Technische Universität Chemnitz-Zwickau
Fakultät Informatik
Lehrstuhl Betriebssysteme

Projektarbeit

Client-Server - Backup für dezentrale Filesysteme auf Basis von *rse*

Angefertigt von Torsten Naumann 03 FIF 91
Chemnitz, den 31. Juli 1996

Betreuender Hochschullehrer : Prof. Dr. W. Kalfa, Fakultät für Informatik
Dipl.-Inf. S. Graupner, Fakultät für Informatik

Aufgabenstellung

In einer heterogenen Workstation-Umgebung mit dezentralen, d.h. nicht montierten Filesystemen auf verschiedenen Workstations, bietet es sich an, eine Client-Server basierte Lösung für das Backup zu verwenden. Das Ziel dieser Projektarbeit ist, eine solche Backup-Lösung für die am Lehrstuhl vorhandenen Filesysteme auf Basis der Arbeit aus [2] herzustellen und zu installieren. Zentraler Teil des Backup-Systems ist ein *Manager*, über den die Abläufe gesteuert werden. Die Backup-Aktivitäten selbst werden durch lokale Proxies aus [2] anhand vorher vom *Manager* geladener Scripte ausgeführt. Es sind jeweils zwei Proxies beteiligt, einer auf der Maschine mit dem Bandlaufwerk und einer auf der Maschine mit dem Filesystem.

Die Anforderungen an das Backup-System sind:

- Backup/Restore wahlweise für alle oder einzelne lokale Filesysteme von Workstations
- Backup/Restore für einen Teilbaum eines Filesystems (im Extremfall einer einzelnen Datei)
- Protokollierung der Backup/Restore-Aktivitäten in einem File (Inhalte, Zeit, Datum, Bandnummer, etc.)
- der *Manager* sollte in geeigneter Form bedienbar sein, um Backup/Restore-Aktivitäten zu veranlassen, zu steuern bzw. eine Konfiguration vorzunehmen (z.B. neue Workstation einbinden, Bandlaufwerk an andere Maschine etc.)
- es ist wünschenswert, das Backup teil-automatisiert ablaufen zu lassen, d.h., auf Veranlassung durch den Bediener sollte eine konfigurierte Folge von Backup-Aktivitäten automatisch ausgeführt werden (Batch-Betrieb)

Ausgabedatum: 01. November 1995
Abgabetermin: 31. Juli 1996

Inhaltsverzeichnis

1 Einleitung **7**

2 Aufgabeanalyse **9**
2.1 Client-Server-Modell 9
2.2 Dezentrale Filesysteme 9
2.3 Backup 9
2.4 Komponenten des Systems 11
2.5 Kommunikation 11
2.6 Protokollierung 13
2.7 Schnittstelle zum Nutzer 14
2.8 Zusammenfassung der Probleme 14

3 Spezifikation **15**
3.1 Basismaschine und Entwicklungsumgebung 15
3.1.1 Beschreibung der *rse*-Komponenten 15
3.2 Spezifikation der funktionellen Anforderungen 17
3.2.1 Beschreibung des Softwaresystems 17
3.2.2 Funktionelle Anforderungen an die Komponenten 18
3.3 Spezifikation der operationellen Anforderungen 19
3.4 Spezifikation der Qualitätsanforderungen 20

4 Entwurf **23**
4.1 Komponenten 23
4.1.1 Aufteilung nach Funktionalität 23
4.1.2 Entwurf der Server 23
4.1.3 Entwurf des Managers (Client) 26
4.2 Kommunikation 28
4.2.1 Protokolle 28
4.2.2 Mechanismen der Kommunikation 29
4.3 Nutzerinterface 30
4.3.1 Befehle 30

5 Implementation **35**
5.1 Systemvoraussetzungen 35
5.2 Implementation der Server 36
5.2.1 ReaderWriter 36
5.2.2 Dumper 36
5.2.3 Start der Server 37
5.3 Implementation des Client 37
5.3.1 Manager 37
5.3.2 Start des `Manag` 39
5.3.3 *rse*-Clients 40

Abbildungsverzeichnis

2.1 Übersicht der WS-Umgebung des Softwaresystems 12
2.2 Flußdiagramm des Softwaresystems 13

3.1 *rse*-Umgebung 17
3.2 Die Komponenten des Systems und ihre Beziehungen untereinander 18

4.1 *multi-process* Architektur 24
4.2 *single-process* Architektur 25
4.3 Architektur des Backup-Systems 26
4.4 Programmablaufplan des `Manag` 28
4.5 Komponenten und Protokolle 29

Tabellenverzeichnis

2.1 Übersicht der WS und ihrer FS-Typen am Lehrstuhl Betriebssysteme 10
2.2 Komponenten des Systems 11
2.3 Erklärung zu Abbildung 2.2 12

4.1 Einstellungen des `RW` 27
4.2 Einstellungen des `DMP` 27
4.3 Manager - ReaderWriter Protokoll 31
4.4 Meldungen des ReaderWriter 32
4.5 Manager - Dumper Protokoll 32
4.6 Meldungen des Dumper 33
4.7 Manager Befehle 34

Kapitel 1

Einleitung

In einer heterogenen Workstationumgebung, wie sie z.B. am Lehrstuhl Betriebssysteme zu finden ist, tritt das Problem auf, Backups von Daten der Filesysteme anzulegen, um gegen eventl. Systemabstürze und damit verbundene Datenverluste abgesichert zu sein.

Ein Backupgerät ist in der Regel nicht an jeder Workstation (WS) zu finden, meist aus Kostengründen. Wenn das Backupgerät transportabel ist, dann bestünde die Möglichkeit, es an die jeweilige Workstation anzuschließen und die Daten abzuziehen. Dagegen spricht, daß Workstations heruntergefahren werden müssen, um ein externes Gerät anzuschließen. Dieser Umstand ist äußerst ungünstig, da die WS vernetzt sind und im Normalfall immer Prozesse darauf laufen, welche Dienstleistungen für andere im Netz erbringen. Ist ein Backupgerät physisch in einer Workstation integriert (also fest eingebaut), so gibt es nur die Möglichkeit die zu sichernden Daten übers Netz auf diese Workstation zu transportieren und dann lokal auf das Backupmedium zu speichern. Das zieht einen relativen großen Arbeitsaufwand nach sich, um sicherzustellen, daß die Daten auch später wieder zurückgeholt werden können (Restore von Daten). Weitere Probleme, die bei dieser Variante auftreten : Es muß genügend Platz zur Zwischenspeicherung auf den Platten der Workstation mit dem Backupgerät sein, Informationen zur Bandbeschriftung und zum Inhalt der Bänder müssen von Hand angelegt und gepflegt werden. Eine weitere Variante wäre das Montieren der Filesysteme per NFS, das erledigt aber lediglich das Problem des Datenzugriffs, d.h. die Daten sind an einer WS verfügbar, nicht jedoch das Problem der Verwaltung der Archive.

Um die beschriebene Aufgabe zu erledigen und dabei die hier beschriebenen Probleme zu umgehen, soll ein Backupsystem auf Basis eines Client-Server-Modells entstehen.

Der Nutzen eines solchen Systems besteht darin, daß ein Nutzer von seiner WS aus alle Daten, die für das Backup in Frage kommen (auch die auf anderen Workstations), in Augenschein nehmen, selektieren und schließlich das Backup starten kann. Die Architektur der einzelnen WS ist dabei völlig ohne Belang ebenso, an welcher sich das Backupgerät befindet.

Kapitel 2

Aufgabeanalyse

2.1 Client-Server-Modell

Diese Modell dient der Beschreibung und Implementierung verteilter Anwendungen. Es wird durch einen Clienten und einen Server beschrieben, welche als Prozesse auf WS zur Ausführung kommen. Die Kommunikation zwischen diesen geschieht über Protokolle. Der Client ist für die Dienstanforderung und der Server für die Diensterbringung zuständig.

Bei diesem Softwaresystem soll der Client in Abhängigkeit der Eingaben des Nutzers die Aktivitäten der Server steuern. Die Server erledigen die Dienste "Daten aufbereiten" und "Daten auf Backupmedium schreiben".

Beispiele für Client-Server-Applikationen:
Network Filesystem (NFS), Filetransfer Protocol (ftp), World Wide Web (WWW)

2.2 Dezentrale Filesysteme

Als dezentrales Filesystem (FS) bezeichnet man physisch getrennte FS auf verschiedenen Workstations, die über ein Netzwerk miteinander verbunden sind. Eine heterogene WS-Umgebung setzt sich aus WS unterschiedlicher Architekturen zusammen, welche zumeist auch unterschiedliche Betriebssysteme und FS-Typen (siehe Tabelle 2.1) integrieren. Die Architektur ist für die Implementierung nicht relevant, da sie auf Script-Ebene durchgeführt wird. Lese- und Schreibvorgang laufen physisch auf Hardware-Ebene oder oberhalb der FS-Ebene (siehe Abschnitt 2.3) ab und sind somit unabhängig vom FS-Typ.

Problem : Die FS befinden sich auf verschiedenen WS, die über ein Netzwerk verbunden sind.

2.3 Backup

Grundlegend kann man zwei verschiedene Arten von Backups unterscheiden. Zum einen das *Platten-Backup* und zum anderen das *FS-Backup*.

Tabelle 2.1: Übersicht der WS und ihrer FS-Typen am Lehrstuhl Betriebssysteme

Host	Architektur	Betriebssystem	Filesystemtyp
bs{0..3}	IBM RISC System/6000	AIX 3.2	jfs
zaphod	SUN Sparc 10	SunOS 5.4	ufs
damokles	SUN Sparc 10	SunOS 5.3	ufs
aurelius	SUN IPC	SunOS 4.1.2	4.2
domitian	SUN IPC	SunOS 4.1.2	4.2
guendel	Intel x86	Linux	ext2

Platten-Backup

Es werden ohne Beachtung der Strukturierung und Belegung der Platte alle Blöcke physisch von der Platte gelesen und sequentiell auf das Backupmedium geschrieben.

- Vorteile:
  - hohe Lesegeschwindigkeit von der Platte, da Streuung der Blöcke keinen Einfluß hat
  - kein Informationen über FS notwendig
- Nachteile:
  - lediglich Archivierung der gesamten Platte
  - selektives Restore nicht möglich
  - jedes Backup hat die volle Größe der Platte (unkomprimiert)
  - keine FS-Informationen verfügbar

FS-Backup

Es werden alles Files des FS entsprechend der vorhandenen Struktur mit allen Attributen gelesen und sequentiell auf das Backupmedium geschrieben.

- Vorteile:
  - FS-Informationen und Attribute werden zugreifbar gespeichert
  - Restore auf beliebige WS
  - selektives Backup/Restore möglich
- Nachteile:
  - Streuung der Blöcke des FS hat Einfluß auf die Lesegeschwindigkeit

Als Backupgerät ist ein Bandlaufwerk (BLW) vorgesehen. Dieses kann theoretisch an jeder beliebigen WS, welche in das Softwaresystem integriert ist, angeschlossen werden. Die Steuerung und das Format der Datenspeicherung auf das BLW übernimmt ein Standard UNIX-Tool wie z.B. `cpio, GNU-tar`. Dadurch ergibt sich die Möglichkeit, ein an WS A geschriebenes Backup auf WS B zurückzuschreiben unabhängig von Architektur und FS-Typ.

2.4 Komponenten des Systems

Für jede der grundlegenden Funktionen ist eine Komponente notwendig. Da diese alle auf verschiedenen WS laufen sollen, ist es günstig, diese ihrer Funktionalität entsprechend zu unterteilen. Zum einen erfolgt eine Unterteilung in Client und Server und zum zweiten nach ihrer Funktion. Daraus ergibt sich Tabelle 2.2. Die Komponenten werden auf den einzelnen WS als Prozesse zur Ausführung kommen. Durch den Einsatz des *rse*-Systems (Remote Scripting Environment [2]) bietet es sich an, diese als Scripte einer interpretierten oder "just in time"-kompilierten Sprache zu implementieren. Es steht dafür eine reichliche Auswahl solcher Sprachen zur Verfügung, als Beispiele seien hier *perl, python, sh, java* genannt.

Tabelle 2.2: Komponenten des Systems

Komponente	Architektur	Funktion	Lokation
ReaderWriter	Server	liest und schreibt FS-Daten	WS mit FS
Dumper	Server	liest und schreibt FS-Daten von dem und auf das Band	WS mit BLW
Manager	Client	steuert und koordiniert den Ablauf, und die Aktivitäten der Server	WS des Nutzers

2.5 Kommunikation

Die vorliegende Dezentralität der Daten und der Umstand, daß das BLW an einer beliebigen WS angeschlossen sein kann, machen es notwendig, das Softwaresystem als eine verteilte Applikation zu entwerfen und zu implementieren. Bei einer solchen müssen die Teilkomponenten zur Steuerung, Synchronisation und zum Datenaustausch miteinander kommunizieren. Da die WS mittels eines Netzwerkes verbunden sind und sich dessen bereits als Kommunikationsmedium bedienen, bietet es sich an, dies mitzubenutzen. Als gemeinsames Protokoll wird das IP (Internet Protocol) genutzt. Zwischen den Komponenten werden unterschiedliche Anforderungen an die Kommunikation gestellt, so daß für jede Verbindung ein Protokoll benötigt wird. Diese sind nicht zu verwechseln mit den Protokollen, welche auf Netzwerkebene ablaufen. Hier handelt es sich um Protokolle, welche im OSI Referenzmodell in der Schicht 7 (Application Layer) anzusiedeln sind, während dessen das IP in die Schicht 3 (Network Layer) [7] eingeordnet werden muß. Die Protokolle realisieren *Daten-* und *Steuerflüsse*. Ein *Datenfluß* beinhaltet lediglich Daten, welche transportiert werden (z.B. `tar`-Daten), jedoch keine Informationen, die die Funktionen der Komponenten beeinflussen. Ein *Steuerfluß* hingegen beinhaltet Informationen, welche als Kommandos interpretiert werden und damit Einfluß auf das Verhalten der Komponenten haben. Kommandos werden genutzt, um die Synchronisation, Nachrichtenübertragung und Ablaufsteuerung zu gewährleisten. Die Kommunikationsstruktur wird in Abbildung 2.2. veranschaulicht. Tabelle 2.3 erklärt die Datenströme.

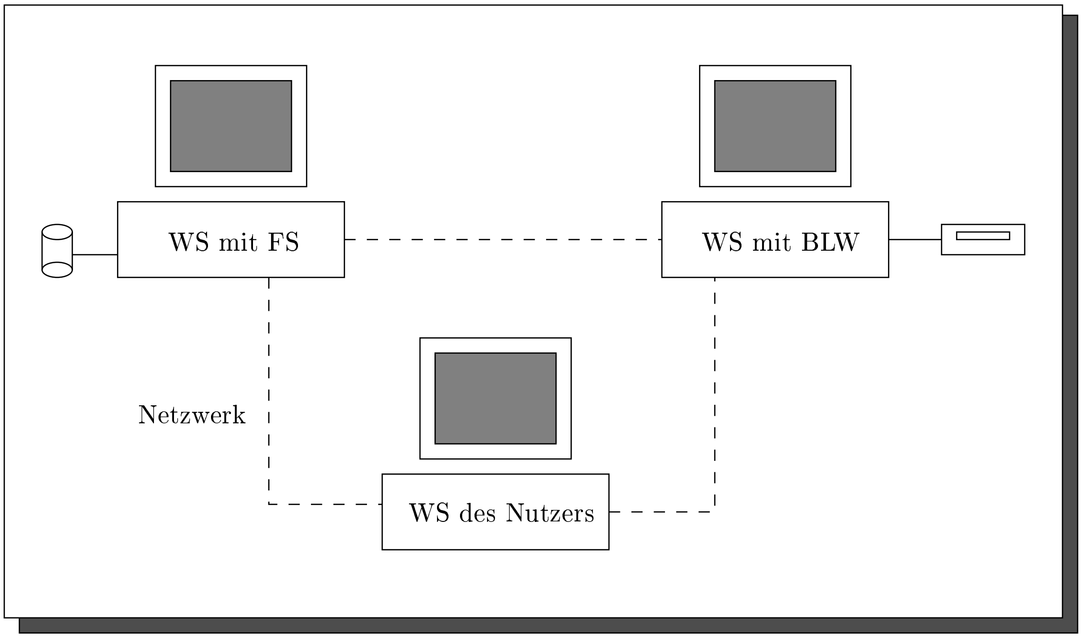

Abbildung 2.1: Übersicht der WS-Umgebung des Softwaresystems

In Abbildung 2.1 ist das Szenario dargestellt. Es wurde nur eine WS mit FS eingezeichnet, es sind aber durchaus mehr möglich. Für das Softwaresystem stellt die Abbildung den *worst case* dar, da sich alle Komponenten auf unterschiedlichen Maschinen befinden. Dieser Ausgangspunkt wird angenommen, weil er alle anderen Fälle abdeckt, so z.B. den, daß das FS, das BLW und der Nutzer sich an einer WS befinden.

Tabelle 2.3: Erklärung zu Abbildung 2.2

Daten/Steuerfluß	Funktion
Steuerfluß(1,2)	Koordinieren und Steuern der entsprechenden Datenflüsse
Datenfluß(1)	Informationen des FS (Filename und -attribute) Daten über den Zustand des Servers Statusmeldungen (Backup gestartet/beendet)
Datenfluß(2)	Daten über das eingelegte Band Daten über den Zustand des Servers Statusmeldungen (Fehler, Schreibvorgang beendet)
Datenfluß(3)	FS-Daten zur Archivierung Backupdaten zum Restore der FS-Daten

Probleme:

- Es ergibt sich die Frage, ob ein einheitliches Protokoll zum Einsatz kommt, oder ein für jede Teilfunktion dediziertes.

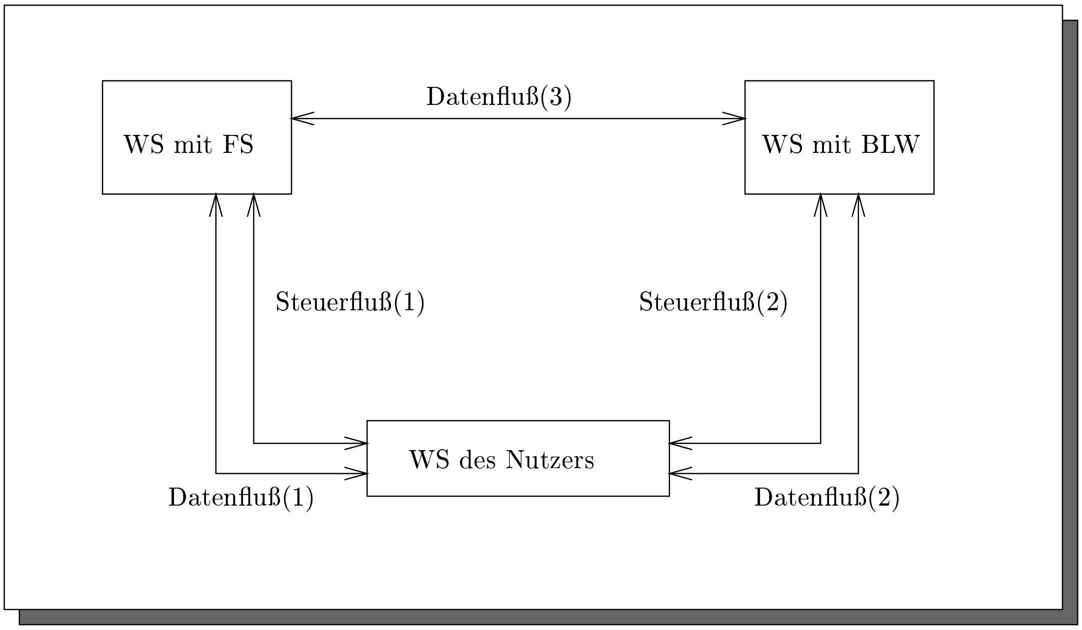

Abbildung 2.2: Flußdiagramm des Softwaresystems

- Welche Kommunikationsart (asynchron, synchron, versuchend, umlenkend) der Prozesse bietet sich hier an?

2.6 Protokollierung

Um gesicherte Daten wieder auf das entsprechende FS zurückschreiben zu können, muß bekannt sein, auf welchem Band sich diese befinden. Das macht eine Protokollierung aller Backup-Aktivitäten notwendig. Da von verschiedenen WS im Netzwerk Backups angelegt werden sollen, sind folgende Informationen für ein späteres *Restore* notwendig.

- Zur Band-Identifizierung
  - Bandlabel
- Für eine Backup-Session
  - Bandlabel
  - Hostname
  - Datum des Backup-Starts
  - ein Eintrag pro File
- Zur File-Identifizierung
  - vollqualifizierter Filename
  - Eigentümer (uid, gid)

- Zugriffsrechte
- Größe
- Modifikationsdatum

Problem: Wird Backup von Host A auf Band archiviert, ist dann die Rücksicherung auf anderen Host als A möglich ?

2.7 Schnittstelle zum Nutzer

Dem Nutzer sollen auf Anforderung die FS-Informationen der im System eingebundenen WS und Statusinformationen über den Verlauf der Aktivitäten angezeigt werden. Die FS-Informationen müssen immer in Verbindung mit dem zugehörigen Hostnamen angezeigt werden. Des weiteren ist eine Möglichkeit zu realisieren, welche dem Nutzer Statusinformationen auf explizite Anforderung zur Verfügung stellt.

Die Kommandoeingabe wird am Prompt des Managers erfolgen. Es wird eine einfache kommandobasierte Schnittstelle zu Verfügung gestellt.

2.8 Zusammenfassung der Probleme

- Die FS befinden sich auf verschiedenen WS, die über ein Netzwerk verbunden sind.
- Es ergibt sich die Frage, ob ein einheitliches Protokoll zum Einsatz kommt, oder ein für jede Teilfunktion dediziertes.
- Welche Kommunikationsart (asynchron, synchron, versuchend, umlenkend) der Prozesse bietet sich hier an?
- Wird Backup von Host A auf Band archiviert, ist dann die Rücksicherung auf anderen Host als A möglich?

Kapitel 3

Spezifikation

3.1 Basismaschine und Entwicklungsumgebung

Als Basismaschinen dienen die WS des Lehrstuhls Betriebssysteme, wie sie in Tabelle 2.1 aufgeführt sind. Die Betriebsystemeplattform ist durchgängig UNIX. Das Einbinden neuer WS in dieses System übernimmt die Softwarelösung aus [2]. Der Einsatz dieser zieht die Implementation der Komponenten in einer Scriptsprache nach sich. Dies gewährleistet gleichzeitig Unabhängigkeit von allen vorhandenen WS-Architekturen, womit gesichert ist, daß dieses Backupsystem in einer heterogenen WS-Umgebung ohne Anpassungen lauffähig ist. Es bestünde die Möglichkeit, den *Manager* in C zu implementieren, da er nicht über den oben genannten Mechanismus verteilt wird, aber aus Gründen der Portierbarkeit wird dieser ebenfalls in einer Scriptsprache implemtiert. Somit ist der *Manager* auf jeder beliebigen WS einsetzbar, ohne daß er neu übersetzt werden muß. Alle Komponenten werden in der Sprache `perl` (Practical Execution and Report Language) [13] implemtiert. Für die Integration des *rse* wird die `rse-C-Library` benötigt. Die Sicherung der Daten auf das BLW übernehmen `GNU-tar` und `dd`. Weiterhin werden noch andere Standard UNIX-Tools wie z.B. `ls, hostname` benutzt.

Konkrete Entwicklungs- und Laufzeitumgebung:

- `perl 5.002`
- `rse (Version 1.0)`
- `GNU-make (Version 3.70)`
- `GNU-tar (Version 1.11.8)`

Nachtrag: Das Softwaresystem *rse* [2] liegt zur Zeit nur für die Architekturen SUN4, SUN5 und Linux vor.

3.1.1 Beschreibung der *rse*-Komponenten

Siehe Abbildung 3.1

Auf jeder beteiligten WS befindet sich ein *rse*-Agent (A), der über ein Basisprotokoll von

Client (C) auf anderen WS angesprochen werden kann. Im Basisprotokoll spiegelt sich die Funktionalität der *rse*-Umgebung wider. Ausführungssubjekte sind *rse*-Nutzer, unabhängig von welcher Maschine sie sich über den Client an den *rse*-Agenten wenden. Nutzer müssen dort bekannt sein und ihre Identität nachweisen, was die Basis für Berechtigungen ist. Beim *rse*-Agenten wird eine lokale Script-Datenbasis verwaltet, in der Nutzer Scripts eintragen, ändern und entfernen können, wenn sie die Berechtigung dazu besitzen. Aus dieser Datenbasis können Scripts zur Ausführung gebracht werden ("linken") und ein "Ausführungs-Agent" (A' = Interpreter) kann gestartet werden. Über das Basisprotokoll wird eine Verbindung A' $\leftrightarrow$ C vermittelt, über die das aufgabenspezifische Dienstprotokoll mit dem Client abläuft. Die Dienstausführung durch A' erfolgt vollständig scriptgesteuert. Über das Basisprotokoll kann auch eine Steuerung von A' erfolgen (Signale). Das genannte Dienstprotokoll bezeichnet die im Text schon erwähnten Protokolle zwischen den Komponenten des Systems.

In Abbildung 3.1 ist der Client sowohl für die Abwicklung des Basisprotokolls als auch für das Dienstprotokoll zuständig. Beide Aufgaben können aber auch in zwei Prozesse entkoppelt werden. Auf der Client-Seite gibt es eine Programierschnittstelle (Bibliothek) und eine kommandoorientierte Dialogschnittstelle für das Basisprotokoll. Das Basisprotokoll umfaßt grob folgende Funktionen:

- Nutzeridentifikation
- Manipulation der Script-Datenbasis (Laden, Entfernen, Ändern von Scripts ...)
- Auswahl, Zusammenstellung von Scripts und Start des Interpreters der Ausführung
- Manipulation von Rechten für Scripts

Die Nutzerverwaltung wird von *rse* selbst vorgenommen, so daß ein *rse*-Nutzer kein UNIX-Nutzer auf einer Maschine sein muß. Bei der Scriptausführung wird dann eine Abbildung vorgenommen: *rse*-Nutzer $\rightarrow$ `uid` des gestarteten Interpreter-Prozesses. Der *rse*-Agent muß dafür unter der `uid` 0 (root) laufen. Die Scriptausführung kann dann prinzipiell unter einer beliebigen `uid` x ablaufen, z.B. als `'nobody'` für Nicht-UNIX-Nutzer oder unter der `uid` eines UNIX-Nutzers oder auch unter der `uid` 0. Die letztgenannte Variante ist vor allem für Aufgaben der entfernten Systemadministration wichtig. Dieses Prinzip schafft viel Spielraum. Nutzer können aufgabenbezogene Scripts unter der `uid` 0 ausführen, ohne ein `root`-login für diese Maschine zu besitzen, natürlich nur, wenn sie autorisiert sind. Für die Verwaltung eines *rse*-Agenten muß es einen speziellen *Administrator* geben, der allein Zugang zur Datenbasis der lokal eingetragenen Nutzer hat und diese Manipulieren darf. Für jeden Nutzer werden dessen Identität und ein öffentlicher Schlüssel zum späteren Nachweis seiner Identität beim *rse*-Agenten gespeichert. Die eigene Verwaltung von Nutzern schafft Unabhängigkeit und wird auch in anderen Systemen praktiziert, z.B. *afs* oder *DCE*. Die *remote-shell* von UNIX und auch die *secure-shell* [15] beziehen sich hingegen auf UNIX-Nutzer.

Der Client benutzt Funktionen der *rse*-Umgebung, die in einer C-Bibliothek bereitstehen. Er beherrscht demzufolge das Basisprotokoll der *rse*-Umgebung und das Dienstprotokoll der eigentlichen Anwendung. Nachdem der Client sich authentifiziert und den entsprechenden

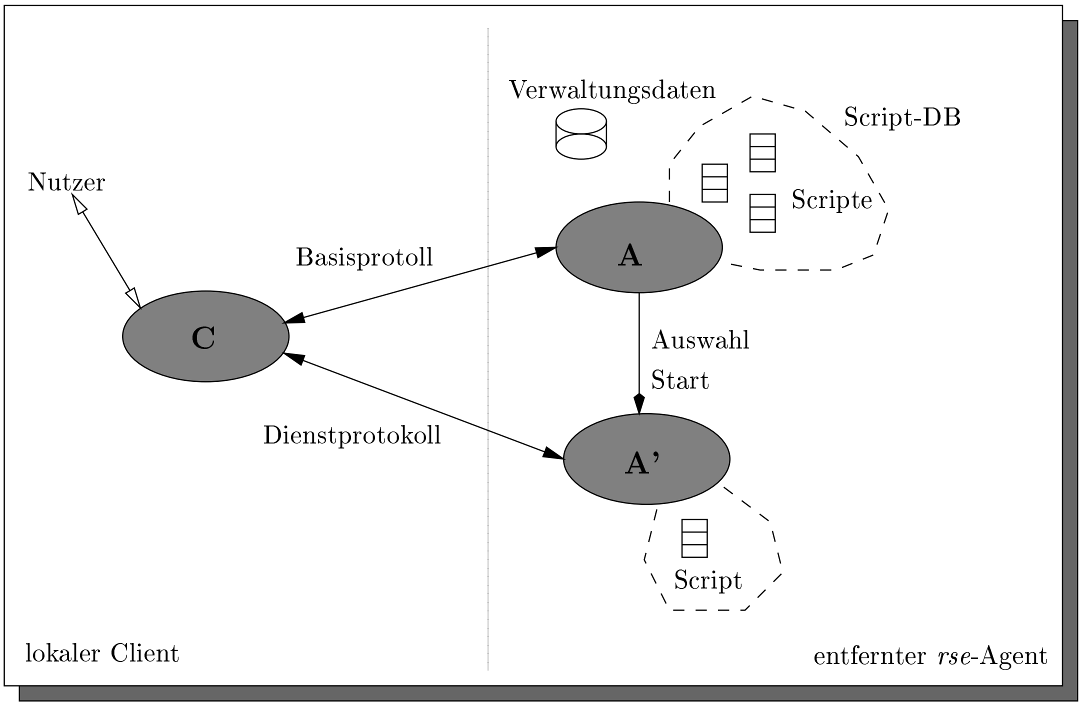

Abbildung 3.1: *rse*-Umgebung

Dienst gestartet hat, läuft die Kommunikation der Komponenten ab wie bei traditionellen Client-Server-Anwendungen. Ab diesem Moment bleibt das Basisprotokoll außen vor.

3.2 Spezifikation der funktionellen Anforderungen

3.2.1 Beschreibung des Softwaresystems

Leistungsumfang:

Das Softwaresystem soll es ermöglichen, daß in einer heterogenen WS-Umgebung von allen WS Backups gemacht werden können, ohne daß dabei die FS zentral, z.B. an der WS mit dem BLW, montiert sind. Es soll über ein Nutzerinterface eine einfache Möglichkeit bieten auf einer ausgewählten WS ein *Backup/Restore* anzuweisen, unabhängig davon, an welcher WS sich das BLW befindet. Zum *Backup* werden der entsprechende Host und dessen zu archivierender Teilbaum des FS ausgewählt und das Kommando zum Start gegeben. Während dieser Zeit kann der Nutzer den Verlauf der Aktivität anhand von Statusinformationen überwachen. Zum *Restore* wird aus den Backup-Protokollen der wiederherzustellende Teilbaum für einen Host ausgewählt. Nach Start des *Restore* kann auch diese Aktivität, wie oben erwähnt, überwacht werden. Das Einspielen eines Backups auf Host A, welches auf Host B gemacht wurde, ist prinzipiell möglich. Es kann allerdings zu Dateninkonsistenzen auf Host A kommen, wenn dieser einen anderen Verzeichnisbaum aufweist. Aus diesem Grund sollte dies nicht getan werden und wird auch vom System nicht explizit unterstützt.

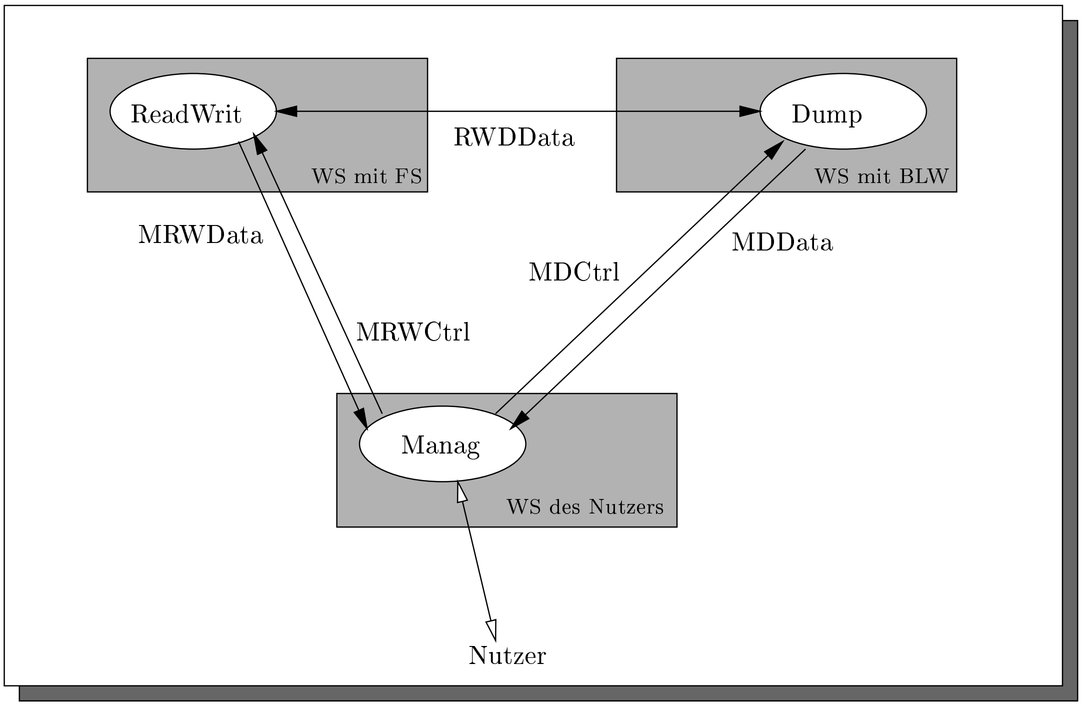

Abbildung 3.2: Die Komponenten des Systems und ihre Beziehungen untereinander

Nutzer:

Vom Nutzer wird angenommen, daß er das UNIX-System und dessen Filesystemkonzept speziell die Attribute (siehe Abschnitt 2.6) kennt.

3.2.2 Funktionelle Anforderungen an die Komponenten

Die folgenden Formulierungen stellen lediglich die grundlegende Arbeitsweise des Softwaresystems auf der Basis logischer Funktionsgruppen dar. Eine feste Abbildung auf konkrete Implementationsformen (z.B. als Funktionen prozeduraler Sprachen o.ä.) findet zu diesem Zeitpunkt nicht statt.

Gesamtfunktion:

Das Gesamtsystem ist in Abbildung 3.2 dargestellt. Die `Manag`-Komponente bedient das ganze System. Sie ist für den Auf-und Abbau der Verbindungen zu den anderen Komponenten und deren Start verantwortlich. Die `Dump`-Komponente steuert das Lesen und Schreiben des Bandes. Sie erhält die zu schreibenden Daten von der `ReadWrit`-Komponente. Die `ReadWrit`-Komponente managt die Lese-und Schreibzugriffe auf das FS. Sie bekommt vom `Manag` die Kommandos zum *Backup/Restore* und initiiert den Verbindungsaufbau zum `Dump`. Ein Kommando besteht immer aus einer Anforderung (*Request*) und einer Antwort (*Response*).

Teilfunktionen:

`Manag`:
Die Komponente kann unabhängig von den anderen gestartet und beendet werden. Sie bietet dem Nutzer an, die anderen Komponenten des Systems zu starten bzw. wenn sie schon laufen, eine Verbindung zu ihnen aufzubauen. Starten und Beenden der Server werden über die Funktionen des *rse* gesteuert.

`ReadWrit: (RW)`
Diese Komponente wird mittels des *rse* auf eine WS transportiert. Die gesamte Steuerung dieser ist nur über die Komponente `Manag` möglich. Die Aufgabe dieser Komponente ist es, den Zugriff auf das FS dieser WS zu ermöglichen und die angeforderten Daten zu den anderen Komponenten zu übermitteln.

`Dump: (DMP)`
Diese Komponente wird mittels des *rse* auf eine WS transportiert. Die gesamte Steuerung dieser ist nur über die Komponente `Manag` möglich, wobei auch `ReadWrit` aktiv Verbindungen zu ihr auf- und abbauen kann. Aufgabe von `Dump` ist die Steuerung das BLW, also Lesen und Schreiben der Daten von dem und auf das Band.

Es ist auch ohne weiteres möglich, alle Komponenten auf ein und derselben WS laufen zu lassen, denn Netzwerkkommunikation ist auch lokal auf einer WS möglich.

Datenströme:

`MRWCtrl`: Steuerfluß für MRWData (Befehle)

`MRWData`: Datenaustausch zischen `Manag` und `RW` (Konfigurationsdaten, FS- und Statusinformationen)

`MDCtrl`: Steuerfluß für MDData (Befehle)

`MDData`: Datenaustauch zwischen `Manag` und `DMP` (Konfigurationsdaten, Band- und Statusinformationen)

`RWDData`: Datenaustausch zwischen `RW` und `Dump` (FS-Daten)

3.3 Spezifikation der operationellen Anforderungen

Die Benutzung des Systems geschieht im Dialogbetrieb, Echtzeitanforderungen bestehen nicht. Es soll jedoch auch die Möglichkeit zum Ausführen eines Kommandostapels vorgesehen werden (Batchbetrieb). Die Bearbeitungszeit eines Auftrages (Backup/Restore) ist in dieser Arbeit nicht relevant. Es werden Datenbasen zur Speicherung der Backup-Protokolle aufgebaut und verwaltet. Die Größe dieser läßt sich im voraus nur schwer abschätzen, da es von solchen Faktoren wie der Größe des zu archivierenden/zurückzuschreibenden Teilbaums und der Länge der benutzten Namen abhängt. Das gleiche gilt auch für die Mächtigkeit der

Datenströme. Wenn z.B. ein Teilbaum archiviert wird mit wenigen aber sehr großen Files, dann ist die Datenmenge der zu schreibenden Daten im Verhältnis zur Datenmenge der Protokollierung wesentlich höher, besteht der Teilbaum aber aus vielen kleinen Files, dann bestünde sogar die Möglichkeit, daß die Datenmenge der Protokollierung gleich dem der zu schreibenden Daten oder mächtiger wird. Das Datenaufkommen der Steuerflüsse kann auf Grund seines geringen Umfangs vernachlässigt werden. Die erzeugte Last und das Lastverhalten des Systems können also lediglich durch statistische Messungen während der Laufzeit abgeschätzt werden.

3.4 Spezifikation der Qualitätsanforderungen

Stabilität:

Das System soll auftretende Fehler weitestgehend behandeln und dem Nutzer in geeigneter Form zu erkennen geben und danach in einen definierten Zustand übergehen. Ein Fehler bei einer laufenden Aktivität sollte lediglich zum Abbruch dieser führen und nicht das gesamte System beeinflussen. Die Nutzerführung geschieht über Kommando-Eingabe, so daß fehlerhafte Eingaben seitens des Nutzers möglich sind, das System sollte diese Art von Eingaben als solche erkennen und begründet zurückweisen. Eine Rücknahme veranlaßter Aktionen ist nicht vorgesehen.

Hardwarefehler können und sollten von einer Applikation nicht behandelt werden. Fehler in der Basissoftware (Betriebssystem, Interpreter, Laufzeitumgebung) und in den verwendeten Mechanismen (*rse*) führen zum Abbruch des Programms, wenn sie die Weiterarbeit schwerwiegend behindern. Es ist nicht sinnvoll, dieses System auf einer WS zu betreiben, welche eine der Anforderungen (Netzwerkanschluß, BLW) nicht erfüllt.

Effizienz:

Die Anzahl der Rechner, die gleichzeitig Daten sichern können, beträgt 1, so daß in einer Laufzeitumgebung mit BLW-Aktivitäten immer nur eine Komponente mit genau einem Partner kommunizieren kann. Es besteht die Möglichkeit, daß sich ein weiterer `Manag` bei den Servern anmeldet, jedoch das Anweisen von Aktivitäten wie z.B. Backup ist nicht möglich, da der Zugriff auf des Band einem Prozeß exklusiv gegeben wird. Es gibt keinen Sinn, daß z.B. mehrere `Dump`-Komponenten gleichzeitig auf ein und dasselbe Band schreiben.

Flexibilität, Erweiterbarkeit:

Durch den Einsatz einer Scriptsprache und des *rse*-Mechanismus beschränkt sich das Einsatzgebiet auf die Architekturen, welche vom *rse* unterstützt werden. Eine Portierung des *rse* auf Architekturen, die zur Implementationszeit dieses Systems noch nicht unterstützt wurden, hat zur Folge, daß dieses sofort auch auf den neu hinzugekommenen Architekturen lauffähig ist. Das gesamte System ist modularisiert, so daß ein Austausch z.B. des Nutzerinterfaces einen geringen Aufwand nach sich zieht.

Verständlichkeit:

Das System setzt die Kenntnis über den prinzipiellen Aufbau vom UNIX-FS und den

Umgang mit einem BLW voraus. Ein Administrator als potentieller Nutzer sollte damit keine Schwierigkeiten haben, das System korrekt zu bedienen.

Kapitel 4

Entwurf

4.1 Komponenten

4.1.1 Aufteilung nach Funktionalität

Die Komponenten des Systems unterscheiden sich in der Art ihrer Funktionalität und in ihrem Design. Es gehören 2 Server unterschiedlicher Funktionalität und ein Client dazu.

Server: ReaderWriter (`RW`), Dumper (`DMP`)

Client: Manager (`Manag`)

Allerdings stimmt diese Aussage nur zum Teil. Bei der Verbindung, über welche die Backupdaten zwischen dem `RW` und dem `DMP` ausgetauscht werden, muß ersterer die Client-Rolle übernehmen, somit beinhaltet dieser auch Client-Funktionalität, wohingegen der `DMP` einer reiner Server ist. Das System muß lediglich auf einer WS installiert werden. Den Service, daß es auch auf anderen WS verfügbar gemacht wird, übernimmt *rse*, welches lediglich im `Manag` enthalten ist. Das System sollte dementsprechend vorzugsweise auf der WS des Backup-Nutzers installiert sein. Wenn die Bedienung auch von anderen WS aus erlaubt werden soll, müssen lediglich der `Manag` und die dazugehörigen Files (Konfiguration, Bibliotheken) auf eben diesen installiert werden, jedoch ist dann von einer solchen kein *Update* bzw. *Laden* des Server-Codes zum *rse*-Agenten möglich.

4.1.2 Entwurf der Server

Folgende Anforderungen müssen von den Servern erfüllt werden:

- hohe Verfügbarkeit
- Fehlertoleranz (kein Absturz bei vom Client verursachten Fehlern)
- Parallele Bearbeitung der Anforderungen mehrerer Clients

Um diesen Anforderungen gerecht zu werden, existieren mehrere Ansätze. [4] enthält eine ausführlichere Beschreibung und Ausführungen zu den Vor-und Nachteilen dieser Lösungen. Grundsätzlich existieren zwei Varianten:

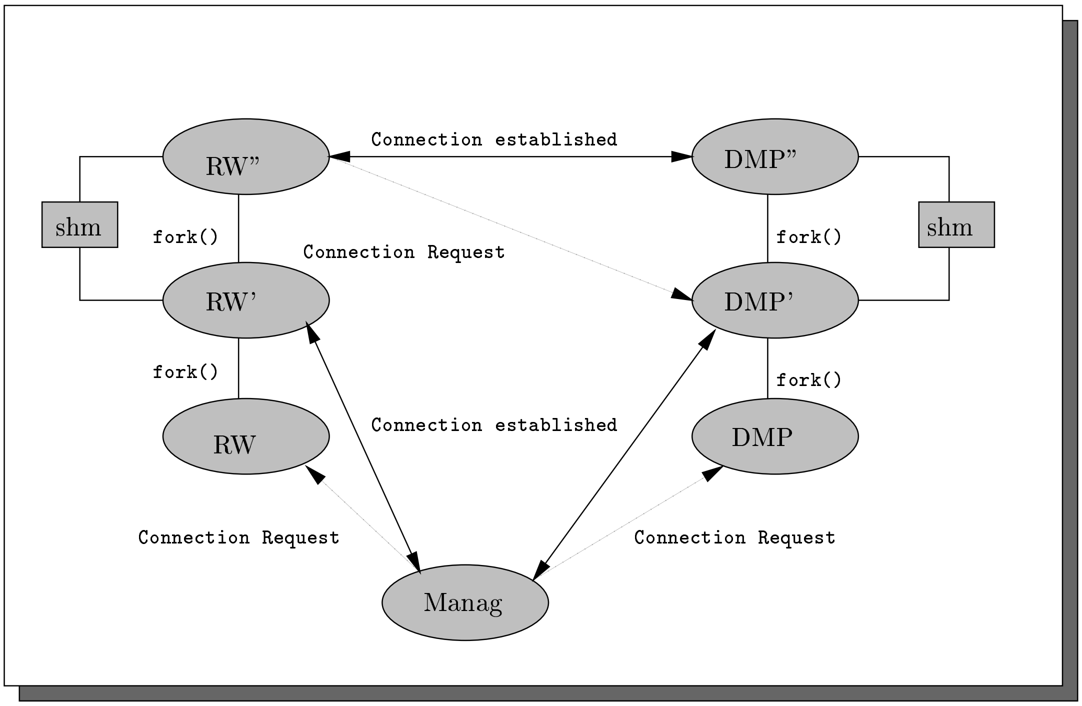

Abbildung 4.1: *multi-process* Architektur

Variante - paralleler Server

Nach dem Start des Servers wartet dieser auf Verbindungswünsche von Clients, bei Eintreten eines solchen nimmt der Server die Verbindung an, erzeugt einen Socket und startet mittels Systemruf `fork()` eine Kopie seiner selbst und geht dann wieder in den Wartezustand. Man nennt diese Architektur auch *multi process* siehe Abbildung 4.1. Das Kind des Servers bearbeitet die Anforderungen des Client, ohne den Ablauf des Master-Servers zu beeinflussen. Lediglich bei seiner Beendigung schickt er das Signal `SIGCHLD` an seinen Elternprozeß zur Kenntnisnahme. Ein Vorteil liegt in einer hohen Verfügbarkeit des Master-Servers, da er keine Bearbeitung macht, sondern lediglich die Verbindung annimmt und zu seinem Kind weiterreicht. Als Vorteil erweist sich die Robustheit gegenüber Abstürzen, da nur ein geringer Teil Code ausgeführt werden muß. Zum dritten erfolgt eine faire Reaktion auf Verbindungswünsche, weil je Verbindung ein Prozeß existiert. Einen weiteren Vorteil stellt die Fehlertoleranz dar, weil es sich um einen zustandslosen Server handelt. Als nachteilig ist festzustellen, daß diese Variante nicht speichereffizient arbeitet, da pro Verbindung ein Prozeß existiert und die Kommunikation zwischen den Prozessen nur über Signale, Shared Memory, Pipes oder FIFOs möglich ist. Diese Mechanismen wiederum bedingen eine Synchronisation des Zugriffes.

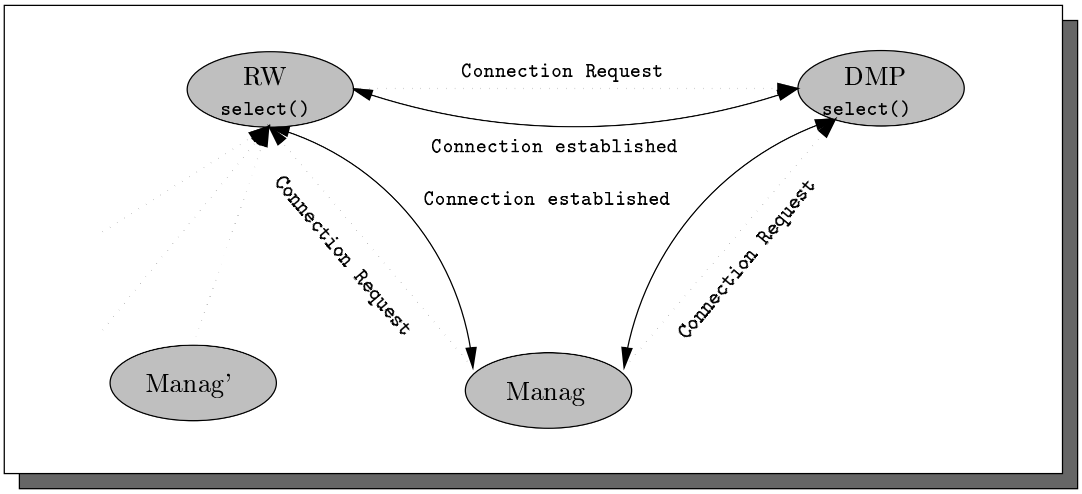

Abbildung 4.2: *single-process* Architektur

Variante - iterativer Server

Nach dem Start des Servers wartet dieser auf Verbindungswünsche von Clients, bei Eintreten eines solchen nimmt der Server die Verbindung an und bearbeitet die Anforderungen des Client. Jeder weitere Client wird von demselben Prozeß akzeptiert und bearbeitet. Man nennt diese Architektur auch *single process* siehe Abbildung 4.2. Um zu gewährleisten, daß die Anforderungen fair bearbeitet werden, ist es notwendig, einen Mechanismus zu implementieren, welcher ständig an allen Sockets hört. Dafür bestünde die Möglichkeit, alle Sockets permanent abzufragen, dies wird als *busy waiting* bezeichnet, da der Prozeß dazu Rechenzeit benötigt. Eine bessere Variante ist, daß der Prozeß in den `sleep`-Zustand wechselt, somit keine Rechenzeit verbraucht und bei Eintreffen von Anforderungen aufgeweckt wird. Diese Funktionalität steht mit dem Systemruf `select()` zur Verfügung. Vorteile sind die Speichereffizienz und das Vorhandensein aller relevanten Daten innerhalb eines Prozesses, was die IPC für diesen Fall überflüssig macht. Die Nachteile sind allerdings nicht zu vernachlässigen, wenn nur ein einizger Client den Server zum Absturz bringt, werden alle anderen Client-Verbindungen in Mitleidenschaft gezogen.

Architektur der Serverkomponenten

Unter Berücksichtigung der aufgezählten Vor-und Nachteile ist folgende Lösung gewählt worden:

Es existiert ein sog. Master-Server, welcher als paralleler Server arbeitet, also für jede neue Verbindung mittels des Systemrufs `fork()` einen Kindprozeß erzeugt, welcher von nun an mit dem Client kommuniziert. Der Kindprozeß arbeitet als iterativer Server, überwacht also alle seine Verbindungen mittels des blockierenden Systemrufs `select()`. Mit dieser Kombination ist gewährleitet, daß für jeden neuen Client ein Kommunikationspartner "geschaffen" wird. Diese Kombination ist das allgemein übliche Design von Client-Server-Anwendungen

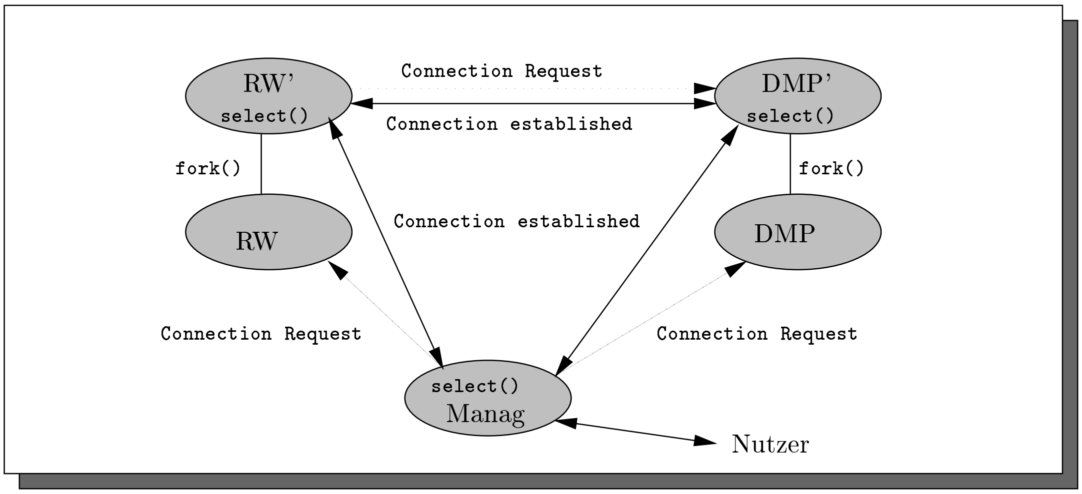

Abbildung 4.3: Architektur des Backup-Systems

auf UNIX-Plattformen, so z.B. `ftp`.

Konfiguration der Server

Die Server lassen sich über das Setzen von Variablen und Anweisen von Aktionen steuern und konfigurieren. Die Einstellungen sind in den Tabellen 4.1 und 4.2 aufgeführt.

Die Befehle zum Manipulieren dieser Variablen sind mit Syntax und Semantik in Abschnitt 4.2.1 erläutert.

4.1.3 Entwurf des Managers (Client)

Der Manager ist die zentrale Komponente in diesem System. Er enthält die Schnittstelle zum Nutzer und muß somit die gesamte Funktionalität dem Nutzer in geeigneter Form zur Verfügung stellen. Seine primären Aufgaben sind die Interaktion mit dem Nutzer, die Verteilung der Server-Komponenten in der WS-Umgebung und die Steuerung dieser. Es besteht zwar die Möglichkeit, das System ohne ihn zu bedienen, da die Protokolle zur Steuerung als Klartextprotokolle ausgelegt sind, aber dies erfordert genaue Kenntnisse der internen Abläufe, so daß diese Variante nur zu Testzwecken herangezogen werden sollte. Zur Steuerung ohne den Manager bietet sich der `telnet`-Client an, welcher auf allen Systemen verfügbar ist.

Wie schon beschrieben, handelt der `Manag` das Basis- und das Dienstprotokoll ab, das bedeutet, er muß für das Basisprotokoll die *rse*-Library benutzen. Dafür gibt es 2 Möglichkeiten für die hier gewählte Scriptsprache `perl`:

1. Erweiterung des `perl`-Interpreter um die Funktionalität des *rse*.
   Vorteil:
   - alle Komponenten liegen komplett in `perl`-Scripts vor

   Nachteil:

Tabelle 4.1: Einstellungen des `RW`

Variable	Bedeutung	Wert
`dumphost`	Host mit BLW und DMP	host.domain oder IP-Adresse
`dumpdataport`	Port beim DMP für die Datenverbindung (wird automatisch vom Manag gesetzt)	1024..65536
`rwtree`	vollqualifizierter Pfad für das Backup	path *space* path
`restorelist`	Files oder Pfade zum selektiven Restore	path/file *space* path/file
`rwafter`	Alle Files ins Backup, die nach `rwafter` modifiziert wurden	`DDMonthYYYY hh:mm:ss` <`timezone`> (22Jul1996 16:00 MET)

Tabelle 4.2: Einstellungen des `DMP`

Variable	Bedeutung	Wert
`dumpdataport`	Port beim DMP für die Datenverbindung (wird automatisch vergeben)	1024..65536
`blocksize`	Blockgröße zum Schreiben auf das Band	Vielfache von 2048 (default: 10240)
`dumpdevice`	Backupgerät	`/dev/rst0` es get auch ein `Filename`
`dumpdevmode`	Mode für das Backupgerät (vom `Manag` automatisch gesetzt)	`read`|`write`

- Neuübersetzung des `perl`-Interpreter für alle Architekturen

2. Integration der *rse*-Funktionalität in einem C-Programm und Aufruf vom `perl`-Script: Vorteil:

   - *rse* transparent aus `perl`-Script nutzbar

   Nachteil:

   - C-Programm auf jeder Architektur übersetzen

Es wird die Variante mit den seperaten C-Programmen gewählt, da bei der anderen der Vorteil der Benutzung einer Scriptsprache verlorengeht und damit auch der der Systemunabhängigkeit.
In Abbildung 4.4 ist der PAP dargestellt.

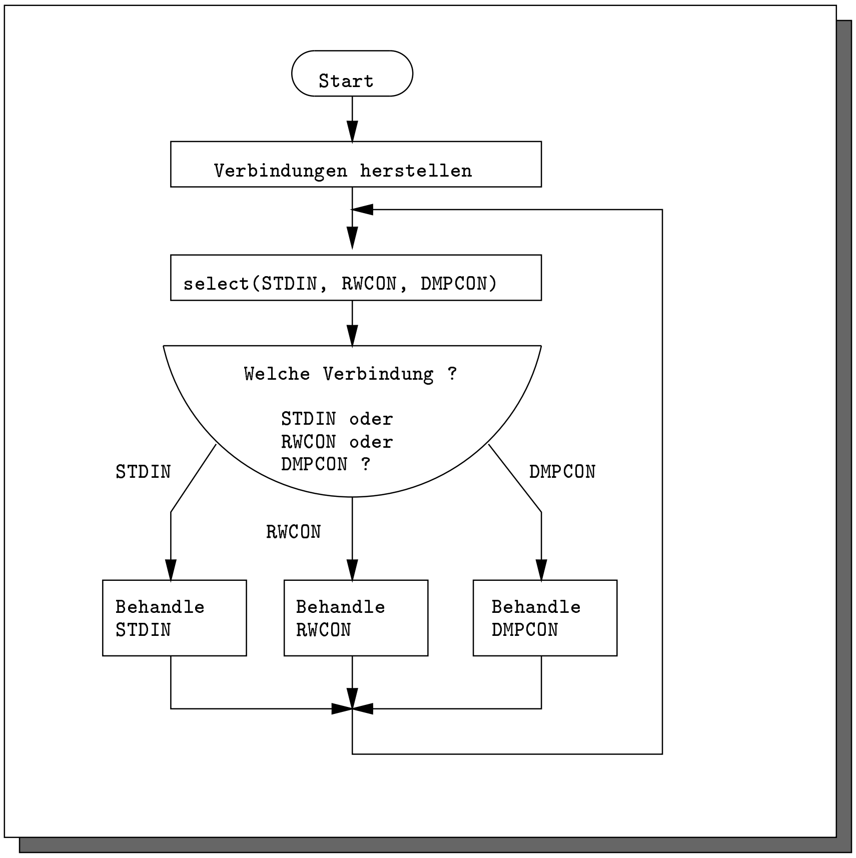

Abbildung 4.4: Programmablaufplan des `Manag`

4.2 Kommunikation

Die Kommunikation wird über Protokolle abgewickelt, wobei zwischen dem Basis- und dem Dienstprotokoll unterschieden werden muß. Das Basisprotokoll wird über die Funktionen in *rse*-Library zur Verfügung gestellt und sind für die weiteren Betrachtungen unerheblich. Im folgenden bezeichnet Protokoll das Dienstprotokoll, welches die Funktionalität des Systems widerspiegelt. Es kommen dedizierte Klartext-Protokolle zum Einsatz. Der Einsatz von Klartextprotokollen erleichtert die Lesbarkeit der Quelltexte, hier ASCII-Files mit `perl`-Syntax, und die Testphase. Der Overhead eines Klartextprotokolls gegenüber einem Protokoll, welches lediglich kurze Codes schickt, die in der weiteren Verarbeitung noch mittels einer Tabelle umgesetzt werden müssen, ist zu vernachlässigen. Für jede Verbindung wird es ein eigenes Protokoll geben (siehe Abbildung 4.5).

4.2.1 Protokolle

Zum Basisprotokoll der *rse* sollen hier keine weiteren Ausführungen gemacht werden, dies alles ist ausführlich in [2] erklärt.

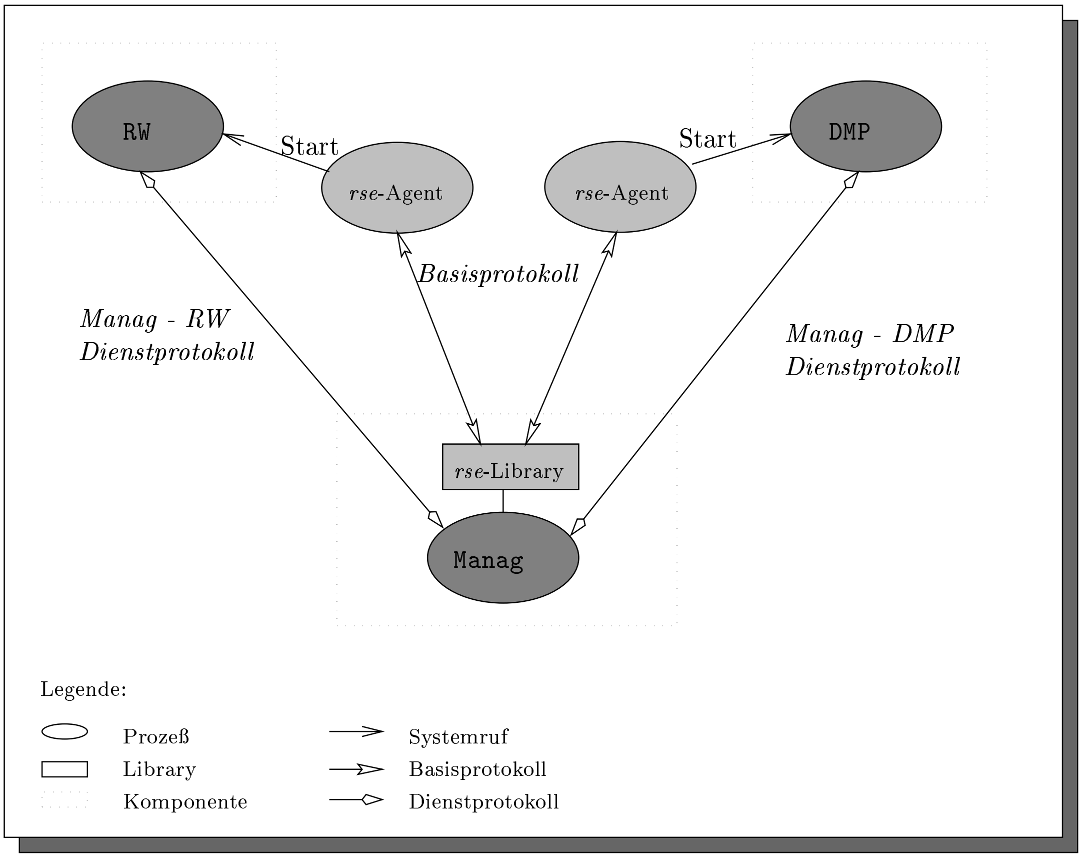

Abbildung 4.5: Komponenten und Protokolle

- `Manag - RW`
  - Anforderungen (Requests) siehe Tabelle 4.3
  - Antworten (Response) siehe Tabelle 4.4
- `Manag - DMP`
  - Anforderungen (*Requests*) siehe Tabelle 4.5
  - Antworten (*Response*) siehe Tabelle 4.6

4.2.2 Mechanismen der Kommunikation

Die traditionellen Methoden der Interprozeßkommunikation wie Pipes, FIFOs, Messages, Shared Memory, Signale und Semaphoren funktionieren nur lokal auf einer WS. FIFOs (auch Named Pipes genannt) setzen ein gemeinsames Filesystem voraus, welches hier nicht zur Verfügung steht. Daraus schlußfolgernd bieten sich hier nur die Möglichkeiten der Netzwerk-Kommunikation. Die 3 grundlegenden bekannten Mechanismen sind:

- TLI (Transport Layer Interface) von System V,
- Sockets vom BSD-UNIX,
- und ONC-RPC[1] (Open Network Computing - Remote Procedure Call),

welcher eine Art Hochsprache im Vergleich zu den anderen darstellt. Zum Einsatz kommen hier die Sockets des BSD-Systems, da diese am weitesten verbreitet und sind und keine Extrainstanzen wie etwa der `portmapper`[2] beim ONC-RPC von Nöten sind. Sockets bieten Kommunikationsendpunkte oberhalb der Transportschicht des OSI-Referenzmodells und gelten heute als Standard auf allen UNIX-Systemen.

4.3 Nutzerinterface

Die Nutzerschnittstelle ist kommandobasiert und interaktiv zu benutzen. Nach dem Start des `Manag` erwartet dieser Eingaben an seinem Prompt. Die Eingaben werden wie in UNIX üblich von `STDIN` gelesen, Meldungen werden auf `STDOUT` ausgegeben, und Fehlermeldungen werden auf `STDERR` ausgegeben, so daß der Benutzung mit Pipes und der Aus- und Eingabeumlenkung nichts im Wege steht. Jedoch ist dies nicht unbedingt empfehlenswert, wenn z.B. das Paßwort für die Benutzung des *rse* abgefragt wird. Nach der Verbindungsaufnahme zu den Servern, zeigt das Prompt den Hostnamen des Rechners, auf welchem der `RW` läuft.

4.3.1 Befehle

Siehe Tabelle 4.7.

Bevor die Server mit den `stop*`-Befehlen beendet werden, muß die Verbindung mit `disconnect` beendet werden.

Die Ausgaben das `Manag` sind größtenteils die der Server, für eine ausführliche Liste dieser siehe Tabelle 4.4 und Tabelle 4.6. Sie haben folgendes Format:

`CODE` *space* `Meldung des Servers`

Die Ausgaben des `Manag` haben keinen vorangestellten `CODE`

Um die Bediensequenz zu vereinfachen, können alle Einstellungen in einem Konfigurationsfile zusammengefaßt werden. Lediglich die einzelnen Aktionen müssen noch "von Hand" angewiesen werden.

[1] Von SUN Microsystems entwickelt worden, hat eine sehr große Verbreitung

[2] Ein Userlevel-Prozeß des ONC-RPC Systems, bei dem neue RPC-Dienste angemeldet werden müssen, ohne ihn geht es nicht.

Tabelle 4.3: Manager - ReaderWriter Protokoll

Befehl	Funktion
`help`	Ausgabe alle möglichen Befehle
`list` *path*	Verzeichnisinhalt von *path* auflisten
`set all`	Anzeige aller definierten Variablen
`set` *variable* = *value*	weist *variable* den *value* zu
`unset` *variable*	löscht *variable*
`stat`	Anzeige von Satusinformationen
`backup`	startet Backup
`restore`	startet Restore
`quit`	schließt die Verbindung zum Client

Tabelle 4.4: Meldungen des ReaderWriter

2xx Informationstexte
200 Command okay.
214 *Help message*
220 *Servicename* Version *nr* on host *host.*
231 Restore ready.
221 Goodbye.
3xx Verbindungszustand zum Dumper
301 Data connection established.
302 Service not available, closing connection.
303 Data connection closed; transfer aborted.
304 Data connection closed normally.
305 There is no data connection to the dump-host.
306 Data connection already opened.
4xx Verbindung zum Manager
401 Service closing control connection.
5xx allg. Fehlermeldungen
500 Syntax error, command unrecognized.
502 Command not implemented.
6xx Konfiguration des Servers
601 Show configuration.
602 Set command succesful.
603 Set command failed.
604 Unset command succesful.
605 Unset command failed.
606 Unset: no such variable.
621 Show state.

Tabelle 4.5: Manager - Dumper Protokoll

Befehl	Funktion
`help`	Ausgabe alle möglichen Befehle
`set all`	Anzeige aller definierten Variablen
`set` *variable* = *value*	weist *variable* den *value* zu
`get` *variable*	liefert den Wert der *variable*
`unset` *variable*	löscht *variable*
`stat`	Anzeige von Satusinformationen
`quit`	schließt die Verbindung zum Client

Tabelle 4.6: Meldungen des Dumper

2xx Informationstexte
200 Command okay. 214 *Help message* 220 *Servicename* Version *nr* on host *host*. 231 Reading from device ready. 221 Goodbye.
4xx Verbindung zum Manager
401 Service closing control connection.
5xx allg. Fehlermeldungen
500 Syntax error, command unrecognized. 502 Command not implemented.
6xx Konfiguration des Servers
601 Show configuration. 602 Set command succesful. 603 Set command failed. 604 Unset command succesful. 605 Unset command failed. 606 Unset: no such variable. 607 Get variable succesful. 608 Get: no such variable. 621 Show state. 631 Set command failed because the backup is running. 641 Writing to device ready. 671 Unknown mode for dumpdevice.

Tabelle 4.7: Manager Befehle

Befehl	Funktion
help|h|?	Ausgabe aller Befehle und deren Funktion
connect|con	mit den eingestellten Servern verbinden
disconnect|dcon	Verbindungen zu den Servern abbauen
readcfg|rc	Konfigurationsfile lesen
showcfg|sc	aktuelle Konfiguration anzeigen
sendcfg|sec	aktuelle Konfiguration zu den Servern transferieren
rset *variable* = *value*	Setzen der *variable* auf *value* und übermitteln zum `RW`
dset *variable* = *value*	Setzen der *variable* auf *value* und übermitteln zum `DMP`
set *variable* = *value*	Setzen der *variable* auf *value* nur lokal
ls *path*	Anzeigen der FS-Informationen von *path* des *RW*
backup|bu	Anweisen der Backup-Aktion
restore|rs	Anweisen der Restore-Aktion
startrw *hostlist*	Starten des `RW` auf den Hosts *hostlist*
startdmp *hostlist*	Starten des `DMP` auf den Hosts *hostlist*
stoprw *hostlist*	Beenden des `RW` auf den Hosts *hostlist*
stopdmp *hostlist*	Beenden des `DMP` auf den Hosts *hostlist*
quit|q|bye|exit|x	Manager beenden (Verbindungen abbauen, aber nicht die Server Beenden)

Kapitel 5

Implementation

5.1 Systemvoraussetzungen

Wie schon in der Spezifikation aufgelistet, werden eine Reihe von Werkzeugen und Bibliotheken zur Realisierung benötigt. Konkret sind dies folgende:

UNIX-Tools

- `/bin/sh`
- `/bin/ls`
- `/bin/hostname`
- `/bin/dd`
- `GNU-tar` Version 1.11.8

`perl5.002`
: Es muß die gesamte Distribution dieser `perl`-Version mit allen Bibliotheken auf den WS lauffähig installiert vorliegen. Genaue Informationen dazu sind unter `http://www.perl.com` und in [10] und [14] zu finden.

rse Version 1.0
: Dieses muß auf allen WS installiert sein. D.h. die Header-Files und die Library werden benötigt, um die *rse*-Funktionalität für den `Manager` zu kompilieren.

C-Compiler
: Ein Standard C-Compiler ist ausreichend. Dieser sollte auf allen Systemen vorhanden sein.

Die Tools können auch an anderen Stellen stehen, wichtig ist, daß die Pfade im Suchpfad des Nutzers stehen, unter welchem die Systemkomponenten ausgeführt werden.

Zu Beachten:
: Beim Aufruf der UNIX-Tools aus `perl` heraus wird `/bin/sh` benutzt. Es ist darauf zu

achten, daß der Sprachumfang dieser von System zu System unterschiedlich ist. So ist `/bin/sh` unter Linux z.B. die `bash` (GNU Born Again Shell) und hat einen wesentlich größeren Sprachumfang als z.B. die Standard `/bin/sh` auf einem Solaris-System.

5.2 Implementation der Server

5.2.1 ReaderWriter

Das Script, welches die Funktionalität des `RW` realisiert, ist komplett in `perl` implementiert. Es existieren zwei Bibliotheken, welche von allen Komponenten benötigt werden. Das Script heißt `readwrit.pl`, die beiden Bibliotheken sind `utils.pl` und `debug.pl`, auf diese wird hier nicht näher eingegangen, da sie nur Routineaufgaben wie z.B. "Socket anlegen" implementieren. Für diese Bibliotheken muß der Suchpfad für `perl`-Bibliotheken erweitert werden, was zur Zeit noch ein Problem darstellt, weil der `rsed` von beliebigen Stellen im FS gestartet werden kann, so daß dieser jedes Verzeichnis als `current working directory (cwd)` haben könnte. Z.Zt. ist dieses Problem so gelöst, daß davon ausgegangen wird, der `rsed` wird in dem Verzeichnis gestartet, unter welchem sich das `etc`-Verzeichnis von ihm befindet. Unter diesem sind dann die Verzeichnisse der *rse*-Nutzer, in denen die Scripte liegen. Es wird also `./etc/$LOGNAME` dem Pfad hinzugefügt. Dafür muß die Umgebungsvariable `$LOGNAME` auf das Login gesetzt sein. Nachteil dieser Variante ist, daß nur Nutzer, unter dessen Login der `rsed` läuft, dies benutzen können. Eine andere Variante ist, das Verzeichnis des *rse*-Nutzers vorher in Erfahrung zu bringen, und dieses in den Code der Servers statisch einzutragen.

`readwrit.pl` realisiert das Lesen/Schreiben des FS. Dies geschieht, indem eine Pipe geöffnet wird, welche gelesen/geschrieben wird. Die Aufrufe, welche die Quell- bzw. Zieldaten erzeugen bzw. auswerten, sehen folgendermaßen aus:

Lesen
: `tar --label` *bandlabel* `--newer` *timestamp* `-zcvpRf - 2>`*logfile* `|`

Schreiben
: `| tar -zxvpRf - 1>`*logfile* `2>&1`

Die Syntax und Semantik der benutzten Optionen sind in der Manual-Page `tar(1)` erklärt.

5.2.2 Dumper

Der `DMP` ist ebenfalls vollständig in `perl` implementiert und benötigt die beiden o.g. Bibliotheken. Der Code befindet sich im Script `dump.pl`. Zur Erweiterung des `perl`-Suchpfades gilt das unter Abschnitt 5.2.1 genannte in gleicher Weise.

`dump.pl` realisiert das Lesen/Schreiben des BLW. Dies geschieht indem eine Pipe geöffnet wird, welche gelesen/geschrieben wird. Die Kommandozeilen, die die Daten von dem bzw. auf das BLW lesen/schreiben, sehen folgendermaßen aus:

Lesen
: `dd if=`*device* `bs=`*blocksize* `|`

Schreiben

`| dd of=`*device* `bs=`*blocksize*

5.2.3 Start der Server

Die Aufrufsyntax ist folgende:

```
readwrit.pl [-d <level> -c <portnumber> -h]
-d <level> debug level
-c <port> portnumber for client
-h shows this page
```

```
dump.pl [-d <level> -p <portnumber> -h]
-d <level> debug level
-c <port> portnumber for command
-h shows this page
```

Die Portnummer für den Client entspricht der Variablen `rwcmdport` im Konfigurationsfile des `Manag`. Der Prozeß geht nicht automatisch in den Hintergrund, wenn dies erwünscht ist, muß dies mit `&` am Ende der Kommandozeile geschehen. Wenn die *rse*-Funktionalität benutzt wird, kann man die Optionen beim Laden der Scripte zum *rse*-Agenten schon angeben. Es ist auch möglich, diese beim Start der Scripte mit zu übergeben. Das System ist auch ohne *rse* lauffähig.

5.3 Implementation des Client

5.3.1 Manager

Der `Manag` wurde wie die Server ebenfalls in `perl` implementiert. Eine Außnahme bildet die *rse*-Komponente, die der `Manag` benutzt. Es existieren für jede Funktion (*load, start, stop, remove*) seperate C-Programme, welche bei Bedarf aufgerufen werden. Sie kommunizieren über die Rückkehr-Codes mit dem rufenden `perl`-Script. Vom `Manag` wird lediglich die Funktionalität des *Startens* und *Stopens* mittels *rse* implementiert. Das Code-Managment, also das *Laden* und *Entfernen* der Scripte beim *rse*-Agenten übernehmen 2 seperate C-Programme. Das `Manag`-Script heißt `manag.pl` und benötigt `utils.pl, debug.pl` und `rsecl.pl`

Die Verbindungen des `Manag` unter anderem auch die mit `STDIN` werden iterativ bearbeitet. Dies wird, wie auch schon in den Server-Komponenten, mittels des Systemrufs `select()` realisiert.

Alle Variablen können hier interaktiv durch Eingaben des Nutzers beeinflußt werden. Diese Arbeit kann durch das Anlegen eines Konfigurationsfiles erleichtert bzw. abgenommen werden.

Konfigurationsfile

Das Konfigurationsfile wird im aktuellen Verzeichnis unter dem Namen `backup.rc` gesucht oder kann als Argument hinter Option `-f` angegeben werden. Die Syntax ist analog den üblichen `rc`-Files in UNIX, also `#` leitet eine Kommentarzeile ein, *variablenname* = *value* ist die Belegung einer Variablen, Leerzeilen werden überlesen. Einziger Unterschied - Kommentare sind nur in seperaten Zeilen erlaubt, also nicht hinter einer gültigen Zuweisung.

```
# Konfigurationsfile fuer Manager des dezentralen Backupsystems
##############################################################
#                                                            #
# MANAGER - Einstellungen                                    #
#                                                            #
##############################################################
logdir = ./log
rserunuser  = tna

##############################################################
#                                                            #
# DUMP - Einstellungen                                       #
#                                                            #
##############################################################
# Hostname oder IP des Rechners, an welchem das Bandlaufwerk
# angeschlossen ist
dumphost    = rac3.informatik.tu-chemnitz.de

# Pfad des Bandlaufwerks
# Hier ist es auch moeglich, ein File anzugeben (/tmp/Backup.tgz)
dumpdevice  = /dev/rst0

# Mode fuer den Dumper
# write - Backup schreiben
# read  - Backup lesen
# wird vom Manager gesetzt
dumpdevmode  = read

# Port, an welchen sich der Manager wenden musz, um mit dem
# DUMP-Server zu kommunizieren
dumpcmdport = 4002

# Port, an welchen sich der RW wenden musz, um Daten
# zu senden/empfangen
# ist jedesmal ein anderer, wird waehrend der Verbindungsaufnahme
```

```
# festgelgt, hier nur der Vollstaendigkeit halber aufgezaehlt
dumpdataport =

# Blockgroesze fuer gnu-tar und dd aufs Band
blocksize = 10240

##############################################################
#                                                            #
# RW - Einstellungen                                         #
#                                                            #
##############################################################
# Hostname oder IP des Rechners, von welchem ein Backup
# gemacht werden soll
rwhost = rac5.informatik.tu-chemnitz.de

# Port, an welchen sich der Manager verbinden musz, um mit dem
# RW-Server zu kommunizieren
rwcmdport = 4001

# Pfad, der gesichert werden soll
# mehrere Pfade moeglich mit <SPACE> separiert
rwtree = /usr/local/lib /usr/sbin

# nur Files, die seit diesem Zeitpunkt veraendert wurden, werden
# im Backup aufgenommen
# Format 'DDMonthYYYY hh:mm:ss <timezone>'
# timezone ist herauszubekommen ueber 'date'
# z.B.    22Jul1996 16:00 MET
rwafter =
```

5.3.2 Start des `Manag`

Die Aufrufsyntax ist folgende:

```
manag.pl [-d <level> -h -f <file>]
-f <file> configuration file
-d <level> debug level
-h shows this page
```

Wenn mit der Debugoption gestartet wird, ist es zu empfehlen, `STDERR` in ein File umzuleiten, da sonst alle regulären und Debugausgaben auf dem Terminal erscheinen, was nicht zur Übersichtlichkeit beiträgt.

5.3.3 *rse*-Clients

Die vom Manag benutzten *rse*-Clienten sind auch als seperate Programme ausführbar.

rseload <rsehost> <rseuser> <interpreter> <version> <description> <script-liste>
: Lädt alle Scripte der *script-list* zum *rse*-Agenten auf dem *rsehost* unter dem Kennzeichen *rseuser*. Die 3 Argumente *interpreter*, *version* und *description* dienen der genaueren Beschreibung des Dienstes, haben aber keine Einfluß auf das Verhalten der Server.

rsestart <rsehost> <rseuser> <execname>
: Startet ein auf *rsehost* vorhandenes Script *execname* als *rse*-Nutzer *rseuser*.

rsestop <rsehost> <rseuser> <execname> <instance>
: Stopt das Script *execname* auf dem *rsehost* und authentifizeirt sich als *rseuser*. *instance* gibt die Instanz des entfernt gestarteten Scriptes an, weil mehrere gleiche Scripte zur Ausführung kommen können ist diese Identifizierung notwendig. Die Instanznummer gibt rsestart nach erfolgreichem Start als positive Ganzzahl zurück.

rseremove <rsehost> <rseuser> <script-list>
: Authentifiziert sich als *rseuser* und entfernt die Scripte der *script-list* aus der Script-Datenbank des *rse*-Agenten auf *rsehost*.

Literaturverzeichnis

[1] Bloomer, John: Power Programming with RPC, O'Reilley & Associates, Inc., 1992

[2] Gläß, André: Dezentrales Systemmanagment in UNIX-Netzen auf Basis rekonfigurierbarer Protokolle, Diplomarbeit, Lehrstuhl Betriebssystme, TU Chemnitz-Zwickau, 1996.

[3] Gläß, A.; Graupner, S.; Naumann, T.: RSE - eine Umgebung für entfernte Script-Ausführung, Beitrag zur SIWORK'96 - Fachtagung, Universität Zürich

[4] Graupner, Sven: Struktukturformen von Servern verteilter Anwendungen unter UNIX, Offene Systeme, Heft 3/94, S.208-215

[5] Hübner, U.: Vorlesungsunterlagen *Rechnernetze*

[6] Hübner U.: Vorlesungsunterlagen *Entwurf verteilter Systeme*

[7] Hunt, Craig: TCP/IP Netzwerk Administration, O'Reilley/International Thomson Verlag GmbH & Co KG, 1995

[8] Kalfa, W.: Betriebssysteme, Akademie-Verlag, Berlin, 1988.

[9] Kalfa, W.: Vorlesungsunterlagen *Betriebssysteme*

[10] Schwartz, Randal L.: Einführung in Perl, O'Reilley/International Thomson Verlag GmbH & Co KG, 1995

[11] Tannenbaum, Andrew S.: Moderne Betriebssysteme, Carl Hanser Verlag München Wien, 1994.

[12] Stevens, W. R.: UNIX Network Programming, Prentice Hall, Englewood Cliffs, 1990

[13] Wall, Larry: Perl programming language, <lwall@netlabs.com>

[14] Wall, Larry and Schwartz Randal L.: Programming Perl, O'Reilley & Associates, Inc. 1992

[15] Ylonen, T.: The SSH (Secure Shell) Remote Login Protocol, Internet-Draft, November 1995.