalls nach den Möglichkeiten die das HTML-Format bietet, im Informations- und Hilfesystem realisiert. Neben der Verknüpfung mittels Hyperlinks, können auch Bilder zur Anwahl von Text oder anderen Bildern zur Verdeutlichung angezeigt werden, oder Bilder selbst als Hyperlink verwendet werden. Während andere Systeme auch die differenzierte Anwahl von Bildteilen zulassen, wurde dies nicht nachvollzogen, obwohl dies mittels Image-Maps auch unter Verwendung von HTML möglich gewesen wäre. Lediglich in diesem Sinne liegt ein Unvollständigkeit im Vergleich mit anderen Systemen vor.

9.2 Zuverlässigkeit

Die Erweiterungen des Datenbankmanager sind formal definiert und unter Verwendung der Klassenbibliothek implementiert. Bei korrekten Datenbankinhalten ist das Auftreten eines Fehlers nicht zu erwarten. Der VDM Server hat seit der Beseitigung des letzten (erkannten) Programmierfehlers, für eine Dauer von ca. 2 Monaten eine außerordentliche Stabilität bewiesen, und gezeigt, daß er für dauerhaften Einsatz geeignet ist.

Unzuverlässige Reaktionen sind jedoch dann zu erwarten, wenn der Festplattenspeicher verknappt, und temporäre Dateien nicht mehr angelegt werden können. In diesem Fall kann die Datenbank des Servers verloren gehen.

Da lediglich eine kurze Verarbeitungsschleife des Servers existiert, die direkt TCL-Code ausführt und alle weiteren Aktiviäten ereignisgesteuert ablaufen, führen Fehler im TCL-Code lediglich zu einem Abbruch der Bearbeitung dieses Ereignisses, gefährden die Zuverlässigkeit des Servers im allgemeinen jedoch nicht.

9.3 Effizienz

Effizienz war bei der Entwicklung des Systems kein Kriterium. Wenn keinerlei Ereignisse verarbeitet werden müssen, wird kaum Rechenzeit in Anspruch genommen. Anders verhält es sich bei der Produktion von HTML-Seiten. Insbesondere bei der Umsetzung in den Latin-1 Zeichencode und der Bearbeitung der Seite mit regulären Ausdrücken wird kurzzeitig erhebliche Systemleistung in Anspruch genommen, die entweder ein entsprechend leistungsstarken Host, oder Geduld des Anwenders erfordern.

9.4 Benutzerfreundlichkeit

Die Benutzerfreundlichkeit des Systems hängt in erster Linie von der Gestaltung des Hypertextes ab. Insbesondere bei der Darstellung von Informationen wurde versucht, eine optimale Präsentation zu bieten. Ein Problem stellt jedoch die Größe der angezeigten Informationen dar, die oftmals das Blättern des Benutzers erfordern, wenn dieser über kein großflächiges Sichtgerät verfügt. Es besteht lediglich die Möglichkeit, daß der Benutzer auswählt, an welcher Stelle die Ausgabe von Text erfolgen soll, um die Bedienung zu vereinfachen. Die Hilfen zur Bedienung des System wurden sehr knapp gehalten, und können zusätzliches Informationsmaterial erforderlich machen.

9.5 Sicherheit

Der Sicherheit wurde zwar Rechnung getragen, indem es möglich ist, jeden RPC-Befehl vor seiner Ausführung auf dem Server zu überprüfen. Die Integration dieser Prüfung in den VDM-Server ist erfolgt, und auch eine Liste der gültigen Kommandos definiert. Da es sich bei Test und Entwicklung jedoch als lästig erwiesen hat, mitunter an die selbstgesetzten Grenzen zu stoßen, wurde dieser Ansatz nicht weiter verfolgt und auskommentiert, so daß jede Art von RPC-Kommandos an den Server abgesetzt werden kann[1].

Eine weitere Sicherheitslücke stellt die Implementierung von Kommandos für die VDM Beispiele dar, da diese nicht auf einen bestimmten Satz begrenzt werden. Kommandos können zwar nur im Verzeichnis des angewählten Beispiels ausgeführt werden, der versierte Eindringling könnte jedoch die Bedeutung der Parameter an das CGI-Gateway verstehen und beispielsweise durch Eingabe von

```
rm /etc/*
```

[1] Die Kommandos befinden sich in der Datei *security.tcl* und wurden in *server.tcl* überprüft

dennoch Schaden in einem anderen Verzeichnis anrichten. Abhilfe würde hier nur die Einschränkung der verwendbaren Kommandos schaffen.

9.6 Portabilität

Die Klassenbibliothek selbst ist nach wie vor für UNIX-Systeme konzipiert und auf diesen portierbar. Der VDM Client kann jedoch auf allen Plattformen eingesetzt werden, für die auch eine TCL-Shell verfügbar ist. Auch für den Bibliotheksbrowser und -manager ist dies unter Verwendung des VDM Clients möglich. Portiert wurden diese drei Bestandteile der Klassenbibliothek auf die 32-Bit Windows Derivate NT und Windows 95.

Anhang A

Online-Dokumentation der Diplomarbeit

Eine Zielstellung aus 1.2.1 ist die Verfügbarkeit der Diplomarbeit selbst im HTML-Format, so daß diese auch Online zu erreichen ist und mit einem geeigneten Browser über das Internet gelesen werden kann. Die folgenden Darstellungen befassen sich mit den Texterstellungs- als auch Konvertierungsarbeiten. Am geeignetsten für ein Zwischenformat zwischen dem Ursprungstext und HTML erschien dem Autor zunächst das Rich Text Format (RTF)[1], da alle ihm bekannten Textverarbeitungsprogramme dieses Format unterstützen, und dieses zudem auch Bilder beinhalten kann.

Dem Diplomanden waren die Schriftsatzsysteme TEX und LATEX zwar bekannt und wurden auch zur Ausgabe fremder Texte verwendet, da er jedoch selbst nie eingiebig mit diesem System gearbeitet hatte, war es ihm aufgrund der (oft unnötigen) Sonderbehandlungen als auch Vielzahl und Kryptik der Steuerkommandos suspekt. Es wird vorweggenommen, daß die Arbeit letztendlich doch mittels LATEX formatiert wurde, wie aus dem Schriftbild auch unschwer zu erkennen ist. Einer der Gründe hierfür ist die hervorragende Qualität des Konverters von Nikos Dragos.

A.1 Aktuelle Textprozessorprojekte

Zunächst soll ein Überblick über den derzeitigen Stand akueller HTML-Textprozessorprojekte vermittelt und beurteilt werden. Es wurden alle Konverter, die derzeit über die URL des W3 Consortiums *www.w3.org* erreichbar waren und im entferntesten zur Lösung des Problems geeignet erschienen, untersucht. Aufgrund der Vielzahl werden diese in ihrer Gesamtheit nicht aufgeführt. [DC95] als auch [TS95] geben einen Überblick über die wichtigsten der Konverter. Mittlerweile ist für die wohl verbreitetste Textverarbeitung WinWord ein Internet-Assistent verfügbar, der auch die komfortable Gestaltung von HTML-Seiten zuläßt.

Das Resümee der Recherche war, daß viele der über das Internet beziehbaren unkommerziellen Textprozessoren, denn nur solche wurden untersucht, scheinbar lediglich zur Lösung individueller Konvertierungsprobleme der Ersteller geeingnet waren.

[1]Er hatte in der Zwischenzeit den Hinweis des Referenten auf den LATEX in HTML Übersetzter von Nikos Dragos vergessen, und den Übersetzer nebst gebundenem Handbuch verlegt

Die Bearbeitung größerer Texte führte zu Abstürzen, unsinnigen Konvertierungen, Abbrüchen und weiteren Fehlfunktionen. Insbesondere die Bearbeitung von Listen, Fußnoten, Tabellen und Bildern bereiteten vielfach Probleme aufgrund der Unterschiedlichkeit der Formate. Viele der Konverter waren mit Perl erstellt, was eine plattformunabhängigkeit sichert, einige in C oder C++ oder selbsterstellten Interpretern. Zahlreiche waren auf ein bestimmtes Betriebssystem, wie DOS, Windows, Windows NT oder UNIX festgelegt und nicht portabel.

Auch die Versuche zahlreicher Gruppen im Internet, SGML zu einem einheitlichen Standard für Dokumente aller Art zu machen, waren nicht so weit fortgeschritten, wie es sich der Autor erhoffte, und ist dem Beispiel gefolgt, sich einen eigenen Textprozessor zu erstellen, der mit WinWord erstellte RTF-Dateien weitgehend in TEX umsetzt, aber auch die direkte Implementierung von komplizierteren TEX -Kommandos in der Quelle zuläßt. Einige Ansätze von Konvertern zur Problemlösung sollen in Kürze dargestellt werden.

Eine der einfacheren Aufgaben für Konverter schien die Umsetzung von HTML, RTF oder ähnlichen Formaten in TEX zu sein. Fast alle Lösungen wurden durch eine Definition von Stilvorlagen für TEX , und Umsetzen der Stilelemente der Quelldatei in TEX -Makros programmiert. Bessere Konverter führten eine erweiterbare Positivliste der Elemente, die umgesetzt werden können.

Aufgrund des geringen Umfanges der Stilmittel von HTML oder Manual-Pages, stellt eine Konvertierung dieser Formate in der Regel kein Problem dar. Sie sind jedoch jedoch zur Texterstellung einer wissenschaftlicher Arbeit aufgrund fehlender Stilmittel und Seitenorientierung von HTML ungeeignet. Das fehlen mathematischer Symbole als auch unzureichende Gestaltungsmöglichkeiten hätte zur Erstellung in HTML zur Erfüllung der Richtlinien für Diplomarbeiten der Fachhochschule Wedel ohnehin nicht ausgereicht.

Eine generelle Feststellung ist, daß die Umsetzung von Formaten schlichten Stilumfangs in umfangreichere leichter zu realisieren ist, als die umgekehrte Konvertierung, bei der viele Konverter aufgrund ihrer Unvollständigkeit scheitern. Umso bemerkenswerter ist der LaTeX2HTML-Translator von Nikos Drakos[2], der in der Version 96.1 verwendet wurde. Die Vorzüge und Eigenschaften dieses Konverters sind:

- Vollständige Übersetzung von LATEX Dokumenten in HTML
- Partielle Ersetzung von Elementen, die nicht direkt übersetzt werden können, durch Bilder, die durch Ausführung von LATEX für das betreffende Fragment erzeugt werden. Bilder werden (aus erstellten EPS-Dateien) im GIF Format erzeugt und sind transparent. Das heißt, sie fügen sich nahtlos in das Schriftbild bei der Wiedergabe durch den Browser ein.
- Umsetzung von Inhaltsverzeichnis, Fußnoten, Referenzen, Verzeichnissen, der Bibliographie und des Indexes in Hyperlinks

Der Übersetzter ist ferner in Perl erstellt und damit (zumindest theoretisch) auch auf allen Plattformen einsetzbar, für die auch LATEXverwendet werden kann, und arbeitet

[2]Nikos Drakos, Computer Based Learning Unix, University of Leeds

mit einem erstellten Cache im Zielverzeichnis um veränderten Text beschleunigt zu übersetzen. Die ohnehin außerordentlich hohe Performance wird damit noch gesteigert.

A.2 Vorgehensweise der Texterstellung

Der Text der Diplomarbeit selbst wurde unter WinWord 6.0 erstellt, und im RTF-Format abgelegt. Zur Konvertierung dieses Formates in gültiges TEX -Eingabeformat wurde ein einfaches Konverterprogramm erstellt, das LL(1) ist und die Umsetzung von Umlauten, Sonderzeichen, der Gliederungen und Schriftauszeichnungen vornimmt. Alle weiteren TEX Kommandos wurden direkt in den Text eingebettet und durch entsprechende Sonderbefehle des Konverters unverändert übertragen.

Abbildungen wurden mit Visio Technical 4.0 gestaltet, mit Photoshop 3.0 bearbeitet und im EPS-Format abgelegt. Die Erstellung des Textes erfolgte durch EmTEX 3.0 von Eberhard Mattes. Abbildungen wurden mit Hilfe der TEX -Umgebung des Postscript-Treibers dvips in der Version 5.58 eingebunden und mit gleichem Treiber zum Druck für einen Tintenstrahldrucker unter Verwendung von Ghostscript vorbereitet.

Bei der Texterstellung haben sich unter Beachtung der Eigenschaften des gewählten Konverters **latex2html** von Nikos Dragos folgende Ziele und obligatorischen Einschränkungen ergeben.

Die Geschwindigkeit der Darstellung eine HTML-Seite durch den Web-Browser hängt neben der Größe der Seite selbst im wesentlichen von den Dateigrößen der verwendeten Bilder ab. Auch wenn der Konverter von Nikos Dragos die durchweg als positiv anzusehende Strategie verfolgt, LATEX -Elemente, die nicht direkte in HTML umgesetzt werden können, als Bild in die Seite einzufügen, sollte dieser Ersatz nach Möglichkeit reduziert werden, da der erstmalige Zugriff auf HTML-Seiten über das WAN bei einer Vielzahl von Bildern zur Geduldsprobe werden kann. Nach dem Erstzugriff werden die Bilder in der Regel durch den Browser lokal im Cache zwischengespeichert, womit sich ein Neuübertragung für eine gewisse Zeit erübrigt.

Die LATEX -Stilvorlage der Texte muß für einseitige Formatierung ausgelegt sein, beispielsweise durch die Verwendung von *article*. Doppelseitige Stilvorlagen wie *book* werden duch den Konverter nicht verarbeitet. Nachteilig ist ferner, daß der Host, auf dem Web-Server ausgeführt wird, vor der Übersetzung in der Konfigurationsdatei des Umsetzers festgelegt werden muß, und die URLs der Zieldokumente auf diesen verweisen, so daß für die Übertragung auf einen anderen Host eine Umsetzung der URLs erforderlich ist.

Das LaTex2HTML Paket von Nikos Dragos, enthält eine detailierte in TEX erstellte Dokumentation, die ein Benutzerhandbuch, bekannte Probleme als auch „Troubleshooting“ enthält. Auf eine Wiedergabe des Inhaltes wird daher verzichtet, und auf das Originaldokument verwiesen.

Bei der Texterstellung wurde auf Umdefinitionen von Umgebungen und Befehlen mit *renewcommand* oder *renewenvironment* verzichtet, um eine äquivalente Umsetzung durch den Übersetzer in HTML nicht zu behindern. Der Text benutzt ausschließlich

Befehle von LaTeX2.09, der selbsterstellte RTF in LaTeX Prozessor produziert Ausgaben, die kein deutsche Stilvorlage voraussetzen, auch wenn dies von **latex2html** unterstützt wird.

Anhang B

Integration von VDM Anwendungsbeispielen

Die Vorgehensweisen für die Einbettung von Beispielen zu der VDM Klassenbibliothek wurden bereits behandelt. Für die Erstellung neuer Beispiele sind detailierte Kenntnisse der Abläufe des Systems nicht erforderlich. Vielmehr wird eine Anleitung benötigt, wie neue Beispiele mit VDM erstellt werden, und auf welche Weise die Dokumentation der Beispiele erfolgen muß, um diese in das Informations- und Hilfesystem zu integrieren.

Die Beispiele zur Klassenbibliothek befinden sich ausgehend vom Verzeichnis **./Test**[1] in einer Verzeichnisstruktur. Sie werden über Textdateien verkettet, die eine übliche HTML-Seite beschreiben und zusätzlich über einen speziellen Tag verfügen, der im Satz der definierten HTML-Tags nicht vorkommt, sondern vom Informations- und Hilfesystem umgesetzt wird.

Dieser Tag ermöglicht nicht nur eine Verknüpfung der Textdateien, die der Baumstruktur der Beispiele folgen, auf jede erdenkliche Weise, sondern läßt auch die Anzeige von Quellcode, Definitions- und Makedateien sowie die Ausführung von Systemkommandos und Programmen zu.

B.1 Interaktiver Test der Beispiele über HTML-Browser

Grundsätzlich sind drei Möglichkeiten vorhanden, über das Hypertext-System auf die VDM Beispiele zuzugreifen. Zum einen ist über Auswahl des Hyperlinks „Hilfe“ aus der Home-Page des Systems ein Verweis auf die Datei **README.html** die im Hauptverzeichnis der Beispiele *./Test* abgelegt ist, gegeben. Diese Datei zeigt eine vollständige Liste der Beispiele an, und erlaubt den Zugriff auf diese. Eine zweite Alternative ist der Aufruf des VDM Klients vom Browser unter Angabe des konkreten Beispiels, das ausgeführt werden soll. Die dritte Möglichkeit ist der Zugriff auf ein Beispiel direkt aus dem Hypertext des Informations- und Hilfesystems heraus, um beispielsweise ein Thema durch dieses Anwendungsbeispiel zu verdeutlichen.

[1] Relativ zum Installationsverzeichnis der Klassenbibliothek

In Abschnitt 5.3 ist definiert, welche Parameter der URL des VDM Web-Klienten hinzugefügt werden müssen, um direkt zu einem Beispiel zu verzweigen. Diese können in der Regel auch bei der Ausführung von Beispielen über das Informations- und Hilfesystem an der URL-Anzeige des Browsers abgelesen werden. Abschnitt 6.1.1 zeigt, wie Links zu Beispielen über Hyptertext-Makroerweiterungen in die Dokumentation der Klassenbibliothek integriert werden.

B.2 Vorhandene Beispiele

Der Autor dieser Arbeit hat bereits einige der zahlreichen Demonstrationsprogramme kommentiert, die als Muster für weitere dienen können. Diese sind

- Die Ursprungsdatei README.html im Hauptverzeichnis der Beispiele,
- das „Astar“ Beispiel,
- das „automem“ Beispiel,
- und ein Skelett, das als Vorlage für neue Beispiele verwendet werden kann

Die obigen Beispiele beinhalten gemeinsam alle Gestaltungsmöglichkeiten. „Astar“ zeichnet sich durch seine Vollständigkeit aus, „automem“ und README.html zeigen, wie auf Beispiele in über- oder untergeordneten Verzeichnissen verwiesen werden kann.

B.3 Beispielskelett

Das Beispielskelett ist im Verzeichnis **./Test/skel** hinterlegt, und beinhaltet eine DSL-Datei, ein Makefile, ein zugehöriges C++ Programm als auch die README.html Datei zur Beschreibung des Skeletts. Die DSL des Beispiels trägt den Namen **skel.vdm** und lautet:

```
DOMAINS
Num is-a Nat0

IMPLEMENTATIONS
Num = + fpr

FUNCTIONS

plus: Num => Num
```

Die Funktion des Beispiels als auch das Hauptprogramm ist implementiert in der Datei **skel.cc**, und führt die Addition von 1 und 1 aus. Übersetzt wird das Beispiel mit einem mustergültigen **Makefile**, in der alle Definitionen und Ziele hinterlegt sind, zur Integration in das Gesamtkonzept definiert sein sollen. Die Datei README.html im Verzeichnis des Skeletts läßt nicht nur die Übersetzung zu, sondern erläutert auch alle Tags, die zur Dokumentierung der VDM-Beispiele zur Verfügung stehen.

B.4 Erweiterung der Beispiele

Bei der Erweiterung der Beispiele sollten zunächst einige Regeln beachtet werden:

1. Das ausführbare Programm nennt sich immer **TEST**
2. Die Makedatei verfügt über die Ziele **clean** und **CLEAN**. Das kleingeschriebene Ziel löscht alle mit der Makedatei erzeugten Dateien, und läßt nur das ausführbare Programm **TEST** bestehen. Das Ziel in Großbuchstaben löscht zusätzlich auch das Testprogramm
3. Die HTML-Datei zur Beschreibung der Beispiele trägt den Namen README.html. Das System erfordert diesen Namen zwar nicht zwingend, es sollte jedoch im Zuge der Vereinheitlichung hierfür Sorge getragen werden. Weitere HTML-Seiten in weiteren Seiten können zusätzlich zu README.html angelegt werden, die Home-Page dieses Beispiels ist jedoch immer diese Datei
4. Für jedes neue Beispiel wird ein Verweiseintrag in der README.html Datei im Hauptverzeichnis der Beispiele eingetragen.

Die speziellen Tags des Informations- und Hilfesystems, welche in die Beispiele eingesetzt werden können lauten:

```
<A EXAMPLE RUN="@Command Parameter">Aufruf eines Kommandos</A>
<A EXAMPLE RUN="Programm Parameter">Start eines Programms</A>
<A EXAMPLE VIEW="Dateiname">Anzeigen einer Datei</A>
<A EXAMPLE REF="HTML-Datei">Verweis zu einer HTML Seite</A>
```

Für alle Tags kann ein absoluter oder relativer Pfad zu den Dateien zusätzlich zum Namen angegeben werden. Aus Sicherheitsgründen bezieht sich jeder relative Pfad auf das aktuelle und jeder absolute Pfad auf das Verzeichnis **./Test** der Klassenbibliotheksinstallation, das für alle Pfadangaben als Hauptverzeichnis gilt.

Auffällig ist, daß zwei Alternativen für die Ausführung existieren. Zum einen eingeleitet durch das „@“-Symbol, zum andern ohne dieses. Der Grund hierfür liegt darin, daß zum einen Systemprogramme wie *ls*, *make* oder *sh* benötigt werden, und zum anderen Programme direkt ausgeführt werden sollen.

Da der VDM Server sein Verzeichnis nicht tatsächlich wechselt, sondern anhand der Startverzeichnisses kombiniert, werden Dateinamen grunsätzlich in absolute Pfade umgewandelt. Solange Beispiele direkt ausgeführt werden, stellt dies kein Problem dar. Werden jedoch Programme wie *make* oder *ls* benötigt, so darf der Zugriff auf diese Programme mit absolutem Pfad aufgrund der Sicherheitsbedingungen nicht möglich sein, weil sie sich in der Regel unter **/usr/bin** befinden. Bei Kommandos, die mit „@“ eingeleitet werden, wird daher folgende Strategie verfolgt:

- Das Programm für das Kommando muß sich im Pfad des VDM Testers **vdmtest** befinden[2].

[2]Dies wird über die Konfigurationsdatei des Servers festgelegt

- Es erfolgt ein expliziter Wechsel in das Verzeichnis des Beispiels
- Das Kommando wird ohne Pfad in diesem Verzeichnis mit den angegebenen Paramtern ausgeführt[3].

Einige der Testprogramme mit dem Namen **TEST** sind lediglich eine Shellkript, das das eigentliche Programm aufruft. Solche Beispiele müssen wie folgt in die Datei README.html eingebunden werden:

```
<A EXAMPLE RUN="@sh TEST">Aufruf des Beispiels</A>
```

Die Tags *EXAMPLE REF* und *EXAMPLE VIEW* unterscheiden sich insofern, als daß ersterer Tag eine HTML-Datei erwartet, die auch weitere *EXAMPLE*-Tags enthalten kann, während letzterer die Datei lediglich als Text anzeigt. Wenn eine VDM Anwendungsbeispiel selbst mehrere Unterverzeichnisse enthält, so sind in diesen weitere **README.html** Dateien anzulegen.

[3] Das dies ohne besondere Maßnahmen keine ausreichende Sicherheit bietet, wird in der Kritik zum System erläutert

Anhang C

Referenzteil

Der Referenzteil beleuchtet die praktische Seite und enthält alle Informationen, die zur erfolgreichen Installation und Konfigurierung des Hypertext Informations- und Hilfesystems notwendig sind. Ebenso werden hier die systemspezifischen Themen behandelt, die weniger von theoretischer Natur sind behandelt. Aufgabe des Referenzteils soll es sein, ein mühsames „herumprobieren“ des Anwenders und Weiterentwicklers durch kurze aber exakte Beschreibungen zu vermeiden, und technische Zusammenhänge zu erhellen.

C.1 Installation

Die Distribution der gesamten VDM Klassenbibliothek in seiner Entwicklerinstallation erfolgt in der Regel durch eine mit GNU-zip (*gzip*) gepackten und durch den Tape-Archiver *tar* archivierten Datei mit dem Namen **flatport.tgz**. Nach Entpacken der Bibliothek gemäß der beiliegenden Informationsdatei, liegt eine Verzeichnisstruktur vor, welche die Erzeugung der Klassenbibliothek einschließlich seiner Tools erlaubt.

Zur Installation wird folgende Software auf dem System benötigt:

1. Tape Archiver tar sowie gzip
2. ANSI C Compiler[1]
3. ANSI C++ Compiler[2]
4. Lex und Yacc (oder kompatible wie flex, bison und andere)
5. TCL Entwicklungspaket
6. TK Entwicklungspaket

Der Aufruf von **make** im Installationsverzeichnis ohne weitere Angaben gibt eine Hilfe über die Installationsoptionen aus. Die Installation unter SUN-OS erfordert die Eingabe von `make SUNFLATBOOT`, unter Linux-und anderen Systemen wird `make FLATBOOT`

[1] Verwendet wird für die Klassenbibliothek in der Regel der GNU gcc

[2] Auch hier wird der GNU Compiler g++ präferiert

erwartet[3]. Alle weiteren Informationen finden sich in der Informationsdatei README.

Die Tools der Klassenbibliothek basieren mittlerweile auf sich selbst, was bedeutet, daß auch der VDM Domaincompiler aus Elementen der VDM-Klassenbibliothek besteht, und daher einen existierenden Domaincompiler voraussetzt. Das Resultat des Domaincompilers ist C bzw. C++ Code, der als solches portabel ist. Die Erzeugung der Datei **flatport.tgz**, welche im übrigen auch durch das *Makefile* im Installationsverzeichnis erfolgen kann, führt vor der Generierung des Archivs die Umsetzung aller VDM-Bestandteile in portablen Hochsprachencode durch, der nach der Portierung auf das Zielsystem übersetzt und gelinkt wird, so daß hiernach auch der Domaincompiler verfügbar ist.

Das Informations- und Hilfesystem arbeitet mit einer Client-Server Struktur und setzt die Verfügbarkeit von TCP-Kommunikation in TCL voraus. Es wird daher insbesondere benötigt:

A. TCL in der Minimalversion 7.5

B. TK in der Miminalversion 4.1

Beta-Version von TCL 7.5 und TK 4.1 werden nicht unterstützt. Für den Fall, daß lediglich ältere Versionen auf dem Zielsystem verfügbar sind, wurde in sofern Vorsorge getragen, daß die sich Klassenbibliothek nach wie vor dennoch vollständig und funktionsfähig erzeugen läßt, jedoch ohne das lauffähige Informations- und Hilfesystem.

Dem Sonderfall, daß TCL in einer älteren Version auf dem System installiert ist[4], wird Rechnung getragen durch eine besondere Art der Nachonfiguration, mit einer separaten, zusätzlichen Installation des benötigen TCL Paketes. Die Vorgehensweise hierfür ist wie folgt:

1. Installieren der Klassenbibliothek mittels *FLATBOOT* bzw. *SUNFLATBOOT*
2. Setzen der Umgebungsvariable *VDMROOT* auf das Installationsverzeichnis und Hinzufügen von $VDMROOT:/bin zu der Umgebungsvariable *PATH*. Dies ist wichtig für die Ausführung von **make prefixtcl** an späterer Stelle, da der Domaincompiler für die Neuerzeugung des Datenbankmanagers verwendet wird.
3. Installation von TCL 7.5 oder einer jüngeren Version in einem wahlfreien temporären Verzeichnis.
4. Ausführen von **configure** im temporären Verzeichnis
5. Editieren der Datei *Makefile*
6. Verändern der Variable *prefixdir* im Makefile des TCL-Quellcodes, die per Vordefinition auf */usr/local* gesetzt ist, auf ein anderes Zielverzeichnis, z.B. das Heimatverzeichnis des Benutzers, jedoch unter Beibehaltung der Endung */usr/local*. Gültig wäre zum Beispiel */home/stud/wi/wi8980/usr/local*

[3] Hinweis: Es wird damit das „flache Modell“ und nicht das Verzeichnismodell der Datenbank erzeugt

[4] Auf der sparc20 der Fachhochschule war dies zur Zeit der Entwicklung des Informations- und Hilfesystems der Fall

7. Erzeugen von TCL durch Aufruf von **make**

8. Installieren von TCL durch Aufruf von **make INSTALL**

9. Wechsel in das Verzeichnis *./tools/g++vdmdbm/* ausgehend vom Installationsverzeichnis der Klassenbibliothek

10. Eingabe von **make prefixtcl**

11. Nachdem die Aufforderung ergeht, Eingabe des gleichen Pfades, in welchen *prefixdir* des Makefiles der TCL-Installation geändert wurde

12. Anschließend Eingabe der TCL-Versionsnummer, der gesonderten Installation

13. Sowie Eingabe von FLAT bzw. SUNFLAT als Ziel für die Neuerzeugung des Datenbankmanagers, nach Verlangen des Systems

14. Eingabe von **make INSTALL**

Bei einer üblichen Installation einer geeigneten TCL Version im System erübrigt sich die vorstehende Installationsanweisung.

Die Makedatei im Installationsverzeichnis der VDM Klassenbibliothek wurde durch den Autor dieser Diplomarbeit um folgende vier Optionen erweitert:

1. **make clean**

   Beseitigen allen Objekt- und Binärcodes sowie aller weiteren erzeugten Dateien. Die Neuerzeugung des Codes kann anschließend mit *make FLATBOOT* bzw. *make SUNFLATBOOT* vorgenommen werden. Eine vollständige Neuübersetzung durch Ausführung von **make FLAT** ist nicht mehr möglich, da auch der Domaincompiler gelöscht wird.

2. **make cleanport**

   Säuberung nach vorstehend beschriebener Methode, sowie Erzeugen des Portierungsarchives **flatport.tgz** mit gesetzten Lese- und Schreibrechten als auch ohne jeglichen Objektcode.

3. **make unixclnt.tgz**

   Erzeugen eines Archives, welches lediglich die Tools *vdmdbm*, *vdmhelp* und *dbmc* mit allen benötigten Dateien für das gleiche UNIX-Zielsystem enthält. Die Tools arbeiten lediglich als Clients und benötigen zum Betrieb den laufenden VDM Server *vdmd* auf gleicher oder einer anderen Maschine.

4. **make winclnt.tgz**

   Generierung eines Archives, das die gleichen Tools wie *unixclnt.tgz* enthält, jedoch für die Zielsysteme Windows NT und Windows 95. Alle externen Programme sind bereits übersetzt, so daß keine weiteren Compiler benötigt werden, der Quellcode dieser Programme nebst Makefiles liegt bei. Das Archiv kann durch *tar* und *gzip* als auch durch *Winzip* entpackt werden. Zusätzlich benötigt wird die Binär- oder Quellcodedistribution von TCL 7.5 und TK 4.1 oder Nachfolgeversionen für 32-Bit Windows[5].

Die VDM Klassenbibliothek wurde neben zahlreichen Veränderungen durch diese Diplomarbeit um drei Programme ergänzt:

1. **vdmd** - der VDM-Server
2. **dbmc** - der VDM-Client
3. **vdmtest** - der VDM Beispieltester

Der VDM-Server **vdmd** ist lediglich ein Softlink auf den konventionellen Datenbankmanager **dbm**. Der Dateiname der Datei bestimmt die Funktionsweise des Programms. Abgesehen von zusätzlichen Funktionen, ist der Aufruf des Datenbankmanagers mit dem Namen **dbm** zu 100% und in allen Funktionen mit der ursprünglichen Fassung kompatibel. Der VDM-Client als auch der VDM Beispieltester sind vollständig in TCL erstellt und kommen auch mit der Original TCL-Shell aus.

Die Aufrufsyntax dieser Programme als auch die geänderte Aufrufsyntax des Bibliotheksmanagers **vdmdbm** sowie Bibliotheksbrowsers **vdmhelp** ist in den Manual-Pages zu der Klassenbibliothek zu finden.

C.2 (Auto-)konfiguration der Makefiles

Viele Softwarepakete verwenden zur Portierung das Paket Autoconf von GNU, welches über die Programme **autoscan** die vollautomatische Portierung durch detaillierte Untersuchung zumindest annähernd erreicht, als auch den Erfahrungsschatz und das Know How von GNU Programmieren in Form von vordefinierten Makros zur Verfügung stellt, um die Portierung mittels **autoconf** zu vereinfachen.

Autoconf ist ein Tool von GNU zur Produktion eines Shell-Skriptes, welches automatisch Quellcode-Pakete auf viele Arten von UNIX Systemen anpaßt. Die durch Autoconf produzierten Skriptdateien benötigen keinen Eingriff der Benutzer oder zusätzliche Software und beinhalten alle Funktionen.

Autoconf produziert anhand einer Vorlage (*template*) mit dem Namen *configure.in*, welche Makros enthält das Skript mit dem Namen *configure*. Der Aufruf von *configure* löst die Untersuchung des Systems aus, um die Besonderheiten und den Typ des Systems festzustellen. Abhängig von den Feststellungen, wird aus einer weiteren Vorlage für eine Makedatei mit dem Namen *Makefile.in* die Makedatei angepaßt an das System erzeugt. Zu den UNIX Varianten, die entdeckt werden gehören u.a. AIX, DYNIX/PTX, IRIX, SUN, ISC-POSIX, MINIX, SCO, XENIX und natürlich LINUX.

Das Konfigurationsskript verwendet keine Features, um die nach UNIX Version 7 (also circa 1977) viele Shells erweitert wurden. Die Vorlage *configure.in* hat folgenden Grundaufbau:

[5] In der Datei WIN41.EXE ist TCL7.5 als auch TK 4.1 enthalten. Der Aufruf dieser Datei startet das Installationsprogramm, so daß die gesamte Installationsdauer einschließlich der VDM Tools nicht mehr als 2 Minuten betragen dürfte.

```
AC_INIT(FILE)
checks for programs
checks of librarys
checks of compiler characteristics
checks for further things
AC_OUTPUT(FILE)
```

Mit `AC_INIT` wird die Datei begonnen und legt den Namen einer Referenzdatei fest. `AC_OUTPUT` erzeugt in der Regel eine oder mehere (Make-)Dateien. Weitere von **autoconf** definierte Makros können zwischen diesen Befehlen eingebunden werden, um bestimmte Eigenschaften festzustellen, die für die Portierung der Software von Bedeutung sind.

Da auch das TCL und TK Paket **autoconf** einsetzt, um die Shell für verschiedene System zu erzeugen, hat der Autor dir Vorlage *configure.in* übernommen, um sie für die Portierung des Datenbankmanagers bzw. VDM Servers zu verwenden, da auch hier eine TCL Shell erzeugt wird. Lediglich die Referenzdatei, die über AC_INIT festgelegt wird, hat der Autor wie folgt angepaßt:

```
AC_INIT(./dbm.vdm)
```

Als Referenz für das aktuelle Verzeichnis wird damit die DSL-Spezifikation der Datenbank verwendet. Bei jüngeren TCL-Versionen kann wiederum, um sich die Beschäftigung mit zahlreichen systemspezifischen Eigenschaften zu ersparen, die Vorlage von TCL verwendet und in `AC_INIT` angepaßt werden.

Einzelheiten über die Variablen und vordefinierten Makros sind in der Dokumentation des GNU-Autoconf Paketes hinterlegt[6].

C.3 Konfiguration des Servers

Die Konfiguration des Servers erfolgt vollständig über eine Konfigurationsdatei. Es sind keinerlei Umgebungsvariablen erforderlich. Wenn dennoch das Arbeiten mit Umgebungsvariablen gewünscht ist, können diese in Form von Makros in die Konfigurationsdatei eingesetzt werden, für die dann durch den Inhalt der entsprechenden gesetzten Umgebungsvariable die Ersetzung erfolgt.

Die Konfigurationsart folgt dem Beispiel von zahlreichen unter UNIX bekannten Dämonen (*daemons*), wie beispielsweise *named*, *inetd* oder auch *httpd*. Der VDM Server kann sowohl als Daemon im Hintergrund ablaufen, als auch auf einem Terminal bzw. Terminalfenster, was jedoch nach Beenden der Terminalsitzung auch die Beendigung des VDM Servers zur Folge hat.

Üblicherweise wird ein Daemon beim Hochfahren des Systems durch die rc-Skripte gestartet und bleibt bis zum Herunterfahren, also auf Lebenszeit aktiv. Da sich der Daemon von seinem aufrufenden Prozess als auch von seinem Steuerterminal abkoppelt, wäre ein Setzen von Umgebungsvariablen zur Vornahme der Einstellungen, die nach dem Start nie wieder benötigt werden, auch im Hinblick auf die zahlreichen Konfigurationsoptionen ein schlechter Ansatz.

[6] Unter Linux findet sich diese üblicherweise unter /usr/local/info

Die Verwendung einer Konfigurationsdatei erlaubt nicht nur die Kommentierung der Optionen in der Datei selbst, sondern auch eine außerordentliche Flexiblität bei Anwendung und Entwicklung. Die einzigste Kommandozeilenoption des VDM Servers ist die Angabe der Konfigurationsdatei, die für die Initialisierung verwendet werden soll:

```
vdmd -c <dateiname>
```

Wird diese Option nicht angegeben, so wird eine Konfigurationsdatei mit dem Namen **vdmd.conf** im gleichen Verzeichnis erwartet und verwendet, in der sich auch der Server **vdmd** befindet.

C.3.1 Statische Einbettung des Konfigurationscodes

Bei dem Datenbankmanager bzw. VDM-Server handelt es sich um eine TCL-Shell, mit problemorientierten Hochsprachenerweiterungen. Ursprünglich wurde eine Zeichenkette, welche TCL-Kommandos für das Booten des Datenbankmanagers enthielt, in den Hochsprachencode implementiert und beim Start des Shell ausgeführt. Im Anschluß an die Vorkonfigurationen durch den Bootcode wurde der Code eine externen TCL-Datei ausgeführt, in der weitere Einstellungen hinterlegt sind.

Die Methode zur Initialisierung wurde geändert, sowohl für den Datenbankmanager, als auch den VDM-Server. Der TCL-Datei **config.tcl** im TCL-Quellverzeichnis des Servers kommt eine besondere Bedeutung zu. Der Inhalt dieser Datei wird mit Hilfe des Stream-Editors **sed** in eine Zeichenkette gewandelt, und anschließend im Teilmodul **tclconfig.cc** als Hochsprachen-Stringdefinition festgeschrieben. Diese Datei wird übersetzt und mit **dbm** gebunden. Anstelle des ursprünglichen Boot-Codes wird diese Zeichenkette in die TCL-Shell aufgenommen. In diesem TCL-Code wird auch anhand des Namens ermittelt, ob es sich um den Datenbankmanager oder den VDM-Server handelt, und im letzteren Fall die Konfigurationsdatei eingelesen umgesetzt. Der ursprüngliche Boot-Code wurde unverändert in die Datei **config.tcl** übertragen, und nach wie vor für den Datenbankmanager ausgeführt.

In der Datei **config.tcl** sind neben den Funktionen zur Bearbeitung der Konfigurationsdatei auch die Standardwerte in einem TCL-Array, dem Konfigurationsprofil, hinterlegt. Dieses kann und muß zum Hinzufügen neuer Optionen ergänzt werden, bevor eine Eintragung in der Konfigurationsdatei zulässig ist.

C.3.2 Besonderheiten der Konfigurationsdatei

Einträge in der Konfigurationsdatei unterliegen einigen einfachen Regeln, die Gestaltungsmöglichkeiten in jeglicher Hinsicht ermöglichen:

- Der Optionsname, der in der ersten Spalte beginnen muß, bildet mit der durch Leerzeichen oder Tabulatoren getrennten Wertzuweisung eine Zuordnung
- Das Doppelkreuz leitet Kommentare ein. Der Rest der Zeile wird ignoriert.

- Sonderzeichen, Leerzeichen also auch das Doppelkreuze müssen zu ihrer Verwendung als Zuordnung in doppelte Anführungszeichen eingeschlossen werden. Es gelten hierbei die Escape-Regeln von TCL.
- Zeilenumbrüche des Optionswertes werden durch einen Rückstrich am Zeilenende vorgenommen.
- Mit „$“ eingeleitete Angaben als Optionswert, werden durch die Umgebungsvariable gleichen Namens ersetzt, die zwingend definiert sein muß, beispielsweise kann hiermit durch das Makro `$VDMROOT` ersetzt werden.

Zur Verarbeitung der Konfigurationsdatei können ferner eine Reihe von Substitutionen vorgenommen werden, als auch Vorgaben gesetzt werden. Dies geschieht durch einen Eintrag in das Konfigurationsprofil in der Datei **config.tcl**. Das Profil die Verarbeitung der Einträge fest, und wird als Zeichenkette angegeben.

Die Syntax und Bedeutungen für einen Profilstring sind wie folgt:

- **Erzwingen einer Option**

  Eine Option, die in der Konfigurationdatei nicht fehlen darf, wird durch ein Ausrufezeichen gekennzeichnet.

- **Vorgabewerte**

  Vorgaben für Optionen werden im Profil als Zeichenkette angegeben. Auch der Leerstring ist zulässig. Fehlt eine Option in der Konfigurationsdatei, wird die Option aus dem Profil verwendet.

- **Pfadersetzung**

  Festlegungen von Pfaden im Profil, die durch ein Doppelkreuz getrennt sind, führen zur Zusammensetzung von Pfaden. Die beispielhafte Angabe von `VdmRoot#./` bewirkt, daß der Pfad aus dem durch die Option *VdmRoot* spezifizierten, und dem Kreuz folgenden zusammengesetzt wird, also in diesem Fall identisch mit dem Eintrag für *VdmRoot* ist. Die Angabe eines relativen Pfades in der Konfigurationsdatei verwendet ebenfalls *VdmRoot* als Basis, fügt diesen jedoch nur dann hinzu, wenn es sich nicht um einen absoluten Pfad handelt. Die Angabe von `VdmRoot#./bin` in dem Profil würde entsprechend als Standardwert ein Unterverzeichnis zu *VdmRoot* bezeichnen.

C.4 Durchführung der Konfiguration

Die Konfigurationsdatei liegt im Entwicklungsverzeichnis *./tools/g++vdmdbm* in zweifacher Ausfertigung vor. Zum einen mit dem Standardnamen *vdmd.conf*, zum anderen als *vdmd.instconf*. Die erstere ist zur Entwicklung vorkonfiguriert, die letztere verwendet fast ausschließlich Standardoptionen, und wird bei der Installation mittels `make INSTALL` als auch `make INSTFLAT` in das Zielverzeichnis übertragen. Dabei wird die Option *VdmRoot* an das Zielverzeichnis der Installation angepaßt und der Dateiname auf den Standardnamen gesetzt.

Die Installationsoptionen sind zwar in der Konfigurationsdatei kurz erläutert, deren Hintergrund sowie wichtige Zusammenhänge sind aber insbesondere für die Weiterentwicklung wichtig, und daher Thema dieses Abschnitts, in dem Beispiele angegeben werden zu denen jeweils eine Tabelle die Erläuerungen vornimmt.

Die Option

```
Version                 1.0
```

muß in jeder Konfigurationsdatei enthalten sein, und vorstehenden Wert beinhalten, der bei einer Fortschreibung mit Änderung des Sinngehaltes einer Optionen für eine eindeutige Indentifizierung sorgt.

C.4.1 Verzeichnisoptionen

Die Verzeichnisoptionen legen fest, an welcher Stelle im Verzeichnisbaum benötigte Dateien für den Server hinterlegt sind. Die Standardeinstellungen entsprechen der derzeit gültigen Verzeichnisstruktur. Tabelle C.4.1 beschreibt alle Konfigurationsoptionen.

Beispiel:

```
VdmRoot                      $VDMROOT/
VdmLibrary                   ./lib

ServerRoot                   /$VDMROOT/tools/g++vdmdbm
ServerTCLDir                ./server

MimeTCLDir                  ./mime
DPTCLDir                    ./dp

ConsoleLogRoot              ./tools/g++vdmdbm/log
ServerLogRoot               ./tools/g++vdmdbm/log/server
AgentLogRoot                ./tools/g++vdmdbm/log/agents

ExamplesDir                 /$VDMROOT/Test
```

C.4.2 Allgemeine Logfunktionen

Die Logfunktionen sind ein wichtiger Bestandteil zur Nachverfolgung von Vorgängen. Ein Mitschreiben hilft, Fehler zu beseitigen, die den VDM Server jedoch keinesfalls terminieren. Die Logoptionen sind in Tabelle C.4.2 aufgeführt.

Beispiel:

```
ConsoleLogStderr        enabled
ConsoleLogStdout        enabled

ServerLogDebug          disabled
ServerDebugDepth        100
ServerLogOSI5           enabled
ServerLogRPC            enabled
```

C.4.3 Agent-Logfunktionen

Die Agent-Logfunktionen beziehen sich sowohl auf die Benutzeragents als auch auf einen eventullen separaten Datenbankprozeß. Die Einstellungen sind zum großen Teil identisch mit den Logfunktionen des Servers. Sie sind in Tabelle C.4.3 genau definiert.

Beispiel:

```
AgentLogSessions       enabled
AgentLogStderr         enabled
AgentLogStdout         enabled
AgentLogDebug          enabled
AgentDebugDepth        100
```

C.4.4 Server Optionen

Die Serveroptionen legen mit die wichtigsten Details zu der Kommunikation des VDM Servers fest. Zum einen werden die verfügbaren Serverfunktionen bestimmt, zum anderen der Port über welchen diese erreichbar sind. Die genaue Beschreibung findet sich in Tabelle C.4.4.

Beispiel:

```
ServerRPC               enabled
ServerTELNET            enabled

PortRPC                 4711
PortTELNET              4712

ManagerPassword         vdm

ShowErrorDetails        enabled

AgentProcess            disabled
```

C.4.5 VDM-spezifische Optionen

Die VDM spezifischen Optionen betreffen die Datenbank, die vom Server zwingend verwendet wird, unabhängig von jeglichen anderweitigen Angaben von Klienten, als auch den VDM Preprozessor, der zur Aufbereitung der Dokumentation als externes Programm benötigt wird. Näheres ist in Tabelle C.4.5 erläutert.

Beispiel:

```
VdmPreprocessor        ./vdmpp
VdmDefaultDatabase     ./vdmdb-c
```

C.4.6 Verschiedene Optionen

Weitere Optionen, die keiner genauen Thematik zugeordnet werden, gibt Tabelle C.4.6 wieder.

Beispiel:

```
ServerRCScript                 /u/vdm/work/g++vdmdbm/vdmdrc

DaemonOperation                disabled
Prompt1                        "vdmd > "
Prompt2                        "> "
```

C.4.7 HTML-Optionen

Die HTML-Optionen betreffen im wesentlichen die Konfigurationen, die für die Produktion von HTML-Seiten für das Informations- und Hilfesystem von Bedeutung sind. Tabelle C.4.7 gibt über die Optionen Auskunft.

Beispiel:

```
TableWidth                4

MimeEncoder               /u/vdm/work/g++vdmdbm/external/mmencode
MimeDecoder               /u/vdm/work/g++vdmdbm/external/mmencode -u

KnownTags                 "H1 H2 H3 H4 H5 H6 BR\
                            /H1 /H2 /H3 /H4 /H5 /H6"

ExampleBrowser            /$VDMROOT/tools/g++vdmdbm/vdmtest
ExampleVdmRoot            /$VDMROOT/
ExampleVdmBin             /$VDMROOT/bin
```

C.5 Start und Beenden des VDM Servers

Nach der Einrichtung einer Konfigurationsdatei oder Verwendung einer bestehenden wird der Server durch Eingabe von

```
vdmd
```

oder

```
vdmd -c myvdmd.conf
```

für die Verwendung einer bestimmten Konfigurationsdatei gestartet. Wenn der Server als Daemon konfiguriert wurde, wird unverzüglich zur Shell zurückgekehrt. Der Server wird grundsätzlich nicht als Hintergrundprozeß von der Shell gestartet, also ohne das Zeichen „&" hinter der Eingabe. Wenn der Server fester Bestandteil des Systems ist, sollte er in die rc-Skripte zum Start eingetragen werden.

Der Server benötigt auf einigen UNIX-Systemen wozu im wesentlichen System V Rel 3.2 und 4.0 zählen dürften, wie auch SUN-OS, die effektiven Rechte der Benutzer als auch Gruppe Null, ergo Supervisorrechte. Der Grund hierfür liegt darin, daß die *setsid*-Systemfunktion als auch Pseudoterminals unter System V diese Rechte benötigen. Abhängig vom System bricht der VDM-Server die Initalisierung mit einer Fehlermeldung ab[7]. Der Start des Servers über ein rc-Skript sichert automatisch Supervisorrechte zu.

Wir der VDM Server ohne Daemon-Funktionalität gestartet, so zeigt sich das Prompt, das in der Konfigurationsdatei hinterlegt wurde. Dem Prompt ist das Wort „console:" vorangestellt. Sollte entegegen aller Wahrscheinlichkeit dieses Prompt nicht erscheinen, so hat der Server die Endlosschleife zur Bearbeitung von Events beendet und ist nicht mehr funktionsfähig. Der Neueinsprung in diese Schleife wird durch Eingabe von

```
console
```

durchgeführt. Die Beendigung des Servers erfolgt durch Eingabe von `exit` oder CTRL-D.

C.6 Steuerung und Fernsteuerung des Servers

Die Steuerung des Servers kann durch direkte Eingabe von TCL-Kommandos am Prompt der Console durchgeführt werden. Diese Kommandos werden jedoch nicht direkt vom TCL-Shell, sondern ereignisgesteuert abgearbeitet. Eingabefehler können daher nicht zu einem Abbruch des Servers aufgrund eines Fehlers führen. Die TCL-Shell arbeitet nicht interaktiv, sondern es handelt sich lediglich um eine Emulation des interaktiven Modus, die durch ein beständig laufendes Programm erzeugt wird. Die Ausführung der Kommandos erfolgt grundsätzlich auf Programmebene 0 von TCL.

[7] Die Fehlermeldung entstammt der Datei *daemon.c*, wo Ursachenforschung betrieben werden kann

Durch Eingabe von `control` am Prompt des Servers wird die Verarbeitung von weiteren Events unterbrochen, und es erscheint ein Menü, welches die Anzeige der Konfiguration als auch das Starten und Beenden von RPC- und TELNET-Server ermöglicht. Erst nach Verlassen des Menüs werden weitere Ereignisse abgearbeitet.

Die Fernsteuerung des Servers wird mit TELNET durchgeführt, es können jedoch nur solche Derivate dieses Programmes verwendet werden, welche zusätzlich zum Hostnamen die Angabe des Portes zulassen. Die Verbindung mit dem VDM-Server wird unter UNIX durch beispielsweise folgende Eingabe aufgenommen:

```
telnet linux 4712
```

Als Portadresse muß diejenige Portadresse angegeben werden, die in der Konfigurationsdatei für den TELNET-Server eingetragen wurde. Nach der Begrüßungsmeldung des VDM Servers wird die Eingabe eines Paßworts erwartet, da der Fernzugriff auf die Serverconsole wechselt, und, wenn der Server mit Supervisorrechten läuft, der Fernbenutzer dazu befugt ist, jegliche Kommandos abzusetzen. Das Passwort ist unkodiert in der Konfigurationsdatei des Servers hinterlegt, woraus folgt, daß die Konfigurationsdatei mit entsprechenden Zugriffsrechten versehen sein muß, um der Datensicherheit Rechnung zu tragen.

Eine Fernsitzung mit dem VDM Server **muss** durch das Kommando

```
quit
```

beendet werden. Die Eingabe von **exit** hätte eine Beendigung der TCL-Shell und damit auch des Servers zur Folge. Auch ein Abbruch der Telnetsitzung ohne Eingabe von **quit** sollte unterbleiben Während einer Fernsteuerung des Servers werden sowohl Standardausgabe als auch Fehlerausgabe auf den Fernbediener umgeleitet. Das Beschreiben einer Logdatei oder Ausgaben auf dem virtuellen Terminal werden daher für die Dauer der Sitzung unterbrochen. Meldet sich ein weiterer Manager über TELNET an während bereits eine Verbindung aktiv ist, so wechselt der Bildschirm auf den neuen Manager über, und kehrt erst nach nach der Beendigung zurück.

C.7 Funktionen des Servers

Eine Erweiterung der Funktionalität des Datenbankmanagers zur Schaffung eines Servers wurde in zweierlei Hinsicht vorgenommen. Zum einen wurden Funktionen dem Kern des Managers[8] hinzugefügt, wie im theoretischen Teil der Diplomarbeit besprochen wird. Zum anderen geschah die Aufnahme systemspezifischer Kommandos in die TCL-Shell, die auf der POSIX-Norm basieren, und daher in der Regel ein UNIX System voraussetzen.

Bei der Entwicklung des Systems wurde zunächst versucht, viele einzelne POSIX-konforme Funktionen in die TCL-Shell einzubetten, um aus diesen einen Server-Daemon zu konstruieren. Als die Umsetzung dieses Vorhabens zu Auswüchsen des TCL-Codes als auch C-Codes führte, fiel der Entschluß, diese Kommandos in einem Befehl zu konzentrieren. Viele der bereits umgesetzten Kommandos wurden ersatzlos gestrichen, und neue Befehle auf ein überschaubares Minimum reduziert.

[8] Sie sind zu finden in der Datei **vdmdbm.cc**, in welcher im wesentlichen die Klasse DBMANAGER ergänzt wurde

C.7.1 Neue TCL Kommandos des VDM-Datenbankmanagers

Zunächst werden die Neuzugänge der Funktionen, die im Hochsprachencode des Datenbankmanagers erstellt sind, dokumentiert. Sie füllen Lücken in der Funktionalität aus und erlauben den Zugang zu der Multisessionfähigkeit.

directimport

Syntax:

```
directimport <flatformat>
```

Das **directimport** Kommando erlaubt das Importieren in die Datenbank äquivalent zum ursprünglich bestehenden Befehl **import**. Bisher war der Import in die VDM Datenbank aus einer Datei mit FLAT-Struktur[9] lediglich durch explizite Angabe eines Dateinamens möglich. **directimport** gestattet die Aufnahme aus einem TCL-String heraus, der äquivalent zum Inhalt einer FLAT-Exportdatei ist. Als boolscher Funktionswert wird der Erfolg des Imports durch das Kommando zurückgegeben.

Das Kommando **directimport** wurde benötigt, um Dateiinhalte, die vom Server über das Netzwerk empfangen werden und von den Klienten stammen, importieren zu können.

exportformat

Syntax:

```
exportformat <adt> <impl> <oper> <comp> <value>
```

Ebenso wie **directimport** für das Arbeiten im Netzwerk erforderlich ist, gilt dies auch für das neue Kommando **exportformat**. Der Export von Datenbankinhalten mittels **export** oder **partialexport** wird im Gegensatz zu den konventionellen Importfunktionen als Funktionswert zurückgegeben, die Inhalte stammen aus der Datenbank. **exportformat** erzeugt aus der TCL-Zeichenkette *value* sowie den Parametern *adt*, *impl*, *oper* und *comp*, die das Element bezeichnen, das gleiche Ausgabeformat. Der Import von Komponenten über das Netzwerk wird durch Erzeugung dieses Formats mit anschließendem Import mittels **directimport** auf dem Server durchgeführt.

session

Syntax:

```
session create

session create <id>

session exists <id>
```

[9]Als FLAT-Struktur bezeichnet der Autor das Format einer aus der Datenbank als Shell-Skript exportierten Datei, das durch Aufruf die Erzeugung der DIRECTORY-Struktur seines Inhaltes vornimmt

```
session release <id>

session <id>

session list

session

session help
```

Die Bedeutung und Theorie der Multisessionfähigkeit des Servers ist im Zusammenhang mit dem Kontextwechsel unter 5.1.5 erläutert. Die hiesigen Funktionen sind äquivalent. Der Zugang zu diesen Funktionen erfolgt über das neu definiert TCL-Kommando **session**. Eine Online-Hilfe kann mit **session help** abgerufen werden.

Die Semantik der weiteren Kommandos ist, in der Reihenfolge ihres Auftretens in der Syntaxbeschreibung,

1. Das implizte Erzeugen einer neuen Sitzung, dessen Identifikationsnummer automatisch vom System vergeben wird und negativ als auch kleiner null ist.

2. Eine Sitzung wird explizit erzeugt, der Parameter *id* legt die Identifikationsnummer fest. Die Nummer muß größer null sein[10]. Bestehende Sessions werden überschrieben.

3. Testen auf Existenz einer Sitzung

4. Freigabe einer Sitzung. Die Freigabe der aktuellen Sitzung hat den Wechsel zur Standardsitzung zur Folge.

5. Wechsel zur Sitzung *id*

6. Ausgabe aller erzeugten Sitzungen

7. Ausgabe der aktuellen Sitzung

Das Sitzungskommando wird ausschließlich vom jeweiligen Benutzer-Agenten (*user agent*), der auf dem Server mit der Aufnahme der Verbindung durch einen Klienten eingerichtet wird, und alleiniger Ansprechpartner des Klienten ist, bedient. Der Agent richtet die Sitzung für den neuen Klienten ein, arrangiert den Wechsel zu ihm bei Anfragen als auch den Rückwechsel zum vorhergehenden und gibt die Sitzung wieder frei, sobald die Klientverbindung abbricht, beendet wird oder ein Ausnahmezustand auftritt.

Da der Agent über Ereignissteuerung agiert, verfügt dieser bei seiner Aktivität über einen eigenen TCL-Interpreter. Die Eingabe des Sitzungskommandos **session** über die Fernwartung oder vom Terminal aus zu Testzwecken ist durchaus zulässig und unschädlich. Das Löschen von Sitzungen hat den Abbruch des Klienten beim nächsten Zugriff zur Folge.

[10]Null ist ausschließlich reserviert für die Standardsitzung, über den auch die Fernwartung durchgeführt wird

C.7.2 TCL-Kommandoerweiterungen des VDM-Servers

Bei der Implementierung der Serverfunktionalität wurde dafür Sorge getragen, daß der VDM Server auch ohne jegliche systemspezifische Funktionen auskommt und lauffähig ist. Dies auch im Hinblick auf eventuelle zukünftige Portierungen auf andere Plattformen, auf denen TCL durchaus lauffähig ist, die verwendeten speziellen UNIX-Funktionen unter Umständen jedoch nicht.

Bei einem Verzicht auf die Erweiterungsfunktionen ergeben sich jedoch fünf Einschränkungen:

1. Es ist keine Daemon-Operation des Servers möglich. Wenn dem Prozeß, der den VDM Server startet ein Kontrollterminal zugeordnet ist, bleibt die Kopplung bestehen. Gleichfalls wird der Server nicht zum eigenen Prozeßgruppenführer. Auch anderer Eigenschaften, über die ein Daemon-verfügt werden nicht geschaffen.
2. Ein Umleitung der Standardausgabe oder Fehlerausgabe auf eine Logdatei ist nicht möglich
3. Der Start eines separaten Datenbankprozesses kann nicht erfolgen
4. Die Fernwartung des Server über TELNET wird nicht unterstützt
5. Der integrierte Debugger ist nicht aktivierbar

Alle TCL-Kommandos wurden aufgrund ihrer Kürze in C erstellt. Der Grund hierfür liegt ferner in dem Verzicht auf objektorientierte Programmierung durch die Ersteller der Tool Command Language und der damit verbundenen Förderung der Einheitlichkeit. Der Code ist nach [GL95] *Lehey, Greg: Porting UNIX Software, 1995* portabel programmiert, und sollte sowohl für alle Releases von System V als auch BSD sowie Linux, was die Funktionen beider Derivate beinhaltet, gültig sein. Auch die Besonderheiten von SUN-OS wurden berücksichtigt. Praktisch getestet werden konnten lediglich SUN-OS als auch Linux. Der Quellcode befindet sich unter *./tools/g++vdmdbm/cmds* der Installation der Klassenbibliothek.

daemon und vt

`vt <masterfd> <slavefd>`

Der Befehl **vt** öffnet ein Pseudoterminal auf dem Host. Ein Pseudoterminal ist ein Gerätepaar, daß aus Master und Slave besteht, und sich wie ein Terminaldevice verhält. Ohne Pseudoterminals (*PTY*) ist eine Fernanmeldung auf UNIX-Hosts über das Netzwerk nicht möglich, daher sind sie in allen bekannten UNIX-Derivaten verfügbar. Das Gerätepaar eines PTY bezieht sich auf ein und dasselbe Gerät. Während der Master das Kontrollterminal des jeweiligen Prozesses ist, wird der Slave über das Netzwerk bedient. Detailierte Informationen über Pseudo-Terminals gibt Stevens in [RS92] [RSE92] und [RS95]. Die Funktion **vt** öffnet ein ein PTY und setzt das Master- als auch Slave-Device als TCL Dateideskriptor in die übergebenen Parametern ein. Der **vt** Befehl wird vom Hypertext Informations- und Hilfesystem nicht direkt, sondern lediglich über das **daemon**-Kommando verwendet.

`daemon [masterfd] [slavefd]`

Über die Funktionsweise von Daemonen (*daemon*) gibt [RS92] Stevens Auskunft. Typische Eingenschaften eines Daemon-Prozesses sind:

- Sie werden bei der Initialisierung des Systems einmal aufgerufen
- Sie sind während der gesamten Einschaltdauer des Systems aktiv. Sie werden nicht unterbrochen und fortgesetzt
- Die meiste Zeit warten Sie auf das Ereignis, bei dessen Auftreten sie in Aktion treten.
- Zum Durchführen von Service-Requests verwenden und erzeugen sie häufig andere Prozesse.

Der neugeschaffene Daemon-Befehl für TCL verwandelt die laufende TCL-Shell mit sofortiger Wirkung in einen Daemon-Prozeß. Im Gegensatz zu üblichen Daemonen wird hierbei ein virtuelles Kontrollterminal zugewiesen, dessen Master- und Slave-Device als TCL-Dateideskriptor in die übergebenen TCL-Variablen gesetzt werden, sofern diese angegeben werden. Von dieser Option wird vom VDM Server jedoch nicht Gebrauch gemacht, da lediglich Kommandozeilen bedient werden müssen.

Die Unterschiede, die zu üblichen Daemon-Prozessen und solchen Prozessen, die mit **daemon** gestartet werden, bestehen sind im Wesentlichen wie folgt:

- Üblicherweise werden alle Datei-Deskriptoren aller geöffneten Dateien geschlossen, der VDM Server läßt die Standardeingabe, -ausgabe und -fehlerkanäle offen.
- Das Arbeitsverzeichnis eines Daemons ist das Hauptverzeichnis „/", um Veränderungen durch mounten bzw. demounten von Dateisystemen zu entgehen. Der VDM Server hat ein eigenes individuelles Verzeichnis.

Alle weiteren Eigenschaften werden mit üblichen Dämonen geteilt. Diese sind Ablauf im Hintergrund durch Abkoppeln von der Prozeßgruppe, ignorieren der Terminal I/O-Signale, Abkoppeln vom Urspungsterminal und Ignorieren des Exit-Status von Kindprozessen.

debug

`debug <level> <file>`

`debug on`

`debug off`

`debug stop`

Der VDM Server verfügt über einen eigenen Debugger, der in der Lage ist, den gesamten Ablauf der TCL-Programme mit allen Substitutionen aufzuzeichnen. Die Einreichtung des Debuggers wird durch Aufruf des Kommandos mit Angabe der Debugtiefe (*level*) und der Ausgabedatei (*file*) durchgeführt, er wird jedoch nicht gestartet.

Erst mittels **debug on** wird die Aufzeichnung aktiviert und mit **debug off** beendet. Durch Einrahmen von TCL-Codesequenzen durch diese Befehle können gezielte Bereiche untersucht werden. Die Debugtiefe *level* legt fest, bis zu welcher Rekursionstiefe die Aufzeichnung vorgenommen werden soll. Sie ist äquivalent zu der Tiele, die mit dem TCL-Befehl *uplevel #level cmd* festgelegt wird. Bei Programmende oder durch **debug stop** wird die Aufzeichnung beendet.

dup und pipe

Die UNIX Systemfunktioonen *dup* und *pipe* zählen zu den Schlüsselkommandos. Ohne diese Funktionen wäre die gesamte Kommunikation, die Konstruktion von Filtern und die Kommunikation von Prozessen undenkbar. Beide Funktionen waren in älteren Versionen als 7.4 von TCL zwar unkommentierte Funktionen, aber definiert. Sie wurden jedoch später vermutlich aufgrund der erweiterten Portabilität entfernt, und daher vom Autor neu geschaffen.

```
dup <oldfd> <newfd>
```

```
dup oldfd
```

```
pipe
```

Das *dup* Kommando mit der Angabe von zwei Dekriptoren entspricht in seiner Funktion der *dup2*-Systemfunktion von UNIX. Der Deskriptor *oldfd* wird auf *newfd* dupliziert und *newfd* dabei nötigenfalls geschlossen. Beide Deskriptoren bezeichnen ein und dieselben Kanal. Das *dup* Kommando ist mit der *dup* Systemfunktion identisch und dupliziert den Quellkanal implizit auf den kleinsten geschlossenen Kanal. Dieser wird zurückgegeben.

Das *pipe* Kommando erwartet keine Parameter und gibt zwei TCL-Deskriptoren als Liste zurück, die eine Pipe definieren. Der erste Deskriptor ist lediglich lesbar, der zweite lediglich beschreibbar[11]. Prozesse kommunizieren miteinander, indem nach Aufruf der *fork*-Funktion der eine Prozeß die Eingabeseite, der andere die Ausgabeseite schließt und die jeweils andere Seite zum Lesen bzw. zum Schreiben verwendet wird. Pipes werden durch Aufruf von Kindprozessen vollständig verererbt, wozu vom Kernel das *dup* Kommando verwendet wird.

fork

```
fork
```

Mit *fork* werden Kindprozesse gestartet. Ein Rückgabewert von 0 bedeutet, daß es sich um den Kindprozeß handelt. Der Vaterprozeß erhält die Prozeßnummer des Kindprozesses als Ergebnis.

[11] Intern wird an die Systemfunktion eine Array aus zwei Integer-Werten übergeben. Ebenso wie die Standardeingabe 0 und die Standardausgabe 1 ist, verhält es sich mit dem Array.

getgid, getuid, setpgrp und ppid

```
getgid
```

```
getuid
```

```
setpgrp
```

```
ppid
```

Diese Funktionen werden vom VDM Server nicht mehr verwendet, sind aber implementiert und beziehen sich auf die (effektive) Gruppen-, (effektive) Benutzeridentifikation als auch das Setzen des Prozeßgruppenführers als auch die Rückgabe der Prozeßnummer des Vaterprozesses in Ergänzung des *pid* Kommandos von TCL. Im weiteren wird auch die Handbuchseiten von UNIX verwiesen.

reopen

```
reopen <fname> <mode> <file> <w|r>
```

Der neugeschaffene TCL-Befehl *reopen* ist vergleichbar mit der UNIX-Systemfunktion *freopen*, jedoch nicht identisch. Er erlaubt das Umleiten eines TCL-Dateideskriptors, der mit *file* angegeben wird, auf eine Datei mit durch *mode* definiertem Zugriffsmodus, der gleiche Werte akzeptiert, wie das *open* Kommando von TCL.

Der reopen-Befehl wird vom VDM Server zur Umleitung der Standardausgabe als auch -fehlerausgabe in eine (Log-)datei verwendet. Dies ist auch für die Standardeingabe möglich, wenn eine Datei als FIFO erzeugt, und in diese von anderer Stelle geschrieben wird. Abhängig von Richtung der Umleitung muß *w* oder *r* angegeben werden. Die Umleitung der Standardeingabe wurde mit Schaffung der Fernwartung überflüssig.

signal und kill

```
signal
```

```
signal <signum>
```

```
signal <signum> [sigaction | SIG_IGN | SIG_DFL]
```

```
kill
```

```
kill <signum> <pid>
```

Die Kommandos *signal* und entstammen auch dem Satz der UNIX-Funktionen. Während *kill* ebensogut auch als externes Programm aufgerufen werden könnte, erlaubt *signal* die asynchrone Ausführung von TCL-Befehlen als Reaktion auf ein externes Signal. Der parameterlose Aufruf von *signal* oder *kill* zeigt alle auf dem jeweiligen UNIX-Host gültigen Signale an. Der Aufruf mit einem Signal gibt den für dieses Signal hinterlegten TCL-Code zurück. Die verbleibende Syntax erlaubt die Hinterlegung des Codes, als auch die Festlegung, ob das Signal ignoriert oder die Standardbehandlung aktiviert werden soll.

Der Signalverarbeitung kommt durch den VDM Server lediglich eine Bedeutung bei der Ausführung der externen Beispiele als Daemon zu, die ein Setzen auf die Standardbehandlung erforderlich macht, da die Beendigung von Prozessen ansonsten von der TCL-Shell nicht erkannt wird. Das *kill* Kommando wird zur Sicherung der Terminierung eines externen Datenbankprozesses verwendet.

wait und waitpid

Zur Überwachung von Kindprozessen kommen unter UNIX viele Möglichkeiten in Betracht, die auch vom Derivat abhängig sind. Kindprozesse, deren Elternprozeß terminiert ist, ohne daß dieser den *exit*-Status erfragt hat, werden als sogenannte *Zombies*, also lebende Tote, bezeichnet (was hiermit nur vereinfachend dargestellt ist). Die folgenden Kommandos *wait* und *waitpid* stellen nur eine Möglichkeit dar, Zombie-Prozesse zu vermeiden, und sind funktional mit den gleichnamigen UNIX-Systemfunktionen identisch.

`<wait>`

`<waitpid> <pid>`

Das *wait* Kommando wartet auf die Beendigung aller Kindprozesse, der *waitpid* Befehl ist differnenziert auf einen bestimmten Prozeß[12], auf den nicht gewartet, sondern für den im voraus bestimmt wird, daß der Exitstatus als bereits abgefragt gilt. Da die Überwachung von Kindprozessen im *fork*-Kommando der TCL-Shell mittels *Tcl_ReapDetachedProcess* bereits integriert ist, werden weder *wait* noch *waitpid* weiter verwendet, können aber dennoch nützlich sein und sind deshalb weiterhin definiert.

[12] Als auch weiteren Optionen, die im Hochsprachencode explizit gesetzt werden

Option	Vorgabewert	Bedeutung
VdmRoot	/usr/local	Hauptverzeichnis der VDM-Installation.
VdmLibrary	/usr/local/lib	Bibliotheksverzeichnis der VDM-Installation. Relative Angaben basieren auf *VdmRoot*.
ServerRoot	/usr/local/lib	Verzeichnis, in welches der Server nach dem Start wechselt. Relative Angaben basieren auf *VdmLibrary*.
ServerTCLDir	/usr/local/lib/vdm/server	Verzeichnis, in dem die TCL-Dateien des Servers hinterlegt sind. Relative Angaben basieren auf *ServerRoot*.
MimeTCLDir	/usr/local/lib/vdm	Verzeichnis, in dem sich die TCL-Dateien zur MIME-Verarbeitung befinden. Relative Angaben basieren auf *ServerRoot*.
DPTCLDir	/usr/local/lib/vdm	Verzeichnis, das die TCL-Dateien für Distriubuted Processing beinhaltet. Relative Angaben basieren auf *ServerRoot*.
ConsoleLogRoot	/usr/local/log	Verzeichnis, in dem die Standardausgabe als auch Fehlerausgabe des Servers in Dateien umgeleitet wird. Relative Angaben basieren auf *ServerRoot*. Die Ausgabe erfolgt nur, wenn die Optionen *ConsoleLogStdout* oder *ConsoleLogStderr aktiviert sind* .
ServerLogRoot	/usr/local/log	Verzeichnis, in dem Protokollierungen der Sitzungsschicht, das Mitschreiben von RPCs und führen der Debugdateien erfolgt. Relative Angaben basieren auf *ServerRoot*. Die Schalter *ServerLogDebug*, *ServerLogOSI5* und *ServerLogRPC* aktivieren die entsprechenden Protokollierungen.
AgentLogRoot	/usr/local/log	Verzeichnis, in dem die Protokolldateien von Benutzeragenten abgelegt werden. Relative Angaben beruhen auf *ServerRoot*. Auch Mitschriften eines selbständigen Datenbankprozesses werden dort geführt, wenn der Schalter *AgentProcess* gesetzt ist.
ExamplesDir	/usr/local/Test	Hauptverzeichnis der VDM Anwendungsbeispiele. Relative Pfadangaben beruhen auf *VdmRoot*

Tabelle C.1: Verzeichnisoptionen des Servers

Option	Vorgabewert	Bedeutung
ConsoleLogStderr	disabled	Protokollierung der Standardausgabe des Servers in einer Logdatei
ConsoleLogStdout	disabled	Protokollierung der Fehlerausgabe des Servers in einer Logdatei
ServerLogDebug	disabled	Einschalten des Debuggers. Sämtlicher TCL-Code wird protokolliert. Da die Größe des Mitschnitts schnell mehrere Megabytes umfassen kann, empfielt sich die Einstellung nur für konkrete Diagnose.
ServerDebugDepth	100	Prozedurtiefe des Debuggermitschnitts. Die Angabe ist äquivalent zum TCL-Befehl *uplevel #level cmd*.
ServerLogOSI5	disabled	Protokollieren der OSI Schicht 5 des Servers. Dies ist die Sitzungsschicht.
ServerLogRPC	disabled	Protokollieren aller Remote Procedure Calls.

Tabelle C.2: Logoptionen des Servers

Option	Vorgabewert	Bedeutung
AgentLogSessions	disabled	Vollständige Protokollierung aller Sitzungen der Benutzeragenten mit Anfragen und Antworten
AgentLogStderr	enabled	Nur gültig bei separatem Datenbankprozeß. Mitschneiden der Fehlerausgabe dieses Prozesses.
AgentLogStdout	enabled	Nur gültig bei separatem Datenbankprozeß. Mitscheiden der Standardausgabe dieses Prozesses.
AgentLogDebug	disabled	Protokollieren des gesamten TCL-Codes des Datenbankenprozesses.
AgentDebugDepth	100	Debugtiefe

Tabelle C.3: Logoptionen der Benutzeragenten

Option	Vorgabewert	Bedeutung
ServerRPC	enabled	Legt fest, ob der RPC-Server gestartet wird. Ohne RPC-Server kann von Klienten auf den VDM Server nicht zugegriffen werden
ServerTELNET	enabled	Option, die festlegt, ob die Fernwartung für den Zugriff über TELNET gestartet wird.
PortRPC	Zwingend Angabe	Listening-Port des RPC-Servers.
PortTELNET	Zwingende Angabe	Listening-Port des TELNET-Servers
ManagerPassword	vdm	Passwort, das bei Zugriff über die Fernwartung eingegeben werden muß. Der Zugriff über die Fernwartung räumt die gleichen Rechte ein, über die der Server selbst verfügt.
ShowErrorDetails	disabled	Fehler des VDM Servers werden jeweils dem Klienten ausgegeben, der den Fehler verursacht hat. Fehler die unabhängig vom Klienten bei Server auftreten, erscheinen auf der Serverconsole. Ein Einschalten dieser Option verursacht die Ausgabe des gesamten TCL-Stacks zusätzlich zum Fehler.
AgentProcess	disabled	Das Einschalten dieser Option veranlaßt den Server, einen separaten Datenbankprozeß zu starten, welcher die Datenbank lädt. Der Serverprozeß selbst leitet alle Anfragen an den separaten Datenbankprozeß weiter.

Tabelle C.4: Optionen des Servers

Option	Vorgabewert	Bedeutung
VdmPreprocessor	/usr/local/lib/vdmpp	Pfadangabe zum Preprozessor. Relative Angaben werden durch die Option *VdmLibrary* ergänzt.
VdmDefaultDatabase	/usr/local/lib/vdmdb-c	Pfadangabe zur VDM-Datenbank. Relative Angaben werden durch die Option *VdmLibrary* ergänzt.

Tabelle C.5: VDM spezifische Serveroptionen

Option	Vorgabewert	Bedeutung
ServerRCScript		Optionales Pfad zu einem TCL-Skript, das beim Start des Servers ausgeführt wird, sofern es angegeben ist
DaemonOperation	enabled	Ausführung des Server als Daemon im Hintergrund.
Prompt1	”vdmd ¿ ”	Prompt der Serverkonsole
Prompt2	”¿ ”	Prompt der Serverkonsole für nicht abgeschlossene Befehle

Tabelle C.6: Sonstige Serveroptionen

Option	Vorgabewert	Bedeutung
TableWidth	4	Tabellenspaltenbreite für platzoptimale Darstellung der Auswahltabelle im Browser
MimeEncoder	/usr/loca/lib/vdm/mmencode	Pfad zum externen MIME-Encoder
MimeDecoder	/usr/local/lib/vdm/mmencode -u	Pfad zum externen MIME-Decoder, der in der Regel auch optional MIME-Encoder sein dürfte
KnownTags		HTML-Tags, die durch Makros vom Preprozessor erzeugt werden, müssen durch diese Option dem VDM Server bekannt gemacht werden, da sie andernfalls als Text dargestellt werden.
ExampleBrowser	/usr/local/lib/vdm/vdmtest	CGI-Interface für die Ausführung und Anzeige der VDM Anwendungsbeispiele. Ein relativer Pfad basiert auf *VdmRoot*.
ExampleRoot	/usr/local	Die Umgebungsvariable *VDMROOT* des Example Browsers wird auf diese Angabe gesetzt. Relative Pfadangaben basieren auf *VdmRoot*
ExampleVDMBin	/usr/local/bin	Verzeichnis, das zur Umgebungsvariable *PATH* des Example Browsers, hinzugefügt wird, damit im wesentlichen der VDM-Domaincompiler zur Übersetzung der Beispiele gefunden werden kann. Relative Pfadangaben basieren auf *VdmRoot*

Tabelle C.7: HTML-orientierte Optionen des Servers

Anhang D

Vorausschau auf mögliche Weiterentwicklungen

Abschließend sollen die Gedanken des Autors, die ihm perspektivisch während der Bearbeitung der Diplomarbeit hinsichtlich von Weiterentwicklungen der Klassenbibliothek in den Sinn kamen, angerissen werden.

Schon bei der Implementation des VDM Servers, der unweigerlich programmiert werden mußte, war der Diplomand der Auffassung, daß dieser eigentlich zu schade sei, um nur für das Informations- und Hilfesystem, den Bibliotheksbrowser als auch -manager verwendet zu werden. Zur Vollständigkeit würde hier der Domaincompiler fehlen, der nach wie vor direkt auf die Datenbank zugreift. Es wurde daher ein Beispiel programmiert, wie mittels Hochsprachencode ein Klient programmiert werden kann, der auf die in TCL implementierte RPC-Schnittstelle oder auch direkt auf die Sitzungsschicht des Servers zugreifen kann[1]. Die Einbindung dieser Schnittstelle in dem Domaincompiler würde die Client-Server Struktur der Klassenbibliothek vervollständigen.

Zu Testzwecken wurde ein Java-Applet programmiert und in der Klassenbibliothek gespeichert, um diese in den Hypertext einzubinden. Bei der Programmierung entstand die Vision, daß die ADTs der Klassenbibliothek auch für Java verwendet werden könnten, zumal sich Java-Quellcode nur geringfügig von C++ unterscheidet. Neben sicheren Zeigern und automatischer Speicherverwaltung, wie sie jetzt durch die Automem-Implementation realisiert wird, bietet Java auch den Vorteil völliger Portierbarkeit. Zudem könnten bei einer Erweiterung der Klassenbibliothek um die Zielsprache Java interaktive Beispiele zu VDM in das Informations- und Hilfesystem aufgenommen werden.

[1] Das Beispiel ist unter ./tools/g++vdmdbm/dp/iface im Entwicklungspaket der Klassenbibliothek vorhanden

Glossar

ADT

Abstrakter Datentyp, der vollständig und ausschließlich durch die Operationen definiert ist, die auf ihn anwendbar sind. Ein ADT kann auch als Klasse bezeichnet werden.

Archie

Archie ist ein Server, der eine Datenbank von anonymen FTP-Servern und deren Inhalte bereitstellt. Er wurde von der Archiv-Gruppe der McGill-Universität in Montreal entwickelt.

Browser

Programm, welches Inhalte interpretiert und darstellt. Ein HTML-Browser stellt HTML-Dokumente dar.

Client

Dienstnutzer, der auf das Angebot des Dienstanbieters zurückgreift.

DNS

Das Domain Name System ist ein verteiltes Datenbank-System, um Computernamen wie www.home.de in ihre numerische Internet-Adresse wie z.B. 193.22.3.1 zu wandeln und umgekehrt. Dem DNS ist es zu verdanken, daß das Internet benutzt werden kann, ohne sich eine Vielzahl von Nummern merken zu müssen.

DSL

Die Domain Specification Language DSL ist eine an META-IV angelehnte Sprache der Klassenbibliothek der FH-Wedel

EPS

Mit ProstScript hat der Hersteller Adobe eine druckerunabhängige Beschreibungssprache geschaffen, die sich als Standard durchgesetzt hat. Encapsulated Postscript (EPS) besteht aus Strukturkonventionen, die den eigentlichen Postscript-Anweisungen in Form eines Prologs und eines Skriptes vorangestellt werden. EPS-Formate sind die mehr oder minder die einzigste Sprache, welche die Satzsysteme TEXund LATEXzur Einbindung von Graphiken verstehen.

FAQ

Frequently Asked Questions. In der Regel geen Experten, nachdem Ihnen die gleiche Frage zum hunterstenmal gestellt wurde, eine Liste mit den Antworten heraus, die als FAQ bezeichnet wird. FAQs werden häufig in Newsgroups verteilt.

Firewall

Ein System, das das Rechnernetz eines Unternehmens oder einer Organisation vor externem Zugriff schützt. Die Organisation erlaubt z.B. den Zugang zum Internet über ein Proxy-System. Der Firewall kann auch über Bastion-Hosts (absolut sichere Hosts als Türsteher des Netzes) realisiert werden, die auch *doorkeeper* genannt werden.

GIF

Graphics Interchange Format, Dateiformat für Graphikdaten. Entwickelt durch CompuServe, Inc. Eines der am häufigsten verwendeten Grafikformate

Gopher

Hilfsprogramm, das Informationen im Internet in einer Menüstruktur organisiert. Gopher stellt die nötigen ftp Verbindungen selbsttätig her.

Home-Page

Der graphische Einstiegspunkt zu allen Informationen, die ein HTML-Server bereitstellt. Die Home-Page ist in der Regel eine Seite, die zahlreiche Informationen und Links zu weiterführenden Seiten beinhaltet.

HTML

Die HyperText Markup Language, eine Teilmenge von SGML, enthält die Sprachelemente für den Entwurf von Hypertext-Dokumenten.

HTTP

Das HyperText Transfer Protocol beschreibt einen festdefinierten Satz von Nachrichten und Antworten, mit denen ein Client und ein Server während einer HTML-Sitzung kommunizieren.

Inline-Image

Als Inline-Image bezeichnet man eine Grafik, die in einem Hypertext-Dokument dargestellt wird.

Invariante

Bedingung für die Wohlgeformtheit eines Objektes. Die Invariante läßt sich als Funktion auffassen, die einen logischen Wahrheitswert zurückliefert.

ISO

Die „International Standards Organization“ (oder auch „International Organization for Standardisation“), hat unter anderem ein Set von Netzprotokollen, die sogennanten ISO/OSI-Protokolle, entworfen. Sie sollten die derzeit gängigen Internet-Protokolle ersetzen, haben jedoch in der Praxis keinerlei Verbreitung gefunden.

LAN

Local Area Network. Eine Menge vernetzter Computer, die räumlich nicht allzu weit voneinander entfernt sind. In der Regel handelt es sich um einen Standort mit einem oder mehreren Gebäuden.

LaTeX

LaTeX ist ein generisches Satzsystem von der Sekretärin Leslie Lamport, das die Formatierungsprozeduren von TeX betreibt. Ein LaTeX-Guru ist eine Person, die alle LaTeX- und internen TeX-Funktionen in- und auswendig kennt. Der Autor dieser Diplomarbeit zählt nicht zu dieser Gruppe.

Linked-Image
Ein Linked Image ist eine Grafik, die nicht direkt im Hypertext-Dokument dargestellt wird, sondern in einer anderen Datei gespeichert ist. Die Graphik wird erst angezeigt, wenn der entsprechende Link angewählt wird.

MAN
Siehe WAN

MIME
Das Mulipurpose Internet Mail Extension Protocol MIME teilt verschiedene Datentypen in eine Anzahl von Haupt- und Untergruppen auf. Die Verwendung dieses Protokolls erlaubt HTML-Browsern, die unterschiedlichsten Dateiformate zu erkennen und zu verarbeiten. Die Hauptgruppe definiert, um welchen Datentyp es sich handelt, also z.B. Grafik, Audio oder Video, die Untergruppe beschreibt den Typ genauer. Ferner ist mit MIME eine zeichensatzübergreifende Codierung von Binärdaten definiert.

OSI-Model
Open System Interconnection Model, Standard für die Kommunikation offener Systeme

Provider
Im Zusammenhang mit dem Internet, Unternehmen oder Organisation, die Anbindungen an das Internet anbietet.

Proxy-Server
Ein Server im Internet, der Systemen den indirekten Zugang zum Netz ermöglicht. Neben Sicherheitsaspekten werden Proxy-Server auch zum zwischenspeichern (cachen) von Daten verwendet.

RFC
RFC steht für „Request for Comments“, also „Aufforderung zu Kommentaren“. Dieser Name ist etwas verwirrend, denn es handelt sich hier um eine Reihe von Dokumenten, welche die Internet-Standards, vorgeschlagene neue Standards und allgemeine, akzeptierte Ideen beschreiben und dokumentieren. Mit RFCs wurden ursprünglich fast alle TCP/IP betreffenden Standards angenommen und etabliert[2].

Server
Dienstanbieter, der eine Service für mehrere Dienstnutzer bereitstellt

SGML
Die Standard Generalized Markup Language ist eine Sprache zur Beschreibung von Dokumenten. Beschrieben wird aber nicht das Aussehen dieser Dokumente, sondern deren Strukur mit Hierarchien und unterschiedlichen Komponenten.

Tags
In HTML werden über Tags die Struktur sowie die Darstellung von Informationen festgelegt.

[2][KW95, S. 133] Washburn und Evans haben RFCs ein ganzes Kapitel gewidmet

TeX

TeX ist ein Satzsystem von Donald E. Knuth, das als ein solches die hohe ästhetische Qualität der Satzkunst beherscht. Die kryptischen und unzähligen Anweisungen von TeX sind ungefähr genauso komfortabel wie die Druckmaschine von Gutenberg.

TCL

TCL (sprich: tickel) steht für Tool Command Language und ist eine in der Hochsprache C erstellte Interpretersprache die sich durch Portabilität auszeichnet. Siehe auch TK.

TCP/IP

Transmission Control Protocol/Internet Protocol. Verbindungsorientiertes Protokoll der Transportschicht, auf dem neben UDP im wesentlichen das Internet basiert.

Telnet

Terminal-Emulator Protokoll, daß die Fernanmeldung an UNIX-Systemen über das Netzwerk erlaubt.

TK

TK ist die Abkürzung für Toolkit und eine graphische Erweiterung von TCL für das X (als auch Microsofts) Windows System.

UDP

User Datagram Protocol. Im Gegensatz zu TCP ein verbindungsunabhängiges Protokoll der Transportschicht, daß sogenannte Datagramme versendet und empfängt.

URL

Ein „Uniform Resource Locator“ ist die Adresse eine Dokuments im World Wide Web. Die Adresse ist Teil des Links auf ein anderes Dokument und wird von der Client-Software entsprechend interpretiert, um die Verbindung zum entsprechenden Server aufzubauen.

UUCP

UNIX to UNIX Copy, UNIX Standardsoftware zur Datenfernübertragung

VDL

Vienna Definition Language. Eine operationale Semantik, die von einer Wiener Forschergruppe verwendet wurde, um die PL/I Programmiersprache zu definieren

VDM

Vienna Development Method. Systematische Softwareentwicklungsmethode mit formalen Methoden

WAIS

Mit WAIS können hunderte Datenbanken und Bibliotheken im Internet mit einer organisierten Struktur durchsucht werden. WAIS untersucht Inhalte und Titel von Dokumenten.

WAN

World Area Network. Überregionales Netz mit Fernverbindungen. Ein Netzwerk das sich über mehr als eine Stadt erstreckt, welches als Metropolitan Area Network (MAN) bezeichnet wird.

WWW

World Wide Web. System für die Suche nach und Zugriff auf Internet-Resourcen

X Windows System

Ein netzbasiertes Windows-System, das ursprünglich am Massachusetts Institute of Technologie (MIT) entwickelt wurde. X-Windows oder kurz X wird in der Regel auf UNIX-Systemen eingesetzt. Der Name X stammt von der Firma Xerox, die bei der Entwicklung von X-Windows genauso maßgeblich war (Gleiches gilt für das Netzwerkprotokoll XNS).

Literaturverzeichnis

[3C90] 3COM Corporation *NDIS Network Driver Interface Specification Version 2.01 FINAL* 3COM/Microsoft LAN Manager 3COM, October 1990

[3C94] 3COM Corporation *Etherlink III Parallel Tasking ISA, EISA, Micro Channel and PCMCIA* Adapter Drivers Reference 3COM, August 1994

[3CA92] 3COM Corporation *Sample Driver Source Code* Etherlink Adapters 3COM, April 1992

[3CB92] 3COM Corporation *Sample Protocol Source Code* Etherlink Adapters 3COM, April 1992

[AA92] Aho, Alfred v. Ravi Sethi Jeffrey D. Ullman *Compilerbau Teil 1* Addison Wesley Deutschland, 1992 ISBN 3-89319-150-x

[AA92] Aho, Alfred v. Ravi Sethi Jeffrey D. Ullman *Compilerbau Teil 2* Addison Wesley Deutschland, 1992 ISBN 3-89319-151-8

[AD91] Diller, Antoni: *Z* , An introduction to formal methods John Wiley & Sons, Inc., 1991 ISBN 0-471-92489

[AI91] Andrews, Derek Ince, Darrel: *Practical formal methods with VDM*, London: McGraw-Hill, 1991 ISBN 0-07-707214-6

[AT92] Tanenbaum, Andrew S.: *Computer-Netzwerke*, Attenkirchen: Wolfram's Fachverlag, 1992 ISBN 3-925328-79-3

[BW95] Welch, Brent B.: *Practical Programming in Tcl and Tk*, New York: Prentice Hall, 1995 ISBN 0-13-182007-9

[DB82] Bjøerner, Dines / Jones, Cliff B., 1982: *Formal speciafication and software development*, Prentice Hall Int., Inc., 1982 ISBN 0-13-329003-4

[DC95] Chandler, David M.: *Running a Perfect Web Site*, Indianapolis: Que Corp., 1995 ISBN 0-7897-0210-X

[DD92] Dougherty, Dale: *sed & awk*, O'Reilly & Associates, Inc., 1992 ISBN 0-937175-59-5

[GB92] Born, Günter: *Referenzhandbuch Dateiformate*, Grafik, Text, Datenbanken, Tabellenkalkulation Bonn: Addison-Wesley, 1992 ISBN 3-89319-446-0

[GL95] Lehey, Greg: *Porting UNIX Software*, Sebastopol, CA: O'Reilly & Associates, Inc., 1995 ISBN 1-56592-126-7

[GS87] Sage, Russel G.: *Tricks of the UNIX Masters*, Indianapolis, Indiana: The Waite Group, Howard W. SAMS & Company, 1987 ISBN 0-672-22449-6

[HH95] Hörcher, Hans-Martin: *The VDM Class Library Generator: Reference Guide*, Fachhochschule Wedel

[IB69] IBM *Einführung in die PL/I-Sprache*, Lehrtext IBM Deutschland, 1969 IBM Form 79967-1

[IB70] IBM *System-Literatur* IBM System /360 Programmierhandbuch PL/I SUBSET IBM Form C12-1000-0

[IG95] Graham, Ian S.: *HTML Source Book*, A Complete Guide to HTML John Wiley & Sons, Inc., 1995 ISBN 0-471-11849-4

[JN90] Nicholls, J.E.: *Z User Workshop, Oxford 1990*, Proceeding of the Fifth Annual Z User Meeting, Oxford Library of Congress Cataloging-in-Publication Data, 1990 ISBN 0-387-19672-2

[JM95] Mohr, Jan Peter: *Benutzerhandbuch für abstrakte Datentypen am Beispiel von VDM*, Fachhochschule Wedel

[JN95] Jones, Russ Nye, Adrian: *HTML und das World Wide Web*, Bonn: O' Reilly / International Thompson Verlag, 1995 / O'Reilly & Associates, Inc., 1995 ISBN 3-930673-34-7

[JS86] Strang, John: *Programming with curses*, O'Reilly & Associates, Inc., 1986 ISBN 0-937175-02-1

[JS92] Spivey, J.M.: *The Z Notation*, A Reference Manual UK: Prentice Hall Int., Ltd., 1992 ISBN 0-13-978529-9

[JW95] Wulf, Jens: *Erweiterung des VDM DSL-Compilers um ADT-Klasse BAG*, Fachhochschule Wedel

[KW95] Washburn, Kevin Evans, Jim: *TCP/IP*, Bonn: Addison-Wesley, 1995 ISBN 3-89319-658-7

[LB95] Goosens, Michel Mittelbach, Frank: Samarin, Alexander *Der LaTeX Begleiter*, Bonn: Addison Wesley, 1995 / New York: Addison Wesley (Original title: LaTeX Companion), 1994 ISBN 3-89319-646-3

[MM92] Mühlhäuser, Prof. Dr. Max Schill, Dr. Alexander: *Software Engineering für verteilte Anwendungen*, Berlin: Springer Verlag, 1992 ISBN 3-54055412-2

[MK93] Kuhn, Michael: *Informationssystem für die VDM-Class Library*, Fachhochschule Wedel, 1993

[MS90] Santifaller, Michael : *TCP/IP und NFS in Theorie und Praxis: Unix in lokalen Netzen* , Bonn: Addison Wesley, 1990 ISBN 3-89319-246-8

[RS92] Stevens, W. Richard: *Programmieren von UNIX-Netzen*, Informationstechnik / Nachrichtentechnik, Wien: Hanser, 1992 / London: Prentice Hall, 1990 ISBN 3-446-16318-2

[RSE92] Stevens, W. Richard: *UNIX Network Programming*, London: Prentice Hall, 1990 ISBN 0-13-949876-1

[RS94] L.Schwartz, Randal / Foreword by Larry Wall: *Learning Perl*, CA: UNIX Programming, O'Reilly & Associates, Inc., 1994 ISBN 1-56592-042-2

[RS95] Stevens, W. Richard: *Programmierung in der Unix-Umgebung* , Die Referenz für Fortgeschrittene, Professional Computing Series Bonn: Addison Wesley, 1995 ISBN 3-89319-814-8

[RW93] Wonneberger, Dr. Reinhard: *Kompaktführer LaTeX* , Bonn: Addison Wesley, 1993 ISBN 3-89319-589-0

[SI92] Schmidt, Uwe, 1992: *Software-Entwicklungs-Methoden*, Vorlesung im Sommersemester 1992 an der FH-Wedel

[SM89] Mikes, Steven: *X Window System Technical Reference*, New York: Addison Wesley, 1989 ISBN 0-201-52370-1

[TS95] Savola, Tom: *Using HTML*, Special Edition Indianapolis: Que Corp., 1995 ISBN 07897-0236-3

Index

Danksagung

Ich möchte mich Bedanken bei

Prof. Dr. Uwe Schmidt für die freundliche und kreative Betreuung der Diplomarbeit

Kei Thomsen für seine Geduld, mir das tcldp auf der Sparc zu installieren

Joachim Schlossarek für den Zugang zu seiner Bibliothek

Reimer Volkers für seine sauberen Mitschriften und

Michael Boguhn für die Bereitstellung seines Fuhrparks

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides Statt, daß ich die vorliegende Arbeit selbständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

Ort, Datum | Unterschrift