View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Qucosa - Publikationsserver der Universität Leipzig

Universität Leipzig
Fakultät für Mathematik und Informatik
Institut für Informatik

Konzeption und Entwicklung eines plattformunabhängigen Tools zum netzwerkweiten Daten–Management und anwendungsspezifische Erprobung

Diplomarbeit

Leipzig, September, 1999

vorgelegt von
Sven Trautmann

Verantwortlicher HSL: Prof. Dr. K. Irmscher
Betreuer: Dr. K. Hänßgen
(Universität Leipzig)
Dr. H. Hayd
(Max–Planck–Institut für
neuropsychologische Forschung, Leipzig)

Inhaltsverzeichnis

Kapitel 1

Einführung

Das Max–Planck–Institut für Neuropsychologische Forschung in Leipzig beschäftigt sich mit der interdisziplinären Forschung an kognitiven Prozessen (z.B. mit Zusammenhängen zwischen Sprache und Gedächtnis) und deren Repräsentation im Gehirn.
Für diese Zwecke verfügt das Institut über mehrere Großgeräte zur Messung der Hirnaktivitäten: einen 3–Tesla–Magnet–Resonanz–Tomograph (MRT), einen 150–Kanal–Magnetencephalograph (MEG) sowie verschiedene Mehrkanal–EEG–Geräte.
Mit Hilfe dieser Geräte wird eine große Menge an Daten gewonnen. So beträgt etwa die Größe eines einzelnen MRT–Datensatzes für einen menschlichen Kopf bereits über 600 Megabyte.
Die so gewonnenen Daten sind noch nicht aussagekräftig und müssen weiterverarbeitet werden. Um z.B. eine bestimmte Sinneswahrnehmung einer Gehirn–Region zuordnen zu können, müssen die MRT–Daten mit MEG–Daten korreliert werden.
Die Software zur Ansteuerung der Großgeräte läuft nur auf bestimmten Hard– und Softwarearchitekturen. Das hat zur Folge, daß die Rohdaten mittels geeigneter Protokolle auf andere Rechner übertragen werden müssen.
Die Programme zur Weiterverarbeitung der Daten bieten meist nur eine sehr eingeschränkte Nutzerfreundlichkeit und liegen auf mehreren Rechnern verteilt vor.
Nachdem die Daten verarbeitet und ausgewertet wurden, müssen sie archiviert werden. Zu diesem Zwecke stehen CD–ROMs und Magnetbänder zur Verfügung.
Ähnlich Szenarien finden sich auch bei anderen Einrichtungen. Deshalb wird

im Rahmen dieser Arbeit ein Konzept entwickelt, welches den geschilderten Problemen Rechnung trägt. Dazu wird es notwendig:

- eine Situationsanalyse für das Daten–Management vorzunehmen,
- den Datenfluss beim Daten–Management in heterogenen Netzwerken zu analysieren,
- die Aufgaben eines Daten–Management–Systems zu bestimmen,
- benutzbare Standards zu vergleichen,
- alternative Verfahren zu diskutieren,
- eine Konzeption für ein Daten–Management–System zu entwickeln,
- die Konzetion in ein lauffähiges Programm umzusetzen und
- dieses an Beispielen zu erproben.

Das Netzwerk im Max–Planck–Institut für neuropsychologische Forschung ist, wie viele andere Rechnernetze, heterogen. Es kommen vier Rechner–Architekturen mit insgesamt 13 verschiedenen Betriebssystemen zum Einsatz. Deshalb ist es wichtig, daß die zu entwickelnde Konzeption weitgehend plattformunabhängig ist.
Deshalb wird ein weiteres Augenmerk auf der Untersuchung der Eignung der plattformunabhängigen Programmiersprache *JAVA* für den Einsatz in größeren Software–Projekten liegen.
Die vorliegende Arbeit gliedert sich in 5 Teile:

- Voraussetzungen für das Daten–Management
- Bestimmung der Anforderungen und Entwicklung einer Konzeption
- Implementierung der Konzeption
- Erprobung der Implementierung
- Zusammenfassung und Ausblick

Im Kapitel "Voraussetzungen für das Daten–Management" wird auf einige Grundlagen in heterogenen Netzwerken eingegangen. Dazu gehören Betrachtungen der *UNIX*–Betriebssystem–Familie, der Netzwerk–Protokoll–Suite *TCP/IP*, des Network–Information–System *NIS*, der Datei–Übertragungs–Protokolle *NFS* und *ftp*, der *remote–login*–Protokolle *rlogin* und *telnet* sowie der Programmiersprache *JAVA*.
Das Kapitel "Bestimmung der Anforderungen und Entwicklung einer Konzeption" beschäftig sich mit einer Begriffsklärung für ein Daten–Management–System und einer Situationsanalyse des bisherigen Daten–Managements. Daraus werden die benötigten Verfahren abgeleitet und auf ihre Tauglichkeit hin verglichen. Ein weiterer Punkt dieses Kapitels ist die

Analyse der Verarbeitungsmöglichkeiten und die Konzeption einer grafischen Benutzeroberfläche.
Das Kapitel "Implementierung des Konzeptes" zeigt die Umsetzung der in der Konzeption gefundenen Ergebnisse sowie die Implementierung der diskutierten Protokolle und Verfahren.
Im Kapitel Erprobung stehen der Einsatz und die Anpassung des geschaffenen Programmes zum Daten–Management im Max–Planck–Institut für neuropsychologische Forschung zur Diskussion.
Die Zusammenfassung zeigt neben den Ergebnissen der Arbeit einen Ausblick auf weitere Möglichkeiten zu Verbesserungen im Daten–Management auf.

Kapitel 2

Voraussetzungen für das Daten–Management

Dieses Kapitel beschäftigt sich mit Standards und Protokollen in heterogenen Netzwerken und geht dabei auf die *UNIX*–Betriebssystemfamilie sowie auf einige Netzwerk–Protokolle der *TCP/IP*–Suite ein. Den Abschluss dieses Kapitels bildet die Vorstellung der Programmiersprache *JAVA*.

2.1 Die *UNIX*–Betriebssystem–Familie

UNIX ist Urahn und Musterbeispiel eines netzwerkfähigen Betriebssystems. Viele Verfahren und Protokolle, welche heute in Netzwerken verwendet werden, wurden innerhalb der *UNIX*–Betriebssysteme erstmals eingesetzt und haben sich durchgesetzt. Deshalb wird auf die *UNIX*–Betriebssysteme eingegangen, um deren Besonderheiten aufzuzeigen. Die *UNIX* Betriebssysteme sind *Multi–User*– und *Multi–Tasking*–Betriebssysteme. Sie erlauben mehreren Benutzern gleichzeitig auf einem Rechner mit mehreren Programmen zu arbeiten. Da *UNIX* keine proprietären Protokolle benutzt und kooperativ ist, empfiehlt es sich für den Einsatz in heterogenen Netzwerken.
Die Geschichte von *UNIX* beginnt im Jahre 1969. Damals wurde *UNIX* von Ken Thompson in den *Bell Laboratories* entwickelt. Der Quellkode des *UNIX*–Betriebssystems wurde kostenlos an Universitäten und Hochschulen verteilt. Dort wurde es vielen zugänglich, die das System verbesserten. Dabei entstand an der Universität von Berkeley ein *UNIX*–Variante mit dem Namen *BSD* (*Berkeley Software Distribution*). Seitdem hat sich eine Viel-

zahl von *UNIX*–Derivaten entwickelt. So hat beinahe jeder Hersteller von UNIX–Workstations sein eigenes Derivat entwickelt (z.B. *IBM — AIX*, *SUN — Solaris*, *HP — HP–UX* usw.). Diese unterschiedlichen *UNIX*e unterliegen jedoch alle einem gemeinsamen Standard, dem *X/OPEN Portability Guide* (*XPG*).
In den letzten Jahren hat sich eine weitere und freie *UNIX*–Implementierung (*Linux*) durchsetzen können. *Linux* wurde von dem Finnen Linus Torvalds für *Intel*–PCs entworfen und wird von einer großen Programmierergemeinde im Internet weiterentwickelt. Der Vorteil für den Anwender ist, daß Linux kostenlos aus dem Internet geladen werden kann und im Quelltext vorhanden ist, somit an die Bedürfnisse der Benutzer angepaßt werden kann und Fehler innerhalb des Betriebssystem schon kurz nach ihrer Entdeckung beseitigt werden können. Inzwischen gibt es *Linux* auch für Hardware–Plattformen mit *Alpha*–, *Sparc*–, *Mips*– und anderen Prozessoren.

2.1.1 Der Mechanismus der Autorisierung

Bei einem *Multi–User*–Betriebssystem muß sich der Benutzer dem System gegenüber autorisieren. Jeder Benutzer hat einen Benutzernamen sowie ein Passwort, mit denen er sich beim System ausweist.
Jedem Benutzer wird mindestens eine Gruppe zugeordnet. Anhand der Gruppen kann das Betriebssystem entscheiden, ob bestimmte Bereiche des Dateisystems oder bestimmte Programme einem Benutzer zugänglich gemacht werden oder vor diesen geschützt werden sollen/müssen. Nicht alle Benutzer des *UNIX*–Betriebssystems haben die gleichen Rechte. Deshalb ist es nötig, daß ein sicheres Verfahren zum Verschlüsseln der Passwörter implementiert ist. Das Verfahren zum Verschlüsseln des Passworts beruht auf dem DES–Algorithmus (*Data Encryption Standard*). Bei diesem wird aus dem eingegebenen Passwort und einigen Informationen aus dem verschlüsselten Passwort eine Zeichenkette erzeugt, die mit dem verschlüsselten Passwort verglichen wird. Das Verschlüsseln des Passworts ist ein Einweg–Algorithmus, d.h. es ist leicht möglich aus dem Klartext das verschlüsselte Passwort zu erzeugen, jedoch ist es nur mit sehr hohem Rechenaufwand möglich aus dem verschlüsselten Passwort den Klartext zu bestimmen. Nähere Information zum DES–Algorithmus und zur *UNIX*–Passwort–Verschlüsselung sind z.B. in [Ape] zu finden.

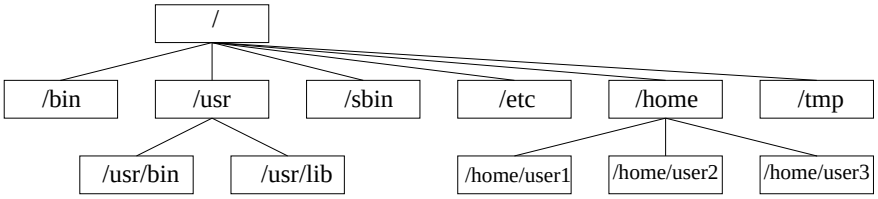

Abbildung 2.1: Die Struktur eines *UNIX*–Dateisystems

2.1.2 Das *UNIX*–Dateisystem

Bei einem *UNIX*–Dateisystem finden sich alle Dateien und Verzeichnisse in einer Baum–Struktur zusammen. Die Wurzel bildet das sogenannte root–Verzeichnis */*. Von diesem aus verzweigen sich die Verzeichnisse weiter. Dies verdeutlicht Abbildung 2.1. Zur Vereinheitlichung der unterschiedlichen *UNIX*–Dateisysteme wurde der *Filesystem Hierarchy Standard* ([Qui]) geschaffen. Die Verzeichnisse nach dem *Filesystem Hierarchy Standard* zeigt Tabelle 2.1 In das *UNIX*–Dateisystem kann man mittels des Befehls `mount` andere Dateisysteme (z.B. CDROM–Dateisysteme oder Netzwerk–Filesysteme *NFS*) "einhängen" (*mounten*) und dann darauf zugreifen, als wären alle Daten auf einem lokalen Datenträger.

2.1.3 Die Vergabe der Zugriffsrechte für Dateien

Jede Datei hat im *UNIX*–Betriebssystem einen Besitzer. Dieser entscheidet, wer die Datei benutzen darf. Dafür gibt es drei Benutzerklassen:

- der Benutzer selbst,
- die Gruppe des Benutzers,
- alle anderen.

Für jede der drei Gruppen kann der Besitzer drei Zugriffsrechte vergeben. Diese Rechte sind:

- Lesen,
- Schreiben und
- Ausführen.

So kann der Benutzer zwar das Lesen seiner Daten für alle Benutzer erlauben, aber das Verändern oder Löschen durch andere Benutzer verhindern. Welche Rechte für eine Datei vergeben wurden, zeigt der *UNIX*–Befehl `ls -l`.

Verzeichnis	Bedeutung
/bin	grundlegende ausführbare Programme
/boot	statische Dateien für den *boot loader*
/dev	Gerätedateien
/etc	Konfigurationsdateien
/home	Nutzerdaten
	jedem Nutzer steht ein Verzeichnis zur Verfügung
/lib	für Programm–Bibliotheken
/mnt	Mount–Point
	Einhängpunkt für temporäre Dateisysteme
/opt	zusätzliche Software–Pakete
/root	Home–Verzeichnis des Systemadministrators (`root`)
/sbin	grundlegende System–Verwaltungs–Programme
/tmp	temporäre Daten
/usr	zweite Hierarchie–stufe
/var	variable Daten

Tabelle 2.1: Die Verzeichnisse eines *UNIX*–System nach dem *Filesystem Hierarchy Standard*

```
drwxr-xr-x 6 trautman users 4096 Jan 30 11:48 program
```

Die ersten zehn Zeichen geben über die Datei–Zugriffsrechte Auskunft.

1. Dateityp `l` = link, `d` = Verzeichnis, `-` = normale Datei
2.-4. Leserechte `r`, Schreibrechte `w` und Ausführrechte für den Besitzer der Datei `x`
5.-7. Leserechte, Schreibrechte und Ausführrechte für die Gruppe des Besitzers
8.-10. Leserechte, Schreibrechte und Ausführrechte für alle anderen

Steht an der jeweiligen Stelle ein "`-`" so ist die betreffende Berechtigung nicht gesetzt.

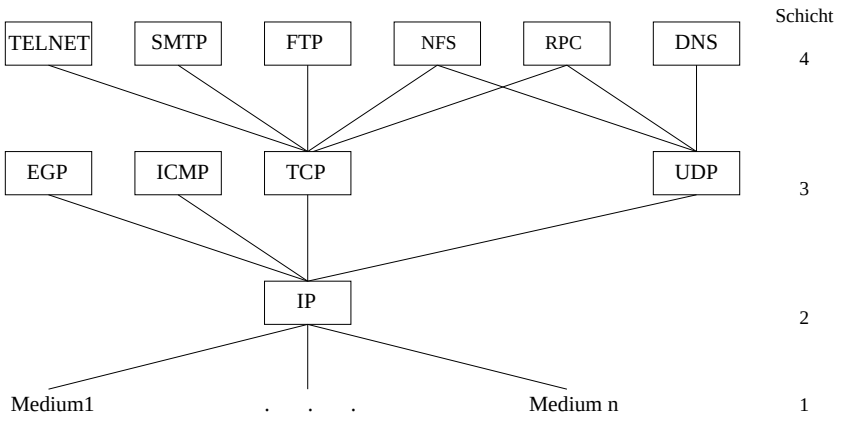

Abbildung 2.2: Ausschnitt aus der *TCP/IP*–Protokoll–Hierarchie

2.2 Die Netzwerk–Protokolle

Für die Kommunikation in heterogenen Netzwerken sind Standards erforderlich. Den bedeutendsten dieser Standards bildet die *TCP/IP*–Protokoll–Suite.

2.2.1 Die *TCP/IP*–Protokoll–Suite

Bis zum Jahre 1978 gab es keine normierten Übertragungsverfahren im Internet bzw. dessen Vorgänger dem *ARPANET*. Jeder Programmierer verwendete für seine Programme das Protokoll, welches ihm am besten geeignet erschien. Um einen einheitlichen Standard zu schaffen, wurden die Kommunikationsprotokolle *Transmission Control Protocol* (*TCP*) (Vgl. [Pos81b]), *User Datagram Protocol* (*UDP* (Vgl. [Pos80] und *Internet Protocol* (*IP*) (Vgl. [Pos81a]) geschaffen. Damit wurden erstmalig standardisierte und offene Übertragungsprotokolle spezifiziert. *TCP/IP* wurde zum Standard für alle Netzwerkknoten im Internet. *TCP/IP* wurde in *BSD* (*Berkeley Software Distribution*), einem in Berkeley geschaffenen *UNIX*–Abart, integriert. Durch die große Verbreitung von *BSD*, besonders an Universitäten, setzt sich *TCP/IP* durch.
Die beiden Protokolle *TCP* und *IP* stellen die Verbindung zwischen Anwendungsprogrammen (Schicht 4) und den verschiedenen Arten von Netz–Hardware (Schicht 1) her. Abbildung 2.2 zeigt einen Ausschnitt aus der *TCP/IP*–Protokoll–Hierarchie. Die *TCP/IP*–Protokoll–Hierarchie gliedert

sich, im Gegenteil zum *OSI*–Schichtenmodell in vier Schichten. Diese Schichten sind:

1. *link layer* — Verbindungsschicht
   Die Verbindungsschicht wird auch manchmal Daten–Verbindungsschicht (*data–link–layer*) oder Netzwerk–Schicht (*network–layer*) genannt. Diese Schicht beinhaltet die Netzwerkkarte und Programme, die direkt auf die Netzwerkkarte zugreifen (z.B. Gerätetreiber innerhalb des Betriebssystems). Hier werden alle Details der Hardware behandelt z.B. welche Schnittstelle oder welches Übertragungsmedium benutzt werden.
2. *network layer* — Netzwerk–Schicht
   Die Netzwerk–Schicht (manchmal auch Internet–Schicht (*internet layer*) genannt) übernimmt die Bewegung der Pakete über das Netzwerk. Hier findet z.B. das *Routing* der Pakete (Vgl. 2.2.3) statt. Innerhalb der *TCP/IP* Protokoll Suite stellt die Netzwerk–Schicht die Protokolle *IP* (*Internet Protocol*), *ICMP* (*Internet Control Message Protocol*) und *IGMP* (*Internet Group Management Protocol*) bereit.
3. *transport layer* — Transport–Schicht
   Die Transport–Schicht stellt den Datenfluß zwischen zwei Rechnern für die Applikations–Schicht bereit. Innerhalb der *TCP/IP–Protokoll–Suite* gibt es zwei völlig verschiedene Protokolle, das *Transmission–Control–Protocol* (*TCP*) und das *User–Datagram–Protocol* (*UDP*).
   TCP bietet einen verläßlichen Datenfluß zwischen zwei Rechnern und sorgt dafür, daß die Daten der Applikations–Schicht in für die Netzwerk–Schicht passende Pakete aufgeteilt werden. Außerdem wird dafür gesorgt, daß empfangene Pakete bestätigt und bei gestörter Verbindung auf verlorene Pakete gewartet wird. Durch diesen verläßlichen Datenfluß braucht die Applikations–Schicht bei Benutzung des *TCP*–Protokolls keine Maßnahmen zur Sicherung der Datenübertragung übernehmen.
   UDP dagegen, bietet der Applikations–Schicht einen weitaus einfacheren Service an. Es sendet nur Pakete, sogenannte *UDP*–Datagramme, von einem Rechner zum anderen. Dabei gibt es keinerlei Garantie, daß die Datagramme auch den anderen Rechner erreichen. Deshalb muß jede Kontrolle der Datenübertragung von der Applikations–Schicht übernommen werden.
4. *application layer* — Applikations–Schicht
   Die Applikations–Schicht übernimmt die Kopplung der Applikationen

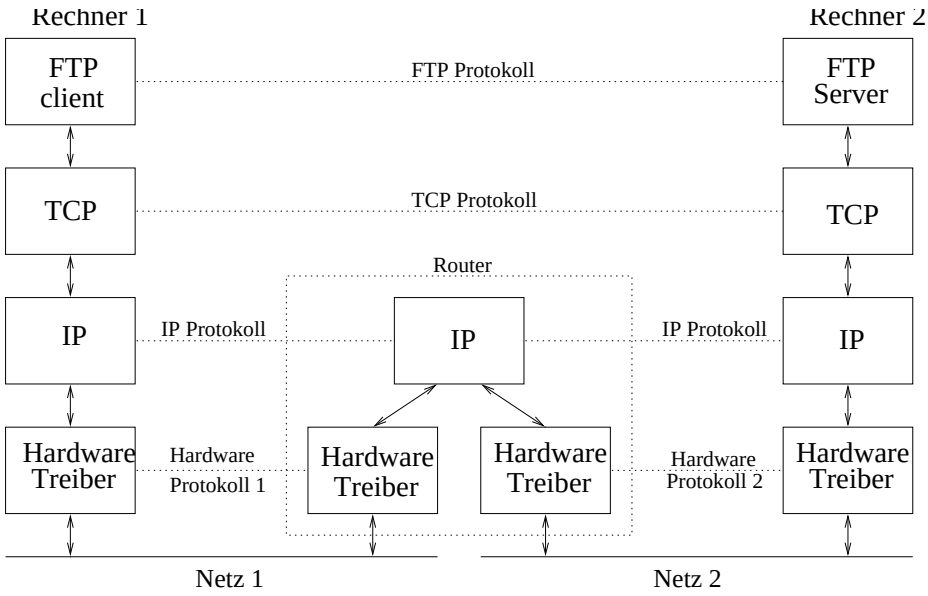

Abbildung 2.3: Beispiel für eine *ftp*–Verbindung zwischen zwei Rechnern in unterschiedlichen Netzen

mit dem Netzwerk. Es gibt viele *TCP/IP*–Applikationen, die fast alle Betriebssysteme anbieten. Beispiele sind:

- *telnet* oder *remote login* (vgl. Kapitel 2.5.1)
- *ftp*, das *File Transfer Protocol* (vgl. Kapitel 2.4.2)
- *SMTP*, das *Simple Mail Transfer Protocol*

Abbildung 2.3 zeigt eine *ftp*–Verbindung zwischen zwei Rechnern in unterschiedlichen Netzen, die durch einen *Router* verbunden sind. Der *Router* wird benutzt, um verschiedene Netze miteinander zu verbinden. Die Daten werden z.B. vom *ftp*–Client Programm über die darunterliegenden Protokolle *TCP*, *IP* und den Hardware–Treiber auf Netz 1 gesendet. Wie dies genau vor sich geht, wird in den Abschnitten 2.2.2 bis 2.2.4 beschrieben. Der Hardware–Treiber des *Routers* nimmt die Daten entgegen und sendet, weil sich der Ziel–Rechner nicht im eigenen Netz befindet, die Daten weiter über den zweiten Hardware–Treiber in das zweite Netz. Dort gelangen die Daten über Hardware–Treiber, *IP* und *TCP* zum *ftp*–Server–Programm.

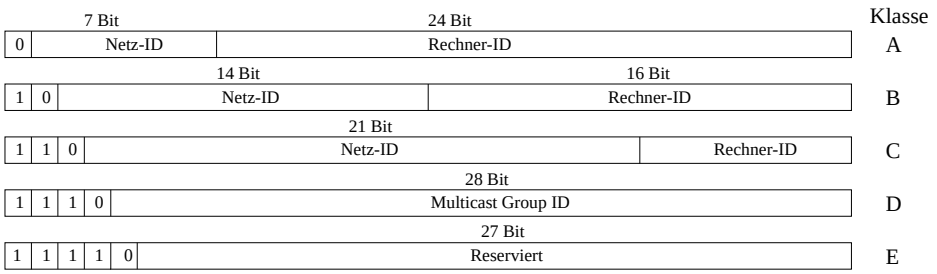

Abbildung 2.4: Die fünf Klassen von Internet–Adressen

Klasse	Teilnetze	Rechner pro Teilnetz
A	128	16777216
B	16256	64516
C	2064512	254

Tabelle 2.2: Anzahl der Teilnetze und Rechner pro IP–Klassen

Internet–Adressen

Jede Netzwerkkarte in einem Teilnetz des Internets muß zur Identifizierung eine eindeutige sogenannte Internet–Adresse haben. Diese Adressen sind 32–Bit Zahlen, die üblicherweise in der *dottet–decimal*–Notation dargestellt werden. Dabei werden die Internet–Adressen als vier, durch Punkte getrennte Zahlen zwischen 0 und 255 dargestellt, die jeweils einem Byte entsprechen, z.B. 194.94.218.45. Bei den Internet–Adressen werden fünf Klassen unterschieden (Vgl. Tabelle 2.4). Es werden jedoch nur die Klassen A,B und C als Rechneradressen verwendet.
Die Klasse D wird für *IP–Multicasts* benutzt. Die Klasse E wird ausschließlich für experimentelle Zwecke genutzt.
Weil jede Netzwerkkarte eine eindeutige IP–Adresse haben muß, gibt es eine zentrale Institution, die die Netzwerk–Adressen vergibt. Diese ist das *Internet Network Information Center*, auch *InterNIC* genannt. Das *InterNIC* vergibt ausschließlich die Netzwerk–IDs. Die Vergabe der Rechner–IDs liegt in der Verantwortung der Netzwerk–Administratoren der einzelnen Teil–Netze.

Domain Name System

Obwohl die Netzwerkkarten in einem Rechner durch ihre IP–Adressen eindeutig bestimmt sind, ist es für menschliche Benutzer einfacher, einen Rechner bzw. dessen Netzwerk–Interface mit einem Namen anstelle der vier Bytes anzusprechen. (Zu beachten ist, daß ein Rechner mit mehreren Netzwerk–Interfaces auch über mehrere Namen angesprochen werden kann und daß einem Netzwerk–Interface ebenfalls mehrere Namen zugeordnet werden können.) Deshalb gibt es eine Datenbank (*Domain Name System*), die für die Zuordnung von *IP*–Adresse und Namen verantwortlich ist. Die Datenbank für die Netzwerk–Namen (*domain names*) wird ebenfalls vom *InterNIC* verwaltet.

Verpacken der Daten

Wenn eine Applikation Daten mittels *TCP* übertragen will, sendet sie die Daten durch den Protokoll–Stapel (*stack*), bis sie als Bitstrom über das Netzwerk fließen. Beim Passieren jeder Schicht im Protokoll–Stapel werden die Daten mit einem Kopf (*Header*), manchmal auch mit einem Schwanz versehen. Ein Beispiel für solch einen Prozeß zeigt Abbildung 2.5.
Zuerst werden die Daten in Segmente aufgeteilt. Jedes der so entstandenen Segmente wird in der *TCP*–Schicht mit einem *TCP*–Header versehen. Dieser beinhaltet Informationen über Zielportnummer, Quellportnummer, Sequenznummer, Windowgrösse u.a. (Vgl. Kapitel 2.2.4).
Dieses Paket wird dann an die nächste Schicht im Protokoll–Stapel, das *IP*–Protokoll, weitergereicht. Dort wird das neue Paket wiederum mit einem Header versehen. Dieser beinhaltet unter anderem die Ziel–*IP*–Adresse, die Quell–*IP*–Adresse, die Protokollnummer und eine Prüfsumme. (Vgl. Kapitel 2.2.3)
Danach wird das so entstandene *IP–Datagram* an die darunterliegende Netzwerk–Schicht weitergegeben.
Die Netzwerk–Schicht kann verschiedene Protokolle unterstützen. In Abbildung 2.5 wird in der Netzwerk–Schicht das *Ethernet*–Protokoll benutzt. Beim *Ethernet*–Protokoll werden die Datagramme wiederum mit einem *Header* versehen, desweiteren wird noch eine Prüfsumme an das Datagram angehängt. Der *Ethernet–Header* enthält: *Ethernet*–Ziel–Adresse, *Ethernet*–Quell–Adresse und einen Typ–Code. Die so verpackten Datagramme werden dann in das Netz abgeschickt und vom Empfänger wieder ausgepackt und in der richtigen Reihenfolge zusammengesetzt.

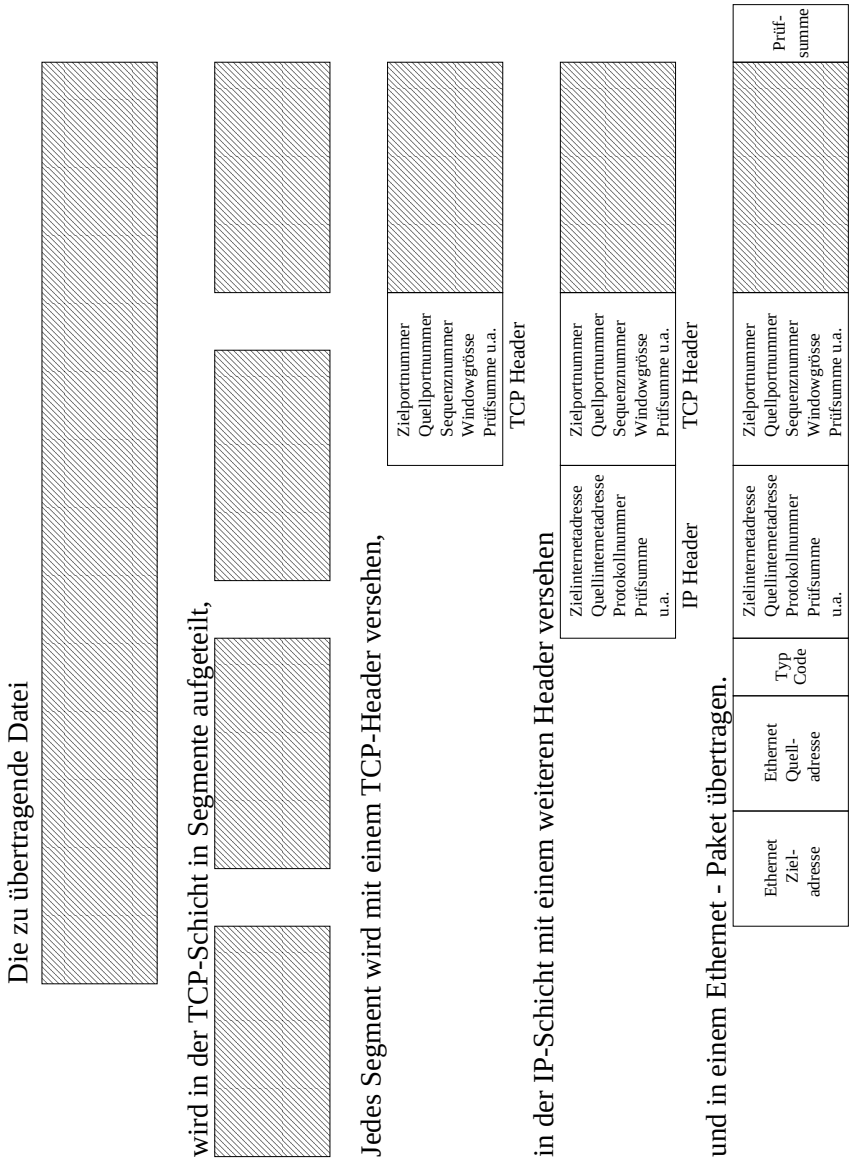

Abbildung 2.5: Die Verpackung der Daten im Protokoll–Stapel (Quelle: [Spr])

Port Nummern

Die Protokolle *TCP* und *UDP* benutzen 16–Bit–Port–Nummern. Jede auf die *TCP*– und *UDP*–Protokolle aufbauende Anwendung benutzt eine eigene Port–Nummer. Damit werden die Pakete unterscheidbar und jede Anwendung erhält nur die für sie bestimmten Pakete. So hat in jeder *TCP/IP*–Implementierung z.B. das *ftp*–Protokoll die *TCP*–Port–Nummer 21.
Eine Übersicht über alle angebotenen Services mit ihren jeweiligen Ports findet sich in *UNIX*–Systemen in der Datei `/etc/services`. Tabelle 2.3 zeigt einen Ausschnitt dieser Datei.

2.2.2 Die Protokolle der Netzwerk–Schicht

Die Netzwerk–Schicht ist die Verbindung zwischen Soft– und Hardware. In ihr werden die Daten auf den Weg in das Netz gebracht und es kommen, in Abhängigkeit von der Hardware, viele unterschiedliche Protokolle zum Einsatz. Einige von ihnen sind:

- *Ethernet*
- *SLIP* (*Serial Line IP*)
- *PPP* (*Point to Point Protocol*)
- *Token Ring*

Weil im Max–Planck–Institut für neuropsychologische Forschung nur ein *Ethernet*–Netzwerk zum Einsatz kommt, wird auf das *Ethernet*–Protokoll, als Protokoll der Netzwerk–Schicht näher eingegangen.
Ein *Ethernet*–Rahmen besteht aus 5 Teilen:

- der Ziel–*Ethernet*–Adresse,
- der Quell–*Ethernet*–Adresse,
- dem Typ (*Type Code*), der die Daten näher beschreibt,
- den Daten (*IP–Datagramme*) selbst und
- einer Prüfsumme

Gemäß der *Ethernet*–Spezifikation, hat jede *Ethernet*–Netzwerkkarte eine eindeutige Adresse. Diese wird vom Hersteller vergeben und sollte danach nicht verändert werden. Damit ist für die Eindeutigkeit der Adressen gesorgt. Es sind somit 2^{48} ($>$ 280 Billionen) Adressen möglich. Der Typ beschreibt die Daten näher. Falls es sich um ein *IP*–Datagram handelt, nimmt das Typ–Feld den Wert 08 00 an. Das Ende des *Ethernet*–Rahmens bildet eine Prüfsumme (Vgl. Abbildung 2.6).

echo	7/tcp	
echo	7/udp	
discard	9/tcp	sink null
discard	9/udp	sink null
systat	11/tcp	users
daytime	13/tcp	
daytime	13/udp	
netstat	15/tcp	
qotd	17/tcp	quote
msp	18/tcp	
msp	18/udp	
chargen	19/tcp	ttytst source
chargen	19/udp	ttytst source
ftp-data	20/tcp	
ftp	21/tcp	
fsp	21/udp	fspd
ssh	22/tcp	
ssh	22/udp	
telnet	23/tcp	
smtp	25/tcp	mail
time	37/tcp	timserver
time	37/udp	timserver
rlp	39/udp	resource
nameserver	42/tcp	name

Tabelle 2.3: Ausschnitt der Datei `/etc/services`

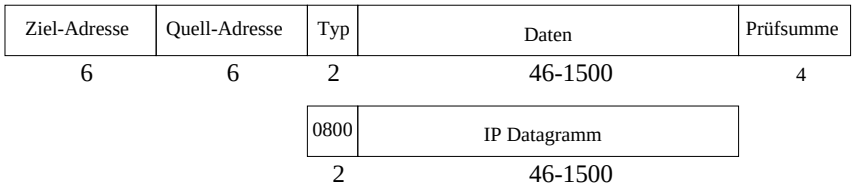

Abbildung 2.6: Das Format der *Ethernet*–Rahmen

2.2.3 Das *Internet–Protocol — IP*

Das Internet–Protokoll ([Pos81a]) ist das grundlegende Protokoll der *TCP/IP*–Protokoll–Suite. Die *TCP*– und *UDP*–Daten werden als *IP*–Datagramme übertragen. *IP* ist ein unzuverlässiger, verbindungsloser Datagram–Liefer–Service. Unzuverlässig bedeutet, daß es fast keine Sicherungen gibt, daß ein *IP*–Datagram seinen Bestimmungsort erreicht. Wenn ein Paket nicht weiter übertragen werden kann, weil z.B. in einem *Router* ein Puffer überläuft, hat das *IP* einen einfachen Fehlerbehandlungsalgorithmus. Dieser sieht vor, daß das Datagram verworfen wird und eine *ICMP*–Meldung zurück an den Absender geschickt wird. Alle Sicherheitsmaßnahmen müssen von darüber liegenden Schichten, wie *TCP* bereitgestellt werden.
Das *IP*–Protokoll ist verbindungslos d.h. jedes Datagram wird unabhängig von allen anderen Datagrammen verarbeitet. *IP*–Datagramme können den Ziel–Rechner in einer anderen Reihenfolge erreichen als sie abgeschickt wurden. Die Ursache dafür ist, daß aufeinanderfolgende Datagramme völlig unterschiedliche Wege durch das Netzwerk nehmen können und dabei ein abgeschicktes Datagram ein vorher abgeschicktes überholen könnte.
Das *IP*–Protokoll verpackt die Daten der übergeordneten Protokolle. Dabei werden die Daten mit einem Kopf (Header) versehen und an die darunterliegende Netzwerk–Schicht weitergegeben. Den Aufbau des Daten–Kopfes zeigt Abbildung 2.7. Der Daten–Kopf enthält folgende Parameter:

- Version
  Hier wird die Version des benutzten IP–Protokolls festgelegt, im Moment ist dies Version 4, bzw. Version 6
- Länge des Kopfes
  Die Länge des Headers in Byte, normalerweise (falls keine Optionen vorhanden sind) ist die Länge 5 Byte.
- Type of Service(ToS)
  Der Service–Typ setzt sich aus einem (meist ungenutzten) 3–Bit *prece-*

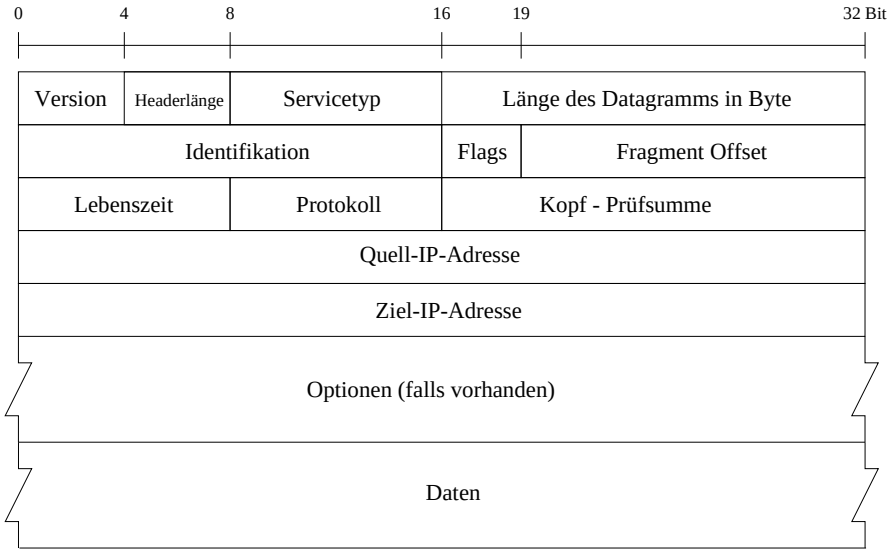

Abbildung 2.7: Das Format des *IP*–Headers

dence field, 4 Service–Typ–Bits und einem ungenutzten Bit zusammen. Die 4 Service–Typ–Bits bedeuten:

1. minimale Verzögerungen
2. maximaler Durchsatz
3. maximale Verläßlichkeit
4. minimale Kosten

Es kann maximal eines dieser vier Bits gesetzt sein. Ist keines gesetzt, wird ein normaler Service angeboten.

- Länge des Datagrams
  Die Länge des Datagrams in Bytes. Anhand der Länge des Datagrams und der Länge des Headers lassen sich die Größe und der Beginn des Datenteils des Paketes bestimmen. Weil dies ein 16–Bit Feld ist, kann die maximale Größe eines *IP*–Datagrams 65535 Bytes betragen.
- Identifikation
  Die Identifikation sorgt dafür, daß jedem Datagram eine eindeutige Zahl zugeordnet wird. Normalerweise wird für jedes Datagram der Wert des vorigen Datagrams um Eins erhöht.

- Flags
  Die Flags bestehen aus drei Bits. Zwei von diesen sind: DF – Don't Fragment und MF – More Fragments. Die beiden Bits DF und MF steuern die Behandlung eines Pakets im Falle einer Fragmentierung.
  Mit dem DF–Bit wird signalisiert, daß das Datagram nicht fragmentiert werden darf.
  Mit dem MF–Bit wird angezeigt, ob einem IP–Paket weitere Teilpakete folgen. Diese Bit ist bei allen Fragmenten außer dem letzten gesetzt.
- Fragment Offset
  Der Fragmentabstand bezeichnet, an welche Stelle relativ zum Beginn des gesamten Datagrams ein Fragment gehört. Mit Hilfe dieser Angabe kann der Zielhost das Originalpaket wieder aus den Fragmenten zusammensetzen.
- Lebenszeit
  Die Lebenszeit gibt an, wieviele *Router* das Datagram passieren kann, bevor es verworfen wird. Sie wird vom Sender definiert (meist 32 oder 64). In jedem *Router*, den das Datagram passiert wird die Lebenszeit um Eins dekrementiert. Ist die Lebenszeit bei Null angekommen, wird das Datagram verworfen und eine ICMP Meldung an den Sender des Datagrams geschickt. Damit wird verhindert, daß Pakete im Kreis geroutet werden.
- Protokoll
  Hier wird angegeben, welches darüber liegende Protokoll das Datagram geschickt hat.
- Kopf–Prüfsumme
  Die Prüfsumme über den Daten–Kopf.
- Quell–*IP*–Adresse
  Die *IP*–Adresse des Absenders.
- Ziel–*IP*–Adresse
  Die *IP*–Adresse des Ziels.
- Optionen
  Das Optionen–Feld hat eine variable Länge. Einige Optionen sind:
  - Sicherheits– und Behandlungsrestriktionen
  - *record route* — Es werden die *IP*–Adressen aller passierten Router aufgenommen.
  - timestamp — Jeder Router hinterläßt neben seiner *IP*–Adresse einen Zeitstempel.

  - *loose source routing* — Es wird eine Liste mit *IP*–Adressen angegeben, die von dem Datagram passiert werden müssen. Auf seinem Weg durch das Netz, kann das Datagram aber auch andere *Router* besuchen.
  - *strict source routing* — Es wird wiederum eine Liste mit *IP*–Adressen angegeben. Diesmal aber müssen die und nur die angegebenen *IP*–Adressen passiert werden.
- Daten

IP–Routing

Wenn zwei Rechner, zwischen denen Daten ausgetauscht werden sollen, direkt miteinander verbunden oder in einem gemeinsamen Netz sind, werden die Datagramme direkt von einem Rechner zum anderen gesendet. Ist dies jedoch nicht der Fall, wird es nötig die Datagramme über Zwischenstationen (sogenannte *Router*) weiterzuleiten. So eine Konstellation ist in Abbildung 2.3 zu sehen. In der *IP*–Schicht des *Routers* gibt es eine Routing–Tabelle. Diese dient zur Behandlung der *IP*–Datagramme. Anhand der Zieladresse des *IP*–Datagrams wird entschieden, wie das Datagram weitergeleitet wird. Ein Eintrag in der Routing–Tabelle setzt sich dabei aus folgenden Bestandteilen zusammen:

- Ziel–*IP*–Adresse.
  Das kann eine komplette Rechner–Adresse oder eine Netzwerk–Adresse sein. Zur Unterscheidung dient ein Flag. Ein Rechner hat immer eine *Host–ID*, die ungleich Null ist. Handelt sich um eine Netzwerkadresse, ist die *Host–ID* immer Null, das bedeutet, daß dieser Eintrag für alle Rechner des Netzwerkes gilt.
- *IP*–Adresse des nächsten *Router*s oder eines direkt verbundenen Netzwerkes.
  Der nächste *Router* ist immer innerhalb eines direkt zu erreichenden Netzwerkes. So bewegt sich das Datagram von *Router* zu *Router*, bis das Ziel–Netz und der Ziel–Rechner erreicht ist.
- Flags. (Vgl. [Ste94])
- das Netzwerkinterface
  an welches das Datagram gesendet werden soll.

Wenn der *Router* keine Informationen über die Ziel–Adresse eines *IP*–Datagrams besitzt, wird das Datagram über einen Standard–Weg (*default route*) weitergeleitet.

2.2.4 Das *Transmission Control Protocol — TCP*

Das *Transmission Control Protocol* ([Pos81b]) ist eines der Protokolle, welche auf dem Internet–Protokoll aufbauen. *TCP* bietet einen verbindungsorientierten, verläßlichen Byte–Strom–Verbindungsservice an. Verbindungsorientiert bedeutet, daß zwei Applikationen, die *TCP* benutzen, bevor sie Daten austauschen, eine Verbindung aufbauen müssen. Die Verläßlichkeit wird durch folgende Maßnahmen erreicht:

- Wenn die *TCP*–Schicht Daten empfängt, wird für jedes empfangene Paket eine Bestätigung an den Absender zurückgeschickt.
- Wird ein Segment verschickt, wird gleichzeitig ein Timer gestartet. Ist bis zu einem bestimmten Zeitpunkt keine Rückmeldung eingegangen, daß das Segment empfangen wurde, wird das Segment erneut übertragen.
- *TCP* vergibt Prüfsummen, sowohl für den *TCP–Header* als auch für die Daten. Stimmt die Prüfsumme nach der Übertragung nicht mit der ursprünglichen überein, so wird das Segment verworfen.
- *TCP*–Segmente werden als *IP*–Datagramme verschickt. Dadurch kann es vorkommen, daß *TCP*–Segmente ungeordnet das Ziel erreichen. Der *TCP*–Empfänger sorgt dafür, daß die Daten wieder in der richtigen Reihenfolge an die Applikations–Schicht weitergeleitet werden.
- Weil *IP*–Datagramme auch doppelt ankommen können, muß das *TCP*–Protokoll dafür sorgen, daß doppelt empfangene Segmente verworfen werden.
- *TCP* bietet eine Fluß–Kontrolle an. Die beteiligten Rechner einer *TCP*–Verbindung haben einen begrenzt großen Puffer. Das *TCP*–Protokoll des Empfängers erlaubt dem Sender nur soviel Daten zu senden, wie in seinen Puffer passen. Damit wird dafür gesorgt, daß ein schneller Rechner nicht den Puffer eines langsameren Rechners überlaufen läßt.

Zwischen den beiden Applikationen werden Ströme von 8–Bit Zeichen ausgetauscht. Das *TCP*–Protokoll interpretiert die zu transportierenden Daten nicht, d.h. das *TCP*–Protokoll überträgt den Daten–Strom, ohne auf dessen Inhalt zu achten. Jedes *TCP*–Segment beinhaltet:

- Quell– und Ziel–Port–Nummer
  Diese dienen zur Identifikation der sendenden und empfangenden Applikationen. Diese beiden Werte sowie die Quell– und Ziel–Adresse im *IP*–Kopf identifizieren eindeutig jede Verbindung.

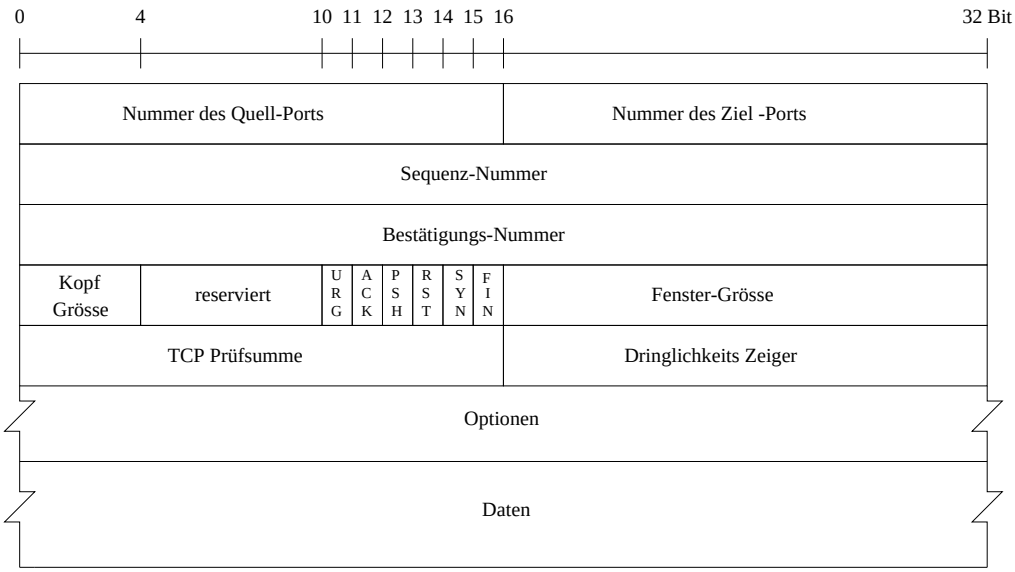

Abbildung 2.8: Das Format des *TCP–Headers*

- Paketlaufzähler
  Der Paketlaufzähler oder die *sequence number* ist eine Nummer, die vom Absender generiert wird und vom Empfänger bestätigt wird.
- Paketbestätigungszähler
  Der Paketbestätigungszähler (*acknowledgement number*) liefert als Bestätigung, die soeben erhaltenen Paketlaufzähler an den Absender zurück.
- Kopflänge
  Die Länge des *TCP–Headers* in 32–Bit–Worten. Dies wird nötig, weil die Länge des Optionen–Feldes variabel ist.
- Flags, wenn das entsprechende Bit gesetzt ist, bedeutet dies:

  0 – URG – der Dringlichkeits–Anzeiger ist gültig

  1 – ACK – die Bestätigungsnummer ist gültig

  2 – PSH – der Empfänger soll diese Daten so schnell wie möglich an die Applikation abgeben

  3 – RST – setze die Verbindung zurück

4 – SYN – synchronisiere die Paketlaufzähler, um eine Verbindung zu initialisieren. Diese Flag ist nur im ersten Paket einer Verbindung gesetzt.

5 – FIN – der Sender hat die Datenübertragung beendet. Dieses Flag ist nur im letzten Paket einer Verbindung gesetzt.

- Fenstergröße
  Legt die Anzahl der Bytes fest, die der Empfänger in seinem Puffer aufnehmen kann. Da dies ein 16–Bit Wert ist, wird die Größe des Puffers auf 65535 Bytes limitiert.
- *TCP*–Prüfsumme
  Eine Prüfsumme über *TCP–Header* und *TCP*–Daten.
- Dringlichkeits–Zeiger
  Der Dringlichkeits–Zeiger (*urgent pointer*) zeigt auf vorrangig zu bearbeitende Daten. Dieser Zeiger wird allerdings nur ausgewertet, wenn das URG–Bit gesetzt ist.
- Optionen
  Hier werden zusätzliche *TCP*–Funktionen definiert.
- Die Daten

2.3 Das Network–Information–System — *NIS*

Das *Network–Information–System* (früher *yellow pages* — gelbe Seiten) wurde von der Firma *SUN Microsystems* entwickelt. Es dient zur Verwaltung von Netzwerken. Das sind Rechnernetze für die eine Anzahl von Nutzern auf jedem Rechner Zugang haben soll, also mit gleichen Passwörtern und gleichen *home*–Verzeichnissen.
Dadurch wird erreicht, daß ein Benutzer nicht nur an einem Rechner Zugang zu seinen Daten erhält, sondern auf allen Rechnern innerhalb eines Netzwerkes. Da dieser Aufwand in großen heterogenen Netzen jedoch sehr hoch ist, wird ein System gebraucht, bei dem Veränderungen an einer zentralen Stelle vorgenommen werden und diese dann automatisch an alle Rechner verteilt werden. Genau diese Aufgaben übernimmt das *Network Information System*. *NIS* sorgt dafür, daß unter anderem die Dateien mit Informationen zu

- Passwörtern,
- Gruppenzugehörigkeit der Benutzer,
- erreichbaren Netzwerken,

- Rechnern im lokalen Netz,
- Mail–Gruppen

für alle Rechner und Benutzer zugänglich sind. Neben diesen Standard–Dateien kann *NIS* dafür sorgen, daß andere Dateien verteilt werden (z.B. Konfigurationsdateien für den Automounter).
Für die Verwaltung wird ein *NIS–Master–Server* benutzt. An diesem werden alle Veränderungen vorgenommen. Daraufhin wird der *NIS–Master–Server* angewiesen, seine Daten an die *NIS–Slave–Server* zu verteilen. Die *NIS–Clients*) beziehen die Informationen mittels RPCs (Remote Procedure Calls) von einem der *NIS–Server*. Dabei ist zu beachten, daß jeder *NIS–Server* gleichzeitig auch *NIS–Client* ist.

2.4 Verfahren zur Datei–Übertragung

2.4.1 Das *Network File System* — *NFS*

Das *Network File System* ([Mic89]) ist ein verteiltes Dateisystem, welches transparenten Zugriff auf Festplatten anderer Rechner bietet. Transparent bedeutet, daß Anwendungen auf dem Client–System nicht entscheiden können, ob die geladene Datei von einer lokalen Platte oder von einem anderen Rechner kommt. Ein Vorzug von *NFS* ist, daß in verteilten Arbeitsumgebungen jeder Benutzer auf jedem Rechner das gleiche Dateisystem vorfindet.
NFS funktioniert folgendermaßen: Auf jedem Rechner, welcher Teile seines Dateisystems für andere Rechner zur Verfügung stellt, läuft ein NFS–Server. In einer Konfigurationsdatei (`/etc/exports`) wird festgelegt, welche Dateisysteme oder Verzeichnisse für welche Benutzer an welchen fremden Maschinen freigegeben sind.
Bevor ein Benutzer auf ein Dateisystem eines NFS–Servers zugreifen kann, muß das Dateisystems des Servers in das Client–Dateisystem eingehängt (gemountet) werden. Die Arbeitsweise des *NFS* verdeutlicht Abbildung 2.9. Wenn eine Anwendung auf einem *NFS*–Client auf eine Datei eines *NFS*–Servers zugreifen will, sendet er eine *RPC*–Anfrage an den Server. Es gibt zwei Varianten von *NFS*. Diese unterscheiden sich durch die Benutzung von *TCP* oder *UDP*. Meistens kommt jedoch *NFS* auf der Basis von *UDP* zum Einsatz. Dies hat den Vorteil, daß es in sicheren Netzen schneller arbeitet, da keine Bestätigungen abgewartet werden müssen. Der *NFS*–Server wartet auf Anfragen der Clients auf *UDP*–Port 2049. Es gibt 15 verschiedene Anfragen. Zu diesen gehören:

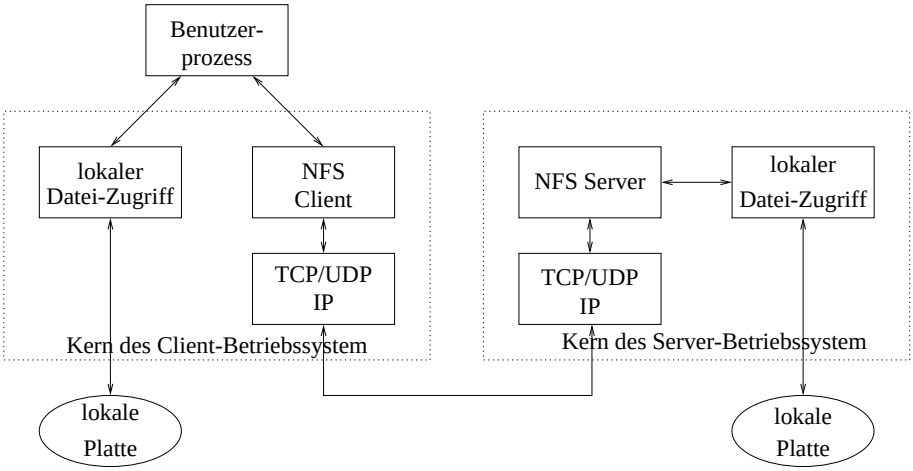

Abbildung 2.9: Ein typisches *NFS*–Szenario

GETATTR/SETATTR liefert oder setzt die Attribute einer Datei z.B. Zugriffsrechte, Besitzer,

STATFS liefert den Status des Dateisystems zurück z.B. den vorhanden Speicherplatz,

LOOKUP liefert ein Datei–Handle, wird bei jedem Öffnen einer Datei auf dem *NFS*–Server ausgeführt und

READ/WRITE liest bzw. schreibt eine Datei vom bzw. auf den *NFS*–Server.

Trifft eine Anfrage ein, gibt der Server diese an das lokale Dateisystem weiter, wo sie verarbeitet wird.
Das *Network File System* baut auf zwei Protokollmechanismen auf. Diese sind *External Data Representation* (*XDR*) und *Remote Procedure Calls* (*RPC*). *XDR* ([Sri95b]) erleichtert die Übersetzung der Daten zwischen heterogenen Computer– und Betriebssystemen. *RPC* ([Sri95a]) sorgt für die Basis der Kommunikation zwischen *NFS*–Client und *NFS*–Server. Auf diesen beiden Mechanismen bauen neben *NFS* noch andere Protokolle auf z.B. *Mount* und *NIS*.
Zur Zeit werden zwei Versionen des *NFS*–Protokolls (Version 2 und 3) in verschiedenen Implementierungen eingesetzt. Diese unterscheiden sich in der Art, wie sie versuchen *NFS* zu beschleunigen. Dadurch arbeiten unterschiedliche Versionen oft schlecht zusammen, was zu Einbrüchen der Übertragungsleistung führen kann.

2.4.2 Das *File Transfer Protocol* — *ftp*

Das File Transfer Protocol *ftp* ([JP85]) ist ein Standard für die Dateiübertragung im Internet. Wie *telnet*, wurde das *ftp*–Protokoll entwickelt, um zwischen Rechnern mit unterschiedlichen Betriebssystemen, unterschiedlichen Dateistrukturen und eventuell unterschiedlichen Zeichensätzen Dateien auszutauschen. Um das Ziel zu erreichen, unterstützt *ftp* nur eine begrenzte Anzahl von Dateitypen (*ASCII*, *binary*, usw.) und Dateistrukturen (*bytestream*– oder *record*–orientierte).
Die *ftp*–Applikation benutzt zwei *TCP*–Verbindungen, um eine Datei zu übertragen.

1. Steuerungsverbindung (*control connection*)
   Die Steuerungsverbindung dient zur Kontrolle der *ftp*–Verbindung. Gesteuert werden alle Dinge, die nicht direkt mit der Übertragung der Dateien zu tun haben. Dazu gehören: Einstellungen

   - zum Daten–Typ,
   - zum Übertragungsmodus,
   - zum Port für die Daten–Verbindung und
   - zur Datei–Struktur.

   Desweiteren werden alle *ftp*–Kommandos über die Steuerungsleitung geschickt.

2. Datenverbindung (*data connections*)
   Im Gegensatz zur ständig geöffneten Steuerungs–Verbindung wird die Datenverbindung nur aufgebaut, wenn Daten übertragen werden sollen. Der Begriff Daten beschränkt sich aber nicht nur allein auf Dateien, die übertragen werden sollen, sondern auch auf Daten, wie die Verzeichnisstruktur.
   Die Datenverbindung sollte auf maximalen Durchsatz ausgelegt werden, da diese für den Datentransport benutzt wird.

Datenrepräsentationen:
Im *ftp* Protokoll werden einige Möglichkeiten angegeben, um die Art der Datenübertragung zu beeinflussen:

1. Dateityp

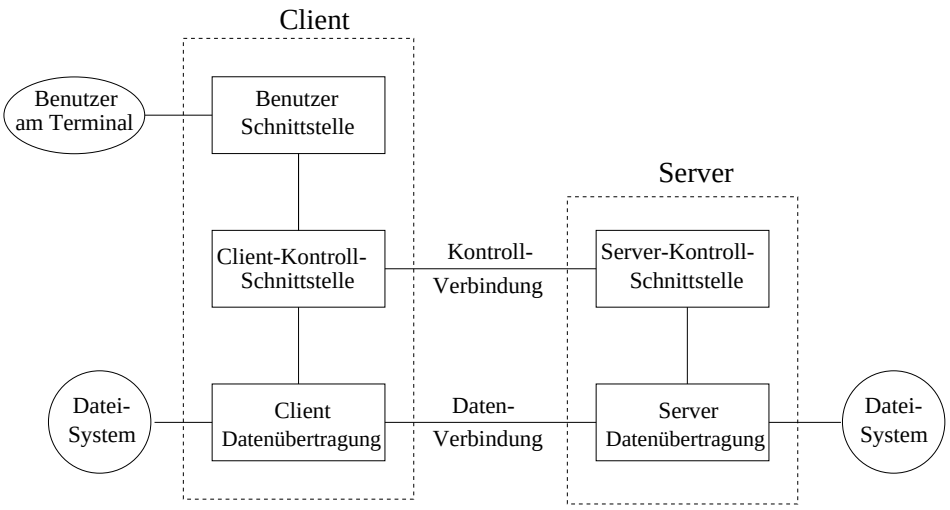

Abbildung 2.10: Der *ftp*–Datenübertragungsprozess

(a) *ASCII*–Dateityp
Dies ist die Standardeinstellung. Die Textdatei wird über die Datenverbindung übertragen. Dabei muß das lokale Textfile in *NVT ASCII* (*Network Virtual Terminal — American Standard Code for Information Interchange*) konvertiert und nach der Übertragung wieder in ein lokales Textfile zurückkonvertiert werden. Das besondere an *NVT ASCII* ist, daß jede Zeile mit einem *carriage return* (Wagenrücklauf) und *linefeed* (Zeilenvorschub) abgeschlossen werden muß.

(b) *EBCDIC*–Dateityp
(*Extended Binary–coded Decimal Interchange Code*) eine alternative Übertragungsform für Text, falls beide Enden *EBCDIC*–Systeme sind.

(c) *Image*–Dateityp
Die Daten werden als ein Strom von Bits übertragen. Dieses Verfahren wird normalerweise benutzt, um Binärdateien zu übertragen.

(d) *Local*–Dateityp
Eine Möglichkeit binäre Dateien zwischen Rechnern mit unterschiedlicher Byte Größe zu übertragen. Die Anzahl der Bits pro Byte wird vom Sender festgelegt. Wenn ein System acht Bit

pro Byte benutzt, ist der *Local*–Dateityp äquivalent zum *Image*–Dateityp.

2. Format–Steuerung
   Diese Auswahl ist nur für *ASCII* oder *EBCDIC* Datentypen möglich.

   (a) *nonprint*
   Die Dateien beinhalten keine vertikale Formatinformation

   (b) *telnet format control*
   Die Dateien beinhalten *telnet vertical format controls*, um sie auf einem Drucker zu interpretieren.

   (c) *fortran carriage control*
   Das erste Zeichen in jeder Zeile ist ein Fortran Format Steuerungs Zeichen.

3. Struktur

   (a) Dateistruktur
   Die Datei wird als zusammenhängender Strom von Bytes angesehen. Es gibt keine interne Dateistruktur.

   (b) Record–Struktur
   Diese Struktur wird nur bei Textdateien benutzt.

   (c) Seiten–Struktur
   Diese Struktur wird nur vom *TOPS–20* Betriebssystem angeboten.

4. Übertragungsmodus
   spezifiziert, wie die Datei über die Datenverbindung übertragen wird.

   (a) Stream mode
   (Default) Die Datei wird als ein Strom von Bytes übertragen.

   (b) Block mode
   Die Datei wird als eine Reihe von Blöcken übertragen, jeder wird mit einem oder mehreren Kopf–Bytes eröffnet.

   (c) Compressed mode
   Die Daten werden mit einem sehr einfachen Kompressionsverfahren behandelt. Dabei werden nur mehrmals hintereinander auftretende Bytes komprimiert.

Normalerweise werden jedoch nicht alle Möglichkeiten implementiert, sondern nur:

Kommando	Beschreibung
`PORT` *n1,n2,n3,n4,n5,n6*	IP–Adresse des Client(n1 – n4) und zu benutzender Port(n5 und n6)
`USER` *username*	sendet Username an den Server
`PASS` *passwort*	sendet Passwort an den Server
`LIST` *filelist*	listet Dateien oder Verzeichnisse auf
`TYPE` *typ*	setzt den Typ der zu übertragenden Daten
`RETR` *Dateiname*	kopiert Datei vom Server auf den Client
`STOR` *Dateiname*	kopiert Datei vom Client auf den Server
`CWD` *Verzeichnis*	wechselt in das angegebene Verzeichnis
`QUIT`	beendet die Verbindung

Tabelle 2.4: Einige ausgewählte *ftp*–Kommandos mit Beschreibung

Typ: *ASCII* oder *Image*

Format Steuerung: nur *nonprint*

Struktur: nur Dateistruktur

Übertragungsmodus: nur *stream mode*

Das *ftp*–Protokoll sieht standardisierte Kommandos und Antworten vor. Diese werden über die Steuerungsverbindung zwischen Client und Server in *NVT ASCII* übertragen. Es gibt mehr als 30 *ftp*–Kommandos, die vom Client an den Server geschickt werden können. Die Kommandos bestehen aus drei oder vier *ASCII*–Zeichen, einige davon mit optionalen Argumenten. Einige Beispiele sind in Tabelle 2.4 zu sehen. Der Server reagiert auf die vom Client gesendeten Kommandos mit standardisierten Antworten (Replies). Die Antworten bestehen aus dreistelligen Zahlen in *ASCII*, welche optional von einer Nachricht abgeschlossen werden können. Der Grund für die Trennung ist: die dreistellige Zahl ist für die Software und die Nachricht für den menschlichen Benutzer. Die Bedeutung der drei Zahlen des Antwort–Codes zeigt Tabelle 2.5. Die Datenverbindung dient nur zum Austausch von Daten. Es gibt drei Möglichkeiten, die Datenverbindung zu benutzen:

1. Übertragen einer Datei vom Client zum Server

Antworten	Beschreibungen
1yz	positive, vorläufige Antwort, eine Aktion wurde gestartet, aber es muß noch eine weitere Antwort abgewartet werden, bevor ein weiteres Kommando gesendet werden kann.
2yz	positive Antwort, ein neues Kommando kann gesendet werden
3yz	positive zwischenzeitliche Antwort, das Kommando wurde akzeptiert, es muß jedoch noch ein weiteres gesendet werden
4yz	die aufgerufene Aktion kann zur Zeit nicht ausgeführt werden. Die Aktion kann später noch einmal gestartet werden
5yz	das Kommando wurde nicht akzeptiert und sollte nicht wiederholt werden
x0z	Syntaktischer Fehler
x1z	Information
x2z	Verbindungen. Antworten, die sich auf die Steuerungs oder Datenverbindung beziehen
x3z	Authentifizierung. Antworten für login oder accounting Kommandos
x4z	Unspezifiziert
x5z	Dateisystemstatus

Tabelle 2.5: Bedeutung der Antworten des *ftp*–Server

2. Übertragen einer Datei vom Server zum Client
3. Übertragen einer Liste von Dateien oder Verzeichnissen vom Server zum Client

Der *ftp*–Server sendet die Verzeichnisdaten über die Datenverbindung, um Zeilenbegrenzungen zu umgehen und es dem Client zu ermöglichen, die Verzeichnisdaten in eine Datei zu schreiben.
Wie bereits erwähnt, besteht die Steuerungsverbindung während der gesamten *ftp*–Verbindung. Die Datenverbindung wird nur aufgebaut, wenn diese auch wirklich benötigt wird. Der Aufbau der Datenverbindung geschieht wie folgt:

1. Der Client hat die volle Kontrolle über den Aufbau der Datenverbindung. Er sendet ein `PORT`–Kommando an den Server. Als Argumente seine *IP*–Adresse sowie die zu benutzende Port–Nummer auf Client–Seite übergeben.
2. Der Server öffnet die Daten–Verbindung und benutzt dabei seinen Port 20.

Es kann notwendig werden, daß nicht der Server die Datenverbindung öffnet, sondern der Client. Dann sieht das Öffnen der Datenverbindung folgendermaßen aus:

1. Der Client sendet das `PASV`–Kommando an den Server. Dies weist den Server an, ein passives Öffnen durchzuführen.
2. Der Server sendet seine *IP*–Adresse sowie die zu benutzende Port–Nummer an den Client.
3. Der Client öffnet die Verbindung zum Server.

Nachdem die Datenverbindung aufgebaut wurde, beginnt die eigentliche Übertragung der Datei. Sie ist dann abgeschlossen, wenn der Server mit einer Meldung über die Kontroll–Verbindung die Übertragung bestätigt. Die Daten–Verbindung wird nach jeder übertragenen Datei geschlossen und muß für jede weitere Übertragung neu aufgebaut werden.

2.5 Verfahren zum *remote login*

2.5.1 Das *telnet*–Protokoll

Telnet ([JP83]) ist eine der ältesten Internet–Applikationen. Sie wurde bereits 1969 im ARPANET, dem Vorgänger des heutigen Internets benutzt.

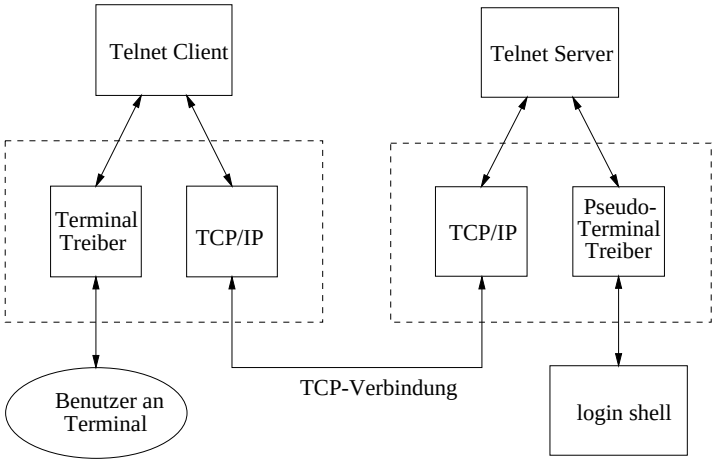

Abbildung 2.11: Aufbau einer *telnet*–Verbindung

Der Name *telnet* ist eine Abkürzung für *telecommunications network protocol* (Telekommunikations–Netzwerk–Protokoll).
Den Aufbau einer *telnet*–Verbindung zeigt Abbildung 2.11. *Telnet* wurde entwickelt, um zwischen beliebigen Rechnern mit beliebigen Terminals zu arbeiten. Um einen kleinsten gemeinsamen Nenner bezüglich der Terminals zu erreichen, wurde in der *RFC* 854 ([JP83]) ein sogenanntes *Network Virtual Terminal(NVT)* spezifiziert. Das *NVT* ist ein virtuelles Gerät, auf das beide, Client und Server, ihre realen Terminals abbilden.
Das *NVT* ist ein Gerät (*character device*) mit einem Ein– (Tastatur) und einem Ausgabeteil (Drucker). Daten, die der Benutzer an der Tastatur eingibt, werden an den Server geschickt, und Daten, die vom Server empfangen werden, werden über den Drucker ausgegeben.
Als Zeichensatz wird *NVT ASCII* benutzt, eine Variante des 7–Bit–US–*ASCII*–Zeichensatzes. Jedes 7–Bit–Zeichen wird als 8–Bit Byte mit einer Null im höchstwertigen Bit übertragen.
Nachdem eine Verbindung aufgebaut wurde, einigen sich der *Client* und der *Server* auf weitere Optionen, welche die virtuellen Terminals zur Verfügung stellen können. *Telnet* benutzt *in–band–signaling* in beiden Richtungen. Das heißt, daß Steuersignale wie Textdaten übertragen werden. Dafür gibt es spezielle Bytefolgen. Jede dieser Kommando–Bytefolgen beginnt mit dem Byte 0xff (255 dezimal), dem `IAC`–Byte (interpret as command (interpretiere als Kommando)). Nach diesem folgt ein Kommando–Byte. Die möglichen

Kommando–Bytes zeigt Tabelle 2.6.

Optionsverhandlungen

Zum Beginn einer *telnet*–Sitzung werden an beiden Enden der Verbindung *NVT*s vorausgesetzt. Der erste Datenaustausch, der stattfindet, ist normalerweise die Optionsverhandlung (*option negotiation*). Die Optionsverhandlungen sind symmetrisch, d.h. beide Seiten können Anfragen an die andere stellen. Es gibt vier verschiedene Anfragen für jede Option.

WILL Der Absender will eine Option selbst bereitstellen.

DO Der Absender fordert den Empfänger auf, eine Option bereitzustellen.

WONT Der Absender will diese Option nicht nutzen.

DONT Der Absender fordert den Empfänger auf, eine Option abzustellen.

telnet erlaubt es, beiden Seiten eine Anfrage entweder zu akzeptieren oder abzulehnen. Daraus ergeben sich dann sechs verschiedene Szenarien, siehe Tabelle 2.7.
Für eine Options–Verhandlung werden mindestens drei Bytes benötigt.

1. das `IAC`–Byte
2. ein Byte für `WILL`, `DO`, `WONT` oder `DONT`
3. ein Byte für die gewünschte Option.
4. eventuell weitere Bytes (abhängig von der gewählten Option).

Es können über 40 Optionen ausgehandelt werden. Einige der Optionen zeigt Tabelle 2.8. Nähere Erläuterungen zu den Optionen aus Tabelle 2.8 sind in den angegebenen *RFC*s zu finden.

Sub–Options–Verhandlungen

Einige Optionen benötigen mehr Informationen als einfach nur "enable" oder "disable". Ein Beispiel dafür ist die Spezifikation des Terminal–Types (Es muß eine ASCII–Zeichenkette, die den Terminal–Typ beschreibt, übertragen werden.).
RFC 1091 definiert die Sub–Options–Verhandlungen (*suboption negotiation*) für den Terminal Typ. Zuerst bittet eine der beiden Seiten (normalerweise der Client), die Option bereitzustellen, indem er die 3–Byte Sequenz

Name	Code	Beschreibung
EOF	236	end-of-file
SUSP	237	suspend current process
ABORT	238	abort process
EOR	239	end of record
SE	240	suboption end
NOP	241	no operation
DM	242	data mark
BRK	243	break
IP	244	interrupt process
AO	245	abort output
AYT	246	are you there?
EC	247	escape character
EL	248	erase line
GA	249	go ahead
SB	250	suboption begin
WILL	251	option negotiation
WONT	252	option negotiation
DO	253	option negotiation
DONT	254	option negotiation
IAC	255	data byte 255

Tabelle 2.6: Kommando–Bytes des *telnet*–Protokolls

1.	WILL	→		Absender will eine Option bereitstellen
		←	DO	Empfänger sagt OK
2.	WILL	→		Absender will eine Option bereitstellen
		←	DONT	Empfänger sagt NEIN
3.	DO	→		Absender will, daß Empfänger eine Option bereitstellt
		←	WILL	Empfänger sagt OK
4.	DO	→		Absender will, daß Empfänger eine Option bereitstellt
		←	WONT	Empfänger sagt NEIN
5.	WONT	→		Absender will eine Option abstellen
		←	WILL	Empfänger muß OK sagen
6.	DONT	→		Absender will, daß der Empfänger eine Option abstellt
		←	WILL	Empfänger muß bestätigen

Tabelle 2.7: Szenarien der Options–Verhandlungen beim *telnet*–Protokoll

Options ID	Name	RFC
1	echo	857
3	suppress go ahead	858
5	status	859
6	timing mark	860
24	terminal type	1091
31	window size	1073
32	terminal speed	1079
33	remote flow control	1372
34	linemode	1184
36	enviroment variables	1408

Tabelle 2.8: Ausgewählte Optionen des *telnet*–Protokolls

`<IAC,WILL,24>`
sendet, wobei 24(dezimal) die Optionen–ID für den Terminal–Typ ist. Wenn der Empfänger damit einverstanden ist, sendet er
`<IAC,DO,24>`
Danach fragt der Sender den Empfänger nach dessen Terminal–Typ.
`<IAC,SB,24,1,IAC,SE>` Die Bedeutung der einzelnen Befehle:

- IAC - Interpretiere als Kommando
- SB - *suboption begin* Kommando
- 24 - spezifiziert die Option Terminal Typ
- 1 - "send your terminal type"
- IAC - Interpretiere als Kommando
- SE - *suboption end* Kommando

Der Client antwortet z.B. mit:
`<IAC,SB,24,0,'I','B','M','P','C',IAC,SE>`

- IAC - Interpretiere als Kommando
- SB - *suboption begin* Kommando
- 24 - spezifiziert die Option – terminal type
- 0 - "my terminal type is"
- IBMPC - Zeichenkette für den Terminal–Typ
- IAC - Interpretiere als Kommando
- SE - *suboption end* Kommando

Die Sub–Options–Verhandlung für den Terminal-Typ ist abgeschlossen. Nachdem alle Optionsverhandlungen abgeschlossen sind, können Befehle und die zugehörigen Antworten übertragen werden.

2.5.2 Das *rlogin*–Protokoll

Ein alternatives Verfahren zum *remote login* ist das *rlogin*–Protokoll. Dies wurde in [Kan91] beschrieben.
rlogin ist ein verhältnismäßig einfaches Protokoll. Es benutzt den *TCP*–Port 513. Die Initialisierung ist sehr einfach: Der Client sendet vier Zeichenketten an den Server.

- ein Nullbyte (einen Leerstring)
- Login–Name des Benutzers auf dem Client–Rechner
- Login–Name des Benutzers auf dem Server–Rechner

- Terminal–Typ/Terminal–Geschwindigkeit

Erkennt der Server die Anfrage an, antwortet er mit einem Nullbyte. Dann wird noch das Passwort abgefragt, und die Verbindung ist aufgebaut.
Jetzt werden die Befehle vom Client an den Server geschickt und die Ausgaben des Servers an den Client.

2.6 Die Programmiersprache *JAVA*

2.6.1 Die Entwicklung

1991 wurde die Programmiersprache *JAVA* ([Fla97]) von der Firma *SUN Microsystems* im Rahmen eines Software–Forschungsprojekts mit dem Namen *oak* (Eiche) für elektronische Haushaltsgeräte, wie Fernseher, Videorecorder, Toaster u.a. entwickelt.
Der Programmiersprache *JAVA* war jedoch kein sofortigen Erfolg im Consumer–Electronics–Bereich bestimmt. Im aufkommenden Boom des *World Wide Web* erkannte man, daß *JAVA* durch die Portabilität, die Übertragung sofort lauffähigen Byte–Codes und die sichere Laufzeitumgebung ideal für Internet–Anwendungen geeignet ist. Kleine *JAVA*–Programme, sogenannte Applets (kleine Applikationen), werden durch eigens dafür definierte Schlüsselworte in HTML–Seiten (Hypertext–Markup–Language — Seitenbeschreibungssprache des Internets([Rag96])) eingebunden. Damit war der Grundstein für den Siegeszug von *JAVA* im Internet gelegt. Aber nicht nur für kleine Anwendungen innerhalb von HTML–Seiten ist *JAVA* geeignet, sondern auch für "normale" Anwendungen.

2.6.2 Die Funktionsweise

Das Besondere an *JAVA* ist, daß es nicht nur auf der Quelltextebene plattformunabhängig ist, sondern auch auf der Binärebene (der sogenannte Byte–Code). Dies wird durch den Einsatz einer virtuellen Maschine, eine sogenannte *JAVA Virtual Machine (JVM)* möglich. Diese *JVM* interpretiert den *JAVA*–Byte–Code, und schickt die interpretierten Befehle an das darunterliegende Betriebssystem. Der Byte–Code ähnelt zwar den Maschinenbefehlen, ist aber nicht an einen speziellen Prozessor–Typ gebunden. Auf jedem Betriebssystem mit einer *JVM*, sind damit alle *JAVA*–Programme ausführbar.
Abbildung 2.12 zeigt den Unterschied zwischen einem normalen und einem *JAVA*–Programm. Während das "normale" Programm direkten Zugriff auf

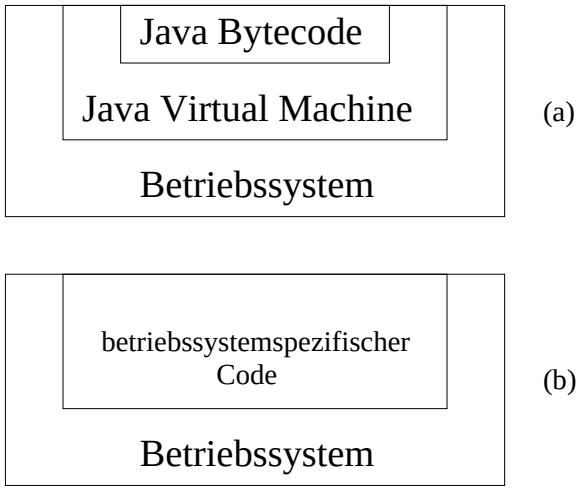

Abbildung 2.12: Schematik der virtuellen *JAVA*–Maschine(a) im Vergleich zu binären Programmen(b)

die Ressourcen des Betriebssystem hat, arbeitet das *JAVA*–Programm in der virtuellen Maschine. Alle Zugriffe auf das Betriebssystem übernimmt die JVM.
Bei anderen Programmiersprachen (z.B. *C* oder *C++*) muß der Quelltext auf jedem System neu übersetzt werden. Das Übersetzen des Quelltextes erweist sich häufig schwieriger als erwartet, weil sich die *C/C++*–Compiler in einigen Details voneinander unterscheiden. Es kann sogar soweit kommen, daß Teile des Quelltextes verändert werden müssen, damit das Programm übersetzt werden kann.

2.6.3 Die Eigenschaften

JAVA ist eine einfache, portable, objektorientierte, multithread–fähige, verteilte, interpretierte, robuste, sichere und dynamische Programmiersprache.

Portabilität

Inzwischen ist *JAVA* auf fast allen Betriebssystemen implementiert[1]. Damit kann man ein Programm unter einem Betriebssystem entwickeln und der *JAVA*–Byte–Code ist auch auf allen anderen Betriebssystemen mit einer Virtuellen–*JAVA*–Maschine ausführbar.

Objektorientierung

Der objektorientierte Programmieransatz ([Cas96]) ist relativ neu. Hier stehen nicht die Verarbeitungsvorgänge im Vordergrund, wie beim funktionalen oder prozeduralen Programmieransatz, sondern wie im "realen Leben" Objekte. Jedes Objekt kann Eigenschaften und Zustände besitzen, Daten mit anderen Objekten austauschen und mit anderen Objekten neue, leistungsfähigere Objekte bilden.
Die Grundelemente einer objektorientierten Sprache sind: Kapselung, Vererbung und Polymorphie.

Kapselung heißt, daß die Daten eines Objektes nur über definierte Methoden zugänglich sind. Die Daten sind also in den Objekten untergebracht. Damit sind auch die Zugriffe auf die internen Strukturen abgeschlossen und für den Anwender des Objektes bzw. den Programmierer unsichtbar.

Vererbung bedeutet, daß Objekte Eigenschaften und Methoden von anderen Objekten "erben", d.h. übernehmen können. In den meisten Sprachen wird dazu der Begriff der Klasse eingeführt. Klassen sind über eine *is–a*–Verknüpfung miteinander verbunden und bilden so eine Klassenhierarchie. An ihrer Wurzel steht die einfachste Klasse, an ihren Enden liegen die spezialisierten Klassen.

Polymorphie heißt, daß Objekte aus verschiedenen Klassen unterschiedlich auf den Aufruf von gleich aussehenden Methoden reagieren.

[1] Es gibt auch Implementierungen, welche dem plattformunabhängigen Konzept zuwiderlaufen. Dies geschieht durch zusätzliche Befehle die nicht zum Standard–*JAVA*. Ein Beispiel ist das *Microsoft–JAVA*(J++)

Multi–Thread–Fähigkeit

Ein Thread (Faden) ist ein sequentieller, meist kurzer Programmteil. Ein Thread ist kein eigenständiges Programm. Er kann aus einem Programm heraus gestartet werden. Der Hauptunterschied zu normalen Programmen ist, daß mehrere dieser Threads parallel zur gleichen Zeit um ihre Ausführung konkurrieren können. Wenn mehrere Threads gleichzeitig ausgeführt werden können, spricht man von Multi–Thread–Fähigkeit. So kümmert sich z.B. ein Thread darum, Daten aus dem Netz zu empfangen, und ein anderer Thread um die Ausgabe der Daten auf den Bildschirm.

Verteiltheit

Die *JAVA*–Klassen–Bibliothek enthält einige Klassen zur Kommunikation über *TCP/IP* (Vgl. Kapitel 2.2). Dies reicht von der Unterstützung von *HTTP* (*Hypertext Transfer Protocol*) und *ftp* (*File Transfer Protocol*) bis hin zum direkten Zugriff auf Sockets. Mit diesen Hilfsmitteln ist es möglich, verteilte Anwendungen mit relativ geringem Aufwand zu programmieren.

Robustheit

Im Gegensatz zu *C/C++* gibt es bei *JAVA* keine Zeiger (*Pointer*). Zeiger werden in *C/C++* benutzt, um direkt auf den Speicher zuzugreifen. Dabei kann es zu Schwierigkeiten kommen, wenn ein Programm im eigenen Adreßraum arbeitet und dabei Teile des Programmcodes überschreibt.
Anstelle von direkten Speicher–Verwaltungsfunktionen wie `malloc` oder `free` bietet *JAVA* einen eingebauten *Garbage Collector* (Müll–Sammler), der dafür sorgt, daß nicht mehr benötigte Speicherbereiche wieder freigegeben werden.
Zur Robustheit von *JAVA* leistet auch die strenge Objektorientierung mit gleichzeitiger strenger Typenprüfung ihren Beitrag. Formale Fehler werden schon beim Compilieren des Byte–Codes entdeckt und damit im entstehenden Programm verhindert.
Der Verzicht der *JAVA*–Entwickler auf *Operator–Overloading*, multiple Vererbung und die Einschränkung der automatischen Typumwandlung tragen zu weniger Programmierfehlern der *JAVA*–Programme bei.

Sicherheit

JAVA–Programme laufen meist in verteilten Umgebungen wie dem Internet. Deshalb werden an *JAVA*–Programme hohe Sicherheitsanforderungen gestellt.
Wenn zum Beispiel ein *JAVA*–Applet von einem unbekannten Rechner eine Klasse lädt und diese versucht auszuführen, darf es nicht zu Komplikationen (wie z.B. Systemabstürzen) kommen.
Um Sicherheit zu gewährleisten, hat *JAVA* ein dreistufiges Sicherheitsmodell eingeführt.
Auf der untersten Ebene:

- verbietet *JAVA* jede Art von direktem Speicherzugriff (kennt keine Zeigertypen),
- werden Zugriffe auf Felder kontrolliert (Bereichsüberprüfung),

Auf der mittleren Ebene:

- wird der geladene Byte–Code überprüft ,

auf der obersten Ebene:

- wird der Programm–Code in einem klar abgegrenzten Rahmen ausgeführt (Sandbox),
- darf nicht auf das lokale Dateisystem geschrieben werden und
- existiert ein Security–Manager, der Zugriffe auf das Dateisystem oder das Netzwerk überwacht.

Die Sandbox existiert nur für Applets, innerhalb eines Web–Browsers. Normale *JAVA*–Programme haben alle üblichen Rechte und dürfen z.B. auf die Datenträger zugreifen. Die Rechte können allerdings durch einen Securitymanager beschnitten werden.

Geschwindigkeit

Ein Nachteil von *JAVA* ist, daß die Ausführungsgeschwindigkeit von *JAVA*–Programmen durch den Umweg über den Byte–Code–Interpreter langsamer ist, als bei Programmen, die direkt auf das Betriebssystem zugreifen. Durch die virtuelle Maschine läuft interpretierter *JAVA*–Byte–Code langsamer als compilierter *C*– oder *C++*–Code. Bei interaktiven grafischen Benutzerschnittstellen oder netzwerkbasierten Anwendungen kommt es jedoch

häufig zu Phasen, in denen das Programm unbeschäftigt ist. Das liegt daran, daß auf ein Ereignis gewartet werden muß, wie etwa eine neue Eingabe des Benutzers, oder das Eintreffen neuer Daten aus dem Netzwerk. In solchen Fällen spielt der Geschwindigkeitsnachteil keine Rolle. Ein Vergleich der Geschwindigkeit von *C*–, *Perl*– und *JAVA*–Code ist in Kapitel 3.15.1 zu finden.

Die GUI-Bibliothek Swing *Swing* ist ein Paket zur einfachen Gestaltung von grafischen Benutzeroberflächen. Es bietet Komponenten wie: Menü–Leisten, Tool–Bars, Dialog–Fenster und vieles mehr. Eine weitere Besonderheit von Swing ist das "plugable look and feel" (veränderbares Aussehen). Damit kann man seine Anwendungen an unterschiedliche Umgebungen anpassen. Unter anderem gehört ein *UNIX/Motif*– und ein *Windows*–Aussehen zu Swing. So kann der Benutzer je nach gewähltem Arbeitsumfeld oder persönlichen Vorlieben seine Oberfläche verändern, ohne den Quelltext eines Programms ändern zu müssen.

Kapitel 3

Bestimmung der Anforderungen und Entwicklung einer Konzeption

In diesem Kapitel, geht es um die Bestimmung der Anforderungen an ein Daten–Management–Systems und die Entwicklung einer daraus resultierenden Konzeption. Zuerst ist es jedoch nötig, den Begriff des Daten–Managements abzugrenzen.

3.1 Begriffsklärung

Der Begriff des Daten–Managements ist nicht eindeutig definiert. Die Definition ist abhängig von der Art der Daten und was mit diesen Daten geschehen soll. So existieren z.B.

- Datei–Management–Systeme
  zur Datei–Verwaltung
- Produkt–Daten–Management–Systeme
  zur Analyse von Produktionsprozessen

- Patienten–Daten–Management–Systeme
  zur Visualisierung und Verarbeitung patientenbezogener Daten (z.B. [Pil99])

Auf Grund der unterschiedlichsten Daten–Management–Systeme wird es nötig, den Begriff des Daten–Managements näher zu spezifizieren. In dieser Arbeit ist unter dem Begriff Daten-Management

- die Erkennung der Art der Daten,
- die Auswahl der geeigneten Verarbeitungsprogramme,
- die Verschiebung der Daten von einem Rechner zu einem anderen zwecks Verarbeitung und die Verschiebung der verarbeiteten Daten auf einen beliebigen Rechner sowie
- die Archivierung von Dateien in heterogenen Netzwerken

zu verstehen.

3.2 Vorbetrachtungen zu Daten–Management–Systemen

Wie beschrieben ist unter Daten–Management die Erkennung der Art der Daten, die Auswahl geeigneter Konvertier–Programme, das Konvertieren der Daten auf speziellen Rechnern sowie die Archivierung von Daten zu verstehen.
Anhand der Begriffsklärung ergeben sich folgende Anforderungen:
Ein Daten–Management–System sollte

- den Datentyp der zu verarbeitenden Daten automatisch erkennen,
- automatisch passende Konvertierprogramme vorschlagen,
- transparent Konvertierungen vornehmen – Dazu gehört:
  - Dateien und Verzeichnissen zwischen Rechnern zu übertragen,
  - die Sicherung der Datenübertragung zu übernehmen und
  - das Starten von Programmen auf entfernten Rechnern zu ermöglichen.

  und
- verschiedene Archivierungsmethoden zur Verfügung stellen.

Die gefundenen Anforderungen sollen in der Konzeption umgesetzt werden. Aufgabe dabei ist es: bereits bestehende Verfahren auf ihre Tauglichkeit für die zu erreichenden Ziele zu untersuchen, neue Verfahren zu entwickeln bzw. vorhandene zu erweitern.
Die Entwicklung der Konzeption gliedert sich in folgende Teile:

- Situationsanalyse des Daten–Managements,
- Analyse des Datenflusses,
- Konzeption der Datenverarbeitungsvorgänge
- Bestimmung der benötigten Verfahren und
- Diskussion der Verfahren.

Dies soll in den folgenden Abschnitten geschehen.

3.3 Situationsanalyse des Daten–Managements

Der Ausgangspunkt für das Management von Daten ist die Gewinnung von Rohdaten. Rohdaten sind Daten, die direkt mittels Meßgeräten gewonnen werden. Nach der Gewinnung gibt es drei Möglichkeiten, was mit diesen geschehen soll:

- Verteilung auf unterschiedliche Rechner, da nicht alle Rohdaten auf jedem Rechner verarbeitet werden können
- Verarbeitung der Daten
- Archivierung auf externen Speicher–Medien

Diese Reihenfolge entspricht dem üblichen Vorgang bei der Daten–Verarbeitung.
Bisher wurden alle drei Teile der Daten–Verarbeitung getrennt behandelt und das setzte nähere Kenntnisse des Netzwerkes sowie der zu benutzenden Programme vorraus. Es existierte bisher keine Möglichkeit, die verschiedenen Aufgaben unter einer gemeinsamen Oberfläche vorzunehmen. Deshalb empfiehlt es sich, die Datenverarbeitungsvorgänge zu managen; d.h. ein Daten–Management vorzunehmen.
Die Notwendigkeit des Daten–Managements soll ein näherer Blick auf das bisherige Vorgehen aufzeigen.

3.3.1 Verteilung

Die Verteilung der Daten geschieht in heterogenen Netzwerken meist mittels des *File–Transfer–Protokolls* (*ftp*) (Vgl. Kapitel 2.4.2), dem *Network–File–System* (*NFS*) (Vgl. Kapitel 2.9), *Andrew–File–System* (*AFS*) oder *SAMBA* (Vgl. [Sha99]).

3.3.2 Verarbeitung

Die Verarbeitung der Daten gliedert sich in zwei Schritte:

- Konvertieren der Daten in eine verwertbare Form und
- Auswerten der Daten

Das Konvertieren der Daten ist meist nur auf speziellen Rechnern mit ausreichender Verarbeitungsgeschwindigkeit und Speicherkapazität möglich. Deshalb müssen die Daten nach ihrer Gewinnung auf diese Rechner kopiert werden und dann mittels eines Konvertierprogrammes gewandelt werden. Bei den Konvertierprogrammen handelt es sich meist um spezielle, kommandozeilenorientierte Programme ohne benutzerfreundliche Oberfläche. Nach dem die Daten konvertiert sind müssen diese eventuell noch zur Auswertung "von Hand" auf spezielle Auswerterechner zurückkopiert werden. Abschliessend werden die Daten vom Benutzer ausgewertet.

3.3.3 Archivierung

Nachdem die Daten ausgewertet wurden, sollen sie archiviert werden. Das Archivieren ist vom beteiligten Medium abhängig. Die entsprechenden Archivierungs–Geräte befinden sich nur an speziellen Servern. Auf die Geräte (CDROM–Brenner, Magnetband–Laufwerk u.a.) kann zum Teil nur lokal zugegriffen werden. Die Daten müssen also zuerst auf diese Rechner übertragen und später auf die jeweiligen Medien geschrieben werden. Die Archivierung erfordert ein Interaktion mit dem Benutzer, da dieser das Archivierungs–Medium bereitstellen muss.

3.4 Die Analyse des Datenflusses

Zu Beginn der Entwicklung der Konzeption steht die Analyse des Datenflusses. Dieser läßt sich in vier Teile aufspalten:

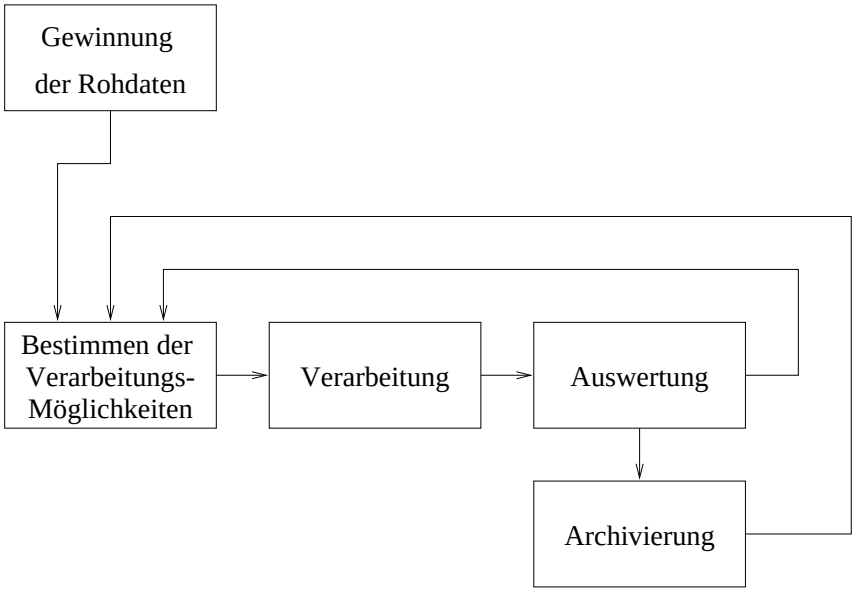

Abbildung 3.1: Schema des Datenflusses

1. Bestimmen der Verarbeitungsmöglichkeiten, dazu gehört die Bestimmung des Datentyps, aus dem die Verarbeitungsmöglichkeiten abgeleitet werden,
2. die Verarbeitung der Daten,
3. die Auswertung der verarbeiteten Daten und
4. die Archivierung der ausgewerteten Daten.

In Abbildung 3.1 ist das Zusammenspiel der Teile des Datenflusses verdeutlicht. Es wird sichtbar, daß vor jeder Verarbeitung eine Analyse des Datentyps stattfindet und daraus die Verarbeitungsmöglichkeiten abgeleitet werden. Bisher musste der Benutzer entscheiden, welche Verarbeitungs–Verfahren sinnvoll sind und welcher Rechner die nötigen Ressourcen zur Verarbeitung der Datensätze bereitstellt. Anschliessend mussten die Datensätze auf den Verarbeitungsrechner übertragen werden und die Verarbeitung gestartet werden. Abschliessend wurden die verarbeiteten Daten ausgewertet, auf einen Rechner mit Archivierungs–Medium übertragen und auf diesen archiviert.
Anhand des beschriebenen Datenflusses soll ein Konzept entwickelt werden,

welches den Datenfluss weitgehen automatisiert. Dazu werden folgende Dinge benötigt:

- Möglichkeit zur Erkennung des Dateityps,
- Ableitung der Verarbeitungsmöglichkeiten aus dem Datentyp,
- Verfahren zur effektiven Datenübertragung,
- Absicherung der Datenübertragung,
- Verfahren zum *remote login*,
- Entwicklung einer grafischen Benutzeroberfläche mit Benutzerführung und
- zur Umsetzung der Konzeption – Auswahl einer Programmiersprache.

In den nachfolgenden Abschnitten werden die notwendigen Verfahren und Standards diskutiert.

3.5 Möglichkeiten der Dateityp–Erkennung

Es gibt zwei Möglichkeiten der Erkennung des Dateityps.

- Erkennung anhand der *magic–number*
- Erkennung anhand der Dateiendung

3.5.1 Dateityperkennung mittels *magic–number*

Wenn eine Datei mit einer solchen magischen Zahl beginnt, kann man anhand einer Datenbank den Dateityp feststellen. Die Dateityp–Erkennung mittels *magic–number* wird nur in *UNIX*–Betriebssystemen verwendet (`file`–Befehl Vgl. [fil]) und betrachtet somit nur *UNIX*–spezifische Datentypen.

3.5.2 Dateityp–Erkennung mittels Dateiendung

Dieses Verfahren ist sehr einfach. Man untersucht den Dateinamen und nimmt die Zeichenkette, die nach dem letzten Punkt im Dateinamen beginnt als Merkmal für den Dateityp. Voraussetzung für dieses Verfahren ist allerdings, daß die Dateiendungen sinnvoll gewählt wurden. Es macht z.B. keinen Sinn, Daten–Dateien mit der Endung `.dat` zu versehen, da diese Endung sehr verbreitet ist und deshalb keinerlei nähere Informationen zum Datei–Typ enthält.

Dateianzahl Verarbeitungsmöglichkeit	0	1 Typ bekannt	1 Typ unbekannt	>1
Kopieren		x	x	x
Bewegen		x	x	x
Löschen		x	x	x
Konvertieren		x		
CDROM schreiben		x	x	x
Tape schreiben		x	x	x
Tape lesen	x			
tar–Archiv erzeugen				x

Tabelle 3.1: Übersicht der Verarbeitungsschritte in Abhängigkeit von der Anzahl und der Art der ausgewählten Dateien

3.5.3 Auswahl des Verfahrens zur Dateityp–Erkennung

Die Datenbank der *magic numbers* beinhaltet keinerlei Informationen über die Datentypen, die im Max–Planck–Institut für neuropsychologische Forschung verarbeitet werden.
Aufgrund der fehlenden Dateitypen und der unterschiedlichen Implementierungen des `file`–Programms fiel die Auswahl des Verfahrens zur Dateityp–Erkennung auf die ungenauere Erkennung mittels Dateiendung.

3.6 Auswahl der Verarbeitungsmöglichkeiten

Die Auswahl der Verarbeitungsmöglichkeiten ist abhängig von der Anzahl und der Art der ausgewählten Datei(en). Eine Übersicht der Verarbeitungsmöglichkeiten in Abhängigkeit von der/den ausgewählten Datei(en) zeigt Tabelle 3.1.
Nachdem die Datei(en) und die Art der Verarbeitung ausgewählt sind werden die Daten auf spezielle Rechner übertragen, die die zur Verarbeitung benötigten Ressourcen bereitstellen. Dazu ist es notwendig die Verfahren zur Datenübertragung näher zu betrachten.

3.7 Vergleich der Verfahren zur Datenübertragung

Die Datenübertragung spielt im Rahmen der Konzeption eine entscheidende Rolle, da sehr große Datenmengen übertragen werden sollen. Es ist also wichtig, ein Verfahren zu benutzen, welches schnellstmöglich die Daten von einem Rechner zum anderen transportiert.
Zur Übertragung der Daten im Netz bieten sich folgende Möglichkeiten an:

- NFS — *Network File System* (Vgl. Kapitel 2.4.1)
- ftp — *File Transfer Protocol* (Vgl. Kapitel 2.4.2)
- http — *Hypertext Transfer Protocol*

Da die zu übertragenden Datensätze sehr groß sind, ist es angebracht, das Protokoll zu benutzen, welches die höchste Übertragungsrate zuläßt. Deshalb folgt ein Leistungs–Vergleich zwischen den Protokollen *ftp* und *NFS*.

Messung der Datenübertragungsraten

Die Messungen zur Datenübertragung (Vgl. Tabelle 3.2) fanden an einem typischen Arbeitstag im Max–Planck–Institut für neuropsychologische Forschung statt, um die Auswirkungen des üblichen Netzverkehrs in die Messungen einfließen zu lassen. Die gemessenen Werte entsprechen in beiden Fällen nicht den maximal erreichbaren Datenübertragungsraten.
Aus den gemessenen Werten ist jedoch abzulesen, daß Übertragungen mit *ftp*–Protokoll mindestens 1,5 mal so schnell sind wie Übertragungen mittels *NFS*. Erfahrungen zeigen jedoch, daß die erreichten Messwerte für das *NFS* von den Implementierungen der *NFS*–Server und den zugreifenden *NFS*–Clients abhängen und in einigen Fällen nur Durchsätze von 30 KByte/s erreicht werden. Die Ursache dafür liegt in der Quittierung der übertragenen Daten. Die Auswahl des Übertragungsprotokolls fällt wegen des höheren Datendurchsatzes auf das *ftp*–Protokoll.
Jedoch sind für die Konzeption noch einige Erweiterungen, welche nicht in einem Standard–*ftp*–Client implementiert sind, nötig.

3.7.1 Erweiterungen des *ftp*–Protokolls

Für die speziellen Aufgaben des zu entwickelnden Programms sind einige Erweiterungen des *ftp*–Clients vorzunehmen. Dazu gehören:

Größe (MB)	t_{ftp}(s)	Rate ftp (MB/s)	t_{NFS}(s)	Rate NFS (MB/s)
3,9	0,72	5,33	1,11	3,46
10	1,56	6,41	2,59	3,86
27	4,81	5,51	7,47	3,54

Tabelle 3.2: Vergleich der Übertragungsrate von *ftp* und *NFS*

- Möglichkeit zur Übertragung von einem Rechner zu einem anderen wobei die Steuerung der Datenübertragung von einem dritten Rechner übernommen wird.
- Möglichkeit zum Übertragen von Dateien über eine Firewall
- Verfahren zum rekursiven Kopieren von Verzeichnissen.

3.7.2 *ftp*–Daten–Verbindung zwischen zwei Servern

Bei einer normalen *ftp*–Verbindung verbindet sich immer ein Client mit einem Server und überträgt Daten entweder vom oder zum Server. Diese Vorgehensweise ist jedoch nicht besonders effektiv, wenn Daten von einem Server auf einen anderen übertragen werden sollen, weil dann zwei *ftp*–Verbindungen geöffnet werden müßten und die Datei zweimal zu übertragen wäre. Dies würde jedoch die Datenübertragungsrate halbieren. So muß zur effektiven Datenübertragung eine Möglichkeit geschaffen werden, um Daten von einem Server direkt auf einen anderen zu übertragen.
Eine Lösung des Problems sieht folgendermaßen aus (Vgl. Abbildung 3.2):

- Der Steuerungs–Rechner baut eine Kontroll–Verbindung zum Server 1 und zum Server 2 auf.
- Der Steuerungs–Rechner meldet sich auf beiden Servern an.
- Der Steuerungs–Rechner veranlaßt auf den Servern, in die gewünschten Verzeichnisse zu wechseln.
- Server 2 wird angewiesen, die Datei zu empfangen.
- Server 1 wird angewiesen, die Datei zu senden.
- Die Übertragung findet statt.
- Der Steuerungs–Rechner wartet, bis er vom Server 2 die Meldung erhält, daß die Daten angekommen sind.

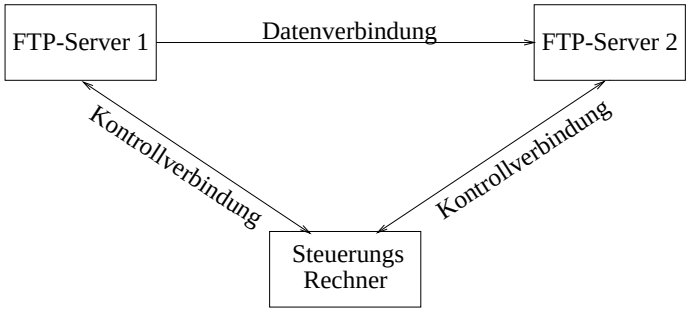

Abbildung 3.2: Dateiübertragung mit *ftp* von Server zu Server unter Kontrolle eines dritten Rechners

Der Nachteil bei diesem Vorgehen ist, man hat keinerlei Kontrolle, wieviel Daten bereits übertragen wurden. Erst wenn die Übertragung abgeschlossen ist oder ein Übertragungsfehler auftritt, erhält man eine Meldung. Dabei tritt das Problem auf, daß der Benutzer nicht einschätzen kann, wie lange die Übertragung dauern wird und ob die Daten überhaupt in annehmbarer Zeit übertragen werden. Es muß also eine Möglichkeit zur Fortschritts–Kontrolle der Übertragung ermöglicht werden.

3.7.3 Übertragung von Dateien über eine Firewall

Bei einer normalen Datenübertragung mit *ftp* wählt der Client einen freien Port aus und sendet mittels des `PORT`–Kommandos dem Server seine *IP*–Adresse sowie die soeben bestimmte freie Portnummer. Der Server öffnet daraufhin die Datenverbindung zu dem übermittelten Port des Client, wobei der Server immer den *TCP*–Port 20 benutzt.
Dieses Szenario ist, wenn der Client innerhalb einer Firewall (Vgl. [BDC96]) steht, nicht möglich. Das liegt daran, daß die Firewall den Versuch des Servers abweist, die Datenverbindung auf dem vom Client vorgegebenen Port zu öffnen. Eine Firewall läßt, wenn überhaupt, nur auf wenigen Ports Verbindungen von außen zu. Es wird also nötig, daß der Client die Datenverbindung aufbaut.
Für diese Zwecke gibt es das `PASV`–Kommando. Damit teilt der Client dem Server mit, daß er auf ein sogenanntes *passiv open* warten soll. Daraufhin teilt der Server dem Client seine IP–Adresse und den zu benutzenden

Port mit. Jetzt kann der Client die Datenverbindung öffnen, ohne daß die Firewall die Verbindung ablehnt.

3.7.4 Rekursives Kopieren von Verzeichnissen

Die ursprüngliche Spezifikation des *ftp*–Protokolls ([JP85]) sieht keine Möglichkeit zum rekursiven Kopieren von Verzeichnissen vor. Zur Erleichterung wird ein Verfahren zum rekursiven Kopieren mittels *ftp* implementiert. Das rekursive Kopieren läuft folgendermaßen ab:

- Eine vorher bestimmte Liste von Datei- und/oder Verzeichnisnamen der zu übertragenden Daten wird übergeben.
- Mittels *ftp* wird festgestellt, ob es sich um eine Datei bzw. ein Verzeichnis handelt.
- Handelt es sich um eine Datei, wird diese kopiert.
- Handelt es sich jedoch um ein Verzeichnis, wird auf dem Ziel–Rechner ein Verzeichnis mit gleichem Namen angelegt und
- rekursiv fortgesetzt

Zur Sicherung der Datenübertragung sind jedoch noch Vorkehrungen zu treffen. Das *ftp*–Protokoll stellt zwar eine gewisse Sicherheit bereit. Jedoch soll zur Erhöhung der Sicherheit ein weiteres Verfahren benutzt werden.

3.8 Sicherung der Datenübertragung

Zur Sicherung der Datenübertragung werden vor und nach jedem Dateitransfer Prüfsummen gebildet und miteinander verglichen. Dazu wird das weit verbreitete Prüfsummen–Programm `md5sum` eingesetzt. Dies wird nötig weil *TCP* zwar eine Fehlerkorrektur beinhaltet jedoch deren Sicherheit nicht ausreichend ist.

Das Prüfsummenprogramm `md5sum`

Das Programm `md5sum` beruht auf dem MD5–Message–Digest–Algorithmus. Der MD5–Algorithmus ist beschrieben in *RFC* 1321 ([Riv92]). Der Algorithmus nimmt als Eingabe eine Nachricht von beliebiger Länge und produziert daraus eine 128–Bit Prüfsumme, auch *fingerprint* oder *message digest* genannt. Der Vorteil dieses Algorithmus ist, daß es beinahe unmöglich ist, zwei Nachrichten zu generieren, welche die gleiche Prüfsumme haben, bzw.

zu einer vorgegebenen Prüfsumme eine Nachricht zu generieren. Daher wird eine fehlerhaft übertragene Datei nicht die gleich Prüfsumme haben wie die Datei vor der Übertragung.
Der Algorithmus zur Absicherung der Datenübertragung gliedert sich in fünf Teile:

1. Anhängen der Füllbits
   Die Nachricht wird erweitert, und zwar, bis ihre Länge (in Bits) kongruent zu 448 modulo 512 ist. Damit wird erreicht, daß die Nachricht nach Anhängen von weiteren 64 Bit ein Vielfaches von 512 Bits lang ist. Das Anhängen der Füllbits wird immer durchgeführt, auch wenn die Länge bereits kongruent zu 448 modulo 512 ist. Das Auffüllen geschieht wie folgt:

   - Anhängen einer Eins,
   - Anhängen weiterer Nullen, bis die gewünschte Länge erreicht ist.

2. Anhängen der Längenangabe
   Eine 64–Bit–Repräsentation der Länge der Nachricht wird an die Nachricht angehängt. Sollte die Länge der Nachricht den Wert 2^{64} überschreiten, werden nur die 64 niederwertigsten Bits benutzt.
   Zu diesem Zeitpunkt ist die Länge der resultierenden Nachricht ein Vielfaches von 512 Bits bzw. ein Vielfaches von 16 Wörtern (`32-bit-words`).
3. Initialisieren des MD Puffers
   Um das *message digest* zu berechnen, wird ein `four-word-buffer` (A,B,C,D) initialisiert, wobei A,B,C und D jeweils 32–bit Register sind. Diese Register werden wie folgt initialisiert:
   word A: 01 23 45 67
   word B: 89 ab cd ef
   word C: fe dc ba 98
   word D: 76 54 32 10
4. Bearbeiten der Nachricht in `16-word`-Blöcken
   Um die Prüfsumme zu generieren, wird eine Hauptschleife (Vgl. Abbildung 3.3) solange durchlaufen, bis die gesamte Datei abgearbeitet ist. Dabei werden jeweils 512 Bit der Nachricht bearbeitet.
   Die Hauptschleife besteht aus vier Runden. Die Funktionsweise der Runden verdeutlicht Abbildung 3.4. Für jede Runde wird eine Hilfsfunktion definiert, welche aus drei `32-bit-words` eine Ausgabe von einem `32-bit-word` generiert. Diese sind:

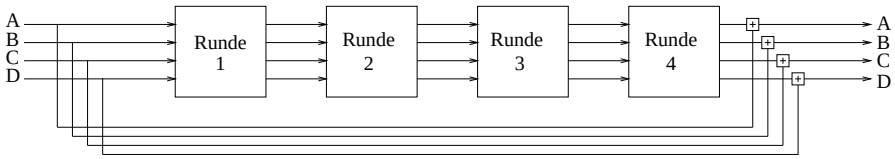

Abbildung 3.3: Die Hauptschleife des MD5–Algorithmus

$F(X,Y,Z) = XY \vee \neg(X)Z$
$G(X,Y,Z) = XZ \vee Y\neg(Z)$
$H(X,Y,Z) = XxorYxorZ$
$I(X,Y,Z) = Yxor(X \vee \neg(Z))$
Die Runden sehen folgendermaßen aus:

- Anwenden der nichtlinearen Funktion (F,G,H oder I) auf b,c und d,
- Addieren von a,
- Addieren des j–ten Teilblocks der Nachricht,
- Addieren einer rundenabhängigen Konstante (Vgl. [Riv92]),
- Linksrotieren des so erhaltenen `32-bit-words` und
- Addieren von b.

Dieser Vorgang wird 16 mal ausgeführt.
Nachdem alle vier Runden absolviert sind, werden die Werte von a, b, c und d zu A, B, C und D addiert.

5. Ausgabe der Prüfsumme
   Die Prüfsumme wird dann aus den vier Ausgaben A, B, C und D generiert. Die Ausgabe beginnt mit dem niederwertigsten Bit von A und endet mit dem höchstwertigen Bit von D

Nachdem die Daten sicher auf einen Verarbeitungs–Rechner übertragen sind wird ein Möglichkeit benötigt, auf entfernten Rechnern Programme zu starten.

3.9 Vergleich der Verfahren zum *remote login*

Um die Daten auf zentralen Rechnern zu konvertieren und die Übertragung zu kontrollieren wird eine Möglichkeit benötigt, sich auf anderen Rechnern

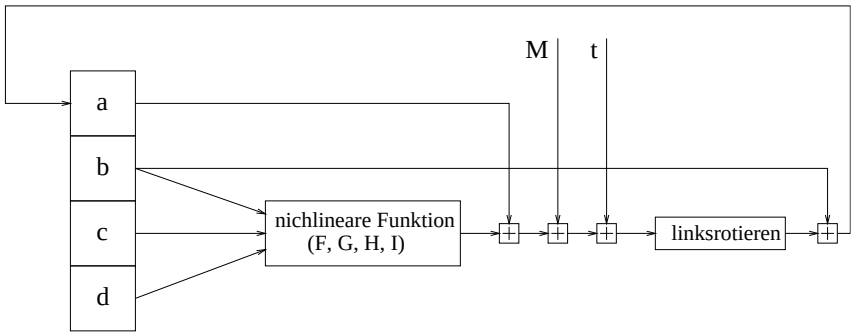

Abbildung 3.4: Eine Runde des MD5–Algorithmus

anzumelden und dort einige Befehle auszuführen.
Zu den gegenwärtig populärsten Internet–Anwendungen gehört das *remote login*. Zu diesem Zwecke gibt es zwei Standard–Programme sowie proprietäre Lösungen. Diese sind:

telnet eine Standard–Applikation, die in beinahe allen *TCP/IP* Implementierungen anzutreffen ist. Sie arbeitet auch mit Rechnern verschiedener Betriebssysteme. *telnet* verhandelt dabei die Optionen (*option negotiation*) zwischen Client und Server, um einen kleinsten gemeinsamen Nenner der Fähigkeiten beider Rechner zu bestimmen. (Vgl. Kapitel 2.5.1)

Rlogin eine Applikation, die aus dem *Berkeley–UNIX* stammt und entwickelt wurde, um zwischen *UNIX*–Systemen zu arbeiten. Dabei sind keine Options–Verhandlungen notwendig. (Vgl. Kapitel 2.5.2)

proprietärer Client und Server ein Client mit zugehörigem Server, der kein Standard–Protokoll benutzt

Bei der Implementierung des *rlogin*–Protokolls trat jedoch das Problem auf, daß es ohne eine Sicherheitslücken zu hinterlassen, unmöglich ist, mit der Programmiersprache *JAVA* einen *rlogin*–Client zu implementieren. Das hat folgenden Grund:
Der *rlogin*–Client benötigt zu seiner Ausführung auf einer *UNIX*–Maschine Systemadministratoren–Rechte. Das liegt daran, daß das *rlogin*–Protokoll einen privilegierten Port benutzt, zu dessen Benutzung man die Benutzer–ID 0 haben muß, also Systemadministrator (`root`) sein muß. Damit jeder

Benutzer des Programms `root`–Rechte hat, muß das Programm dem Systemadministrator gehören und zusätzlich muß das *SetUID*–Bit gesetzt sein. Ob das Bit gesetzt ist, erkennt man an dem `s` im ersten Byte der Dateirechte.

```
-rwsr-xr-x 1 root root  10088 Oct 16 14:14 /usr/bin/rlogin
   ^
```

Aufgrund der Architektur von *JAVA* ist es nicht möglich, einzelnen *JAVA*–Klassen dieses Recht einzuräumen. Man müßte dem *JAVA*–Interpreter die Systemadministratorenrechte geben. Damit wäre es aber möglich, das System zu beschädigen oder Zugriff auf Daten zu erlangen, die dem Datenschutz unterliegen (etwa Patientendaten). Das dadurch entstehende Sicherheitsrisiko wäre nicht akzeptabel, so daß von diesem Protokoll wieder Abstand genommen wurde.
rlogin ist außerdem ein Protokoll, welches nur unter *UNIX* implementiert ist und deshalb dem plattformübergreifenden Ansatz der Konzeption zuwiderläuft.
Als dritte Möglichkeit zum *remote login* bietet sich ein proprietäres Client–Server–Protokoll an. Das Problem bei einem proprietären Client und Server ist, daß auf jedem der beteiligten Konvertier–Rechner der Server laufen müßte.
Bei *UNIX*–Systemen hat jede Datei einen Besitzer sowie Zugriffsrechte (Vgl. Kapitel 2.1.3). Die Zugriffsrechte entscheiden, wer die Datei lesen, schreiben oder ausführen darf. Dies führt zu einem weiteren Problem. Es kann passieren, daß der Benutzer der Datei die Rechte so vergeben hat, daß außer ihm keiner die Datei lesen kann. Weil das Konvertierprogramm mit einem anderen Benutzer–ID laufen muß, kann es dazu kommen, daß der Server keine Rechte hat, die Datei zu lesen, also auch keine Möglichkeit, die Daten zu konvertieren. Das liegt daran, daß der Server–Prozess nur genau einen Besitzer haben kann, aber Rechte für alle Benutzer haben müßte.
Genauso könnte es dazu kommen, daß das Konvertierprogramm zwar die Daten lesen und konvertieren kann, aber der Benutzer keine Rechte hat, um die konvertierte Datei vom Konvertier–Rechner zurückzukopieren. Eine Lösung wäre, dem Server–Programm Systemadministratoren–Rechte zu geben und nach jedem Konvertier–Vorgang den Benutzer der Datei (mittels des *UNIX*–Kommandos `chown` ([cho])) zu verändern. Aber auch hier taucht das Problem der Sicherheit auf. So könnten geschickte Angreifer die Rechte des Programms ausnutzen, um auf Rechnern mit aktivem Server Systemadministratoren–Rechte zu erlangen.

Bewertung der Verfahren zum Remote–Login

Aufgrund der Sicherheitsprobleme bei *rlogin* und einem proprietären Client und Server fiel die Auswahl auf das sicherere und kompliziertere *telnet*–Protokoll.
Somit steht ein Verfahren zur Verfügung, Programme auf entfernten Rechnern zu starten. Jetzt soll analysiert werden, was nötig ist, um die einzelnen Verarbeitungs–Schritte durchzuführen.

3.10 Die Dateiverarbeitungsvorgänge

Zu den Dateiverarbeitungsvorgängen zählen:

- der Kopier–Vorgang,
- der Beweg–Vorgang,
- der Lösch–Vorgang und
- die verschiedenen Konvertier–Vorgänge
- der `tar`–Vorgang

Diese werde in den folgenden Abschnitten näher betrachtet.

3.10.1 Der Kopier–Vorgang

Der Kopier–Vorgang dient zum Kopieren von Dateien auf andere Rechner oder in andere Verzeichnisse. Er gliedert sich in drei Teile:

Generierung der Prüfsumme

Es wird eine *telnet*–Verbindung zum Quell–Rechner aufgebaut. Dort wird dann mit dem Programm `md5sum` die Prüfsumme für die Datei erstellt und zwischengespeichert.

Die eigentliche Datenübertragung

Die Datenübertragung erfolgt mit *ftp*. Es werden vom Client aus zwei *ftp*–Verbindungen aufgebaut, die erste zum Quell–Rechner, die zweite zum Ziel–Rechner. Auf den beiden Rechnern wird in die gewünschten Verzeichnisse gewechselt. Anschließend wird an den Quell–Rechner das `PASV`–Kommando

gesendet und die so erhaltene Port–Nummer mit dem `PORT`–Kommando an den Ziel–Rechner geschickt. Jetzt wird der Ziel–Rechner angewiesen, die Datei zu empfangen und der Quell–Rechner die Datei zu senden. Die Übertragung findet statt.

Generierung der zweiten Prüfsumme

Es wird wiederum eine *telnet*–Verbindung aufgebaut, diesmal jedoch zum Ziel–Rechner und erneut mit dem Programm `md5sum` eine Prüfsumme der übertragenen Datei erstellt und mit der Prüfsumme der ursprünglichen Datei verglichen. Sind sie identisch, ist der Vorgang erfolgreich beendet, ansonsten muß eine Fehlermeldung ausgegeben werden.

3.10.2 Der Beweg–Vorgang

Der Beweg–Vorgang ist ein Kopier–Vorgang, bei dem nach jeder Übertragung und Kontrolle der Übertragung die Datei mittels der noch geöffneten *ftp*–Verbindung und dem `DELE`–Kommando gelöscht wird.

3.10.3 Der Lösch–Vorgang

Beim Lösch–Vorgang wird eine *telnet*–Verbindung genutzt. Nachdem die Verbindung geöffnet ist, wird in das Quell–Verzeichnis gewechselt und für jedes ausgewählte Listenelement (Datei oder Verzeichnis) ein `rm` (*UNIX*–Befehl zum Löschen von Dateien oder Verzeichnissen) mit den Optionen `-rf` gestartet. Die Optionen bedeuten:

`r` Löschen aller Unterverzeichnisse, falls vorhanden und

`f` Löschen ohne nochmalige Nachfrage beim Benutzer.

Die Nachfrage ist bereits vor dem eigentlichen Löschvorgang, innerhalb des Programms erfolgt.

3.10.4 Der Konvertier–Vorgang

Der Konvertier–Vorgang ist der komplexeste unter den Datei–Aktionen. Er besteht aus einem Kopier–Vorgang, dem eigentlichen Konvertieren, und einem Beweg–Vorgang:

- Bestimmen der ersten Prüfsumme der Ausgangsdatei,
- Übertragen der Ausgangsdatei auf den Rechner, auf dem die Daten konvertiert werden sollen,
- Bestimmen der zweiten Prüfsumme der Ausgangsdatei und Vergleich mit erster Prüfsumme,
- Starten des Konvertierprogramms,
- Bilden der ersten Prüfsumme der konvertierten Datei,
- Bewegen der konvertierten Daten auf den Ziel–Rechner und
- Bestimmen der zweiten Prüfsumme der konvertierten Datei und vergleichen mit ersterer.
- Löschen der temporären Dateien auf dem Konvertier–Rechner

3.10.5 Der `tar`–Vorgang

Der `tar`–Vorgang ist ein spezieller Konvertier–Vorgang, bei dem mehrere Dateien und Verzeichnisse zu einer Datei zusammengefasst werden können. Es wird eine *telnet*–Verbindung zum Quell–Rechner geöffnet, in das aktuelle Verzeichnis gewechselt und die `tar`–Datei erzeugt. Dies geschieht mit folgender Kommandozeile:

```
tar cf Dateiname.tar Datei1 Datei2 Verzeichnis1 ...
```

Wobei Dateiname.tar der Name der zu erzeugenden Datei ist und Datei1 Datei2 bzw Verzeichnis1 die zusammenzufasenden Dateien und Verzeichnisse.
Bevor die Verarbeitungsvorgänge zur Archivierung näher betrachtet werden noch einige Betrachtungen zu den die Archivierung betreffenden Verfahren.

3.11 Möglichkeiten zur Datenarchivierung

Die Archivierung spielt bei großen Datenmengen eine wichtige Rolle, da die Datenmenge ohne externe Archivierung bald die gesamte Kapazität der Festplatten überschritten hätte. Somit ist es nötig, geeignete Verfahren zur Datenarchivierung einzusetzen. Zur Archivierung bieten sich zur Zeit zwei Medien an:

- CD (Kapazität zwischen 650 und 700MB)
- Magnetbänder z.B. DLT–Tapes (Kapazität zwischen 10 und 35 GB)

Neben diesen etablierten Archivierungsmedien gibt es noch einige andere, die auf den ersten Blick geeignet erschienen. Dazu gehört die *Digital Versatile Disc* (*DVD*). Bisher gibt es allerdings kaum Software, die dieses Medium unterstützt und selbst die Recorder Hardware (*DVD–RAM*) ist bisher nicht lieferbar. Ein weiteres Problem der *DVD* ist die Unklarheit des Kopierschutzes. Deshalb kann *DVD* nicht in die Betrachtungen einbezogen werden.

Verfahren zur Sicherung auf Magnetband

Zur Sicherung von Daten auf ein Magnetband bieten sich im *UNIX*–Umfeld drei standardisierte Verfahren an.

- `tar`
- `dump/restore`
- `cpio`

Das Programm `tar`

Das `tar`–Programm (tape archiver) ([tar]) ist ein Standard–Programm zur Sicherung von Daten auf Magnetbändern. Es ist unabhängig von der benutzten Hardware und dem benutzten Medium.

Die Programme `dump/restore`

Die Programme `dump` und `restore`([rdu]) erlauben es, gesamte Dateisysteme auf Magnetband zu schreiben und wieder zurückzuladen. Darin liegt aber der Nachteil der beiden Programme. Es können keine einzelnen Dateien oder Verzeichnisse auf Magnetband geschrieben werden, sondern nur gesamte Dateisystem. Damit einher geht das Problem, daß `dump/restore` für ihre Arbeit Informationen aus den Tiefen des Dateisystems benötigen und sich deshalb die Implementierungen von Dateisystem zu Dateisystem unterscheiden.

Das Programm `cpio`

Das Programm `cpio` ([cpi]) ist mit dem `tar`–Programm verwandt es schreibt Dateien mit den zugehörigen Dateiinformationen (Dateiname, Besitzer, Zeitstempel und Zugriffsrechte) in eine Archiv–Datei. Der Unterschied zum `tar`–Programm besteht darin, daß `cpio` Dateinamen mit mehr als 100 Zeichen unterstützt.

Das Hilfs–Programm `mt`

Auf einem Magnetband werden die Datensätze hintereinander auf das Band geschrieben. Somit ist es nötig, um einen anderen als den ersten Datensatz zu lesen, das Magnetband an die richtige Stelle zu positionieren. Als Hilfsmittel für den Umgang mit Magnetbändern benötigt man ein Programm namens `mt` [mt-] (*magnetic tape manipulating program*). Dieses Programm dient dazu:

- das Magnetband zurückzuspulen und
- die Position des Lese–Schreib–Kopfes auf dem Band zu verändern.

Befindet sich das Band in der gewünschten Position, kann mit dem Lesen oder Schreiben des Bandes begonnen werden.

Auswahl des Verfahrens zur Sicherung auf Magnetband

Ein wichtiges Kriterium für die Sicherung von Daten auf Magnetband ist die Möglichkeit, Daten, die von Rechner A auf Magnetband geschrieben wurden auch auf einem Rechner B, welcher nicht das gleiche Dateisystem besitzt wieder zurückschreiben zu können. Auf Grund der fehlenden Universalität der Programme `dump/restore`, kommen die Programme `tar` und `mt` zum Einsatz.

Verfahren zum Erstellen von CD-ROMs

Um eine CD zu erstellen, sind drei Schritte erforderlich.

- gesicherte Übertragung der Daten auf einen Rechner mit CD–Brenner,
- Generieren eines Abbildes (*Image*) der CD aus den vorgegebenen Dateien und Verzeichnissen und
- Schreiben des Abbildes auf den CD–WORM–Rohling.

Zum Erstellen des Abbildes der CD und zum Brennen unter *UNIX* existieren folgende Programme:

- `mkisofs` ([You]), zum Erzeugen des Abbildes,
- `cdrecord` ([Sch]), zum Schreiben des Abbildes auf CD–WORM–Rohling.

3.12 Die Archivierungsvorgänge

Zu den Archivierungsvorgängen gehören:

- Erstellen einer CDROM
- Speichern auf Magnetband
- Lesen vom Magnetband

3.12.1 Der CD–Brenn–Vorgang

Der CD–Brenn–Vorgang gliedert sich in drei Teile:

- Übertragen der Daten auf den Brenner–Rechner mittels *ftp*–Protokoll,
- Erstellen des Abbildes der CD mittels `mkisofs` und *telnet*–Protokoll und
- Brennen der CD mittels `cdrecord` und *telnet*–Protokoll.

3.12.2 Der Magnetband–Schreib–Vorgang

Der Magnetband–Schreib–Vorgang gliedert sich in drei Teile:

- Übertragen der Daten auf den Rechner mit Bandlaufwerk mittels *ftp*–Protokoll
- Positionieren des Bandes mittels `mt` und *telnet*–Protokoll und
- Beschreiben des Bandes mittels `tar` und *telnet*–Protokoll

3.12.3 Der Magnetband–Lese–Vorgang

Der Magnetband–Lese–Vorgang gliedert sich in drei Teile:

- Positionieren des Bandes mittels `mt` und *telnet*–Protokoll,
- Auslesen des Bandes mittels `tar`und *telnet*–Protokoll und
- Übertragen der Daten auf den Ziel–Rechner mittels *ftp*–Protokoll

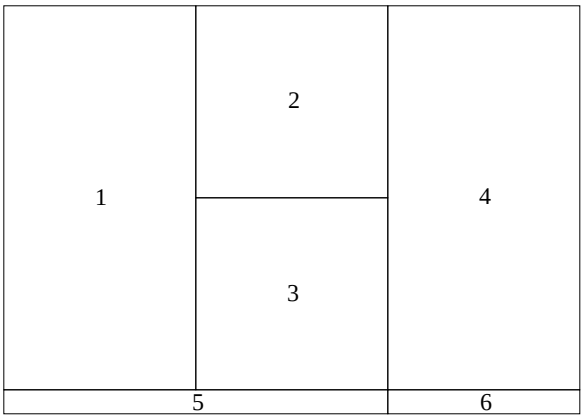

Abbildung 3.5: Der prinzipielle Aufbau der Benutzeroberfläche

3.13 Die Optionen der Verarbeitungs–Vorgänge

Einige Verarbeitungsvorgänge benötigen weitere Optionen, bevor diese ausgeführt werden können. Die Verarbeitungsvorgänge mit den entsprechenden Optionen sind:

Konvertier–Vorgang	Kommandozeilen–Optionen für das Konvertierprogramm
`tar`–Vorgang	Name der zu erzeugenden `tar`–Datei
Magnetband–Schreib–Vorgang	Name des Rechners mit Magnetband–Laufwerk sowie das zu benutzende Device
Magnetband–Lese–Vorgang	Name des Rechners mit Magnetband–Laufwerk sowie das zu benutzende Device

Nachdem alle Verarbeitungsvorgänge näher untersucht wurden, wird es nötig, den Aufbau der grafischen Benutzeroberfläche zu spezifizieren.

3.14 Die grafische Benutzeroberfläche

Die Benutzeroberfläche gliedert sich in sechs Teile. (Vgl. Tabelle 3.3 und Abbildung 3.5). Dabei sind nicht alle Bereiche ständig aktiv, sondern wer-

Nr.	Name	Beschreibung
1	Quell–Bereich	Übersicht des aktuellen Verzeichnis des Quell–Rechners
2	Aktions–Bereich	Bereich zum Auswählen der Aktionen
3	Options–Bereich	Auswahlbereich für weitere, von der gewählten Aktion abhängige Optionen
4	Zielbereich	Übersicht des aktuellen Verzeichnis des Ziel–Rechners
5	Statuszeile	dient zur Ausgabe von Status- und Hilfsinformationen
6	Beenden–Taste	Taste zum Beenden des Programmes

Tabelle 3.3: Die Bereiche der grafischen Benutzeroberfläche

den nach Bedarf eingefügt oder wieder gelöscht.

Der Quell–Bereich (1) gliedert sich in eine Rechner–Auswahlbox und eine Übersicht des aktuellen Verzeichnisses.

Der Aktions–Bereich (2) gliedert sich in drei Teile: den Teil für die Datei–Aktionen (Kopieren, Bewegen und Löschen), den Teil für die Konvertierprogramme (abhängig von den ausgewählten Dateien) und den Teil für die Datensicherungs–Aktionen.

Der Options–Bereich (3) ist abhängig von den ausgewählten Aktionen. Im einfachsten Falle enthält der Options–Bereich nur eine Taste, um die ausgewählte Aktion zu starten. Der Options–Bereich kann aber auch weitere Optionen wie z.B. Optionen zu Konvertierprogrammen oder Informationen über Magnetband–Laufwerke enthalten.

Der Ziel–Bereich (4) gliedert sich, wie der Quell–Bereich, in Rechner–Auswahlbox und eine Übersicht des aktuellen Verzeichnisses.

Die Statuszeile (5) liefert in Abhängigkeit vom derzeitigen Programm–Zustand eine kleine Hilfe.

Die Beenden–Taste (6) dient zum Beenden des Programms.

Die Bereiche der grafischen Benutzeroberfläche sind jedoch nicht immer belegt. Dies dient der Übersichtlichkeit. So soll dafür gesorgt werden, daß Bereiche, die bei einer entsprechenden Aktion keine Belegung haben müssen, auch nicht belegt werden. Der Quell– und der Aktions–Bereich sind ständig

belegt. Der Options–Bereich wird erst aktiv, wenn eine Aktion ausgewählt wurde. Er enthält die in Kapitel 3.13 beschriebenen Optionen sowie eine Taste zum Auslösen des Vorgangs. Der Ziel–Bereich ist bei folgenden Vorgängen aktiv:

- Kopier–Vorgang,
- Beweg–Vorgang,
- Konvertier–Vorgang,
- `tar`–Vorgang und
- Magnetband–Lese–Vorgang.

Als Abschluss der Konzeption steht die Auswahl der Programmiersprache, welche zur Implementierung herangezogen werden soll.

3.15 Vergleich der Programmiersprachen

Zur Programmierung in heterogenen Netzwerken ist nicht jede Programmiersprache gleich gut geeignet. Bei der Auswahl ist zu berücksichtigen, daß die Programmiersprachen auf allen (wenigstens den wichtigsten) Rechnerplattformen implementiert sein sollten. Die Programmiersprache sollte eine Möglichkeit zur einfachen Netzwerk–Programmierung anbieten und einfache Möglichkeiten zur Programmierung von grafischen Benutzeroberflächen zur Verfügung stellen.
Daher fiel eine Vorauswahl auf: *C/C++* ([Str97]), *JAVA* ([Fla97]) und *Perl* ([LW96]).

C/C++

- Vorteile:
  - ausgereifte Programmiersprache,
  - erzeugt schnellen Code
- Nachteile:
  - plattformabhängig, d.h. Programme müssen auf jedem Betriebssystem neu übersetzt werden,

  - komplizierte, unsichere und fehleranfällige Programmierung durch Möglichkeit des direkten Speicherzugriffs und Komplexität der Sprache.

JAVA

- Vorteile:
  - plattformunabhängig,
  - einfach, weil viele Bibliotheken zur Vereinfachung der Programmierung bereits im *JAVA Development Kit* enthalten sind,
  - erzeugt sicheren Code,
  - einfache Netzwerk–Programmierung durch umfangreiche Programmier–Bibliotheken,
  - strenge Objektorientierung.
- Nachteile:
  - langsam durch Interpretation des *JAVA–Byte–Code*,
  - relativ neue Programmiersprache

Ausführlichere Informationen zur Programmiersprache *JAVA* in Kapitel 2.6.

Perl/TK

- Vorteile:
  - auf allen relevanten Plattformen verfügbar,
  - einfache Programme lassen sich einfach umsetzen
  - ausgereift,
  - umfangreiche Bibliotheken für fast jede Aufgabe
- Nachteile:
  - Inkompatibilitäten zwischen verschiedenen Versionen von *Tk*
  - geringe Ausführungsgeschwindigkeit durch Interpreter

3.15.1 Geschwindigkeitsvergleich zwischen den in Frage kommenden Programmiersprachen

Um die Geschwindigkeit von *JAVA*–Byte-Code, *C*–Code und *Perl*–Skripten vergleichen zu können, ist es nötig, zwei verschiedene Szenarien zu betrachten.

1. rechenintensive Verfahren
2. nicht–rechenintensive Verfahren

Diese Unterscheidung ist notwendig, da nicht–rechenintensive Programme (z.B. mit grafischer Benutzeroberfläche oder Programme, welche auf Komponenten zurückgreifen, die Daten nicht so schnell liefern können, wie sie verarbeitet werden könnten) den Prozessor nicht bis an seine Grenzen belasten. Als Beispiel für ein rechenintensives Verfahren wird die Berechnung von π mittels des Leibnitz–Verfahrens herangezogen. Zum Vergleich der Programmiersprachen bei nicht–rechenintensiven Verfahren wird der in Kapitel 4.1 beschriebene *JAVA–ftp*–Client sowie ein *C–ftp*–Client und ein *Perl–ftp*–Client benutzt.

Das Leibnitz–Verfahren zur Berechnung von π

Zur Berechnung von π gibt es eine Vielzahl von Möglichkeiten. Eine davon ist das Leibnitz–Verfahren. G. W. Leibnitz fand im Jahre 1763 folgende Gleichung:

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} \ldots \tag{3.1}$$

Die Quelltexte für das Leibnitz–Verfahren zur Berechnung von π sind im Anhang zu finden. Zusätzlich zu den drei Programmiersprachen wird noch eine Messreihe mit einem *just–in–time–Compiler* aufgenommen. Dieser sorgt dafür, dass *JAVA*–Code nicht interpretiert wird, sondern vor der Ausführung in Maschinen–Code übersetzt wird. Dadurch kann es zu erheblichen Verbesserungen der Geschwindigkeit kommen.
Als Messplattform stand ein *Linux*–PC mit AMD K6-2 (233 MHz, 64MB RAM) zur Verfügung. Folgende Compiler/Interpreter–Versionen kamen zum Einsatz:

C	gcc Version 2.7.2.3
Perl	5.004_04
JAVA	JDK Version 1.1.7
JIT	TYA Version 1.4

Schleifen	t_C	t_{Perl}	t_{JAVA}	t_{JIT}
10^7	2.2	56.9	13.4	4.2
10^8	22.2	502.8	131.2	38.1
10^9	220.7	5522.1	1307.6	378.3

Tabelle 3.4: Vergleich der Geschwindigkeiten verschiedener Implementierungen des Leibnitz–Verfahrens (Alle Zeitangaben (t_C,t_{Perl},t_{JAVA} und t_{JIT}) in Sekunden)

Geschwindigkeits–Vergleich der rechenintensiven Anwendung

Die Messergebnisse des Geschwindigkeitsvergleich zum Leibnitz–Verfahren sind in Tabelle 3.4 zu finden. Die Ergebnisse zeigen, daß compilierter *C*–Code die besten Ergebnisse liefert. Knapp doppelt so lange benötigt der *JAVA*–Code welcher mit dem *JIT*–Compiler ausgeführt wurde. Der interpretierte *JAVA*–Bytecode folgt auf dem dritten Rang mit einer Ausführungszeit, die etwa 6–mal so lange ist wie die *C*–Ausführungsgeschwindigkeit. Die längste Ausführungszeit benötigte das interpretierte *Perl*–Skript. Es benötigt etwa 20-40 mal so lange wie der compilierte *C*–Code.
Nachdem bei einer rechenintensive Anwendung die Geschwindigkeit verglichen wurde, steht die Betrachtung einer nicht–rechenintensiven Anwendung im Vordergrund.

Ergebnisse des Geschwindigkeits–Vergleich der nicht–rechenintensiven Anwendung

Bei der Geschwindigkeitsmessung für eine nicht–rechenintensive Anwendung wurden drei Dateien mit einer Größe von 10MB, 100MB und 1GB mittels *ftp*–Protokoll übertragen. Dazu wurden *ftp*–Clients in den Programmiersprachen *JAVA* und *Perl* implementiert. Die Implementierungen sind im Anhang zu finden. Für die Implementierung in *C* wurde auf das *UNIX*–Standard–Programm `ftp` zurückgegriffen.
Die Ergebnisse der Messungen sind in Tabelle 3.5 zusammengefasst. Die Geschwindigkeits–Messungen zeigen, daß bei netzwerkintensiven Anwendungen (z.B. das Übertragen von Dateien mittels *ftp*–Protokoll) keine der Implementierungen einen Vorteil gegenüber den anderen vorweisen kann. Das liegt daran, daß bei netzwerkintensiven Anwendungen nicht der Prozes-

Größe	t_C	t_{Perl}	t_{JAVA}	t_{JIT}
10 MB	12.8	13.0	13.3	11.6
100 MB	110.0	106.2	116.0	107.4
1 GB	1080	1094	1103	1101

Tabelle 3.5: Vergleich der Geschwindigkeiten verschiedener Implementierungen des *ftp*–Protokolls (Alle Zeitangaben (t_C,t_{Perl},t_{JAVA} und t_{JIT}) in Sekunden)

sor der minimierende Faktor ist, sondern die Netzwerkkarte. Da die Abweichungen der Messergebnisse alle innerhalb der Fehlergrenzen (10 %) liegen, sind diese als gleichwertig anzusehen.

3.15.2 Auswahl der Programmiersprache

Aufgrund der im Vergleich zu *C* besseren Plattformunabhängigkeit, sichereren Programmierung und der im Vergleich zu *Perl* höheren Arbeitsgeschwindigkeit fiel die Auswahl der Programmiersprache auf *JAVA*.

Kapitel 4

Implementierung der Konzeption

Dieses Kapitel beschäftigt sich mit der Umsetzung der in der Konzeption gefundenen Ergebnisse in *JAVA*–Programmcode.

4.1 Implementierung des *ftp*–Clients

Die Implementierung des *ftp*–Clients gliedert sich in zwei Teile.

1. Implementierung eines *ftp*–Clients nach der *RFC* 959 ([JP85]) und
2. Implementierung der anwendungsspezifischen Besonderheiten

4.1.1 Implementierung eines *ftp*–Client nach *RFC* 959

Die grundlegenden Funktionen eines *ftp*–Clients nach *RFC* 959 sind:

- Öffnen der Kontroll–Verbindung zum *ftp*–Server
  Diese Aufgabe übernimmt der Konstruktor der Klasse `ftp.java`

```
// Bestimmung der IP-Adresse des Hostes
hostaddr=InetAddress.getByName(host);
// Oeffnen der Verbindung zum ftp-Port
ftpsocket= new Socket(hostaddr,21);
// Zuordnung der Ein- und Ausgangsstroeme
outstream = new PrintWriter(ftpsocket.getOutputStream());
inputstream = new DataInputStream(ftpsocket.getInputStream());
```

Es wird ein Socket zum *ftp*–Server auf *ftp*–Port 21 geöffnet und die Ein– und Ausgabeströme zugewiesen.

- Anmelden des Benutzers
  Das Anmelden übernimmt die Methode *open*

```
public int open(String user, String pass){
  command("user "+user);
  int result=command("pass "+pass);
  return result;
}
```

- Übertragen eines Kommandos an den *ftp*–Server
  Zu diesem Zwecke existieren zwei Methoden: `command` und `realCommand`. Die Methode `realCommand` reicht das übergebene Kommando direkt an den *ftp*–Server weiter, nachdem ein `carriage return/linefeed`–Paar angehängt wurde.

```
public int realCommand(String command){
  command=command+"\r\n";
  // senden des Kommandos
  outstream.println(command);
  outstream.flush();
  // auslesen der Antwort
  int code=getreply(inputstream);
  return code;
}
```

  Die Methode `command` übernimmt die Kommandos: `LIST` (Anzeigen des Verzeichnisinhaltes), `STOR` (übertragen einer Datei zum *ftp*–Server) und `RETR` (übertragen einer Datei vom *ftp*–Server) und führt, je nach Kommando, die entsprechende Methode aus.

- Empfangen der Antworten des *ftp*–Servers
  Zum Empfangen der Antworten des Servers sind zwei Methoden implementiert: `getStringReply` und `getreply`. `getStringReply` liefert die gesamte Ausgabe, ohne diese auszuwerten.

```
private String getStringReply(DataInputStream inputstream){
 String SocketOutput;
 // warten, bis Daten eintreffen
  do {
    count++;
  }
```

```
  while(inputstream.available()==0);
  // auslesen der Ausgabe
  SocketOutput=inputstream.readLine();
  return (SocketOutput);
}
```

`getreply` liefert nicht die komplette Antwort zurück, sondern nur die erste Ziffer des *Reply–Codes* (Vgl. Tabelle 2.5).

```
do {
  SocketOutput=inputstream.readLine();
}
// test ob Ausgabe mit einer dreistelligen Zahl beginnt.
while(!(Character.isDigit(SocketOutput.charAt(0)) &&
          Character.isDigit(SocketOutput.charAt(1)) &&
          Character.isDigit(SocketOutput.charAt(2)) &&
          SocketOutput.charAt(3)==' '));
value=Integer.parseInt(SocketOutput.substring(0,1));
// rueckgabe der ersten Ziffer
return (value);
```

Damit läßt sich feststellen, ob ein Kommando erfolgreich war, oder ob ein Fehler aufgetreten ist.

- Bestimmen des Ports für die Daten–Verbindung.
  Die Datenverbindung wird für jede zu übertragende Datei neu geöffnet. Normalerweise geschieht dies mit dem `PORT`–Kommando. Sollen Dateien von einem Server zu einem anderen übertragen werden, wird das `PASV`–Kommando benutzt. Der *ftp*–Client sendet an den *ftp*–Server das `PASV`–Kommando. Daraufhin sendet der *ftp*–Server sechs Zahlen: seine eigene *IP*–Adresse (die ersten vier Zahlen) sowie den für die Daten–Verbindung zu benutzenden Port.

```
// senden des PASV-Kommandos
outstream.println("PASV");
outstream.flush();
// Lesen der Antwort bis diese mit 227 beginnt
do {
  pasv=inputstream.readLine();
  }
while(!((pasv.charAt(0)=='2')&&(pasv.charAt(1)=='2')&&(pasv.charAt(2)=='7')));
// parsen der Antwort
begin=pasv.indexOf("(")+1;
end=pasv.indexOf(")");
data=pasv.substring(begin,end);
```

```
StringTokenizer parse=new StringTokenizer(data,",");
// Bestimmen der IP-Adresse und des Ports
while(parse.hasMoreTokens()){
  if (count<4) {
    addr=addr+parse.nextToken()+".";
    }
  else {
    dataport[count-4]=Integer.parseInt(parse.nextToken());
    }
  count++;
  }
addr=addr.substring(0,addr.length()-1);
port2=dataport[0]*256+dataport[1];
}
```

- Übertragen einer Datei vom *ftp*–Client zum *ftp*–Server
  Das Übertragen einer Datei vom Client zum Server übernimmt die Methode `upload`. Zuerst wird auf dem Client in das gewünschte Verzeichnis gewechselt. Anschließend wird das Kommando `TYPE I` an den Server geschickt, um eine binäre Übertragung zu vereinbaren. Danach wird eine Datenverbindung aufgebaut und über die Datenverbindung das Kommando zum Speichern einer Datei (`STOR`) abgeschickt. Die Datenübertragung beginnt. In 1024–Byte–Blöcken wird die Datei übertragen. Zum Abschluß wird die Datenverbindung geschlossen.

```
// Aufbau der Verbindung
int port=pasvinit(controlin,controlout);
try {
  dataSocket=new Socket(hostname,port);
}
catch (IOException e){
  System.err.println("could not get port: "
                     +dataSocket.getLocalPort()+", "+e);
}
// Setzen des Uebertragungsmodus auf binaer
controlout.println("TYPE I");
controlout.flush();
getreply(controlin);
// Absetzen des STOR--Kommandos
controlout.println("STOR "+filename);
controlout.flush();
int result=getreply(controlin);
//Uebertragen der Datei
if (result==1){
  try {
    OutputStream outdataport = dataSocket.getOutputStream();
```

```
    byte b[]=new byte[1024];
    filename=dir+filename;
    RandomAccessFile infile;
    infile =new RandomAccessFile(newdir+filename, "r");
    int amount;
    while ((amount =infile.read(b))>0){
      outdataport.write(b,0,amount);
    }
    infile.close();
    outdataport.close();
    dataSocket.close();
    result=getreply(controlin);
  }
  catch (IOException e){
    e.printStackTrace();
  }
}
else{
  System.err.println("Error calling for upload");
}
return result;
```

- Übertragen einer Datei vom *ftp*–Server zum *ftp*–Client
  Diese Aufgabe übernimmt die Methode `download`. Dabei müssen zwei Varianten unterschieden werden. Entweder sollen die übertragenen Daten in eine Datei geschrieben werden (beim Befehl `RETR`) oder direkt ausgegeben werden (beim Befehl `LIST`). Die Variable `save` gibt an, ob die Datei gespeichert werden soll oder nicht. Sollen die Daten gespeichert werden, wird das Übertragungs–Verfahren auf *binär* gesetzt und der Dateiname bestimmt. Anschließend werden die Datenverbindung geöffnet, die Daten entweder in die Datei oder in die Variable `listreply` geschrieben und die Datenverbindung wieder geschlossen.

```
// Oeffnen der Verbindung
int port=pasvinit(controlin,controlout);
dataSocket=new Socket(hostname,port);
StringTokenizer stringtokens = new StringTokenizer(command);
String newcommand=stringtokens.nextToken();
String filename="";
if (save) {
  // Setzen des Uebertragungsmodus auf binaer
  controlout.println("type i");
  controlout.flush();
  getreply(controlin);
  filename=stringtokens.nextToken();
  String filename2=filename.substring(filename.lastIndexOf("/")+1);
```

```
  command=newcommand+" "+filename2;
}
controlout.println(command);
controlout.flush();
int result=getreply(controlin);
if (result==1){
  // Lesen der Datei und speichern unter filename2
  InputStream is=dataSocket.getInputStream();
  int amount;
  if(save){
    byte b[]=new byte[4096];
    RandomAccessFile outfile=new RandomAccessFile(filename, "rw");
    while((amount=is.read(b))!=-1){
      outfile.write(b,0,amount);
      }
    outfile.close();
    }
  else {
    listreply="";
    String erg;
    byte b[]=new byte[1024];
    while((amount=is.read(b))!=-1){
      erg=new String(b,0,amount);
      listreply=listreply+erg;
      }
    }
  code=getreply(controlin);
  is.close();
  dataSocket.close();
  }
}
```

- Schließen der Kontroll–Verbindung
  Das Schließen der Kontroll–Verbindung übernimmt die Methode `close`. Zuerst wird über die Kontroll–Verbindung der Befehl `QUIT` gesendet. Anschließend werden der Ein– und der Ausgabestrom sowie der *ftp*–Socket geschlossen.

```
public void close(){
  command("quit");
  outstream.close();
  inputstream.close();
  ftpsocket.close();
}
```

4.1.2 Implementierung des 3–Rechner–*ftp*

Die Implementierung des 3–Rechner–*ftp* ist in der Klasse `mainWindow.java` in der Methode `ftp_file` zu finden. Zuerst werden zwei Datenverbindungen geöffnet: die erste zum Quell– und die zweite zum Ziel–Rechner. Anschliessend wird in die gewüenschten Verzeichnisse gewechselt und der Übertragungsmodus auf binär gesetzt. In der Methode `getport` wird der zu benutzende Port des Quell–Rechners bestimmt und dann mittels `PORT`–Kommando an Ziel–Rechner geschickt. Damit ist die Datenverbindung initialisiert und die Datenübertragung wird durch die Befehle `STOR` und `RETR` gestartet.

```
// Oeffnen der Verbindungen
ftp source=new ftp(sourcehost);
ftp target=new ftp(targethost);
// Anmelden des Benutzers
source.open(login,passwd);
target.open(login,passwd);
// Wechseln in Verzeichnisse
source.command("CWD "+path);
target.command("CWD "+targetpath);
// setzen des Uebertragungsmodus
source.realCommand("TYPE I");
target.realCommand("TYPE I");
// bestimmen des Ports
String portstring=source.getPort();
int portresult;
portresult=target.realCommand("PORT "+portstring);
while (portresult!=2);
// Beginn der Datenuebertragung
target.realCommand("STOR "+file);
source.realCommand("RETR "+file);
```

4.1.3 Erweiterungen des *ftp*–Clients

Neben den grundlegenden Methoden wurden zur Erleichterung noch einige weitere innerhalb des *ftp*–Clients implementiert. Diese dienen zur Auswertung der Ausgaben des `LIST`–Befehls. Zu diesen Methoden gehören:

- `getDirEntries`
  liefert die Namen der Dateien im aktuellen Verzeichnis zurück

- `getLinkDestination`
  liefert die Position der realen Datei, falls es sich bei der Datei um einen link handelt
- `getFileSizes`
  liefert die Größe der Dateien im aktuellen Verzeichnis.

4.2 Der *telnet*–Client

Der ursprüngliche *telnet*–Client

Das *telnet*–Protokoll (Vgl. Kapitel 2.11) ist ein sehr umfangreiches Protokoll. Da bereits Umsetzungen des *telnet*–Protokolls existieren und die Implementierung eine umfangreiche Aufgabe ist, wurde statt einer erneuten Programmierung auf einen bereits existierenden *telnet*–Client ([Jug99]) zurückgegriffen.

Anpassungen des *telnet*–Clients

Die existierende Implementierung mußte jedoch einigen Anpassungen unterzogen werden. Bei der ursprünglichen Umsetzung handelt es sich um ein Applet, welches in HTML (HyperText Markup Language) ([Rag96]) eingebunden und mittels eines *JAVA*–fähigen Web–Browsers ausgeführt wird.
Deshalb war es nötig, aus dem Applet einen Bestandteil einer *JAVA*–Anwendung zu machen. Dazu mußte eine *JAVA*–Klasse implementiert werden, welche die Kernmethoden des existierenden *telnet*–Client der Anwendung zur Verfügung stellt. Dies geschah innerhalb der Klasse `telnet.java`. Bei Methoden ohne Auszug aus dem Quelltext werden die Routinen des ursprünglichen *telnet*–Clients benutzt. Die zur Verfügung gestellten Methoden sind:

- `wait`
  wartet bis ein vorgegebenes Zeichen (`token`) oder eine Zeichenkette vom *telnet*–Server eintrifft, aber nur solange bis eine gewisse Zeit

(`timeout`) vergangen ist. Wird z.B. benutzt , wenn das Ende eines Befehls abgewartet wird.

```
String tmp = "";
long deadline = 0;
if(timeout >= 0)
  deadline = new Date().getTime() + timeout;
  do {
    if(timeout >= 0){
      while(tio.available() <= 0){
        if(new Date().getTime() > deadline)
                throw new TimedOutException();
        Thread.currentThread().sleep(100);
      }
    }
    tmp = new String(tio.receive());
  }
  while(tmp.indexOf(token) == -1);
}
```

- `receiveBytes`
  empfängt die Daten, welche vom *telnet*–Server kommen als Feld von `bytes`.
- `available`
  gibt die Anzahl der Bytes wieder, die der *telnet*–Server bereits abgeschickt hat, aber der *telnet*–Client noch nicht ausgewertet hat.
- `receive`
  empfängt die Daten, welche vom *telnet*–Server kommen als Zeichenkette.
- `receiveUntil`
  empfängt die Daten, welche vom *telnet*–Server kommen als Zeichenkette bis eine bestimmte Zeichenkette eintrifft.
- `send`
  sendet Daten an den *telnet*–Server
- `sendLine`
  sendet ebenfalls Daten an den *telnet*–Server, hängt aber einen Zeilenvorschub `carriage return` an.

- `login`
  übernimmt das Anmelden des Benutzers beim *telnet*–Server. Es wird gewartet, bis der *telnet*–Server das Login verlangt. Dann wird der Login–Name übertragen und darauf gewartet, daß der *telnet*–Server die Eingabe des Passworts erwartet. Das Passwort wird gesendet und der Anmelde–Vorgang ist abgeschlossen.

```
wait("ogin:");
send(loginname + "\r");
wait("assword:");
sendLine(passwd + "\r");
```

- `setPrompt`
  setzt die interne Variable prompt
- `disconnect`
  trennt die Verbindung zum *telnet*–Server

4.3 Die grafische Benutzeroberfläche

Zur Implementierung von Benutzeroberflächen mit *JAVA* bieten sich mehrere Möglichkeiten an. Die erste ist das *Abstract Window Toolkit* (AWT). Dies stellt die grundlegenden Werkzeuge zur Erzeugung von grafischen Benutzeroberflächen zur Verfügung.
Seit einiger Zeit hat sich zusätzlich zum AWT ein weiteres Grafikpaket durchsetzen können, *Swing*. Dabei handelt es sich um eine Klassenbibliothek, die auf dem Konzept des AWT aufbaut und dieses erweitert. Zu den Erweiterungen gehören:

- Neue Elemente zur Programmierung grafischer Benutzeroberflächen, u.a. erweiterte Tasten und Fortschrittsanzeigen
- Neue Layoutmanager
  Ein Layout–Manager dient dazu, verschiedene Komponenten (z.B. Eingabefelder oder Text–Bereiche) in einem Programm–Fenster zu

plazieren. Die Layout–Manager des AWT lassen jedoch nur eine Gestaltung in engen Grenzen zu. Deshalb wurden neue Layout–Manager in das *Swing*–Paket aufgenommen.

- Plugable look and feel
  Eine Möglichkeit, das Aussehen der Benutzeroberfläche an das umgebende Betriebssystem anzupassen, zur Verfügung stehen: *UNIX/Motif*, *Windows* und das *JAVA*–Look–and–Feel

Das Hauptfenster

ist von der Klasse `JPanel` abgeleitet. Diese Klasse ist eine Container–Klasse. Die Container (`JPanel`, `JFrame`) bilden den Rahmen für alle Komponenten, die zur grafischen Benutzeroberfläche gehören.
Wie bereits in Kapitel 3.14 beschrieben, besteht die GUI aus sechs Komponenten, die mittels `GridBagLayout` positioniert werden.

Das `GridBagLayout` ist ein Layout–Manager, der Teil des *AWT*–Paketes ist. Er ist der komplexeste und flexibelste der Standard–*JAVA*–Layout–Manager.
Ein `GridBagLayout` ordnet die Komponenten in einem Container als Gitter an, wobei zu jeder Komponente Informationen zur Lage, Größe u.a. (die `GridBagConstraints`) angegeben werden können. Anhand der Informationen wird das Gitter aufgebaut und die Komponenten eingepasst. Um ein Element in das Layout einzufügen, wird mittels der Methode `add` eine Komponente mit ihren `GridBagConstraints` an die gewünschte Stelle im Layout positioniert.
Der Datentyp `GridBagConstraints` kann folgende Eigenschaften einer Komponente beeinflussen:

gridx, gridy gibt die Gitterposition der Komponente im Raster an.

gridwidth, gridheight gibt die Breite und Höhe der Komponenten im Raster an.

fill gibt an, in welche Richtung eine Komponente wachsen soll, wenn ihr mehr Platz als die Minimalgröße zur Verfügung steht.

ipadx, ipady gibt einen Größenzuwachs in Pixel an, um den die Komponente in der entsprechenden Richtung größer sein soll als die Standardminimalgröße.

insets ist der Abstand zu benachbarten Komponenten.

anchor gibt an, an welcher Position in der Gitterzelle eine Komponente angezeigt werden soll, wenn sie kleiner als die Zelle ist.

weightx, weigthy gibt an, wie ein zusätzlicher Platz im Container, z.B. durch Vergrößern des "Fensters", in horizontaler oder vertikaler Richtung aufgeteilt werden soll.

Zur besseren Übersichtlichkeit wurde die Methode `getConstraints` entwickelt. Diese übernimmt die Einstellungen für die Position, die Größe und die Ausdehnung der Komponente.

```
GridBagConstraints getConstraints(int x,int y,int w,int h,int fill){
  GridBagConstraints test=new GridBagConstraints();
  test.gridx=x;
  test.gridy=y;
  test.gridwidth=w;
  test.gridheight=h;
  test.fill=fill;
  return test;
}
```

Damit sieht die Anordnung der sechs Komponenten (`inputpane`, `actionpane`, `optionpane`, `outputpane`, `status` und `close`) des Hauptfensters im Quelltext so aus:

```
add(inputpane,getConstraints(0,0,1,2,GridBagConstraints.BOTH));
add(actionpane,getConstraints(1,0,1,1,GridBagConstraints.BOTH));
add(optionpane,getConstraints(1,1,1,1,GridBagConstraints.BOTH));
add(outputpane,getConstraints(2,0,1,2,GridBagConstraints.BOTH));
add(status,getConstraints(0,2,2,1,GridBagConstraints.BOTH));
add(close,getConstraints(2,2,1,1,GridBagConstraints.BOTH));
```

Bei `inputpane, actionpane, optionpane` und `outputpane` handelt es sich um von `JPanel` abgeleitete Klassen, bei `status` um ein Textfeld (`JTextField`) und bei `close` um eine Taste (`JButton`).

Die Bereichs–Arten

Im Layout des Hauptfensters existieren vier verschiedene Bereiche (Panels (*panes*) Vgl. Abbildung 3.5). Zwei von diesen sind ständig gleich belegt. Diese sind `inputpane` und `actionpane`.
Die Inhalte der Bereiche `optionpane` und `outputpane` werden je nach ausgewählten Datei(en) ,Verzeichnis(sen) und Aktionen dynamisch belegt.

- Das Panel `inputpane`
  Das `inputpane` besteht aus zwei Komponenten. Aus einer Auswahlbox für die Rechner im oberen Teil und einer Liste von Dateien und Verzeichnissen im unteren Teil, welche in ein Scroll–Bereich eingelagert ist. Diese dienen zur Auswahl des Quell–Rechners und der Quell–Datei(en) und/oder Verzeichnis(se).
- Das Panel `actionpane`
  Das `actionpane` besteht aus sechs Komponenten, jeweils drei Bezeichnern (`JLabel`) und drei Auswahlboxen (`JComboBox`) für die Aktionen.
- Das Panel `outputpane`
  Das `outputpane` ist nicht ständig aktiv. Es kann zwei Zustände annehmen, entweder ist es frei oder aber wie das `inputpane` aufgebaut. Das ist von der aktuell ausgewählten Aktion abhängig.
- Das Panel `optionpane`
  Das `optionpane` kann wie das `outputpane` in Abhängigkeit von der ausgewählten Aktion unterschiedliche Aussehen annehmen.
- Die Statuszeile
  Die Statuszeile ist vom Typ `JTextField`. Dabei handelt es sich um ein nicht–veränderbares Textfeld. Je nachdem, in welchem Zustand sich die Anwendung befindet, werden über die Statuszeile unterschiedliche Texte ausgegeben. Anfangs wird die Statuszeile mit einem Text initialisiert.

Der Austausch der Bereiche

Am Beispiel des `optionpane` soll demonstriert werden, wie die Bereiche neu belegt werden können. Zuerst wird das alte `optionpane` aus dem Hauptfenster entfernt. Daraufhin wird ein neues `optionpane` angelegt, initialisiert und an die Stelle des alten positioniert. Anschließend muß nur noch das Hauptfenster neu gezeichnet werden, und der Austausch ist abgeschlossen. Diesen Vorgang verdeutlicht ein Ausschnitt der Datei `mainWindow.java`.

```
mainwindow.remove(optionpane);
optionpane=new JPanel();
optionpane.setMaximumSize(new Dimension(200,200));
optionpane.setMinimumSize(new Dimension(200,200));
optionpane.setPreferredSize(new Dimension(200,200));
mainwindow.add(optionpane,getConstraints(1,1,1,1,GridBagConstraints.BOTH));
mainwindow.validate();
mainwindow.repaint();
```

Der Aufbau der Bereiche

Der Aufbau der einzelnen Bereiche soll anhand des `ftpPanel1` demonstriert werden. Dieser Bereich wird nach dem Start der Applikation in den Bereich `inputpane` geladen.
Der Bereich `ftpPanel1` beinhaltet zwei Komponenten: `compList` und `myScrollPane`. Diese beiden Komponenten werden mittels des `BorderLayouts`, einem der einfachsten Layouts, plaziert. `myscrollPane` ist von der Klasse `JScrollPane` abgeleitet und sorgt dafür, daß, wenn die darin liegende Liste (`Liste1`) zu groß wird, ein Rollbalken (`scrollbar`) eingeblendet wird.
Im Quelltext sieht der gesamte Einbettungs–Vorgang so aus:

```
JComboBox compList =new JComboBox(comp);
Liste1=new JList(model1);
JScrollPane myScrollPane= new JScrollPane(Liste1,
            ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED,
            ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED);
setLayout(new BorderLayout());
setBorder(new LineBorder(Color.darkGray));
add("North",compList);
add("Center",myScrollPane);
```

4.4 Der Programmablauf

4.4.1 Die Initialisierungen

Beim Start des Programmes werden einige Initialisierungen vorgenommen. Diese sind:

- Anmelden des Benutzers (Vgl. Abbildung 4.1)
- Auslesen der Konfigurations–Dateien (Vgl. Kapitel 4.5)
  Diese Aufgabe übernehmen die Klassen: `ReadCompConf.java`, `ReadYpDirs.java` und `ReadFormats.java`

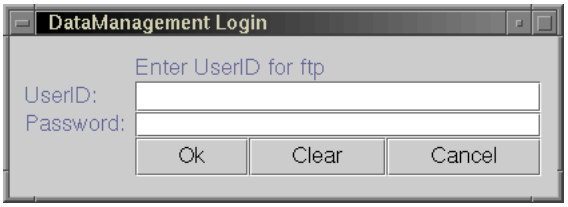

Abbildung 4.1: Das Anmeldefenster der Applikation

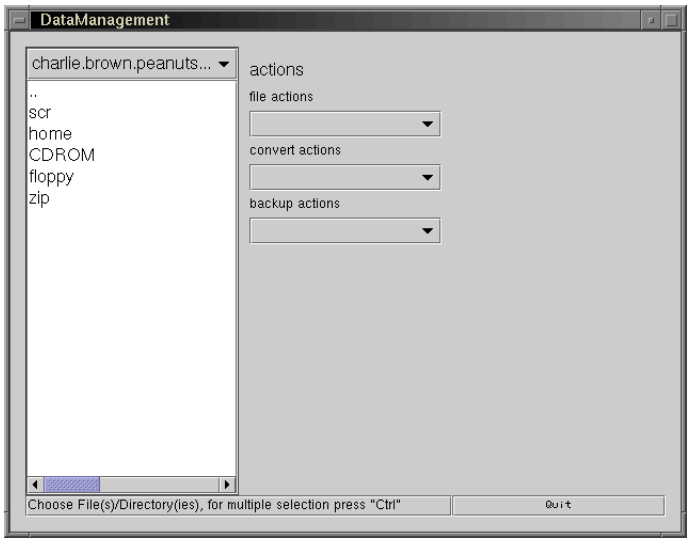

Abbildung 4.2: Startbild des Programmes zum Daten–Management

- Bestimmen der erreichbaren Rechner
  Es wird überprüft, welche der in der Konfigurations–Datei `computer.defs` zur Startzeit des Programmes erreichbar sind.
- Initialisierung der grafischen Benutzeroberfläche
  Die Bereiche `inputpane`, `actionpane` sowie die Statuszeile `status` und die Beenden–Taste `close` werden dargestellt. (Vgl. Abbildung 4.2)

4.4.2 Gewinnen der Verzeichnissübersichten

Nach dem Start des Programmes ist im Eingabe–Bereich `inputpane` neben der Auswahlbox für die Rechner eine Verzeichnisübersicht. Zur Gewinnung

dieser wird eine *ftp*–Verbindung, entweder zum ersten Rechner in der Initialisierungsliste, oder zum lokalen Rechner aufgebaut. Welcher der beiden Rechner ausgewählt wird, hängt davon ab, ob der lokale Rechner einen *ftp*–Server besitzt. Ist dies der Fall, wird der lokale Rechner als Quell–Rechner bestimmt. Sollte der lokale Rechner keinen *ftp*–Server besitzen, wird der erste Rechner aus der Initialisierungsliste benutzt.
Nachdem die Verbindung (falls noch keine geöffnet ist) zum Quell–Rechner aufgebaut ist, werden mittels der *ftp*–Befehle `CWD` (*change working-directory* – Wechseln des Verzeichnisses) und `LIST` (Anzeigen des Verzeichnisinhaltes) die Information für den Quell–Bereich gewonnen (geschieht innerhalb der Methode `getDirEntries`) und im Bereich `inputpane` dargestellt. Das geschieht wie folgt:

- an den *ftp*–Server wird der Befehl `LIST` geschickt,
- der Server schickt die Datei–Informationen an den Client,
- der Client filtert die Dateinamen aus den Datei–Informationen heraus, dies übernimmt die Methode `getDirEntries()` der Klasse `ftp` und
- die Dateinamen werden in einer Liste dargestellt. (Zur Darstellung einer Liste wird in *JAVA* `model1` vom Typ `DefaultListModell` verwendet.

Im Quelltext sieht dies folgendermaßen aus:

```
if (ftp1==null){
  ftp1=new ftp(hostname);
  ftp1.open(login,passwd);
}
entries1=ftp1.getDirEntries(path1);
if (entries1!=null){
  model1.removeAllElements();
  model1.addElement("..");
  for (int i=0;i<Array.getLength(entries1);i++){
    if ((!entries1[i].equals(".."))&&(!entries1[i].equals("."))){
      model1.addElement(entries1[i]);
    }
  }
}
```

4.4.3 Auswahl der Aktionen

Nachdem die Datei(en) und/oder Verzeichnis(se) mittels Maus und `STRG`–Taste ausgewählt sind, folgt die Auswahl der Verarbeitungsschritte. Wird während der Auswahl die `STRG`–Taste gedrückt ist eine Auswahl mehrerer Dateien oder Verzeichnisse möglich.
Für die Auswahl der Aktionen existieren drei verschiedene Möglichkeiten.

1. Dateiverarbeitung
   dazu gehört: das Kopieren, das Bewegen und das Löschen von Dateien.
2. Konvertierung
   dazu gehören, je nach Dateityp, unterschiedliche Konvertierprogramme
3. Archivierung
   dazu gehört die Möglichkeit, Daten auf externe Medien wie CDs und Magnetbänder, auszulagern sowie Daten von Magnetband einzulesen.

Zur Auswahl der Aktionen dienen die Auswahlboxen im Aktionsbereich `actionpane` der grafischen Benutzeroberfläche. Dabei handelt es sich um drei Auswahlboxen, eine für die Datei–Aktionen, eine für die Konvertier–Aktionen und eine für die Archivierungs–Aktionen. Als Beispiel dienen die Abbildungen 4.3 und 4.4.

Nachdem die Auswahl getroffen wurde, werden die Options– und/oder Ziel–Bereiche aktiviert.
Die Aufgabe der Darstellung der zur Aktion gehörenden Bereiche (Vgl. Kapitel 3.14) übernimmt die Klasse `comboBoxListener` in der Datei `mainWindow.java`. Der `comboBoxListener` überwacht alle drei Auswahlboxen. Die Verbindung der einzelnen Auswahlboxen mit dem `comboBoxListener` sieht im Quelltext folgendermaßen aus:

```
move.addActionListener(new comboBoxListener());
conv.addActionListener(new comboBoxListener());
back.addActionListener(new comboBoxListener());
```

Wobei `move`,`conv` und `back` die Auswahlboxen für die Datei–Verarbeitung, die Konvertier– und die Archivierungs–Vorgänge sind. Die Methode `addActionListener` sorgt dafür, daß, wenn eine Veränderung (Aktion) an einem Objekt auftritt, diese dem *Listener* (Zuhörer) mitgeteilt wird und

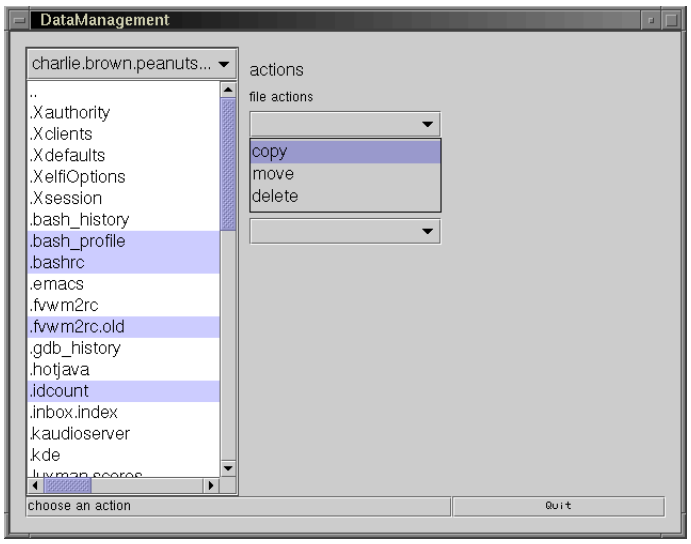

Abbildung 4.3: Die Auswahlbox für die Dateiverarbeitungs–Vorgänge

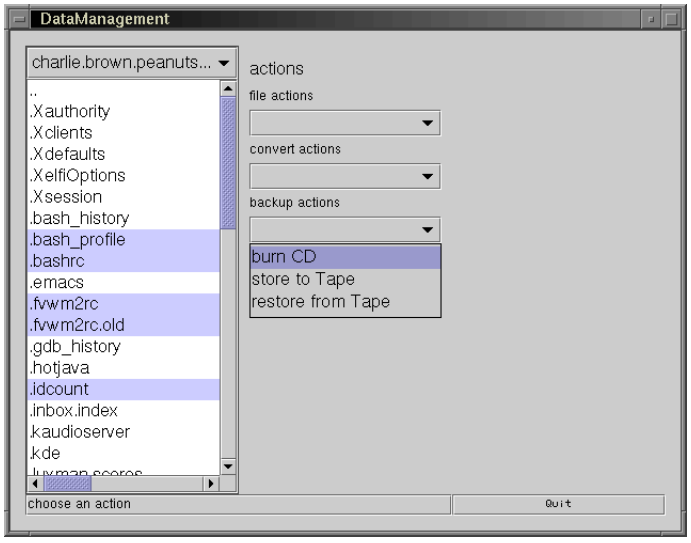

Abbildung 4.4: Die Auswahlbox für die Archivierungs–Vorgänge

dieser auf die Aktion reagieren kann.
Der `comboBoxListener` selbst orientiert sich anhand der Adresse der Aktions–auslösenden Auswahl–Box. Je nach Auslöser nehmen die Panel (`optionpane` und `outputpane`) unterschiedliche Werte an. Dies zeigt folgender Ausschnitt aus der Klasse `mainWindow.java`.

```
JComboBox cb= (JComboBox)e.getSource();
if (cb==move){
      action="move "+move.getSelectedItem();
      mainwindow.remove(optionpane);
      mainwindow.remove(outputpane);
      optionpane=new normalOptionPanel();
      mainwindow.validate();
      mainwindow.repaint();
}
if (cb==back){
      action="back "+back.getSelectedItem();
      mainwindow.remove(optionpane);
      mainwindow.remove(outputpane);
      if (...){
        optionpane=new tapePanel();
        outputpane=new JPanel();
      }
      else if (...){
        optionpane=new tapePanel();
        outputpane=new ftpPanel2(host2,path2);
      }
      else {
        optionpane=new normalOptionPanel();
        outputpane=new JPanel();
      }
}
if (cb==conv){
      mainwindow.remove(optionpane);
      mainwindow.remove(outputpane);
      outputpane=new ftpPanel2(host1,path2);
      optionpane=new tarPanel();
 }
 mainwindow.add(outputpane,getConstraints(2,0,1,2,
 GridBagConstraints.BOTH));
 mainwindow.add(optionpane,getConstraints(1,1,1,1,
 GridBagConstraints.BOTH));
 mainwindow.validate();
 mainwindow.repaint();
```

Abbildung 4.5: Das Fenster zur Bestätigung der Aktionen

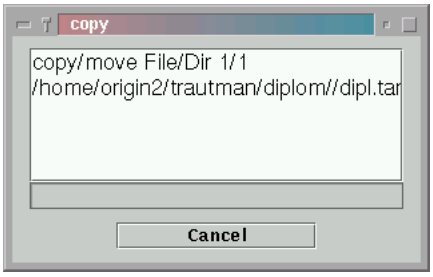

Abbildung 4.6: Informationsfenster zur laufenden Aktion

4.4.4 Bestätigung der Aktionen

Nachdem eine Aktion ausgewählt und gestartet wurde, öffnet sich ein Fenster zur Abfrage, ob diese Aktion wirklich ausgeführt werden soll. Ein Beispiel zeigt Abbildung 4.5. Nachdem die Eingabe bestätigt wurde, wird ein Fenster geöffnet, welches über den aktuellen Zustand der Aktion Aufschluß gibt. Ein Beispiel zeigt Abbildung 4.6 Dieses Fenster gliedert sich in drei Teile.

- Konsole zum Anzeigen von Meldungen
- eine Fortschrittsanzeige
- ein Schalter zum Abbrechen der Aktion

Die Ausgaben in der Meldungs–Konsole unterscheiden sich, je nachdem, welche Aktion ausgewählt wurde.

4.4.5 Die *Threads* zur Verarbeitung der Daten

Intern wird für jede Aktion ein *Thread* gestartet. Ein *Thread* ist im Prinzip ein Unterprogramm, welches unabhängig vom Hauptprogramm arbeitet

(Vgl. Kapitel 2.6.3). In der Klasse `mainwindow.java` existieren sieben dieser *Threads*:

```
class CDThread extends Thread implements ActionListener{
class TapeReadThread extends Thread implements ActionListener{
class TapeStoreThread extends Thread implements ActionListener{
class DeleteThread extends Thread implements ActionListener{
class CopyThread extends Thread implements ActionListener{
class TarThread extends Thread implements ActionListener {
class ConvertThread extends Thread implements ActionListener {
```

Alle diese *Threads* sind von der *JAVA*–Klasse `Thread` abgeleitet. Als Beispiel für einen der Verarbeitungs–*Threads* dient die Klasse `DeleteThread`. Ein Thread hat mehrere Methoden. Die wichtigste ist die Methode `run`. In dieser wird definiert, was der Thread ausführen soll. Im Beispiel–*Thread* ist das folgendes:

- die Namen aller zu löschenden Dateien werden in ein Feld gespeichert,
- ein Dialog–Fenster wird geöffnet, welches über den Fortgang des Löschens informiert,
- die Dateien werden gelöscht,
- der Bereich `inputpane` wird aktualisiert und
- der `deletethread` wird beendet.

Der zugehörige Quelltext:

```
class DeleteThread extends Thread implements ActionListener{
  public void run() {
    if (filearray==null){
      files=new String[1];
      files[0]=filename1;
    }
    else {
      files=filearray;
    }
    dialog=new MsgDialog();
    dialog.setSize(300,200)
    dialog.setTitle("Delete");
    dialog.cancelbut.addActionListener(this);
    dialog.setVisible(true);
    deleteFiles(host1,path1,files,dialog.text);
    dialog.setVisible(false);
    changeList1(host1,path1);
```

```
    deletethread.stop();
  }
}
```

4.5 Liste der Konfigurations–Files

Für die Definition des Systems werden Konfigurationsdateien benutzt. Diese befinden sich im Unterverzeichnis `defs`. Es gibt Konfigurationsdateien für:

- alle Konvertierprogramme
- jedes einzelne Konvertierprogramm
- die benutzbaren Computer
- die `home`–Verzeichnisse
- die `scr`–Verzeichnisse
- alle Computer mit Bandlaufwerk
- jeden einzelnen Computer mit Bandlaufwerk

Die Konfigurationsdateien sind alle mit einem einfachen Texteditor veränderbar. Das zugrundeliegende Schema gilt für alle Konfigurationsdateien. Im Prinzip wird für jeden Eintrag eine Zeile benutzt. Jede Zeile enthält wiederum mehrere Teile. Diese sind durch das *pipe*–Symbol "|" getrennt. Beispiele für die Konfigurations–Dateien sind in Kapitel 5.4 zu finden.

4.6 Benutzerführung

4.6.1 Der Kopier– bzw. Beweg–Vorgang

Um eine oder mehrere Dateien bzw. Verzeichnisse zu kopieren oder zu bewegen, wählt man mit der Maus und der Taste *CTRL* die gewünschten Dateien aus. Dann wird aus der Auswahlboxen `file actions copy` oder `move` gewählt. Jetzt öffnen sich der Options– und der Ziel–Bereich. Der Options–Bereich beinhaltet nur eine Taste, um die Aktion zu starten. Der Zielbereich dient zum Auswählen des Zielverzeichnisses. Dies zeigt Abbildung 4.7. Nachdem die Start–Taste gedrückt wurde, erscheint ein Dialogfenster, in dem die Aktion bestätigt werden muß. Danach erscheint ein Fenster, das den aktuellen Stand des Vorgangs wiedergibt und erlaubt, den Vorgang abzubrechen.

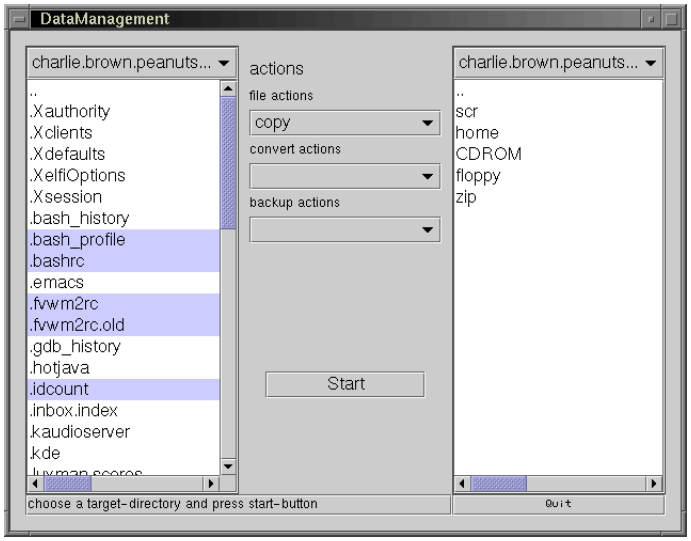

Abbildung 4.7: Die Benutzeroberfläche vor dem Kopier–/Beweg–Vorgang

4.6.2 Der Lösch–Vorgang

Beim Lösch–Vorgang werden wie beim Kopier–Vorgang die zu löschenden Dateien ausgewählt. Nachdem in der Auswahlbox `file actions delete` ausgewählt wurde, verändert sich nur der Options–Bereich. Abbildung 4.8 zeigt die Oberfläche vor dem Start des Lösch–Vorgangs.

4.6.3 Die Konvertier–Vorgänge

Die Erleichterung der Konvertiervorgänge waren der eigentliche Grund für die Entwicklung des Konzeptes. Zu Beginn eines jeden Konvertiervorganges wird mit der Maus die zu konvertierende Datei ausgewählt. Anhand der Endung des Dateinamens sucht das Programm nach passenden Konvertierprogrammen in der Datei `convert.defs`. Diese werden in der *convert actions*–Auswahlbox angezeigt.
Jetzt wird das gewünschte Programm ausgewählt. Danach ändert sich der Options– und Ziel–Bereich. Im Options–Bereich erscheinen eine Auswahlbox und eine Start–Taste. In der Auswahlbox kann, falls nötig, zwischen unterschiedlichen Optionen für das Konvertierprogramm gewählt werden. Im Ziel–Bereich wird das Ziel–Verzeichnis ausgewählt.
Ein Beispiel zeigt Abbildung 4.9.

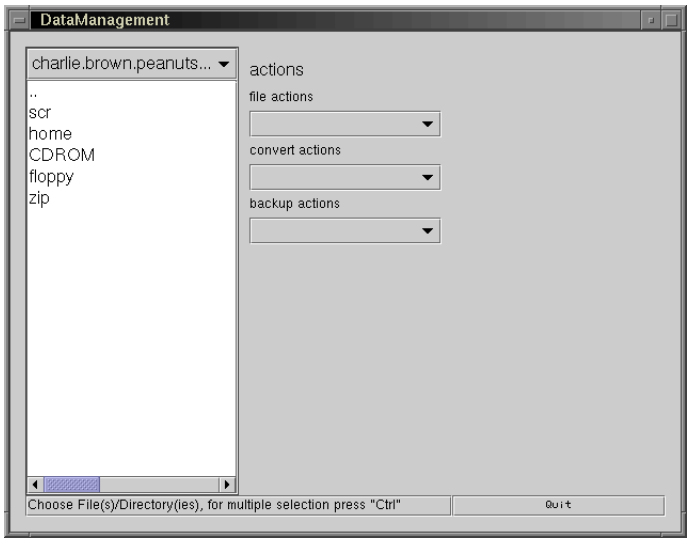

Abbildung 4.8: Die Benutzeroberfläche vor dem Lösch–Vorgang

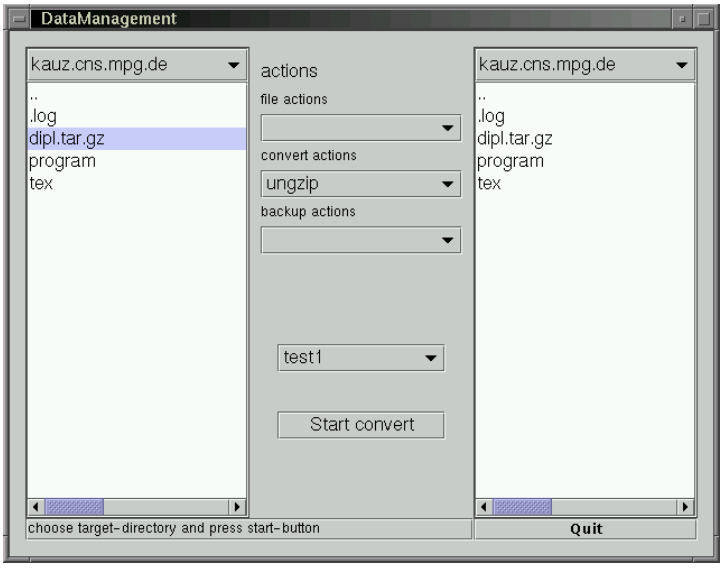

Abbildung 4.9: Die Benutzeroberfläche vor dem Konvertier–Vorgang

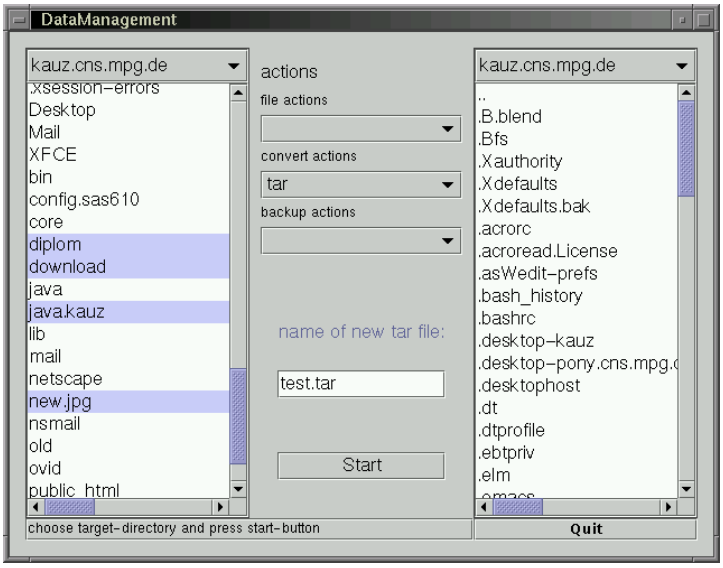

Abbildung 4.10: Die Benutzeroberfläche vor dem *tar*–Vorgang

Der `tar`–Vorgang

Das `tar`–Programm (tape archiving utility) ist kein Konverter, sondern dient dazu, mehrere Dateien oder Verzeichnisse auf ein Bandlaufwerk oder in eine Datei zu schreiben. Dadurch unterscheidet sich der `tar`–Vorgang von den Konvertiervorgängen. Beim `tar`–Vorgang werden mehrere Dateien ausgewählt. Danach wird in der `convert actions`–Auswahlbox die Option `tar` ausgewählt und der Options– und Ziel–Bereich werden aktualisiert. Im Options–Bereich werden ein Eingabefeld, in dem der Name der zu erzeugenden `tar`–Datei eingetragen werden kann und eine Start–Taste sichtbar. Im Ziel–Bereich kann der Ziel–Rechner und das Ziel–Verzeichnis verändert werden. Nachdem alle Auswahl getroffen ist, wird die `tar`–Datei durch Drücken der Start–Taste erzeugt. Dies zeigt Abbildung 4.10.

Der CD–Brenn–Vorgang

Um eine CD zu brennen, wählt man mittels Maus und der *STRG*–Taste die Dateien und Verzeichnisse aus, die auf die CD sollen und wählt bei den Archivierungs–Aktionen `Burn CD` aus. Anschließend wird die *Start*–Taste gedrückt. Es öffnet sich ein Fenster, welches Auskunft über den Fortgang

des Vorganges gibt.

Der Tape–Schreibe–Vorgang

Zur Daten–Archivierung auf Magnetbänder sind die gewünschten Dateien und Verzeichnisse auszuwählen. Bei den Archivierung–Aktionen ist `store to tape` auszuwählen. Im Optionen–Bereich werden zwei Auswahlboxen (für den Rechner und das Gerät) sowie eine Start–Taste sichtbar. Nachdem der Rechner sowie das Bandgerät ausgewählt ist, wird die Aktion mittels der *Start*–Taste ausgeführt.

Der Tape–Lese–Vorgang

Um Dateien und Verzeichnisse von einem Bandlaufwerk zurückzulesen, muß aus dem Archivierungs–Aktions–Feld die Option `restore from tape` ausgewählt werden. Daraufhin erscheint derselbe Optionen–Bereich wie beim Tape–Schreibe–Vorgang. Dort werden Rechner sowie Band–Gerät ausgewählt. Zusätzlich zum Optionen–Bereich wird der Ziel–Bereich aktiv. Dort kann der Ziel–Rechner sowie das Ziel–Verzeichnis angegeben werden. Nachdem Optionen und Ziel eingestellt sind, beginnt der Rücklese–Vorgang mittels der *Start*–Taste.

4.7 Dateistruktur der Applikation

Das Anwendungsprogramm verteilt sich auf ein Verzeichnis mit zwei Unterverzeichnissen. Im Hauptverzeichnis befinden sich die neu entwickelten Klassen. Im Unterverzeichnis `socket` befindet sich der *telnet–Client* und im Verzeichnis `defs` die Konfigurations–Dateien. Eine Übersicht der Dateien im Hauptverzeichnis zeigt Tabelle 4.1

4.8 Erzeugung des `jar`–Archivs

Nachdem alle Klassen implementiert und getestet sind, werden die Klassen mittels `jar`–Programm (*JAVA–Archiv*) zusammengefaßt. Dies geschieht mit dem Befehl:(

```
jar xcf classes.jar *.class socket/*.class
```

Datei/Verzeichnis	Aufgabe
`ReadCompConf.java`	Auslesen der Konfiguration der einbezogenen Rechner (Vgl. Kapitel 5.4.1)
`ReadFormats.java`	Auslesen der Konfigurationsdatei der bekannten Konvertierprogramme (Vgl. Kapitel 4.5)
`ReadTapeConf.java`	Auslesen der Konfigurationsdatei der Daten der Rechner mit Magnetband–Laufwerk (Vgl. Kapitel 5.4.4)
`application.java`	Die Hauptklasse, startet zuerst den Anmelde–Vorgang und danach die Klasse `mainWindow`
`converterType.java`	Hilfsklasse für die Klasse `ReadFormats`
`(defs)`	Verzeichnis der Konfigurationsdateien
`dialog.java`	Klasse für das Dialogfenster der Verarbeitungs–Threads
`ftp.java`	Implementierung des *ftp*–Clients
`login.java`	Das Anmelde–Fensters
`mainWindow.java`	Die Hauptklasse des Programms beinhaltet das Hauptfenster und die Verarbeitungs–Threads
`(socket)`	Verzeichnis mit den Klassen für den *telnet*–Client
`telnet.java`	Wrapper–Klasse für den *telnet*–Client
`ypdirs.java`	liefert die *home*– und *scr*–Verzeichnisse

Tabelle 4.1: Übersicht und Beschreibung der Dateien

Das so erzeugt Archiv kann mit einer *JAVA*–Laufzeit–Umgebung *JAVA–Runtime–Enviroment*) auf den verschiedenen Clients installiert werden.

Kapitel 5

Erprobung der Implementierung

Die exemplarische Umsetzung der geschaffenen Daten–Management–Konzeption wurde am Max–Planck–Institut für neuropsychologische Forschung in Leipzig durchgeführt.
Als Grundlage diente eine bereits bestehende Patientendatenbank, welche durch das geschaffene Daten–Management–Tool erweitert werden soll.

5.1 Die Patienten–Datenbank

Im Rahmen einer früheren Arbeit([Els98]) entstand eine Patienten–Datenbank mit grafischer Benutzeroberfläche. Diese Datenbank erlaubt es, persönliche Patientendaten mit Versuchsdaten zu kombinieren. Abbildung 5.1 zeigt das Startbild. Die Oberfläche setzt sich aus vier Teilen zusammen:

links	Übersicht der patienten– und versuchsbezogenen Datenblätter,
rechts	das jeweils aktive Datenblatt,
oben	eine Menuleiste und
unten	einer Statuszeile.

Mit dieser Patienten–Datenbank ist es möglich, Patientendaten und Ver-

Abbildung 5.1: Das Startbild der Patienten–Datenbank

suchsdaten zu kombinieren. Sie erlaubt es jedoch nicht, die Versuchsdaten direkt weiterzuverarbeiten oder zu archivieren.
Die Patienten–Datenbank wurde ebenfalls mit der Programmiersprache *JAVA* implementiert.
Bevor die Erprobung stattfinden kann, müssen die Systemvoraussetzungen für die Clients sowie die Server festgelegt werden.

5.2 Systemvoraussetzungen

5.2.1 Anforderungen an die Clients

An die Clients werden folgende Anforderung gestellt:

- Es muß ein *JRE* (*JAVA Runtime Enviroment*) mindestens in der Version Version 1.1 existieren.
- Die Clients müssen mittels eines *TCP/IP*–basierten Netzwerkes mit den Servern verbunden sein.

5.2.2 Anforderungen an die Server

Die Server müssen einen *ftp*–Server und *telnet*–Server bieten. Zusätzlich muß das Programm `md5sum` auf jedem Server im Verzeichnis `/usr/local/bin` installiert sein, um die übertragenen Dateien kontrollieren zu können. Die Programme:

- `ls`,
- `echo`,
- `tar`,
- `rm`,
- `ln` und
- `cd`

müssen erreichbar sein. Hinzu kommt, daß der *ftp*–Server das passive Öffnen erlauben muß.

5.3 Installationsvorgang

Es wird eine funktionierende *JAVA*–Implementierung (mindestens Version 1.1.x) vorausgesetzt. Daneben muß das *Swing*–Paket (bzw. *JAVA* Foundation Classes) der Firmen *SUN* und *Netscape* in der Version 1.0.1 installiert sein. Dieses muß in die *CLASSPATH*–Variable eingetragen werden, um von der virtuellen *JAVA*–Maschine benutzt werden zu können. Nachdem das *Swing*–Paket und die *JAVA*–Laufzeit–Umgebung (JRE) bzw. das *JAVA*–Entwicklungs–Kit (JDK) installiert ist, werden die zum Programm gehörigen *JAVA*–Klassen (zusammengefaßt in der Datei `classes.jar`) in das `lib`–Verzeichnis der *JRE* installiert. Damit sollte der Installationsvorgang abgeschlossen sein und das Programm mittels des Befehls:

```
java application
```

gestartet werden können.

5.4 Konfiguration

Vor der ersten Ausführung müssen die Konfigurationsdateien im Unterverzeichnis `defs` angepaßt werden, ansonsten kommt es dazu, daß das Programm nur eingeschränkt oder gar nicht funktioniert.

5.4.1 Liste der Server

In die Liste (`computer.defs` im `defs`–Verzeichnis) werden alle Server aufgenommen, auf denen der Benutzer den gleichen Login–Namen sowie das gleiche Passwort hat und die zum Arbeiten mit dem Programm vorgesehen sind. Ein Beispiel sieht folgendermaßen aus:

```
kauz.cns.mpg.de
pony.cns.mpg.de
rabe.cns.mpg.de
origin.cns.mpg.de
```

Es ist auch möglich, die Rechner ohne Domainnamen anzugeben. Es wird dann die lokale Domain benutzt. Diese Liste erscheint dann, falls die *ftp*–Server der Rechner erreichbar sind, in der Auswahlbox für die Rechner.

5.4.2 Liste der Konvertierprogramme

Die Liste der Konvertierprogramme befindet sich in der Datei `convert.defs`. Ein Ausschnitt sieht so aus:

```
gzip||.gz|kauz(/tmp/)|defs/gzip.defs
gunzip|.gz||kauz(/tmp/)|defs/gunzip.defs
```

Die Bedeutung der Einträge sind:

1. Name des Konvertierprogrammes, wie es im Programm erscheinen soll
2. Endung der Dateien, die von diesem Konverter bearbeitet werden können
3. Endung der Datei, nachdem die Konvertierung durchgeführt wurde.
4. Name des Rechners sowie ein Verzeichnis auf diesem Rechner, welches genügend Speicherplatz zum Konvertieren anbietet.
5. Name der Datei, in der die Kommandozeilen–Optionen angegeben sind.

5.4.3 Konfigurationsdateien für die Konvertierprogramme

In den Konfigurationsdateien für die Konvertierprogramme werden weitere Optionen gespeichert. Ein Beispiel für das Kompressions–Programm `gzip` ist in der Datei `gzip.defs` gespeichert und sieht folgendermaßen aus:

```
fast|gzip -1 $1
normal|gzip $1
good|gzip -9 $1
```

Für jede gewünschte Option des Konvertier–Programmes wird eine Zeile benutzt. Eine Zeile setzt sich aus zwei Teilen zusammen. Diese sind:

1. der Name der im Programm erscheint (Dieser sollte für den Benutzer verständlich gewählt werden.) und
2. die genaue Kommandozeile, wobei der Name der zu konvertierten Datei mit $1 maskiert wird und der Name der konvertierten Datei mit $2

5.4.4 Konfigurationsdatei für die Rechner mit Bandlaufwerk

In der Konfigurationsdatei für die Rechner mit Bandlaufwerk befindet sich eine Liste der Rechner mit Bandlaufwerk. Für das Max–Planck–Institut für neuropsychologische Forschung gibt es zwei Rechner mit Bandlaufwerk. Damit sieht die Datei `tape.defs` so aus:

```
scholle
hering
```

5.4.5 Konfiguration für einzelnen Rechner mit Bandlaufwerk

In der Datei `scholle.defs` werden die Devices (Geräte) angegeben, über die z.B. der Rechner Scholle verfügt. Ein Beispiel sieht so aus:

```
/dev/rmt0a
/dev/rmt0h
/dev/rmt0l
/dev/rmt0m
/dev/nrmt0a
/dev/nrmt0h
/dev/nrmt0l
/dev/nrmt0m
```

5.4.6 Anpassungen für den CD–Brenner

Das Brennen von CDs ist im Max–Planck–Institut für neuropsychologische Forschung bereits mit einigen Skripten vereinfacht worden. Der Ablauf ist folgender:

- Erstellen der Datei `cdrom.defs` mit einer Liste der Dateien und Verzeichnisse im *home*–Verzeichnis des Benutzers
- Starten des Skriptes `prepare_cd` (kopiert die Daten auf den Rechner mit CDROM).
- Brennen der CD mit dem Skript `make_cd`

Deshalb muss nicht das Vorgehen aus Kapitel 3.12.1 benutzt werden. Es reicht aus, die Datei `cdrom.defs` zu schreiben und mittels *telnet* das Skript `prepare_cd` zu starten.

Der dritte Schritt (das Brennen der CD) wird vom Benutzer am CD–Brenner–Rechner ausgeführt, da es direkte Interaktion mit dem Benutzer voraussetzt (das Einlegen der CD).

5.5 Test–Szenarien

Zur Erprobung des Programmes ist es nötig, das Programm auf den verschiedenen Rechner–Plattformen zu installieren und zu testen. Zu den durchgeführten Tests gehören:

- Kopieren einer Datei von einem Server auf einen anderen.
- Konvertieren eines Datensatzes
- Sicherung eines Datensatzes auf ein Archivierungs–Gerät.

5.6 Testplattformen

Als Testplattformen kam *Microsoft Windows 3.11* nicht zum Einsatz, weil keine *JAVA*–Implementierung (Version 1.1 oder höher) für *Microsoft Windows 3.11* existiert.

5.6.1 *SGI–IRIX*

Als erster Testrechner diente eine *O2* mit *R10000* Prozessor der Firma *Silicon Graphics*. Dort wurde die *JAVA* Version "3.0.1 (Sun 1.1.3)" installiert. Alle Test–Szenarien (Vgl. Kapitel 5.5) wurden erfolgreich durchlaufen.

5.6.2 Linux

Der zweite Testrechner war ein Linux–PC mit *JAVA* Version 1.1.7 von *http://www.blackdown.org*. Auch hier liefen alle Tests erfolgreich ab.

5.6.3 Microsoft Windows NT 4.0

Der dritte Testrechner war ein PC mit *Windows NT 4.0 Workstation*. Nach kleinen Veränderungen am Programm wurden auch hier alle Test–Szenarien erfolgreich durchlaufen.

5.7 Performance–Messungen

Um die Leistung des Programms zu messen, wird von zwei unterschiedlichen Szenarien ausgegangen.

1. zu konvertierende Datei und Konvertierprogramm befinden sich auf einem Rechner, und die konvertierte Datei wird wieder auf denselben Rechner zurückgeschrieben.
2. die zu konvertierende Datei liegt auf Rechner 1, der Konvertier–Vorgang findet auf Rechner 2 statt und die konvertierten Daten werden abschliessend auf Rechner 3 geschrieben.

Um vergleichbare Ergebnisse zu erzielen, werden die gleichen Daten konvertiert. Als Konvertierprogramm kommt das Packprogramm `gzip` zum Einsatz. Die auszupackende Datei war 53 MByte groß, im ausgepackten Zustand betrug ihre Größe 109 MByte.

5.8 Auswertung der Performance–Messungen

Tabelle 5.1 zeigt die Messergebnisse des Geschwindigkeitsvergleichs. Auffällig ist, daß es egal ist, ob eine Datei auf demselben Rechner oder auf einem anderen konvertiert wird. Dies liegt daran, daß die Übertragungsleistung der Festplatten der begrenzende Faktor ist. Das bedeutet: Es ist langsamer, eine Datei von einer Festplatte auf dieselbe Festplatte zu kopieren, als die Datei über das Netzwerk auf einen anderen Rech-

Aktion	T_{Fall1} in Sek.	T_{Fall2} in Sek.
Erstellen der Prüfsumme der Ausgangsdatei	12.0	10.9
Übertragen der Datei auf Konvertierrechner	13.8	10.7
Erstellen der Prüfsumme der kopierten Datei	13.1	13.3
Konvertieren der Datei	23.9	16.4
Erstellen der Prüfsumme der konvertierten Datei	23.8	23.7
Übertragen der Datei auf Ziel–Rechner	25.8	23.7
Erstellen der Prüfsumme der kopierten und konvertierten Datei	31.5	28.0
Löschen der temporären Dateien auf dem Konvertierrechner	0.1	0.3
Gesamtzeit	144.2	127.0

Tabelle 5.1: Performance–Vergleich der unterschiedlichen Konstellationen beim Daten–Management.
Fall1 ist die Konvertierung einer Datei auf einem Rechner.
Fall2 die Übertragung einer dieser Dateien auf den Konvertier–Rechner und das anschliessende Kopieren der konvertierten Datei auf einen dritten Rechner

ner zu schreiben. Beim gleichzeitigen Lesen und Schreiben auf einer Festplatte muss der Lese–Schreib–Kopf der Festplatte zwischen dem zu lesenden Block und dem zu schreibenden Block hin- und herbewegt werden. Wird die Datei jedoch nur geschrieben, muß der Lese–Schreib–Kopf der Festplatte nicht ständig bewegt werden. Dies führt zu einem erhöhten Datendurchsatz. Die unterschiedlichen Zeiten beim Erstellen der Prüfsummen sind auf systematische Fehler zurückzuführen. Auffällig ist, daß das Erzeugen der Prüfsummen einen hohen Anteil an der Gesamtzeit einnimmt. Die zum Erzeugen der Prüfsummen notwendige Zeit ist in etwa doppelt so lang wie die reine Übertragungszeit der Dateien.

Kapitel 6

Zusammenfassung und Ausblick

Das Auftreten von Datenmengen im Mega– und Gigabyte–Bereich spielt heute und in noch höherem Masse zukünftig ein wichtige Rolle. Damit einher geht der Wunsch, die Daten möglichst effektiv und einfach zu verarbeiten. Da nicht jeder über entsprechende Fachkenntnisse verfügt, ist es nötig, dem Benutzer eine einfache, intuitiv bedienbare Oberfläche anzubieten.
Heterogene Netzwerke auf Basis von *TCP/IP* sind weit verbreitet und bilden die Grundlage für das Internet. Deshalb werden diese auch in absehbarer Zukunft weit verbreitet sein.
Aus diesen Gründen entstand der Gedanke, eine Konzeption für ein Daten–Management–System zu erstellen und in ein Programm umzusetzen. Bevor die Konzeption erstellt werden konnte, musste das bisherige Daten–Management analysiert werden. Dabei kam es zu dem Ergebnis, daß die Hauptaufgaben eine Datenmanagement–Systems sind: die Erkennung der Art der Daten, die Auswahl der geeigneten Verarbeitungsprogramme, die Verschiebung der Daten von einem Rechner zu einem anderen zwecks Verarbeitung und die Verschiebung der verarbeiteten Daten auf einen beliebigen

Rechner sowie die Archivierung von Dateien in heterogenen Netzwerken zu übernehmen.
Zu Beginn eines jeden Daten–Management–Vorgangs steht die Erkennung der Art der Daten. Es wurde die einfacherer Methode der Erkennung anhand der Dateiendung benutzt.
Es hat sich herausgestellt, daß das effektivste Verfahren zur Datenübertragung in *TCP/IP*–Netzwerken das *ftp*–Protokoll ist. Es überträgt Daten mindestens 1,5 mal so schnell wie das *NFS*–Protokoll. Das liegt daran, daß unterschiedliche Implementierungen des *NFS*–Protokolls Probleme mit der Bestätigung der übertragenen Daten haben können.
Obwohl das *ftp*–Protokoll bzw. das darunterliegende Protokoll *TCP* bereits eine Sicherung der Datenübertragung vornimmt, kam zur zusätzlichen Sicherung der Datenübertragung das Programm `md5sum` zum Einsatz. Der dort implementierte *MD5*–Algorithmus sorgt dafür, daß falls Übertragungsfehler auftreten sich diese in einer erzeugten Prüfsumme niederschlagen.
Nachdem die Daten übertragen sind, ist es nötig, Programme auf entfernten Rechnern zu starten. Zu diesem Zwecke bieten sich mehrere Protokolle an. Aus Sicherheitsgründen wurde das *telnet*–Protokoll verwendet.
Ein weiterer Punkt dieser Arbeit beschäftigte sich mit dem Vergleich der Programmiersprachen *C*, *Perl* und *JAVA* im Hinblick auf ihre Eignung für größere Anwendungen in heterogenen Netzwerken. Es wurde die Tauglichkeit der Programmiersprache *JAVA* für größere Projekte, im Vergleich zu Miniatur–Programmen (sog. Applets), welche innerhalb von Web–Browsern zur Auflockerung von HTML-Seiten dienen, nachgewiesen. Dabei kam es zu dem Ergebniss, daß *JAVA*–Programme, insbesondere beim Einsatz eines *just–in–time*–Compilers, in *C* geschrieben Programmen kaum noch in der Geschwindigkeit nachstehen. Der Geschwindigkeitsnachteil von *JAVA* läßt sich nur noch bei rechenintensiven Programmen nachweisen, und selbst dort ist der Geschwindigkeitsfaktor unter zwei. Hingegen zeigt sich bei der Programmiersprache *Perl* ein erheblicher Geschwindigkeitsnachteil gegenüber

C–Programmen. So sind rechenintensive Programme in Perl etwa 20 bis 40 mal langsamer als das vergleichbare *C*–Programm.
Bei nicht–rechenintensiven Programmen liegen die drei Programmiersprachen etwa gleich auf, da der Prozessor des Systems nicht vollständig ausgelastet ist. Dies wurde anhand der Übertragung mehrerer Dateien mittels *ftp* nachgewiesen.
Java ist somit für größere Anwendungen geeignet, ganz besonders, wenn die Anwendung mit dem Benutzer in Interaktion treten und/oder den Transport von Daten über ein Netzwerk übernehmen soll. Darüber hinaus ist die Programmiersprache *JAVA* durch ihre Portabilität in heterogenen Umgebungen jeder anderen Sprache mit Hinsicht auf die Portierbarkeit überlegen.
Das geschaffene Konzept ist durch die Einbindung eines auf TCP/IP basierenden *ftp*– und *telnet*–Clients in UNIX–Umgebungen universell einsetzbar und an die Erfordernisse der Benutzer anpassbar. Durch die Konfiguration über Textdateien ist das Programm einfach zu konfigurieren. Da sich jeder Nutzer mit seinem Loginnamen und Passwort anmelden muss, ist für die System–Sicherheit gesorgt. Jeder Benutzer hat nur genau die Rechte, die ihm der Systemadministrator zugewiesen hat.
Als Fortsetzung der Arbeit könnte die Verbindung von Patientendaten und Versuchsdaten ausgebaut werden. Dazu müsste eine Datenbank erstellt werden, welche sowohl die Kontrolle über die Patientendaten als auch über die Versuchsdaten hat. Dadurch könnte ein Überblick über alle zu einem Projekt gehörenden Daten geschaffen und eventuell doppelt ausgeführte Datenverarbeitungen ausgeschlossen werden (Analyse des Workflow). Diese Aufgabe ist jedoch sehr umfangreich und erfordert die genaue Kenntnis aller möglichen Verarbeitungsschritte und Informationen über den Datenfluss.
Eine Erweiterung des Programms könnte in einer Implementierung des `file`–Programms in *JAVA* bilden, um die Dateityp–Erkennung eindeutig zu gestalten, ohne die Dateiendungen benutzen zu müssen. Zukünftig könnte die Konzeption um andere externe Speichermedien erweitert werden.

Kapitel 7

Anhang

7.1 Quellcode des *C*–Programmes zum Leibnitz–Verfahren

```
void main (void)
{
  double pi_viertel=1.0;
  int count, maxcount, x;
  maxcount=10000000;
  x=3;
  for (count=2;count<=maxcount;count++){
    if (!(count%2)){
      pi_viertel-= 1.0/(double) x;
    }
    else {
      pi_viertel+= 1.0/(double) x;
    }
    x+=2;

  }
  printf (" %12d  %1.16f    \n", count, pi_viertel*4.0);
}
```

7.2 Quellcode des *JAVA*–Programmes zum Leibnitz–Verfahren

```
public class pi {
  public pi(){

  }
  public static void main(String args[]){
    double pi_viertel=1.0;
    int maxcount, x;
    maxcount=1000000000;
    x=3;
    for (int count=2;count<=maxcount;count++){
      if((count%2)==0){
        pi_viertel=pi_viertel- (double) 1.0/ (double) x;
      }
      else {
        pi_viertel=pi_viertel+ (double) 1.0/ (double) x;
      }
      x=x+2;
    }
    System.out.println(pi_viertel*4);
  }
}
```

7.3 Quellcode des *Perl*–Skriptes zum Leibnitz–Verfahren

```
#!/usr/bin/perl -w
use strict;
use vars qw($pi_viertel $count $maxcount $x );
$pi_viertel=1.0;
$maxcount=1000*1000000;
$x=3;;
for ($count=2;$count<=$maxcount;$count++){
    if (!($count%2)){
        $pi_viertel-= 1/ $x;
    }
    else {
        $pi_viertel+= 1/ $x;
    }
    $x+=2;
  }
printf (" %12d  %1.16f    \n", $count, $pi_viertel*4.0);
```

7.4 Quellcode des *JAVA*–Programmes zur *ftp*–Messung

```
public class ftptest {
  public ftptest (){
  }
  public static void main(String args[]){
    String portString;
    ftp a = new ftp("windhund");
    a.open("username","password");
    a.command("cwd /sdb1/");
    a.realCommand("TYPE i");
    a.command("RETR 10MB.dat");
//    a.command("RETR 100MB.dat");
//    a.command("RETR 1GB.dat");
    a.close();
  }
}
```

7.5 Quellcode des *Perl*–Skriptes zur *ftp*–Messung

```
#!/usr/bin/perl -w
use strict;
use Net::FTP;
use vars qw($ftp);
$ftp= Net::FTP->new("seekuh.cns.mpg.de");
$ftp->login("loginname","password");
$ftp->cwd("/tmp");
$ftp->binary();
$ftp->get("10MB.dat");
#$ftp->get("100MB.dat");
#$ftp->get("1GB.dat");
$ftp->quit;
```

7.6 Das Max–Planck–Institut für neuropsychologische Forschung

Das Institut (http://www.cns.mpg.de) wurde 1995 gegründet und gliedert sich in zwei Arbeitsbereiche:

- Neurologie

  Der neurologische Arbeitsbereich befaßt sich mit der Analyse und Beschreibung der individuellen Neuroanatomie mit Methoden der Magnetresonanzoptik und der Neuroinformatik. Zweiter Arbeitsschwerpunkt ist die Erforschung der zerebralen Implementierung von Gedächtnisfunktionen mit Hilfe der funktionellen Magnetresonanztomographie. In Zusammenarbeit mit der Tagesklinik für kognitive Neurologie des Universitätsklinikum Leipzig werden Erkenntnisse der Grundlagenforschung für Diagnostik und Therapie nutzbar gemacht sowie Patienten zum besseren Verständnis von Arbeitsgedächtnis– und Sprachverstehens–Prozessen untersucht.
- Neuropsychologie

  Im Zentrum des Forschungsinteresses der Neuropsychologie steht die Frage, wie das Gehirn komplexe mentale Fähigkeiten, wie Sprachverarbeitung und Gedächtnisleistungen, realisiert. Mit Hilfe von Verfahren, wie elektroenzephalographischer (EEG) und magnetenzephalographischer (MEG) Messungen wird die Gehirnaktivität bei Sprach– und Gedächtnisprozessen analysiert. Mit diesen beiden Verfahren sowie der Magnetresonanztomographie (MRT) eröffnet sich die Möglichkeit, ein kohärentes Bild neurokognitiver Verarbeitungssysteme des menschlichen Gehirns zu erstellen.

7.7 Das Rechnernetz des Max Planck Instituts für neuropsychologische Forschung

Im Rechnernetz des Instituts wird mit verschiedenen Hardware–Architekturen und Betriebssystemen gearbeitet. Die wichtigsten Betriebssysteme sind:

- *Linux*,
- *SGI–IRIX* 5.3, 6.3 und 6.4,
- *Microsoft Windows* 3.11 *for Workgroups*,
- *DEC UNIX*,
- *Microsoft Windows NT 3.51* und *NT 4.0*,

Die Hardware–Architekturen sind:

- PCs
- *SGI Indy*, *O2*, *Octane*, *Origin 2000*
- *DEC Alpha*

Diese unterschiedlichen Rechner sind mittels eines *TCP/IP*–Netzwerkes verbunden. Zentrale Dateisysteme sind mittels *NFS* und Samba zugänglich.

7.8 Besonderheiten des institutsweiten Dateisystems

Das Max–Planck–Institut für neuropsychologische Forschung hat einige Besonderheiten im Dateisystem. Neben den Standard–*UNIX*–Dateisystem existieren noch drei weitere Dateisysteme, die bei Bedarf netzwerkweit jedem Rechner zur Verfügung stehen:

- `/home`–Verzeichnisse,
- `/scr`–Verzeichnisse und
- `/usr/local`–Verzeichnisse

Das `/home`–Verzeichnis dient in erster Linie dazu, von jedem Rechner aus seine eigenen Dateien und Verzeichnisse benutzen zu können. Da die `/home`–Verzeichnisse auf unterschiedlichen Rechnern verteilt liegen, muß je nach Nutzer das `/home`–Verzeichnis auf einem anderen Rechner zugänglich gemacht werden. Die `/home`–Verzeichnisse sind wie folgt organisiert:

1. /
2. home
3. /
4. Rechnername
5. Nummer der eingebauten Festplatte

Daraus ergeben sich z.B. `/home/origin2` oder `/home/kauz1`. Wobei origin und kauz Rechernamen sind.
Das `/scr`–Verzeichnis dient dazu, temporär größere Datenmengen im Netz zugänglich zu machen. Die Daten werden jedoch nicht ins Backup einbezogen, müssen also leicht wiederherstellbar sein.
Das `/usr/local`–Verzeichnis dient zur zentralen Bereitstellung von Software für alle *UNIX*–Systeme. Dazu gehören die Auswerteprogramme zur Verarbeitung der im Institut gewonnen Daten. Da sich die Programme für die verschiedenen Betriebssysteme unterscheiden, müssen auch in Abhängigkeit von der Architektur unterschiedliche Daten zugänglich gemacht werden. Dazu stehen auf dem Server Origin für die drei Betriebssysteme

DEC/OSF, *SGI/IRIX* und *PC/Linux* jeweils die passenden Binaries zur Verfügung, welche sich dann unter `/usr/local/bin`, `/usr/local/lib` und `/usr/local/share` auf jedem System wiederfinden.
Für die automatische Bereitstellung der Verzeichnisse wird das `automount`–Programm (Vgl. [Anv]) benutzt.
Soll in ein Verzeichnis gewechselt werden, welches zur Zeit nicht bereitgestellt ist, tritt das `automount`–Programm in Aktion und bindet das Verzeichnis ins lokale Dateisystem ein.
Die Information darüber, welche Platte an welchem Rechner erreichbar ist, beinhaltet das *Network Information System* NIS (Vgl. Kapitel 2.3).
Innerhalb des Programms werden anhand der Informationen aus dem *NIS*, falls in eines der Verzeichnisse `scr` oder `home` gewechselt werden soll, die noch nicht geladenen lokalen Verzeichnisse eingebunden. Da der Automounter auf allen *UNIX*–Rechnern installiert ist, genügt es, in diese Verzeichnisse zu wechseln, um die Verzeichnisse zu mounten.
Im Quelltext sieht das (am Beispiel der `home`–Verzeichnisse) so aus:

```
void openHomeDirs(String hostnm){
  String name=hostnm.substring(0,hostnm.indexOf("."));
  int count=yp.getNumberOfHomeDirs(name);
  String[] dirs=new String[count];
  dirs=yp.getHomeDirs(name);
  try {
    telnet opendirs=new telnet(hostnm,login,passwd);
    for (int i=0;i<count;i++){
      opendirs.sendLine("cd /home/"+dirs[i]);
    }
    opendirs.disconnect();
  }
  catch (IOException e){
  }
}
```

Mittels einer telnet–Verbindung wird in die rechnerspezifischen lokalen Verzeichnisse gewechselt, so daß diese via `automount` eingehängt werden. Analoges gilt für die `/scr/`–Verzeichnisse.

Literaturverzeichnis

[Anv] ANVIN, H. PETER: *Linux Reference Manual* - `automount`.

[Ape] APEL, J.: *Kryptographie.* Vorlesung SS 1997.

[BDC96] BRENT D. CHAPMAN, ELIZABETH D. ZWICKY: *Einrichten von Internet-Firewalls.* O'Reilly & Associates, Inc., 1996.

[Cas96] CASTAGNA, G.: *Object Oriented Programming. A Unified Foundation.* Birkhäuser, Biel-Benken, 1996.

[cho] *UNIX Reference Manual* - `chown`. 4.2 Berkeley Distribution.

[cpi] *UNIX Reference Manual* - `cpio`. 4.2 Berkeley Distribution.

[Els98] ELSNER, MATTHIAS: *Interaktive Benutzeroberfläche für eine verteilte Datenbank-Anwendung mit sicherer Client-Server-Kommunikation.* HTWK Leipzig, 1998.

[fil] *UNIX Reference Manual* - `file`. 4.2 Berkeley Distribution.

[Fla97] FLANAGAN, DAVID: *Java in a Nutshell.* O'Reilly & Associates, Inc., 1997.

[JP83] J. POSTEL, J.K. REYNOLDS: *RFC 854 - Telnet Protocol Specification.* IETF, May-01-1983.

[JP85] J. POSTEL, J.K. REYNOLDS: *RFC 959 - File Transfer Protocol.* IETF, Oct-01-1985.

[Jug99] Jugel, Matthias L.: *The Java^{TM} Telnet Applet.* `http://siemens.first.gmd.de/persons/leo/java/Telnet`, 1999.

[Kan91] Kantor, B.: *RFC 1282 - BSD Rlogin.* December 1991.

[Kop96] Kopka, Helmut: *LaTeX.* Addison-Wesley, 1996.

[LW96] Larry Wall, Tom Christiansen, Randal L. Schwartz: *Programming Perl.* O'Reilly & Associates, Inc., 1996.

[Mic89] Microsystems, Sun: *NFS: Network File System Protocol Specification.* March 1989.

[mt-] *UNIX Reference Manual* - `mt`. 4.2 Berkeley Distribution.

[Ous94] Ousterhout, J.K.: *Tcl and the Tk Toolkit.* Addison-Wesley, 1994.

[Pil99] Pilz, Uwe: *Patienten Daten Management System COPRA.* `http://www.thuenet.de/intensivehelp/copra.html`, 1999.

[Pos80] Postel, J.: *RFC 768 - User Datagram Protocol.* ISI, Aug-28-1980.

[Pos81a] Postel, J.: *RFC 791 - Internet Protocol.* IETF, Sep-01-1981.

[Pos81b] Postel, J.: *RFC 793 - Transmission Control Protocol.* IETF, Sep-01-1981.

[Qui] Quinlan, Daniel: *Filesystem Hierarchy Standard – Version 2.0.*

[Rag96] Raggett, D.: *HTML 3:Electronic publishing on the world wide web.* Addison-Wesley, 1996.

[rdu] *UNIX Reference Manual* - `rdump`. 4.2 Berkeley Distribution.

[Riv92] Rivest, R.: *The MD5 Message-Digest Algorithm.* MIT Laboratory for Computer Science and RSA Data Security, Inc., April 1992.

[Sch] Schilling, Joerg: *Linux Reference Manual* - `cdrecord`.

[Sha99] Sharpe, Richard: *What is Samba.* `http://anu.samba.org/cifs/docs/what-is-smb.html`, 1999.

[Spr] Spruth, Prof. Dr.-Ing. Wilhelm: *Vorlesungsunterlagen, Verteilte Systeme.*

[Sri95a] Srinivasan, R.: *RPC: Remote Procedure Call Protocol Specification Version 2.* Sun Microsystems, August 1995.

[Sri95b] Srinivasan, R.: *XDR: External Data Representation Standard.* Sun Microsystems, August 1995.

[Ste92] Stern, Hal: *Managing NFS and NIS.* O'Reilly & Associates, Inc., 1992.

[Ste94] Stevens, W. Richard: *TCP/IP Illustrated, Volume1.* Addison-Wesley Publisching Company, 1994.

[Str97] Stroustrup, Bjarne: *Die C ++ Programmiersprache.* Addison-Wesley, 1997.

[tar] *UNIX Reference Manual* - `tar`. 4.2 Berkeley Distribution.

[You] Youngdale, Eric: *Linux Reference Manual* - `mkisofs`.

Abbildungsverzeichnis

Tabellenverzeichnis

Glossar

American Standard Code for Information Interchange (ASCII) Amerikanischer Zeichen–Kode. Ursprünglich eine Zeichenkodierung mit 7 Bits. Inzwischen arbeiten die meisten Rechner mit einer erweiterten 8–Bit–Version.

Andrew File System (AFS) Dateiverwaltungssystem, ähnlich wie NFS. AFS verfügt über zusätzliche Möglichkeiten wie Kerberos–Authentifizierung, lokales Zwischenspeichern von Daten und verfeinerte Zugriffskontrolle auf Dateisysteme.

ARPANET Ein paketvermitteltes Netz aus den frühen 70ern. Ein Vorgänger des heutigen Internet.

Berkeley Software Distribution (BSD) Ein an der Universität von Berkeley entstandenes *UNIX*–Derivat.

CDROM Archivierungsmedium mit 650 MB Kapazität.

Data Encryption Standard (DES) Ein Standard für Datenverschlüsselung mit einer Schlüssellänge von 56-Bit. Wird bei der UNIX–Passwort–Verschlüsselung benutzt.

Datagram In Übertragungsnetzen ein Datenpaket, das die vollständige Empfängeradresse und Absenderangaben enthält. Es kann so als eigene Einheit im Netz direkt vom Sender zum Empfänger weitergeleitet

werden, ohne daß vorher Absprachen zwischen den Kommunikationspartnern stattfinden. Der Vorteil: Protokoll Overhead zum Aufbau der Verbindung wird weitgehend vermieden.

Domain Name Service (DNS) sorgt für die Umsetzung der Klartextnamen der Netzwerkkarten in IP–Adressen

Extended Binary-coded Decimal Interchange Code (EBCDIC) Ein 8-Bit Zeichen–Kode, von IBM entwickelt. Auf einigen Großrechnern anstelle des ASCII-Codes verwendeter Code zur Darstellung von Buchstaben und Ziffern. EBCDIC und ASCII sind nicht miteinander kompatibel.

External Data Representation (XDR) Von Sun Microsystems entwickelter Standard zur maschinenunabhängigen Darstellung von Datenstrukturen.

File Transfer Protocol (FTP) Übertragungsprotokoll für Dateien zwischen zwei Rechnern

Filesystem Hierarchy Standard (FHS) Standard für das *UNIX*–Dateisystem.

Firewall System zur gegenseitigen Abschottung von Netzen unterschiedlichen Sicherheitsbedarfs. Mit Hilfe einer "Software-Brandschutzmauer" werden dabei die Netze über Filterfunktionen mit einem Zugriffsschutz versehen.

Hypertext Markup Language (HTML) Seitenbeschreibungssprache des Internets

Hypertext Transfer Protocol (HTTP) Client-Server-TCP/IP-Protokoll, das im Internet für den Austausch von HTML-Dokumenten benutzt wird. Im Normalfall benutzt es Port 80.

Internet Adressen Zahlenkombination, die eine Netzwerkkarte im Internet eindeutig bezeichnet. IP-Adressen haben folgende Struktur: a.b.c.d (a, b, c, d sind natürliche Zahlen zwischen 0 und 255.)

Internet Control Message Protocol (ICMP) Das Protokoll, das von Internet-Protokoll-Ebene (IP von TCP/IP) benutzt wird, um Steuerinformationen für die Wegeauswahl auszutauschen.

Internet Group Management Protocol (IGMP)

Internet Protocol (IP) Übertragungsprotokoll der Netzwerk–Schicht. IP transportiert Datenpakete über mehrere Netze zu einem Empfänger.

JAVA Objektorientierte Programmiersprache, die sich besonders durch ihre Plattformunabhängigkeit auszeichnet.

JAVA Virtual Machine (JVM) Interpretiert den *JAVA*–Code und gibt ihn an das darunterliegende Betriebssystem weiter.

Linux Ein freier Ableger der *UNIX*–Betriebssystemfamilie. Wurde von dem Finnen Linus Torvalds für IBM–PC entwickelt und ist inzwischen auf vielen Hardware–Plattformen implementiert.

Magnetbänder Archivierungsmedium mit Kapazitäten von 80 MB bis 35 GB

Multi-Tasking-Betriebssystem Ein Betriebbsystem, das es erlaubt, mehr als einen Prozess gleichzeitig auszuführen.

Multi-User-Betriebssystem Ein Betriebssystem, das es erlaubt, mehrere Benutzer arbeiten zu lassen.

Network File System (NFS) Dateiverwaltungssystem der Firma Sun. Mit NFS können File-Systeme, die auf verschiedenen Rechnern liegen, zu einem einzigen logischen Verzeichnisbaum kombiniert werden.

Network Information System (NIS) Von der Firma Sun entwickeltes Client-Server-Protokoll zur Verwaltung von Konfigurationsdaten wie Host- und Nutzernamen in verteilten Systemen. Ist identisch mit dem "Yellow Pages"-Dienst.

Network Virtual Terminal (NVT) Virtueles Terminal, welches z.B. vom *telnet*–Protokoll benutzt wird. Bildet den kleinsten gemeinsamen Nenner bei Client–Server–Anwendungen, die ein Terminal benutzen.

OSI-Schichtenmodell Das OSI–Schichtenmodell ist universelles, hierarchisches Modell der Datenkommunikation. Im Gegensatz zum TCP/IP–Schichtenmodell gliedert es sich in 7 Schichten: Physikalische Schicht, Verbindungsschicht, Netzwerkschicht, Transportschicht, Sitzungsschicht, Darstellungsschicht und Anwendungsschicht.

Port Die Kommunikation zwischen Rechner erfolgt über sogenannte Port–Nummern. Für Standard–Dienste werden die Port–Nummern von der IANA(Internet Assigned Numbers Authority) vergeben.

Remote Login (rlogin) Protokoll zur Ausführung von Programmen auf entfernten Rechnern. Wird nur auf *UNIX*–Systemen eingesetzt.

Remote Procedure Call (RPC) Mechanismus verteilter Systeme, der es ermöglicht, in einem Client Programm die Dienstschnittstellen von im Netz verteilten Servern so aufzurufen, als wären sie auf dem eigenen Computer.

Request for Comments (RFC) Das Internet und TCP/IP betreffende Standarddokumente. Tragen ihren Namen, weil Standards im Entstehungsprozeß öffentlich diskutiert werden.

Router Ein spezialisierter Computer oder eine Software, die die Verbindung zwischen zwei oder mehr Netzwerken ermöglicht. Der Router ist in einem Netzwerk unter einer Adresse, in dem anderen Netzwerk

unter einer anderen Adresse bekannt. Werden im ersten Netz Pakete an eine Station im zweiten Netz gesendet, so erkennt dies der Router und sendet diese Pakete in das zweite Netz, an das der Router mit der zweiten Adresse ebenfalls angebunden ist.

Routing Vermittelung von Paketen in verschiedene Netze.

SAMBA Ein Protokoll zur Bereitstellung von Dateien und Druckern in heterogenen Netzwerken.

Simple Mail Transfer Protocol (SMTP) Protokoll für E-Mail im Internet.

Socket Eine Zahl, die den logischen Kanal identifiziert, auf dem das IP Protokoll senden soll.

TCP–Segment Ein Teil der zu übertragenden Daten, welche mit einem Header versehen werden, der Angaben u.a. über Quelle, Ziel, Sequenznummer, Prüfsumme enhält.

Telecommunications Network Protocol (telnet) Internet-Dienst, der es ermöglicht, auf anderen Rechnern im Netz so zu arbeiten, als ob man direkt als Terminal angeschlossen wäre.

Transmission Control Protocol (TCP) verbindungsorientiertes Übertragungsprotokoll in Transport–Schicht der *TCP/IP*–Protokoll–Suite, das einen verläßlichen Datenfluß bereitstellt.

UNIX Offenes Betriebssystem, für das Varianten aller gängigen Rechnerhersteller vorhanden sind, z.B. AIX (IBM), Digital Unix (DEC), HP-UX (Hewlett Packard), Irix (SGI), Solaris (Sun), UNICOS (Cray).

User Datagram Protocoll (UDP) verbindungsloses Übertragungsprotokoll in Transport–Schicht der *TCP/IP*–Protokoll–Suite, das nur Datagramme versendet, ohne deren Eintreffen am Ziel–Rechner zu überwachen.

X/OPEN Portability Guide (XPG) gemeinsamer Standard aller *UNIX*–Betriebssysteme

Ich versichere, daß ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Ort	Datum	Unterschrift
Leipzig	30.9.1999	