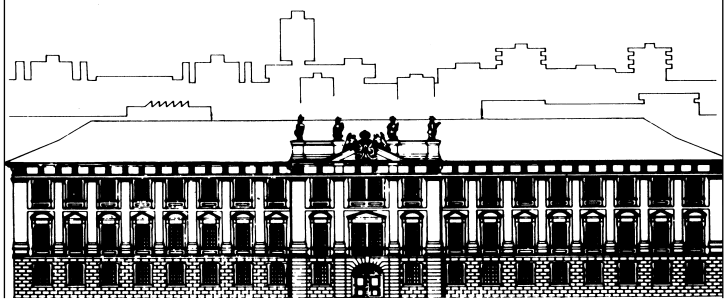
Archivierung von hierarchisch organisierten Dateisystemstrukturen

Frank Kardel

Dissertation

Institut für
Mathematische Maschinen
und Datenverarbeitung
der
Friedrich-Alexander-Universität
Erlangen-Nürnberg



Lehrstuhl für Informatik IV
(Betriebssysteme)

Archivierung von hierarchisch organisierten Dateisystemstrukturen

Der Technischen Fakultät der
Universität Erlangen-Nürnberg

zur Erlangung des Grades

DOKTOR-INGENIEUR

vorgelegt von

Frank Kardel

Erlangen – 1995

Als Dissertation genehmigt von
der Technischen Fakultät der
Universität Erlangen-Nürnberg

Tag der Einreichung: _____

Tag der Promotion: _____

Dekan: _____

Berichterstatter: _____

_____

Kurzfassung

Durch die größeren Massenspeicherkapazitäten verschärfen sich die Probleme der Datenhaltung, -sicherung und -archivierung. Trotz der dramatischen Entwicklung im Software- und Hardwarebereich besteht noch erheblicher Nachholbedarf an adäquater Backup- und Archivierungssoftware. Gründe sind die Trends zu dezentralem Rechnen und die "Offenen Systeme". Die Entwicklung von zentral verwalteten Rechnern mit wenigen Betriebssystemen zu "Offenen Systemen" mit Rechnern verschiedener Architektur und Betriebssystemen muß zur Entwicklung neuer Softwarestrukturen für Datenhaltung, -sicherung und -archivierung führen.

Die zunehmende globale Vernetzung und Diversifizierung von Daten stellt neue Anforderungen an die langfristige Datenspeicherung. Das gilt besonders für den Einsatz neuer Technologien (Multimedia). Architektur-, Anwendungs- und Betriebssystemwechsel müssen berücksichtigt werden. Die herkömmliche Art der Datenspeicherung (einfache Dateien) ist innerhalb dieses Umfeldes nicht mehr zeitgemäß.

Im folgenden Text werden zu den gebräuchlichen Verfahren Lösungsansätze für die oben beschriebenen Probleme gegeben. Diese gehen von einem stark dezentralen, vernetzten System aus. Sie setzen voraus, daß das Lokalitätsprinzip seine Gültigkeit behält; die Vernetzung aber den Zugriff auf weltweit verteilte Betriebsmittel erlaubt. Dieses hat Auswirkungen auf die Skalierbarkeit des Systems zukunftssicherer Datenarchivierung.

Wichtige Ziele sind:

- Langfristige Interpretierbarkeit der Daten.
- Interoperabilität aller Komponenten des Systems.

Da die Entwicklung innerhalb der Datenverarbeitung schwer vorauszusagen ist, können nur Verfahren angegeben werden, die mit den verwendeten und den zu erwartenden zurechtkommen. Dadurch müssen hohe Anforderungen an das Softwaresystem gestellt werden.

Ein heterogenes, verteiltes Archivierungssystem für die langfristige Speicherung von Daten ist erforderlich. Die zentralen Eigenschaften werden in der vorliegenden Arbeit erörtert.

Inhalt

Inhalt

Inhalt

Inhalt

Liste der Abbildungen

Liste der Tabellen

1 Einleitung

Der folgende Text befaßt sich mit dem Problemkreis der Datensicherung und Archivierung in heterogenen, vernetzten Rechnerinstallationen. Das zweite Kapitel wird sich mit Datenhaltungssystemen befassen. Hier werden die besonderen Anforderungen und Stärken ausgesuchter Datenhaltungssysteme dargestellt. Es werden auch die besonderen Anforderungen an die `längerfristige Datenhaltung` beleuchtet. Ein Vorschlag zur Realisierung eines `erweiterbaren Datensicherungssystems` findet sich im dritten Kapitel. Das hier beschriebene `BackStage-System` versucht, mit heutigen Mitteln einige Lösungsvorschläge für die längerfristige Datenhaltung zu machen. Schwerpunkt ist hier das `flexible Archiv`, das in der Lage ist, alle heute gebräuchlichen Dateisystemstrukturen abzubilden. Wegen der zu erwartenden, vielfältigen Datenformate wird im vierten Kapitel eine Dateisystemerweiterung für Unix Dateisysteme beschrieben. Sie erlaubt es, die im `BackStage-Archivkonzept` (drittes Kapitel) beschriebenen Mechanismen auch innerhalb eines Dateisystems zu nutzen. Das fünfte Kapitel befaßt sich mit der zentralen Frage der `langfristigen Archivierung`. Hier wird die Klassifikation der gespeicherten Daten verfeinert und auch im Umfeld der technischen Entwicklung betrachtet. Die sich hieraus ergebenden Ergebnisse werden verwendet, um Aussagen über die `Interpretierbarkeit` und `Wiederherstellbarkeit` von Datenbeständen zu gewinnen. Diese Ergebnisse erlauben es, die `Dateninflation` einzuschränken und die Fortpflanzung von Informationen in Datenbeständen auf Konsistenz zu überwachen.

1.1 Entwicklung von Speicherkapazitäten in Rechnerinstallationen

Innerhalb der letzten 9 Jahre sind die Preise für leistungsfähige Rechner mit großen Externspeicherkapazitäten dramatisch gefallen. Entsprechend ist der Einsatz von Externspeichern gestiegen. Erfahrungswerte zeigen, daß man mit einem Speicherplatzanstieg von 60 - 100% pro Jahr [CM90] rechnen muß. So hatte ein 1986 in der Forschung verwendeter Dateiserver eine Plattenkapazität von etwa 300-600Mb.
1995 veranschlagt man für einen Rechner mit demselben Einsatzgebiet etwa 20 Gb. Die Beobachtungen an dem Lehrstuhl für Betriebssysteme an der Universität Erlangen-Nürnberg bestätigen diese Erfahrungswerte. Mit der immensen Vergrößerung der Externspeicherkapazitäten und den gesteigerten Anforderungen an die Verfügbarkeit und den Aufgabenstellungen von Datenverarbeitungsanlagen hat die Entwicklung der Software für Datensicherung nicht schrittgehalten. Der Bereich Datensicherung ist vielschichtig und umfaßt die Aspekte:

- Backup
  Schutz vor Datenverlust durch Geräte- und Bedienfehler,
- Archivierung
  Dokumentation ausgezeichneter Datensammlungen.

Probleme der Archivierung werden im Zusammenhang mit der rasanten Entwicklung der Software und Speichertechnologien unterschätzt. Ein Problemfeld ist die längerfristige Interpretierbarkeit der Daten.

1.2 Backup, Archivierung und dezentrales Rechnen

Die Aussage, daß die Backup- und Archivierungssoftware nicht mit der Entwicklung der Massenspeichersysteme wie Magnetbandbibliotheken mit automatischem Zugriff, optischen Speichern und magneto-optischen Verfahren [CM90] schrittgehalten hat, ist nicht ganz haltbar. Diese Software für Backup und Archivierung war schon immer vorhanden (besonders im Großrechnerbereich, Rechenzentrumsbetrieb). In dem Maße, wie sich die Verfügbarkeit von Massenspeichersystemen ausgeweitet hat, sind die entsprechenden Werkzeuge jedoch nicht angepaßt worden. So ist es heute nicht ungewöhnlich, daß Personalcomputer (PC) mit mehreren Gigabyte Plattenplatz ausgestattet sind, aber es existieren keine Mechanismen, um diesen Plattenplatz vernünftig zu sichern oder zu archivieren. Mechanismen dieser Art waren und sind auch heute noch nicht weit verbreitet. Neuerdings gibt es die ersten Produkte, die Datensicherung und Archivierung in der dezentralen Welt (besonders im PC-Bereich) anbieten. Für die noch geringe Verbreitung gibt es mehrere Gründe. Ein Grund ist die geringe Nachfrage. Selten werden Sicherungsstrategien angewendet, die dem verwendeten Massenspeicherplatz gerecht werden. Wie wertvoll häufige Sicherungen jedoch sein können, zeigt sich meistens erst, wenn die 9Gb-Platte defekt ist. Verlorengegangene Daten sind oft nicht wiederzubeschaffen, wenn keine adäquate Datensicherung durchgeführt wurde. Es ist mit den immer preiswerter und schneller werdenden Mikroprozessoren möglich geworden, leistungsfähigere Betriebssysteme einzusetzen. Der Trend geht von autonomen 1-Benutzerbetriebssystemen zu Multitasking/Multiuser-Betriebssystemen. Diese unterstützen auch die Vernetzung. Bei Verwendung dieser Systeme bietet sich das Mittel der Vernetzung zu gemeinsamer Nutzung teurer Betriebsmittel, wie großer Platten oder digitaler Bandgeräte mit hoher Aufzeichnungsdichte an. Die Vernetzung bringt noch einen weiteren Vorteil. Meistens sind die Datenbestände auf autonomen Rechnern in einem nicht konsistenten Zustand. Überall liegen Kopien von Daten. Es ist schwer, die aktuelle Kopie der Daten zu finden. Die Vernetzung aber erlaubt hier eine zentrale Datenhaltung und leichtes Transferieren von Daten.
Vernetzte Systeme sind aber (bis auf wenige Ausnahmen) aus heterogenen Komponenten aufgebaut. Diese Heterogenität der Systeme führt zu einem erhöhten Administrationsaufwand. Mit der Anzahl der verschiedenen Betriebssysteme und Maschinenarchitekturen nimmt dieser zu. Ebenso vielfältig wie die Betriebssysteme und Maschinenarchitekturen gestalten sich die Sicherungsverfahren und die zu sichernden Datenbestände.

Die administrative Komplexität begrenzt das sinnvolle Wachstum einer Rechnerinstallation. Man kann davon ausgehen, daß die Grenze bei etwa drei bis fünf verschiedenen Betriebssystemen pro Administrator liegt. Dieser Wert ist allerdings sehr von der An-

zahl der verwendeten Rechner und den Einsatzgebieten abhängig. Je gleichartiger beide Bereiche strukturiert sind, desto einfacher ist es, sie zu unterstützen.
Das M.I.T. mit seinem *Project Athena*[BLP85][CGR90] ist ein gutes Beispiel. Dort werden etwa 5 Personen pro 1000 Workstations benötigt[Tre88]. Erreicht wird dieser gute Wert durch die Verwendung des Unix Betriebssystems mit vielen lokalen Modifikationen (Kerberos [SNS88], Palladium, Moira[RGL88], Zyphyr[Del88]) und einer einheitlichen Verwaltung aller Maschinen. Das Beispiel der Systemadministration am M.I.T. zeigt, daß mit der Einführung einer homogenen – zumindest logischen – Systemumgebung der Administrationsaufwand erheblich reduziert werden kann. Für den Bereich der Datensicherung und -archivierung müssen diese Mechanismen in Zukunft eine einheitliche Struktur haben. In Anbetracht der verschiedenen Systeme und der schnellen Entwicklung im Betriebssystem- und Anwendungsbereich muß die Handhabbarkeit und Sicherung der großen Datenbestände gewährleistet werden.

1.3 Offene Systeme und Standards

Die Welt der offenen Systeme ist nicht ganz so einfach, wie der Begriff vermuten läßt. Dennoch können die ersten Ergebnisse zur Homogenisierung der Systeme durchaus genutzt werden. Es gibt auch viele Ansätze, einheitliche Schnittstellen und Mechanismen in der Welt der offenen Systeme zu schaffen. Als Beispiel seien hier die Bemühungen der Standardisierungsgremien wie zum Beispiel: Distributed computing Environment (DCE) [Osf94], Common Object Request Broker Architecture (CORBA) [Leg93] und POSIX[Bro88] erwähnt. Die letzten Jahre haben gezeigt, daß es relativ lange dauert, bis ein Standard entstanden ist und sich auch behauptet. Auch ist festzustellen, daß einzelne Standardisierungsgremien sich gegen Alternativvorschläge durchzusetzen haben. Es ist auch nicht immer klar, welche der konkurrierenden Gruppen sich durchsetzen wird. Meistens entsteht ein Nebeneinander der Standards, und häufig setzen sich auch Kompromisse, die Bestandteile beider ehemals konkurrierenden Vorschläge beinhalten, durch. Diese Situation ist die eigentlich treibende Kraft der Innovation. Es ist zu erwarten, daß es zumindest in der Datenverarbeitung keine endgültigen Standards und Verfahren geben wird. Viele Probleme können nur mit Heuristiken gelöst werden. Darin sind viele Lösungswege enthalten. Es gibt keine optimale Lösung, wohl aber viele Lösungen mit verschiedenen Schwerpunkten. Es muß also ein Verfahren gefunden werden, das mit den vielfältigen Varianten der zu sichernden Daten zurechtkommt.

1.4 Backup und Archivierung heute

Als Betriebssysteme verwendet man derzeit in der kleinen und mittleren Datenverarbeitung vorwiegend:

– Unix[1] und seine Derivate

- Mach
- OSF/1[1]
- die PC-Betriebssysteme Windows, Windows-NT[2] und Novell Netware[3]

Diese Betriebssysteme sind überwiegend neu oder werden erst seit kurzem im kommerziellen Bereich eingesetzt. Daraus ergeben sich die eingeschränkten Datensicherungswerkzeuge. Viele der Werkzeuge sind genau auf das verwendete Betriebssystem zugeschnitten. Diese bieten häufig für das jeweilige System gute Anwendungseigenschaften. Meistens werden aber alle Eigenschaften der vom Betriebssystem unterstützten Datenhaltungssysteme berücksichtigt. Dieser Vorteil der guten Systemanpassung wird häufig mit Einschränkungen in der Kompatibilität erkauft. Schnelle Systeme verwenden oft Datenstrukturen, die eng an die vom jeweiligen Betriebssystem verwendeten Datenstrukturen angelehnt sind. Die Anpassungen gehen zum Teil so weit, daß sich das Datensicherungssystem von Betriebssystemversion zu Betriebssystemversion unterscheidet. Diese Unterschiede führen dazu, daß ältere Datensicherungen nicht mehr ohne weiteres rekonstruierbar sind.

Neben der Betriebssystemabhängigkeit der verbreiteten Datensicherungssysteme spielen noch die Fähigkeiten dieser Systeme eine Rolle. Nicht alle vom Hersteller angebotenen Datensicherungssysteme sind auch zur ernsthaften Datensicherung einzusetzen. Viele Werkzeuge stammen noch aus der Entwicklungszeit der jeweiligen Betriebssysteme und tragen aus diesen Gründen meist noch Restriktionen in sich, die den Einsatz zur Datensicherung nicht angeraten scheinen lassen.

In der Unix Betriebssystemumgebung lassen sich hier die Einschränkung auf 100 Zeichen für Pfadnamen im *tar*-Format und 16-Bit Inodenummern im *cpio*-Format nennen. Eine Übersicht der Stärken und Schwächen gebräuchlicher Unix Datensicherungswerkzeuge findet sich in LISAV[Zwi91]. Viele der dort beschriebenen Fehler und Unzulänglichkeiten lassen sich auf Beschränkungen im Datenformat, das aus Kompatibilitätsgründen meist beibehalten werden muß, zurückführen. Andere Probleme entstehen durch bestimmte Dateisystemeigenschaften, die nicht über die normale Betriebssystemschnittstelle zur Verfügung stehen. Hierzu gehören die in Unix verwendeten Dateien mit “Löchern”. Bei diesen Dateien werden nur dann Datenbereiche auf der Platte belegt, wenn sie explizit geschrieben sind. Diese Dateien sind hervorragend geeignet für Daten, die nach Hashkriterien (Dateien, die mit Hilfe der *dbm* Bibliotheksfunktion erstellt werden) abgelegt werden. Der Nachteil dieser Dateien liegt darin, daß es über die normale Betriebssystemschnittstelle nicht möglich ist, zu erfahren, welche der Datenblöcke wirklich belegt sind. Um diese Information zu extrahieren, ist man auf direkten Gerätezugriff und entsprechende Kenntnis der Dateisystemstrukturen

1. Unix ist eingetragenes Warenzeichen der Santa Cruz Operation (SCO).
1. OSF ist eingetragenes Warenzeichen der Open Software Foundation.
2. Windows und Windows-NT sind eingetragene Warenzeichen von Microsoft.
3. Novell Netware ist eingetragenes Warenzeichen der Novell Inc.

angewiesen. Es ist kaum zu erwarten, daß sich hier Verbesserungen in der Schnittstellendefinition ergeben. Es ist eher damit zu rechnen, daß dieser Zustand bestehen bleibt. Weiter ist zu vermuten, daß er sich noch in dem Maße verschlechtert, in dem neue Speicherungssysteme für spezielle Anwendungen (Multimedia-Dateisysteme mit Anforderungen an Lesegeschwindigkeit und flexible Erweiterbarkeit) entstehen. Spezielle Sicherungsverfahren wird es hier geben müssen.

1.5 Neue Entwicklungen im Bereich der Datenspeicherung

Um einige der oben beschriebenen Probleme zu lösen, sind von vielen Gruppen Vorschläge für Datenspeicherung, Massendatenspeicherung und Vernetzung von Systemen und ihren Daten gemacht worden. Die Lösungen lassen sich grob in die Themenbereiche gliedern.

- Backup
  Sicherung vor Verlust von Daten durch Geräteausfall,
- Datenspeicherung
  Sicherung ausgezeichneter Dateisystemzustände.

Innerhalb dieser zwei Themenbereiche gibt es mehrere Lösungsansätze, die sich stark im investiven Entwicklungsaufwand unterscheiden.

- Backup
  - Scripten unter Verwendung der mitgelieferten Werkzeuge (tar, cpio, dump, restore, volcopy),
  - Eigene, lokale Lösungen (unter Umständen unter Verwendung frei erhältlicher Software),
  - Erweiterung vorhandener Werkzeuge (meist durch den Hersteller),
  - Software von Drittherstellern zur Datensicherung.
- Datenspeicherung
  - Neue Dateisysteme,
  - Dateisysteme mit Erweiterungen (um Massenspeichersysteme zu unterstützen).

Diese Lösungsansätze (Scripten und eigene Lösungen) versuchen teilweise, das Beste aus der gegebenen Situation zu machen, oder sie vermeiden das Problem ganz (Dateisystemansatz und zentralistische Verwaltung in Form von Dateiservern). Dieses Vorgehen ist für den Bereich der Kurzzeitspeicherung (bis etwa drei bis fünf Jahre) durchaus legitim. Probleme treten jedoch bei Fragen der Heterogenität und der Abbildung der jeweils vom Dateisystem zur Verfügung gestellten Semantiken auf die unterschiedlichen Betriebssystemschnittstellen auf. Mittlerweile beschäftigen sich Drittheresteller von Sicherungssystemen mit diesem Problem. Häufig wird nur der funktio-

nal kleinste gemeinsame Nenner zur Verfügung gestellt (NFS Semantik[SGK85] oder ftp Protokoll). Manchmal wird auch die Verwendung spezieller Clientsoftware verlangt (AFS-Client). Auch das Zurückziehen auf etablierte "Standards" ist hier häufig zu beobachten. Aus Effizienzgründen und wegen beschränkter Entwicklungszeit für die Software wird diese Struktur dann auch in den Datenstrukturen festgelegt. Nicht immer lassen sich diese Strukturen nachträglich an neue Entwicklungen anpassen. Zur Zeit findet ein großer Innovationsschub im Bereich der Multimedia-Anwendungen statt. Es sind Impulse für die Entwicklung im Bereich der Datenspeicherung zu erwarten[Mey91].

2 Datenhaltungssysteme von Rechenanlagen

In diesem Kapitel werden die üblichen Datenhaltungssysteme von Rechenanlagen beleuchtet. Es gibt verschiedene Systeme. Sie sind durch die Anforderungen charakterisiert. Diese reichen von temporären Speicherstrukturen für den Systembetrieb bis hin zu Datenbanken. Für das Themengebiet `Archivierung` werden hier primär die `Dateisysteme` genauer betrachtet. Diese stellen immer noch die am häufigsten anzutreffenden Datenhaltungssysteme dar. Sie werden von dem Betriebssystem bereitgestellt. Ausgehend von den Dateisystemen wird die Sicherungsform des Archivs betrachtet und wie herkömmliche Dateisysteme heute archiviert werden. Aus den unterschiedlichen existierenden Dateisystemen werden die für ein Archiv zu erfüllenden `Anforderungen` definiert. Alle Anforderungen berücksichtigen den Einsatz in einer `heterogenen Umgebung`. Zum Schluß wird die Struktur des *Mass Storage Reference Models*, sowie dessen Nachfolger, das *Reference Model for Open Storage System Interconnection*, beleuchtet.

2.1 Grundsätzliche Konzepte

Neben der Speicherhierarchie in den heutigen Rechenanlagen (Register, Cache, Hauptspeicher, Externspeicher) kann man weitere Speicherungsformen unterscheiden. Hierzu gehören:

- Backing-Store für virtuellen Speicher,
- Dateisysteme,
- Datenbanken,
- Object-Store,
- Langzeitspeicherung.

2.1.1 Backing-Store

Der Backing-Store wird nur für den unmittelbaren Betrieb (Programmablauf / Seitenauslagerungsstrategien) benötigt und ist ungültig nach Systemzusammenbruch.

2.1.2 Dateisystem

Das Dateisystem ist die heute verbreiteste Form der Datenhaltung. Sie wird normalerweise vom Betriebssystem bereitgestellt, und sie wird für die Dienstleistungen des Betriebssystems (Programmstart und Ablauf) genutzt. Die Ausnahmen sind die neueren Betriebssystemformen, die sich in einen Microkernel und weitere Systemkomponenten gliedern. Das Dateisystem gehört auch zu den ausgelagerten Komponenten.

2.1.3 Datenbanken

Die Verwendung von Datenbanken geht mit Anwendungen einher, die auf eine sichere gemeinsame Datenbasis in koordinierter Weise (Transaktionskonzept) zugreifen. Hauptziele der Datenbanken sind die sichere Speicherung, der koordinierte Zugriff, die vielfältige Abfrage (Views) und die Verknüpfungsmöglichkeiten von Datenbeständen.

2.1.4 Object-Store

Die Object-Stores sind eine neue Entwicklung aus den objektorientierten Betriebssystemen [Kle92], die von den herkömmlichen Betriebssystemstrukturen abweichen. Sie bieten als einziges, aber mächtiges Strukturierungsmittel, die Objekte an.

2.1.5 Langzeitspeicherung

Bei der Langzeitspeicherung werden die Daten der Direktzugriffspeicher meistens auf relativ langsamen, zumeist sequentiellen Datenträgern gespeichert (unter Umständen sogar auf Langzeitspeichern, wie sie durch das CDROM dargestellt werden). Diese Art der Speicherung hat drei Zielsetzungen:

- Schutz vor Datenverlust bei Ausfall der Direktzugriffsspeicher (Backup),
- Dokumentation (Kopie eines konsistenten Standes),
- Datenübermittlung (Verbreitungsmedium).

Durch folgende drei Verwendungsmöglichkeiten sind unterschiedliche Forderungen an die Form der Datenspeicherung auf den externen Speichermedien zu stellen. Diese Datenspeicherungformen sind Backup, Datenübermittlung und Archivierung.

Backup
: Große Datenmengen müssen schnell auf Externspeichermedien transferiert werde. Da auch hier auf konsistente Dateisystemzustände geachtet werden muß, führt eine Backupoperation in einem Rechnersystem fast unweigerlich zu einer Betriebseinschränkung, wenn nicht gar zu einer Betriebsunterbrechung. Nur wenige Implementierungen erlauben ein Backup während des Betriebs [Kol91][Shu91][Enq91][Shu91a]. Eine Ausnahme bilden hier die Dateisysteme, die mit Massenspeicherunterstützung arbeiten und eine nicht verwendete Datei innerhalb des Staging Prozesses auf Externspeicher transferieren. Dennoch gibt es auch für diese Systeme Backupmechanismen. Beim Backup wird die Gesamtmenge des Dateisystembestandes als verlustgefährdet betrachtet.

Datenübermittlung
: Daten müssen kompakt in einem Standardformat (*tar, cpio*) gespeichert werden, damit sie vom beim Empfänger wieder extrahiert werden können.

Archivierung
: An die Archivierung werden noch schärfere Anforderungen gestellt als an die Datenübermittlung. Während man bei der Datenübermittlung von einigermaßen ähnlichen Systemumgebungen ausgehen kann, ist dieses bei der langfristigen Speicherung nicht mehr möglich. Der Zugriff auf die Daten kann unter Umständen mehrere Jahre nach der Abspeicherung erfolgen. Um nach so langen Zeitspannen die Daten noch verwenden zu können, müssen besondere Vorkehrungen getroffen werden. Um dem magnetischen Verfall vorzubeugen oder um neue Speichertechnologien nutzen zu können, müssen die Datenbestände von Zeit zu Zeit umkopiert werden.

Die unterschiedlichen Verwendungsarten der auf externen, auswechselbaren Datenträgern gespeicherten Daten bringen neue Probleme mit sich:

- Die Lebensdauer der Daten übersteigt meistens die Lebensdauer der Datenträger.
- Die Datenträgertechnologie überholt sich.
- Die Geräte zum Zugriff auf die Datenträger sind nicht mehr verfügbar.
- Die Lebensdauer der Daten übersteigt die Lebensdauer
  - der Anwendung (ca. 2 - 5 Jahre),
  - der Architektur (ca. 1 - 2 Jahre),
  - des Betriebssystems (ca. 10 - 20 Jahre).

Hier sind umfangreiche Pflegemaßnahmen für die langfristig zu speichernden Daten nötig. Einmal archivierte Daten sollten über längere Zeit interpretierbar sein. Um dieses zu erreichen, wird ein adäquates Modell benötigt. Es sollte möglichst lange die zu erwartenden Datenmengen ohne gravierende Änderungen verwalten können. Gerade hier fehlt es heute noch bei den Massenexternspeicherverwaltungssystemen. Dieses trifft besonders für dateisystemorientierte Systeme zu.

Es ist zu beachten, daß bei der Speicherung von Daten über längere Zeit sich die Umgebung, unter der die Daten erzeugt wurden, im Laufe der Zeit verändert. Diese Veränderung ist umso problematischer, je komplexer die Daten strukturiert sind. Zu der Umgebung gehören:

- die Maschinenarchitektur (Prozessor),
- das Betriebssystem (einschließlich Werkzeuge und Bibliotheken)
- die Anwendung (alle Versionen des datenverarbeitenden Programms).

Die Annahme, daß diese Umgebungseinflüsse durch die Verwendung von Standardformaten minimiert oder beseitigt werden können, trifft nur teilweise zu.

Gegen allgemeingültige Datenformate sprechen:

- Jeder Standard hat ein bestimmtes Ziel und damit eine eingeschränkte Gültigkeit.
- Neue Entwicklungen fordern die Erweiterung oder neue, mächtige Standards.

Jeder veränderte Standard führt zu einem neu zu unterstützenden Datenformat [Mey91]. Als Beispiel mögen hier die Bemühungen und Diskussionen um einen internationalen Zeichensatz dienen (ASCII[ISO91], ISO8859-x[ISO87], Unicode[ISO93]).

Die Inflation der Standards verschärft sich durch das langfristige Speichern von Daten. Die gespeicherten Daten werden nicht automatisch aktualisiert.

Diese Problematik zeigt sich:

Bei dem derzeitigen Datenbestand ist das Datenformat nicht immer bekannt. Es ist nicht dokumentiert, oder die Dokumentation ist nicht verfügbar (herstellerspezifische Datenformate). In diesem Falle sind die Daten nur im Zusammenhang mit den dazugehörigen Programmen nutzbar. Diese Programme unterliegen meistens jedoch selbst Veränderungen.

Nicht immer ist ein Programm in der Lage, alle vorherigen Versionen der Daten zu lesen. Das gilt besonders dann, wenn diese Versionen mehr als zwei Jahre alt sind. Hersteller versuchen, dieses Problem mit Konvertierungsprogrammen zu lösen. Diese sind auch der Ansatz, der für den aktuellen Datenbestand durchzuführen ist. Hierbei kann es besonders dann zu Problemen kommen, wenn Installationen mit verschiedenen Versionsständen kooperieren müssen. Deutlich wird dieses bei “shared libraries”. Programme, die auf Installationen mit neuerem Versionstand generiert wurden, laufen unter Umständen nicht auf Installationen mit älterem Versionsstand.
Leider erfassen die Konvertierungsprogramme ganz selten Daten, die nicht zu dem im unmittelbaren Zugriff befindlichen Datenbestand gehören. Damit werden im ungünstigsten Falle automatisch alle Daten ungültig, die ein älteres, nicht konvertiertes, Datenformat haben.

Die Größe des Verlustes, der sich durch diese Nichtinterpretierbarkeit ergibt, ist von den Daten abhängig. Wird dennoch eine Interpretierbarkeit dieser Daten gewünscht, so ist dieses nur durch nachträglichen Aufwand möglich. Für Dokumente mit gesetzlichen Anforderungen an die Lagerung ist dieser Aufwand sehr hoch. Minimal ist er für abgeleitete Daten aus anderen noch vorhandenen Dateien. Dieses ist bei Objektdateien, die wieder aus Quelldateien erstellt werden können, der Fall. Deshalb haben Objektdateien, da sie systemabhängig sind, einen geringen Archivierungswert, wenn die Quelldateien zur Verfügung stehen. Diese Aussage ist allerdings nur solange haltbar, wie es möglich ist, die Objekt- und Programmdateien aus den Quelldateien zu generieren.

Selten sind Quelldateien vollständig umgebungsunabhängig. Häufig sind Softwaresysteme von anderen abhängig. Es kann einen großen Aufwand bedeuten, solche Abhängigkeiten durch Portierung anzupassen. Dieser Aufwand kann so hoch werden, daß es sich nicht lohnt, ihn durchzuführen [Mul89]. Dennoch haben Quellen einen höheren Archivierungswert, selbst wenn sie nicht mehr übersetzbar sind. Sie dokumentieren die Algorithmen für den menschlichen Leser. Binärformate haben diese Eigenschaften meist nicht.

Die "Wertehierarchie" der Interpretierbarkeit von Daten findet sich nicht nur im Bereich der Softwareentwicklung. Sie läßt sich für fast alle Arten von Daten angeben und ist eng mit der jeweiligen Abstraktionsebene der Daten verknüpft. Je weniger Annahmen über die Umgebung gemacht werden, desto abstrakter sind die Daten, und sie können leichter und damit länger aufbewahrt werden.

Da es im allgemeinen nicht möglich ist, die Interpretierbarkeit von Daten automatisch zu bestimmen, sollten im Falle der längerfristigen "Offline"-Speicherung Mechanismen gefunden werden, die es ermöglichen, entsprechende Aussagen über die in den Dateien gespeicherten Daten zu machen.

Diese Angaben gehen üblicherweise über die in den von den Dateisystemen gespeicherten Angaben hinaus. Sie sind für den unmittelbaren Dateizugriff irrelevant. Sie kommen erst bei Umgebungsveränderungen zum Tragen. Erste Ansätze für die Abspeicherung von Zusatzinformationen finden sich in dem von Apple entwickelten Betriebssystem MacOS (Bindung zwischen Applikationen und Daten). Systeme, die eine dem Apple-Finder ähnliche Oberfläche anbieten (Windows, SunOS Binder), greifen auch zu diesem Mittel. Von diesen Systemen ist vermutlich das MacOS Resource / Data Konzept am weitesten fortgeschritten.

2.2 Herkömmliche Dateisysteme

Zu den normalen Dienstleistungen verbreiteter Betriebssysteme gehören herkömmliche Dateisysteme. Sie ermöglichen dem Betriebssystem den Anwendungsbetrieb aufzunehmen. Hauptsächlich werden in den Dateisystemen die Konfigurationsdaten und Dienstprogramme gespeichert. Zusätzlich werden noch die Anwendungsprogramme und Anwendungsdaten hinterlegt.

2.2.1 UNIX-Dateisysteme

Unix-Dateisysteme sind vom Namensraum her hierarchisch organisiert. Unterschiedliche Geräte werden über die *mount*-Operation in einen einheitlichen Namensraum eingebunden. Haupteigenschaft des Unixdateisystems ist die Integration von Geräten und Dateien unterschiedlichen Typs innerhalb des Namensraums des Dateisystems. In einem Unix Dateisystem werden im allgemeinen folgende Dateitypen unterstützt:

- ungepufferte Gerätedatei (*character special* Gerätetreiber),
- gepufferte Gerätedatei (*block special* Gerätetreiber),
- reguläre Datei,
- Verzeichnis,
- symbolischer Verweis,
- FIFO Datei.

Diese Liste von Dateitypen wird von einigen Herstellern noch um besondere Dateitypen erweitert (symbolische Verweise, deren Inhalt von zusätzlichen Parametern abhängt).

Aufgrund der einheitlichen Abstraktion der Dateisystemobjekte durch den Namensraum ist auch eine einheitliche Attributmenge in Form der *stat*-Struktur vorhanden. Diese Attribute umfassen:

- diverse Zeiten
  (Zugriff, Modifikation, Attributänderung),
- den höchsten Dateioffset,
- die Anzahl belegter Blöcke,
- die Gerätekennung,
- die Zugriffsrechte,
- den Datei-Typ.

Die *stat*-Struktur ist statisch. Keine Erweiterung ist hier möglich. Das Unixdateisystem stellt zur Datenspeicherung nur uninterpretierte Bytefolgen zur Verfügung[Bac86]. Alle höheren Dateiabstraktionen (ISAM, Record, B-tree,...) müssen durch die Anwendung, beziehungsweise Bibliotheksfunktionen, realisiert werden. Diese geringe Abstraktion erlaubt vielfältige Datenformate. Sie können unter Umständen nur durch die entsprechende Anwendung interpretiert werden.

Um dennoch mit den meisten in der Praxis vorkommenden Dateitypen zurechtzukommen, hat sich die Konvention durchgesetzt, am Anfang einer Datei eine charakteristische Kennung zu hinterlegen. Berücksichtigt man diese Praxis, so zeigt sich hier doch eine implizite Typisierung der Dateien.

2.2.2 DOS-Dateisystem

Das DOS-Dateisystem gehört zu den einfachsten Dateisystemen. Als Abstraktionen werden Dateien auf einem Datenträger (durch den Gerätenamen identifiziert) angeboten. Der Namensraum ist hierarchisch organisiert, Namen umfassen 8 Zeichen für den Namen und 3 Zeichen für eine eventuelle Erweiterung des Namens.

Das DOS Dateisystem unterscheidet zwei Namenstypen:

- Gerätenamen
  CON:, LST:,..., A:...P:
- Dateinamen
  bestehend aus dem Paar: Gerät und Dateiname.

Namenskomponenten werden gewöhnlicherweise durch "\" getrennt. Es gibt eine feste Anzahl von Dateiattributen (Zeit, Größe, Schreibschutz,...). Außer dem Schreibschutz gibt es keine weitere Rechtekontrolle (spezielle Netzwerkimplementierungen ausgenommen).

2.2.3 NOS-Dateisystem

Das NOS-Dateisystem leitet sich aus dem Multics-Projekt ab. Das Dateisystem besteht aus Verzeichnis und Datenobjekten. Alle Dateisystemobjekte besitzen eine umfangreiche Liste von Attributen, die den Dateiinhalt, ihre Verwendung, die Zugriffsrechte und weitere Aspekte zur Verwaltung der Datei spezifizieren. Die Zahl der definierten Attribute liegt bei etwa 200.

2.2.4 MacOS-Dateisystem

Die neueren MacOS-Dateisysteme[App86] sind hierarchisch, gerätegebunden organisiert. Bemerkenswert ist die Struktur der Dateien. Jede Datei ist zweigeteilt. Sie besteht aus einem *Resourcenanteil* und einem *Datenanteil*. Zusätzlich zu dieser Strukturierung der Datei gibt es noch eine Menge fester Attribute (Größe, Zeiten, Flags), die die Datei als Dateisystemobjekt charakterisieren. Diese Aufteilung der Datei in zwei Bereiche erlaubt einerseits die Speicherung von Daten wie in gewöhnlichen Dateisystemen und andererseits die Speicherung von Resourcen, die meistens die für den Programmablauf wichtige Daten (Programmsegmente) und Parameter (z. B. Textübersetzungen) enthalten. Das Laufzeitsystem definiert mit Hilfe der Resourcen sogar eine Vererbungshierarchie über die Suchreihenfolge. Das MacOS Dateisystem erlaubt hiermit eine vielfältige Speicherung von zusätzlichen Informationen innerhalb der Resourcen.

2.2.5 AFS-Dateisystem

Das AFS-Dateisystem[Mor86] gehört zu den Netzwerkdateisystemen. Es zeichnet sich durch folgende Eigenschaften aus:

- hierarchischer Namensraum,
- unixartige Attribute,
- Erweiterung um Zugriffskontrollisten für Verzeichnisse.

Primäreinsatzgebiet von AFS ist die effiziente Bereitstellung von Dateien im Netzwerk mit Hilfe geeigneter Caching-Strategien.

2.2.6 VMS-Dateisystem

VMS hat ein hierarchisches Dateisystem mit geräteorientierter Namensgebung. VMS besitzt, wie auch viele andere frühe Betriebssysteme, eine Vielzahl von Dateitypen zur Unterstützung verbreiteter Dateiorganisationen (ISAM, fixed/variable RECORD).

2.3 Archive

Nach den Betrachtungen über Dateissysteme und deren Vielfältigkeit sollen nun Archive untersucht werden. Hier geht es nicht um sorgfältig von Menschen betreute Bandarchive. Die Abbildung von Dateisystemstrukturen in längerfristigen Speicherungsstrukturen soll im Vordergrund stehen.

Bei Archiven werden hauptsächlich dokumentarische Zwecke verfolgt. Die Speicherung von Daten in Archiven soll in strukturierter Form stattfinden. Meistens handelt es sich um einen logisch konsistenten Dateisystemzustand. Dieser Dateisystemzustand stellt eine Art "offizielle Version" dar. Eine *tar*-Datei von einem Dateisystemteilbaum ist in diesem Sinne ein Archiv, auch wenn es sich um eine sehr rudimentäre Form handelt. An Archive für längerfristige Dokumentation sind erhöhte Anforderungen zu stellen. Die Kriterien sind:

- Erweiterbarkeit,
- Betriebssystemunabhängigkeit,
- Dateisystemunabhängigkeit.

Unter Berücksichtigung dieser Kriterien können heute verbreitete Werkzeuge (*tar, cpio*) nicht bestehen, da sie weitgehend Unix-spezifisch sind und unter Umständen nicht alle Aspekte eines Unixdateisystems sichern können (partiell allozierte Dateien, Gerätedateien, Zugriffskontrollisten, kontextabhängige Dateien, etc.). Die Dateisystemsicherung mag zwar nicht vorrangig Ziel eines Archivs sein, ist aber mit Sicherheit ein Kriterium für die Leistungsfähigkeit der Archivstruktur, sich an neue Datenformate anzupassen.

Zwei Systeme werden für die Archivierung eingesetzt:

- Dokumentationssysteme
  meistens optische Speichermedien, Scannertechnologie,
  Indizierung von Dokumenten,
- Dateisysteme
  vornehmlich von Dateiservern mit Migrations- und Stagingverfahren.

Hauptziel der Dokumentationssysteme ist das Erfassen und Indizieren von existierenden Dokumenten in elektronischer Form (Scannen) zur automatischen Suche und Reproduktion.

Die Zielrichtung bei den Dateisystemen liegt in der Bereitstellung des unbegrenzten Dateisystems. Dieses kommt der "Jäger und Sammler"- Mentalität der meisten Benutzer entgegen. Bemerkenswert ist jedoch, daß sich die Meinung gebildet hat, circa 80% der gespeicherten Daten sind entbehrlich. Das Problem ist, daß es nicht mit vertretbarem Aufwand möglich ist, die noch wertvollen 20% der nutzbaren Daten zu extrahieren. Man steht hier vor einer ähnlichen Schwierigkeit wie bei den Seitenauslagerungsstrategien. Es ist nicht bekannt, welche Daten in der Zukunft referenziert werden. Um auch nur einigermaßen mit diesem Problem umgehen zu können, werden weitere Zusatzinformationen benötigt. Kaum eines der heute verbreiteten dateisystemorientierten Systeme für langfristige Speicherung enthält Vorkehrungen, um solche Informationen zu verarbeiten.

2.4 Massenspeichersysteme zur Archivierung

2.4.1 Problemumgebung

An ein Archivierungssystem für dateisystemartig organisierte Daten werden vielfältige Anforderungen gestellt. Die härtesten erwachsen aus den beiden Teilbereichen:

- Langzeitspeicherung
- Betriebssystemunabhängigkeit.

Ein Archiv muß verschiedene Dateisysteme unterstützen. Es sollte die Möglichkeit besitzen, zusätzliche Informationen (Kommentare, Gültigkeitsinformationen, erweiterte Zugriffsrechte und Indizierung) zu spezifizieren.

Die Unterstützung von verschiedenen Dateisystemen bedingt Erweiterbarkeit und Betriebssystemunabhängigkeit. Bei der stetigen Entwicklung der Software (zum Beispiel die DOS Versionen, Unix Versionen, Unix Nachfolger und DOS Nachfolger) müssen archivierte Daten auch nach Systemwechsel noch zugreifbar sein, auch wenn sie unter Umständen nur noch mit Aufwand interpretierbar sind.

Längerfristige Verfügbarkeit der Daten bedeutet jedoch nicht, daß die Archivierungssoftware statisch ist und eventuell auf ein bestehendes Betriebssystem zugeschnitten ist, wie es bei Dateisystemen der Fall ist. Die flexible, offene Datenstruktur muß im Vordergrund stehen. Die Realisierung eines Archivsystems mit der entsprechenden Datenstruktur würde auch dazu führen, daß die Archivierungsfunktionalität auf vielen verschiedenen Plattformen zur Verfügung steht. Ein weiterer Aspekt ist die Vernetzungsfähigkeit. Auch wenn heute noch viele Personalcomputer ohne Vernetzung betrieben werden, so wird doch die Vernetzung, besonders bei zentralen Diensten wie Archivierung, eine immer größere Rolle spielen.

2.4.2 Anforderungen an ein Langzeitarchivierungssystem

Hauptprobleme bei der Realisierung eines Langzeitarchivierungssystems liegen in drei Bereichen:

- Namensraum,
- Dateibegriff,
- Rechteverwaltung.

Bei den Namensräumen haben sich zwei wesentliche Verfahren herauskristallisiert. Das erste Verfahren ist der flache Namensraum. Hier sind alle Namen gleichberechtigt. Der Nachteil hierbei ist, daß Namen nicht mehr frei von einer Anwendung gewählt werden können. Häufig treten Namenskonflikte auf, oder es entstehen sehr lange, unhandliche Namen.

Das zweite Verfahren ist der hierarchische Namensraum. Diese Namensräume vermeiden die Nachteile der flachen Namensräume weitgehend. Hier wird die Eindeutigkeit von Namen nur noch innerhalb einer Ebene gefordert.

Wegen der größeren Strukturierbarkeit hat sich der hierarchische Namensraum bei den Dateisystemen verbreitet. In letzter Zeit haben sich die Anregungen zur Entwicklung alternativer Namensräume vermehrt. Insbesondere Namensräume, die über Suchfunktionen für Attributwerte verfügen, wurden vorgeschlagen [CS92]. Diese neuen Strukturen sind meistens mit hierarchischen Namensräumen gekoppelt und bilden nicht selten eine Erweiterung der kanonischen hierarchischen Namensräume.

Eine weitere interessante Beobachtung ist die Verknüpfung von Namensräumen und Rechten. Im Bereich der hierarchischen Namensräume werden nicht selten Rechte auch mit Verzeichnissen verknüpft, um so das Erzeugen von, oder den Zugriff auf weitere Namen und Objekte kontrollieren zu können. Hier müßte eine striktere Trennung zwischen Rechten für die Namensraummodifikation und dem Objektzugriff angestrebt werden.

Die Rechteverwaltung stellt ein ganz eigenes Themengebiet dar. Rechteverwaltung variiert von überhaupt nicht vorhanden (häufig bei Einzelbenutzerbetriebssystemen) über “discretionary access control” bis zur “mandatory access control”. Die Rechteverwaltung ist in jedem Falle abhängig von den lokalen Gegebenheiten (Policy). Nicht jedes Verfahren ist für jedes Einsatzgebiet geeignet [Mul89]. Dennoch ist es wünschenswert, verschiedene Verfahren zu unterstützen. Dieses erlaubt, heterogene Umgebungen mit einem Archivsystem ohne allzuviel Funktionalitätsverlust zu bewältigen. Spezifische Lösungen werden allerdings in jedem Falle im Sinne der Effizienz besser sein. Das geschieht aber meistens auf Kosten der Interoperabilität.

2.4.3 Dateibegriff

Sogar heutige Dateisysteme weisen bei dem Datei– (Datenobjekt–) begriff unterschiedliche Ausprägungen auf. Diese gehen von den einfachen uninterpretierten Binärdateien (MS/DOS, Unix) über strukturierte Dateien (ISAM, VSAM, Record, Variable-Length-Record – MVS, VMS) bis hin zu zusammengesetzten Dateien (Resource- und Data-fork) in MacOS. Für alle diese Ausprägungen ist ein Modell zu finden, das in der Lage ist, diese Dateiobjekte homogen zu beschreiben. Ebenso vielfältig wie die Dateisysteme selbst sind die in einem Dateisystem zusätzlich zu den Dateien abgespeicherten Attribute.

2.4.4 Betriebsanforderungen

Ein netzwerkweites Archivierungssystem muß natürlich auch die Eigenschaften der Netzwerke berücksichtigen. Man kann bei Netzwerken von bidirektionalen, zuverlässigen Verbindungen ausgehen. Fehlerquellen sind vielfältig:

Das Problem der Netzwerkpartitionierungen bleibt. Man kann Teile des Netzwerks nicht erreichen. Normalerweise sind diese Ausfälle von zeitlich begrenzter Natur und im allgemeinen schnell behoben.

Der Ausfall ganzer Rechner ist eine weitere Fehlerquelle. Der Ausfall ist zunächst nicht von einer Netzpartitionierung zu unterscheiden. Die Ausfallerkennung geschieht normalerweise über eine Mitteilung bei Wiederanlauf des zuvor ausgefallenen Rechners [Nel81].

Weitere Gründe für Störungen können im Betriebsmittelmangel liegen. Häufig sind auch diese Fehler temporär.

Fehler gibt es auch bei den Datenträgern. Sie sind nie 100% fehlerfrei. Insbesondere ergeben sich Datenfehler infolge von Alterungsprozessen.

Genannt werden müssen aber auch die Bedienfehler. Es muß immer damit gerechnet werden, daß falsche Datenträger eingelegt werden oder falsche Einstellungen an Geräten vorgenommen (Schreibschutz) werden.

Daraus folgt, daß das Programmsystem auf obengenannte Fehler vorbereitet sein muß, um eine möglichst präzise Fehlermeldungen abgeben zu können.

Weitere wesentliche Eigenschaften eines Archivsystems sind:

- Stabilität,
  Bei fehlerfreien Komponenten muß ein kontinuierlicher Betrieb gewährleistet sein.
- Überprüfbarkeit,
  Der Zustand des Systems ist jederzeit einsehbar (entsprechende Rechte vorausgesetzt), und Fehler sind erkennbar (keine “lautlose” Verklemmung).

– Fehler-”Toleranz”.
  Das System erwartet das Auftreten von Fehlern und dokumentiert sie, oder schlägt alternative Wege ein (Datenträgerfehler).

2.5 Mass Storage Reference Model des IEEE

Die Version 4 stammt von 14. Juli 1991. Eine neue Version ist in Vorbereitung und steht kurz vor der Verabschiedung. Das Mass Storage Reference Model identifiziert die wesentlichen Komponenten und Abstraktionen für ein dateisystemartiges Massenspeicherverwaltungssystem.

Wesentliche Begriffe des Mass Storage Reference Models sind:

- Bitfile,
  uninterpretierter Bitstrom, der auf einem Massenspeicher geschrieben, gelesen und gelöscht werden kann.
  Bitfiles haben keine Größenbeschränkung.

- Bitfile Server,
  Server, der Erzeugen, Zugriff und Löschen von Bitfiles kontrolliert.
  Der Bitfileserver ist für den Aufbau der Bitfileabstraktion zuständig.

- Storage Server,
  Server, der die Speicherung von einzelnen Datensegmenten übernimmt, aus denen Bitfiles aufgebaut sind.

- Physical Volume Repository,
  Speicherungssubsystem, das automatischen Zugriff auf Massenspeicherdatenträger über Wechselautomaten erlaubt.

Die in dem Mass Storage Reference Model primär identifizierten Systemkomponenten sind:

- Bitfile Client,
  Anbindung der Anwendung an die Komponenten des Mass Storage Reference Models

- Bitfile Server,
  Aufbau und Verwaltung der Bitfile Abstration

- Bitfile Mover,
  Transport von Bitfile Daten

- Name Server,
  Abbildung des von der Anwendung verwendeten Namensraums auf Bitfile Identifikationen

- Storage Server,
  Verwaltung von physischen Datenträgern

- Site Manager.
  Verwaltungsfunktionen

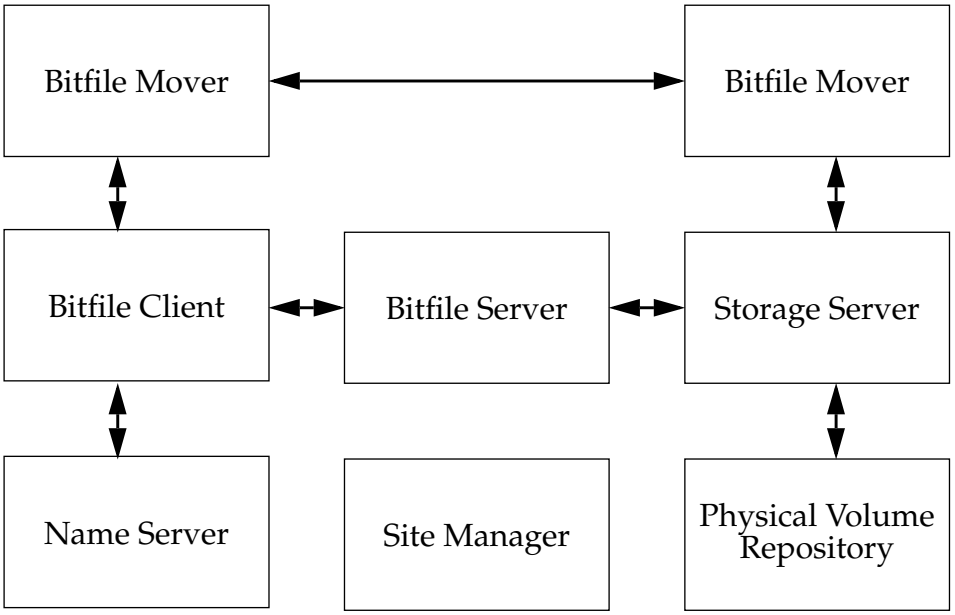


Abb. 2.1 IEEE Mass Storage Reference Model Struktur

Das *Mass Storage Reference Model* benennt drei Abstraktionsebenen.

Die oberste Ebene ist die Ebene des Bitfile Servers. Hier werden Bitfiles erzeugt, geschrieben, gelesen und zerstört. Der Bitfile Server übernimmt den Aufbau dieser Abstraktionsebene. Bitfiles werden über maschinennahe Bitfile-Identifikatoren (BitfileID) angesprochen. Diese BitfileID ist global eindeutig und enthält keinen Hinweis auf den Ort des darüber referenzierten Bitfiles. Ein Bitfile ist ein Bitstring unbegrenzter Länge, der wahlfrei geschrieben und gelesen werden kann. Zusätzlich zu dem Bitstring können noch Attribute (Name, Wert Paare) verwaltet werden. Authentisierung und Autorisierung werden ebenso wie die Abrechnung durch separate Module übernommen. Ein Bitfile Server verwaltet eine Anzahl von Bitfiles. Jedes Bitfile wird durch seine Attribute beschrieben (Zugriffsrechte, Lebensdauer,...).

Die Storage Server bilden die mittlere Abstraktionsebene im MSRM. Die eigentliche Speicherung des Bitfiles geschieht in Segmenten, die einzelnen Storage Servern zugeordnet sind. Diese Segmentliste (bestehend aus Segmentdeskriptor) wird als Attribut des Bitfiles gespeichert. Ein solcher Segmentdeskriptor enthält die Kennung eines logischen Volumes, den Versatz und die Länge eines Segments in einem Storage Server. Der Storage Server übersetzt Segmente von logischen Volumes auf Segmente auf physikalischen Volumes und steuert über die physische Segmentinformation die Geräte an.

Die Geräte bilden die unterste Abstraktionsebene. Bei Vorhandensein eines Systems mit automatischem Datenträgerwechsel (Roboter) wird dieses über ein Physical Volume Repository, der die Buchführung über den momentanen Zustand der Datenträger hält, angesprochen.

Alle Ebenen des Mass Storage Reference Modells sind konsistent nach immer demselben Schema aufgebaut. Es wird eine Abstraktion mit Hilfe von Tabellen aufgebaut, die dann mit Hilfe von schwächeren Abstraktionen realisiert wird. Die einzelnen Bestandteile haben meistens nur buchhalterische Funktionen. Problematisch bei diesem Vorgehen ist, daß auf jeder Abstraktionsebene Verwaltungsinformationen anfallen, die gesichert werden müssen. Sollten die Daten zerstört werden, so erscheinen die Daten der tieferen Abstraktionsebenen wertlos. Auf den untersten Ebenen werden keine Zusatzinformationen gehalten, die eine Rekonstruktion ermöglichen.

Das Modell erlaubt Skalierbarkeit durch Replikation der Dienste und Migration der Daten. Das Problem der Sicherheitsstrategien (Site policy) und des Namensraumes wird delegiert.

Insgesamt erlaubt das *Mass Storage Reference Model* einen Aufbau eines großen Dateispeichers. Es sind bewußt keine Vorschläge für Semantiken gemacht worden, sondern nur die generelle Struktur ist beschrieben. Auch werden keine Strukturierungshilfsmittel (außer vagen Redundanzanforderungen) angegeben.

Betrachtet man das Mass Storage Reference Model aus der Sicht der Dateisysteme, so bietet sich das Bild eines großen Dateisystems mit flachem Namensraum (BitfileIDs), dessen Dateien auf unterschiedlichen Datenträgern gespeichert sein können.

2.6 Open Storage Systems Interconnection

Seit September 1994 existiert eine Nachfolgeversion des Mass Storage Reference Modells das *Reference Model for Open Storage Systems Interconnection* (OSSI) [IEEE94]. Da diese Version noch nicht verabschiedet ist, hat sie nur vorläufigen Charakter. Die wesentlichen Änderungen liegen in der Neustrukturierung der Komponenten und einer Vereinheitlichung der Mechanismen.

So wurde zum Beispiel der Begriff des Bitfiles durch den des Adreßraumes ersetzt. Adreßräume sind aus Segmenten aufgebaut und können geschachtelt werden. Dieses entspricht den Verfahrensweisen, wie sie in neueren Betriebssystemen zur Bildung von Adreßräumen verwendet werden. Ebenso wurde eine vereinheitlichte Darstellung für Datenträger und ihrer Komposition gefunden. Diese erlaubt eine konsistentere Beschreibung der Systemkomponenten und ihrer Interaktion.

Durch die zusätzlich eingeführten Abstraktionen verschärft sich allerdings das Datenkonsistenzproblem zwischen Nutzdaten und Metadaten (Verwaltungs- und Zusatzinformationen). Erfreulicherweise ist eine vermehrte Integration von Metadaten aus allen Abstraktionsebenen des OSSI zu verzeichnen. Diese scheinen allerdings immer noch als zusätzlicher Bestandteil zu gelten. Alles in allem erscheint die neue Version des OSSI eine bessere Basis als das MSRM zu bieten. Sie eignet sich gut als Strukturierungskonzept für Massenspeicherung von Daten mit Migrationsmechanismen. Es muß hier nochmals bemerkt werden, daß nur die Strukturierung beschrieben wird.

Zur Implementation und den Mechanismen werden im OSSI keine Aussagen gemacht. Man geht, wie im MSRM, davon aus, daß zur Realisierung die entsprechende Standardtechnologie verwendet wird. Man hofft, daß die nach dem OSSI entwickelten Systeme sich leicht miteinander kombinieren lassen. Dieses mag innerhalb gewisser Grenzen sogar zutreffen, wenn man sich die Implementationsansätze der heutigen nach dem MSRM entwickelten Systeme betrachtet. Diese nutzen fast zwangsläufig die verbreiteten Mechanismen (gnu-tar, cpio, afio, rsh, TCP/IP, ftp, NFS, AFS, NovellNet).

Weiter ist allerdings zu beobachten, daß die Verwendung von (Quasi-) Standards nicht unbedingt eine Vereinheitlichung im Sinne eines zukunftssicheren Konzepts (Datenformats) darstellt. So können immer noch nicht Daten von PCs (Novell), wenn sie im OSSI System gespeichert werden, von einer Unix Maschine interpretiert werden. Dieses liegt daran, daß auf den PCs sehr spezifische Datenformate (Binärkompatibilität) verwendet werden. Die Interaktion mit anderen Systemen ist noch nicht sehr verbreitet. Gerade diese Interaktion wird sich mit dem Voranschreiten der Entwicklung der Vernetzung verstärken und auch die Forderung nach einfacherer Interaktion zwischen den Systemen verschärfen.

Erste Ansätze hierzu mögen die Standarisierungsbemühungen für Zeichensätze, Dokumentbeschreibungssprachen und maschinenunabhängige Binärformate sein. Die Einführung von zentralen (administrativ gesehen - die Realisation kann und sollte verteilt geschehen) Massenspeichersubsystemen sollte den Datenaustausch zwischen den verschiedenen beteiligten Systemen unterstützen. Diese Forderung wird auch durch den Langlebigkeitsaspekt derartiger Datenbestände unterstützt.
Die mittlere Lebensdauer der datengenerierenden Anwendungen und Systeme ist meistens kürzer als die der Daten selbst. Neben dem Aspekt der reinen Datenorganisation ist auch immer der Aspekt der Datengültigkeit und -interpretierbarkeit zu berücksichtigen. Dieser Aspekt wird nur peripher von dem OSSI gestreift und in die Semantiken der Storage Server verlagert. Dieses ist auch aus Sicht des OSSI eine gültige Vorgehensweise, da hier nur die Mittel und Strukturen zur Haltung von Datenbeständen geschaffen werden sollen.

2.7 Zusammenfassung

In diesem Kapitel wurde auf die verschiedenen, heute verwendeten Speicherungsformen in Rechenanlagen eingegangen. Aus diesen Speicherungsformen wurden die Anforderungen an ein Archivierungssystem extrahiert. Ein solches Archivierungssystem muß folgende Eigenschaften besitzen:

- Unterstützung der wesentlichen Eigenschaften der Datenobjekte in den existierenden Datenhaltungssystemen.
- Offenheit für zukünftige Erweiterungen.
- Unterstützung von vernetzten heterogenen Systemen.

- Unabhängigkeit von aktuellen Betriebssystemen.
- Unabhängigkeit von aktuellen Dateisystemen.

Diese Eigenschaften stellen die Mindestanforderungen an ein zukunftsorientiertes Archivierungssystem dar.

Weiterhin wurden die heute verwendeten Datenhaltungsmethoden diskutiert. Die Vielfalt der existierenden Dateisysteme und die immer mehr voranschreitende Vernetzung stellen besondere Anforderungen an ein Archivierungssystem.

Da die Daten gewöhnlich eine längere Lebensdauer haben als die sie generierenden Softwaresysteme, wird für längerfristige Datenspeicherung ein flexibles Archivierungssystem gefordert. Es muß in der Lage sein, die verschiedensten heutigen und möglichst auch zukünftigen Dateisysteminhalte abbilden zu können. Weiter müssen die längerfristig gespeicherten Daten interpretierbar bleiben. Hinsichtlich der Interpretierbarkeit der Daten gibt es in den Datenbeständen unterschiedliche Qualitäten. Je nach Darstellungsart ist die Interpretierbarkeit mehr oder weniger an die Umgebung (Programme, Rechner, Betriebssysteme) gebunden. Je größer diese Abhängigkeit ist, desto aufwendiger kann eine spätere Interpretation werden. Ob dieser Aufwand gerechtfertigt ist, hängt im wesentlichen vom Nutzen der jeweiligen Daten ab. Fest steht, daß die Informationen, die heutzutage mit den eigentlichen Daten gesichert werden, selten genug Hinweise für eine spätere Interpretation enthalten. Angaben, die es erlauben würden, den Aufbewahrungswert der Daten abzuschätzen, fehlen ganz.

Die betrachteten Standardisierungsvorschläge der IEEE P1244 Arbeitsgruppe stellen einen Fortschritt in der einheitlichen Darstellung von Massenspeichersubsystemen dar. Sie befassen sich allerdings nur mit der eigentlichen Speichertechnologie. Diese Standards können eine gute Basis für ein Speichersubsystem bilden, zumal hier schon alle Besonderheiten eines heterogenen, vernetzten Betriebes berücksichtigt werden.

3 BackStage

Dieses Kapitel beschreibt die `Zielsetzungen` und `Konzepte` des BackStage-Archivierungssystems. BackStage wurde aufgrund der Datensicherungsprobleme innerhalb der heterogenen Rechnerumgebung des Institutes für mathematische Maschien und Datenverarbeitung (IMMD) entwickelt. Wegen der vielfältigen Systeme wurde BackStage konsequent für den robusten, vernetzten Betrieb konzipiert. Nach Definition der Anforderungen wird auf die wesentlichen `Basismechanismen` innerhalb von BackStage eingegangen. Ziel ist es, mit wenigen Basismechanismen leistungsfähige `Abstraktionen` zu schaffen. Mit diesen Abstraktionen wird dann die eigentliche BackStage-Funktionalität verwirklicht. Der erste Komplex beschreibt die `Transaktionsmechanismen`, sowie die Namensraumbildung für die `BackStage-Dienste`. Weitere wesentliche Komponenten sind die Betriebsmittelverwaltung und Authentisierung.

Im nächsten Abschnitt werden die BackStage-Komponenten beschrieben. Der `Volumemanager` bildet hierbei die Basis für die Speicherung von Daten auf externen Datenträgern. Der Volumemanager stellt große Dateien auf externen Datenträgern zur Verfügung. Aufbauend auf dem Volumemanager verwaltet der `Archivmanager` die BackStage Archive. Diese Archive sind so strukturiert, daß die in den vorhergehenden Kapiteln genannten Anforderungen an Archive erfüllt werden können. Zum Schluß wird dargelegt, wie der `Backupmanager` mit Hilfe des Archivmanagers die Durchführung der Backups beaufsichtigt.

3.1 BackStage Entwicklung

BackStage ist ein Archivierungssystem mit Backupfunktionalität. Es wird seit Sommer 1990 an der Friedrich-Alexander-Universität entwickelt. Ausgangslage für die Entwicklung von BackStage war das Fehlen einer Backuplösung für heterogene Systeme. Aufgrund des Personalmangels ist es nicht möglich, Backup für jedes einzelne System mit den für dieses System mitgelieferten Werkzeugen durchzuführen. Ein weiterer Hinderungsgrund ist, daß nicht jedes System über ein Massenspeichersystem (Bandgerät) verfügt. Es wurden deshalb mehrere Eigenentwicklungen auf der Basis von *rsh*, TCP-Verbindungen und *cpio/afio* durchgeführt. Die Erfahrungen bei der Verwendung und Portierung dieser Systeme sind gesammelt worden. Sie dienen als Basis für BackStage. Die Wurzeln dieses universitätseigenen Backupsystems reichen zurück bis zu 1983. Parallel zur Entwicklung des Backupsystems sind die Versionen des Mass Storage Reference Models entstanden. Obwohl beide Systeme große Ähnlichkeit in ihrer Grundstruktur aufweisen, sind diese Strukturierungskonzepte bis 1992 unabhängig voneinander entstanden. Einige der Mass Storage Reference Model Überlegungen sind in die BackStage Entwicklung einflossen.

3.2 BackStage Umgebung

Ursprüngliche Zielsetzung von BackStage war es, Backups in heterogenen Systemen durchführen zu können. Die Strukturierungsüberlegungen haben aber dann den Schwerpunkt in Richtung langfristiger Archivierung von Dateisystembäumen mit Backup als Spezialfall verlegt. Um auch in großen Installationen die Archivierung und Datensicherung durchführen zu können, werden erhöhte Anforderungen an die Automatisierung der BackStage-Systemfunktionen gestellt. Eingriffe durch Menschen sollen in diesem System auf das unbedingt notwendige Minimum beschränkt werden.

3.2.1 Anforderungen

Die allgemeinen Anforderungen, die an BackStage gestellt werden, sind:

- korrekte, flexible Authentisierung,
  netzwerkweit, offen für neue Verfahren
- Verwaltung der Massenspeicher Datenträger (Bänder, Platten),
  Lebensdauerüberwachung
- Unterstützung von Parallelarbeit,
- flexible Auftragsbearbeitung,
- Bediener (Operator) Schnittstelle,
- Konfiguration,
- Wiederherstellung der Datenbasen mit Hilfe der vom System verwendeten Mechanismen,
- Zugriff auch von Benutzern,
- Erweiterbarkeit,

Massenspeicherverwaltungssysteme sind komplex [CM90]. Gründe sind dafür der Wunsch nach Zuverlässigkeit, der gemeinsamen Verwendung von Betriebsmitteln und der Interoperabilität. Um dieser Komplexität Rechnung zu tragen, wurde BackStage unterteilt.

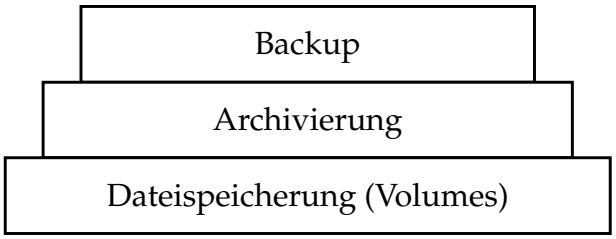


Abb. 3.1 BackStagesystemstruktur

Backup, Archivierung und Dateispeicherung sind also die Hauptabstraktionsebenen von BackStage.

Die erste Ebene bildet der Volumemanager. Der Volumemanager verwaltet die Massenspeicher und ihre Datenträger. Außerdem stellt er die Mechanismen zum Datentransport zwischen den Massenspeichern und den Anwendungen zur Verfügung. Als Abstraktion werden beliebig lange Dateien bereitgestellt.

Dateien werden zu einem Volume zusammengefaßt. Volumes können mit bestimmten Datenträgern assoziiert werden. Auch ist es möglich, ganze Volumes aus dem Volumemanager zu entfernen.Diese können dann zu einem anderen Volumemanager übertragen werden.

Von der Art der verwalteten Einheiten entspricht der Volumemanager dem, was das IEEE Mass Storage Reference Model realisiert.

Eine weitere Aufgabe des Volumemanagers ist es, die Daten, die durch ihn verwaltet werden, verfügbar zu halten. Dieses betrifft insbesondere das notwendige Umkopieren von Daten auf magnetischen Datenträgern, wie Bändern, weil diese nur eine begrenzte Lebensdauer haben.

Die zweite Ebene wird durch den Archivmanager gebildet. Der Archivmanager nutzt den Volumemanager, um seine Aufgaben zu erfüllen. Die Aufgabe des Archivmanagers ist es, baumartig strukturierte Objekthierarchien wie Dateisysteme, unter Berücksichtigung ihrer zeitlichen Entwicklung, darzustellen. Die Abstraktion des Archivmanagers ist also die Darstellung zeitlich veränderlicher Objektmengen in hierarchischen Namensräumen.

Die dritte Ebene bildet der Backupmanager. Er hat zwei Aufgaben. Einerseits sichert er regelmäßig alle von ihm betreuten Dateisysteme innerhalb von Archiven, andererseits verwaltet er die netzweiten Namensräume und ihre Beziehung zu den Archiven.

3.3 BackStage Basismechanismen

In einem verteiltem Programmsystem, wie es BackStage darstellt, ist es notwendig, den Programmablauf nachvollziehbar zu gestalten. Dieses gilt besonders in einer Umgebung, in der Fehler nie ausgeschlossen werden können. In der Literatur werden häufig Transaktionen für derartige Systeme vorgeschlagen. Auch innerhalb von BackStage werden Transaktionen verwendet. Jede längerdauernde komplexe Operation wird in BackStage als Transaktion realisiert. Es sind geschachtelte (nested) Transaktionen erlaubt. Sie dürfen auch Nebenläufigkeit erzeugen. Mit Instantiierung einer Transaktion wird der anfordernden Transaktion die neue Transaktionsidentifikation (TID) mitgeteilt. Diese Transaktionsidentifikation kann dann weiter verwendet werden, um die Transaktion abzubrechen (*abort*), zu untersuchen (*status*) oder darauf zu warten (wait: *commit*, *abort*). Transaktionsidentifikationen haben mehrere Funktionen:

Eine TID

- ist ein Name für einen Dienst,
  Dienste sind lang laufende Transaktionen
- wird verwendet, um Operationen eines Dienstes anzufordern
  status, *abort*,...,
- bestimmt die Lebensdauer eines Dienstes.

Eine TID beschreibt eine Transaktion, die entweder selbst Transaktionsschritte durchführt oder eine Dienstleistung anbietet.
Die erste Art der Transaktion kann man mit üblichen Transaktionen vergleichen, indem die normalen Operationen die eigentliche Aufgabe der Transaktion sind. Zusätzlich wird die *abort()* Operation angeboten.
Die zweite Verwendung der Transaktion entspricht dem althergebrachten "Server", der Anforderungen entgegennimmt und abarbeitet. Bei der Abarbeitung kann es zur Erzeugung neuer Transaktionen kommen.
Der Sinn, auch einen Server als Transaktion längerer Lebensdauer aufzufassen, liegt in der Vereinheitlichung des Interaktionsmodells (Wechsel der Rolle von Client und Server). Demnach stellen Transaktionen sich ähnlich wie Objekte eines objektbasierten Systems dar [Weg90]. Sie umfassen allerdings nicht gleich den objektorientierten Anspruch (Vererbung).

Mit der TID-Abstraktion werden drei Aspekte gleichzeitig verfolgt:

- Transaktionssemantik,
  auch über Knotengrenzen,
- Objektbasierung,
  Strukturierung und Kapselung (zumindest logisch)
- Fernaufrufmechanismus.
  Direkte Ausführung oder neue Transaktion.

Wenn man Transaktion und Objekt koppelt, ergeben sich neue Sichtweisen für die Ausführung.
Es werden nur kurze Kommunikationsphasen zur Auftragserteilung und Ergebnisübergabe benötigt. In der Zwischenzeit dürfen die Kommunikation oder die beteiligten Rechner zusammenbrechen.
Bei Wiederanlauf werden begonnene Transaktionen weiter fortgesetzt. Dieses erlaubt ein Fortschreiten der gesamten Anwendung auch dann, wenn die beteiligten Partner zeitweilig ausfallen. Diese lokal fortsetzbaren, langfristigen Aufträge haben auch einen Vorteil. Nur der lokale Zustand muß rekonstruiert werden (siehe auch [Nel81]). Interaktionen mit anderen Knoten beschränken sich auf Auftragserteilung (*invoke*), Statusabfrage (*await*) und Ergebnistransport (*commit* & *abort*).

Diese Art der Auftragsbearbeitung ist allerdings mit dem Problem der verwaisten Aufrufe behaftet. Eine Transaktion kann nicht terminieren, bevor nicht alle ihre Untertransaktionen terminiert sind. Es kann vorkommen, daß die Kommunikation zwischen den Knoten längerfristig gestört ist, oder ein Knoten ganz außer Betrieb geht (Geräteausfall oder Verkauf). In diesem Falle würden Transaktionen übrigbleiben, die ohne weitere Maßnahmen bestehen bleiben. Um solche Transaktionen auch im verteilten Fall unter Kontrolle zu halten, erhält jede Transaktion eine Lebensdauer. Nach Ablauf der Lebensdauer wird die Transaktion mit *abort()* abgebrochen. Damit sterben verwaiste Transaktionen automatisch ohne (größere) Seiteneffekte aus [Nel81]. Hierbei ist zu bemerken, daß die Transaktionen nur als zuverlässiger Fernaufruf genutzt werden. Im allgemeinen gibt es kein gemeinsames *commit* oder *abort*. Jede Transaktion muß ihre voneinander unabhängigen Untertransaktionen selber überwachen. Werden die Untertransaktionen nicht mehr betreut, so beenden sie sich aufgrund der zeitlichen Lebensdauerbegrenzung selbst. Ist ein gemeinsames *commit* oder *abort* gewünscht, so muß dieses innerhalb der Transaktion programmiert werden. Bis jetzt erscheint es noch nicht notwendig, diese starke Forderung als Basismechanismus zu realisieren. Ein gemeinsames *commit* und *abort* wäre in diesem Falle nur schwer handhabbar, wenn man die korrekte Exterminierung aufrechterhalten möchte[GR93]. Weil die Ergebnisse einer Untertransaktion zuverlässig an die übergeordnete Transaktion übertragen werden müssen, kommt es hier in der Ergebnisübermittlungsphase grundsätzlich zu Exterminierungsproblemen. Solange eine begonnene Ergebnisübermittlung nicht korrekt bestätigt ist, kann die Transaktion nicht mehr automatisch terminiert werden, wenn eine *commit*-Entscheidung dieser Transaktion vorliegt. Diese kritische Phase des Kommunikationsprotokolls sollte so kurz wie möglich gehalten werden.

3.3.1 Transaktionsimplementierung

Wegen der geringen Verfügbarkeit von portierbaren Transaktionsbibliotheken für C/Unix wurde ein eigener Ansatz zur Realisierung der Transaktionssemantik in Verbindung mit dem Fernaufruf gewählt. Um eine weite Verbreitung (heterogene Systeme) zu erreichen, wurde C/Unix als Entwicklungsumgebung gewählt. Als sichere, atomare Systemaufrufe mit dauerhaftem Effekt gelten die Systemaufrufe *creat*(), *link*() und *rename*().

Auf der Basis dieser Systemaufrufe lassen sich Daten stabil speichern. Weiterhin muß gewährleistet werden, daß alle geschriebenen Daten auch auf dem eigentlichen, nicht flüchtigen Datenträger hinterlegt sind. Ein Systemzusammenbruch kann somit toleriert werden. Bei BSD-Systemen bietet sich der *fsync()*-Systemaufruf an. Er gewährleistet, daß alle Daten der Datei auf dem Datenträger hinterlegt sind. Bei System V Varianten der Unix Betriebssysteme bietet sich hier die *open()*-Operation mit der Option *O_SYNC* an.

Als Fehlermodell kommen für die Implementierung Betriebsmittelknappheit, Programmierfehler und Bedienfehler in Betracht.

Bei Betriebsmittelknappheit wird abgebrochen, ebenso bei Rechnerausfall und Bedienfehlern.
Programmierfehler gliedern sich in zwei Klassen:
Die erste Klasse enthält Fehler, bei denen das Programm durch das Betriebssystem abgebrochen wird. Diese Fehler sind wie Rechnerausfall zu werten, können durch Neustart aber zu Schleifen führen. Dieses läßt sich durch höhere Überwachungsinstanzen und eventuelle Blockierung der Transaktion vermeiden.
Die zweite Klasse ist nicht über das Transaktionskonzept zu erfassen. Hier zerstören Programmierfehler die Konsistenz der Daten (im Sinne des ordnungsgemäßen Ablaufs). Eine leistungsfähigere Sprache als C könnte hier helfen; aber das Problem der semantischen Fehler ist nicht zu beseitigen.

Die Transaktionen werden also mit Hilfe von Unix-Prozessen realisiert. Wegen der möglichst gering zu haltenden Anforderungen an die Umgebung (Portierbarkeit) wird auf die Verwendung spezieller Threadpakete zur Zeit verzichtet, da ein Wiederaufsetzen eines Threadkontextes nach Wiederanlauf nicht ohne besonderen Aufwand möglich ist.

3.3.2 Transaktionen

Eine Transaktion besteht aus:

- einer Datenstruktur für stabile Daten,
- einem flüchtigen Datenbereich,
- einer Liste von Aktionen (*C-Funktionsaufruf, invoke, wait*),
- sowie einem Index für die Folgeaktion für den *commit-*, *abort-* und *repeat*-Fall.

Diese Datenstruktur wird automatisch aus einer entsprechenden Transaktionskurzbeschreibung generiert.

Jede Instanz einer Transaktion wird von einem Transaktionsscheduler verwaltet und ausgeführt. Dieser Transaktionsscheduler entspricht nur teilweise dem Transaktionsmonitor in [GR93]. Der Scheduler kümmert sich nur um die Speicherung der stabilen Daten, sowie um die Transaktionszustände und die Wiederanlauffunktion. Er sorgt nur dafür, daß die lokalen Transaktionen voranschreiten können und überwacht ihre Zustände. Es ist nicht Aufgabe des Transaktionsschedulers, ein verteiles *commit* durchzuführen. Dieses muß durch die Transaktion selbst durchgeführt werden. Im Normalfall werden die Unterstransaktionen, die *comitted* sind in der *wait*-Phase so akzeptiert (d. h. die *commit*-Entscheidung wird sofort bestätigt).

Durch dieses 2-Phasen *commit*-Protokoll geraten die Transaktionen nach Übermittlung ihres Ergebnisses in den “in-doubt”-Zustand [GR93], bis die Entscheidung von der übergeordneten Transaktion eintrifft. In dieser Phase sind keine Zeitschranken mehr möglich. Die Transaktion muß auf die Antwort warten. Tritt jetzt eine längerfristige Störung der Kommunikation auf, dann ist das System bis zur Wiederaufnahme der

Kommunikation blockiert. Das bedeutet, daß der automatische Exterminierungsmechanismus nur bis zur Ergebnisübermittlung funktioniert. Treten Störungen in der Ergebnisübermittlungsphase auf, so können die Transaktionen nicht mehr automatisch terminiert werden, da nicht bekannt ist, ob die übergeordnete Transaktion das Ergebnis schon korrekt erhalten hat.

Treten Störungen vor der Ergebnisübermittlungsphase auf, so können die untergeordneten Transaktionen gefahrlos terminiert werden, da eine Lebensdauerüberschreitung als *abort()* interpretiert wird.

Transaktionen können 9 Zustände haben:

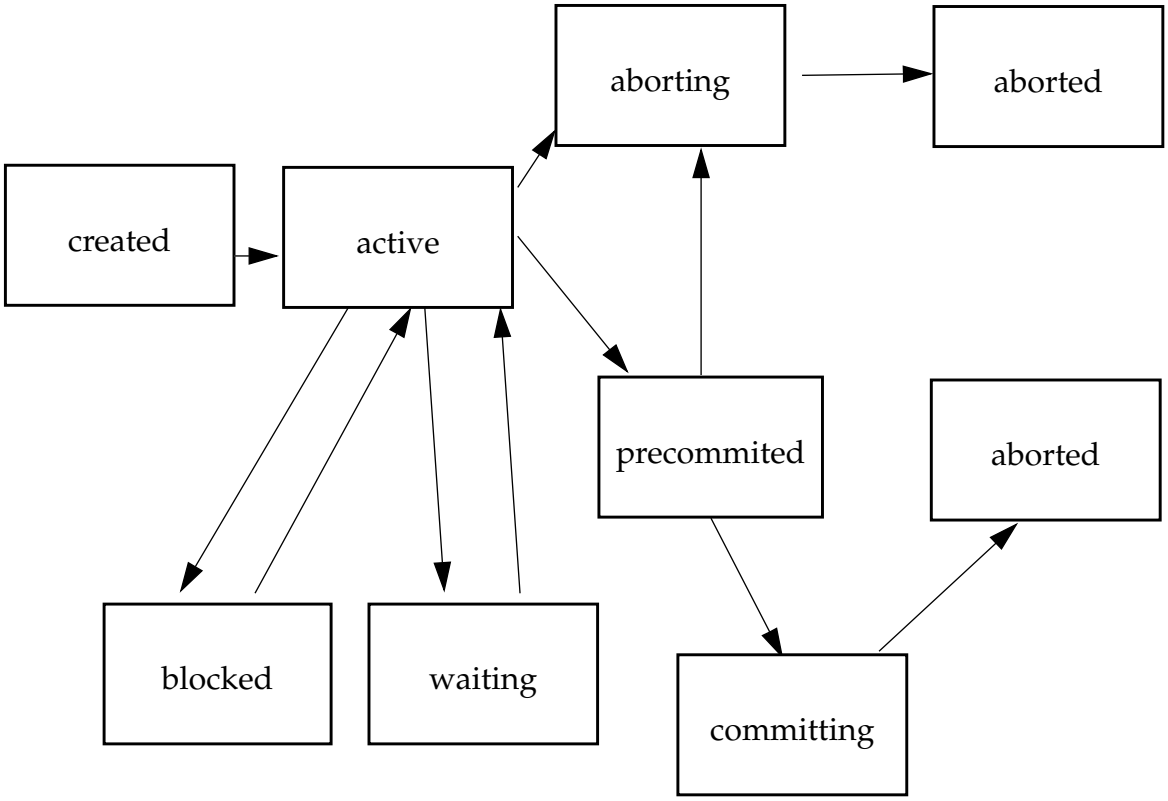


Abb. 3.2 Transaktionszustände

Diese Zustände, wie auch der Transaktionsschrittzähler, werden stabil gespeichert. Bei Wiederanlauf wird nur eine "recovery"-Routine zum Wiederaufbau der flüchtigen Daten durchlaufen. Der letzte Transaktionsschritt wird mit "RESTART" als Argument wieder aufgesetzt. Bis auf die Wiederanlaufeigenschaften entspricht dieses Modell der Transaktion dem "stabilen" Prozessor [Nel81]. Es kann so mit einem Minimum an Rekonstruktionsaufwand die Arbeit fortgesetzt werden. Diese Eigenschaften werden jedoch mit einem Preis bezahlt. Die stabilen Daten einer Transaktion müssen immer zwischen den Transaktionsschritten gesichert werden.

Als Transaktionsschritte stehen zur Verfügung:

- C-Funktionsaufruf,
  Implementierung der Funktionalität
- *invoke*(TID, Parameterblock),
  Starten von weiteren Transaktionen (Ausführung zwischen den Transaktionsschritten)
- *await*(List),
  Blockieren einer Transaktion, bis eine der Untertransaktionen terminiert ist. Hierbei wird das Ergebnis dieser Transaktion übertragen.
- *abort*(List),
  Abbrechen der angegebenen Transaktionen.
- *commit*,
  Entscheidung dieser Transaktion für *commit.*
- *acceptCommit*(List),
  Bestätigen des *commits* einer Untertransaktion.
- *block*(TID, Cause),
  Blockieren der angegeben Transaktion mit dem Grund *Cause.*
- *unblock*(TID, Cause, Code),
  Deblockieren der angebenen Transaktion und Übergabe des Codes.

Koordinierungsprimitive sind nicht vorgesehen, sind aber leicht mit den *block()/unblock()*-Primitiven realisierbar. Zusätzlich ist jedes C-Code Fragment atomar. Die mit den Primitiven *invoke, commit, acceptCommit, await*, *abort*, *block* und *unblock* assoziierten Datenstrukturen werden stabil gespeichert.

Dieses Transaktionsmodell ist nicht so mächtig und komfortabel wie andere Systeme, aber es ist aus derzeitiger Sicht ausreichend. Die Anwendung spezifischer, auf Transaktionen zugeschnittener, Programmiersprachen und Laufzeitumgebungen [Lis87] wäre an dieser Stelle von Vorteil.

3.3.3 Auftragserteilung

Die Phasen Auftragserteilung und Ergebnisüberstellung stellen in sich kleine Transaktionen dar. Die Auftragserteilung ist im Sinne der Exterminierung noch relativ unkritisch, weil diese Transaktion sich durch die Lebensdauerbegrenzung selbst terminieren. Die Exterminierung wird dann problematisch, wenn angefangen wurde ein *commit*-Ergebnis zu übermitteln. Nachdem eine Übermittlungsversuch mit der *commit*-Entscheidung durchgeführt wurde, kann die Untertransaktion nicht mehr bis zum Eintreffen der Bestätigung oder einer von der übergeordneten Transaktion stammenden *abort*-Nachricht automatisch exterminiert werden.

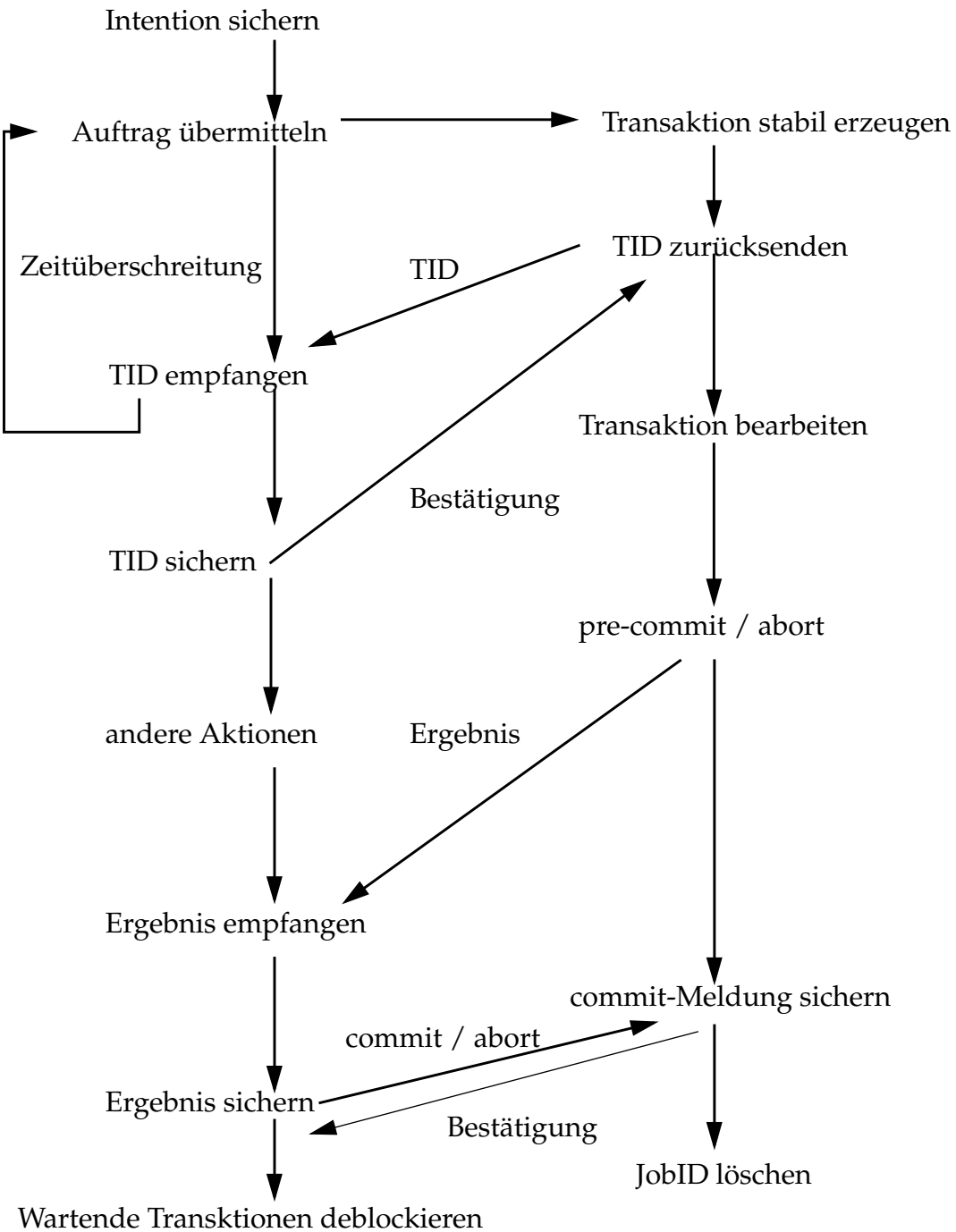


Abb. 3.3 Transaktionsprotokoll (2 Phasen Commit)

Der Aufruf ist asynchron. Ergebnisse werden in der *await*-Phase übertragen. Mögliche *await*() - Ergebnisse sind:

- Anzahl der Kommunikationsversuche überschritten,
- *error* (Authentisierung, Operation, Namenauflösung, Dienst),
- TID,
- *commit*, *abort*(Reason).

Die Möglichkeit, die *commit*-Meldung separat zu senden, erlaubt es auch, traditionelle transaktionsmanagerartige Abläufe zu gestalten. Hier muß jedoch auf das Problem der korrekten Exterminierung geachtet werden.

3.3.4 Verwaltung von Transaktionen

Um die verschiedenen im Netzwerk verteilten Transaktionen erreichen zu können, wurde ein Namensschema entwickelt. Die Namen der Transaktionen sind Bestandteil der Transaktionsidentifikatoren. Die Übersetzung von Namen in aktuelle Adressen und Prozesse geschieht durch den BackStage Portmapper-Daemon (vgl. 3.3.4.2 auf Seite 35). Der Portmapper-Daemon hält einerseits die auf einem Knoten (als Erweiterung auch innerhalb einer administrativen Domain) angebotenen Dienste. Zum anderen speichert er die jeweils gültige Adreßübersetzung. Die Übersetzung kann sich dynamisch ändern, da Prozesse durch Programmfehler oder durch freiwillige Terminierung versterben.

3.3.4.1 Aufbau einer Transaktionsidentifikation

Eine Transaktionsidentifikation hat folgende Struktur:

- Name,
- Gültigkeitsbereich,
- Gültigkeitsdauer,
- Verifikator.

Die beiden wesentlichsten Bestandteile einer Transaktionsidentifikation sind der Dienstname und der Gültigkeitsbereich. Der Dienstname ist innerhalb eines Gültigkeitsbereichs eindeutig. Der Gültigkeitsbereich ist normalerweise auf einen Rechner beschränkt, kann aber auf eine ganze administrativ zusammengehörige Rechnergruppe ausgedehnt werden. In diesem Fall ist der die Transaktionsidentifikatoren übersetzende Portmapper verteilt zu implementieren.
Die Gültigkeitsdauer beschränkt die Lebensdauer einer Transaktionsidentifikation und wird dazu verwendet, das Problem der verwaisten Aufrufe (Exterminierung) zu lösen. Da es bei der Auftragserteilung zu Verzögerungen kommen kann, werden für die Gültigkeitsdauer relative Zeiten angegeben. Nachdem ein Auftrag erteilt wurde, entsteht eine Transaktionsidentifikation mit einer absoluten Zeitangabe. Dieses Vorgehen wird normalerweise in der Datenverarbeitung nicht verwendet. Da hier jedoch ein Problem gelöst werden muß, das durch das Fehlen einer gemeinsamen Kommunikation erst entsteht, bietet sich der physikalische Zeitbegriff an.

Mittlerweile ist es technisch möglich, weltweit auf eine gemeinsame Zeitbasis innerhalb vertretbarer Toleranzen – weniger als eine Sekunde – zugreifen zu können [Mil94]. Durch den Bezug auf ein unabhängiges Synchronisationssystem, wie es die physikalische Zeit darstellt, können die verwaisten Aufrufe nach Ablauf ihrer Lebens-

dauer terminiert werden. Das ist auch ohne eine Rückfrage beim erzeugenden System möglich. Auch wenn keine Kommunikation stattfindet, kann entschieden werden, wann eine Transaktion als terminiert betrachtet werden muß, solange sie nicht Bestandteilt eines gemeinsamen *commits* ist, und auf die Entscheidung der koordinierenden Transaktion wartet. Dieser Zustand sollte im Hinblick auf eine erfolgreiche Exterminierung von verwaisten Aufrufen vermieden werden.

3.3.4.2 Der Portmapper

Dienste werden mit ihrem Namen beim Portmapper registriert. Zusätzlich zum Namen wird noch das realisierende Programm angegeben. Die Rechte werden aus der Authentisierungsinformation abgeleitet. Weitere Informationen geben an, ob dieser Dienst wiederanlauffähig ist oder nicht. Der Portmapper übernimmt hier die Abbildung zwischen Transaktionsidentifikation und dem Betriebssystem, sowie die Übersetzung der Netzwerkadressen. Durch die stabile Speicherung der Dienste erhält der Portmapper auch die längerfristigen Transaktionen. Diese müssen natürlich wiederanlauffähig sein.

Wird eine Transaktionsidentifikation gesucht, so wird der dafür zuständige Portmapper befragt, der im Gültigkeitsbereichsanteil angegeben ist. Bei der Übersetzung einer Transaktionsidentifikation sind 4 Fälle zu unterscheiden:

- Ein Prozeß für die gesuchte Transaktion existiert.
  - Rückgabe der aktuellen Adresse
- Für die gesuchte Transaktionsidentifikation besteht kein Prozeß, aber die Transaktionsidentifikation ist gültig.
  - Prozeß starten
  - Adressauflösung beenden nach Anmeldung des neuen Prozesses
- Der Prozeß ist defekt (Abbruch durch das Betriebssystem).
  - Rückgabe “defekt”
- Die Transaktionsidentifikation ist nicht bekannt.
  - Rückgabe “unbekannt”

3.3.5 Strukturierung des Dienstenamensraumes

Da ein Betriebssystem-Prozeß unter Umständen mehrere Transaktionen und damit Transaktionsidentifikationen beinhalten kann, wurde der Namensraum der Dienste naheliegenderweise hierarchisch organisiert. Jede interne Transaktion bildet eine Unterkomponente des Prozesses. Hieraus folgt, daß bei Nichtauffinden des Transaktions-

namens die Adresse desjenigen Prozesses geliefert wird, die das längste übereinstimmende Präfix des Namens hat. Existiert keine solche, so gibt es die Transaktionsidentifikation und damit auch die Transaktion nicht.

Name	Programm	Status	Adresse
vold	/local/backstage/bin/vold	running	tcp:1613@faui45.informatik.uni-erlangen.de
device-control.exabyte.8500	/local/backstage/bin/dc -t EXABYTE_8500 /dev/nrst10	running	tcp:2046@faui45.informatik.uni-erlangen.de
device-control.dat	/local/backstage/bin/dc -t DAT /dev/nrst2	ready	–

Tab. 3.1 PortmapperTransaktionstabelle

Die Adressauflösung geschieht als Teilfunktion der *sendtotid* Operation. Es ist erlaubt, die Adressen zwischenzuspeichern und die Betriebssystembindung bestehen zu lassen (Kommunikationskanäle). Diese Informationen müssen im Falle eines Verbindungszusammenbruchs neu beschafft werden. Sie sind nicht unbeschränkt gültig.

Eine weitere Funktion des Portmappers ist die Überwachung der Dienstprozesse. Bei Fehlfunktion werden diese Transaktionen vom Portmapper blockiert. Das ist dann notwendig, wenn die diese Transaktionen realisierenden Prozesse zu häufig unerwartet abbrechen.

Der Portmapper stellt somit das Bindeglied zwischen Betriebssystem und Transaktionsumgebung dar.

Die Form der Adresse erinnert an die durch World Wide Web eingeführten *Universal Resource Locators.*Die BackStage Transaktionsidentifikatoren wurden vor den URLs definiert. Beide Namensschemata lösen das Adressierungsproblem dadurch, daß zusätzlich zu den eigentlichen Addressinformationen auch noch die Protokolle angegeben werden, zu denen die Adressen gehören. Die weite Verbreitung des *World Wide Web* zeigt die Einsetzbarkeit dieses Verfahrens.

3.3.6 Parameterübergabe

Da es sich bei den BackStage-Diensten um unvollständige Dienste im Sinne von späteren Erweiterungen handelt, wurde auch ein entsprechend einfaches Parameterübergabeprotokoll verwendet. Die übertragenen Einheiten sind Vektoren von Name/Wertpaaren. Die Namen und Werte sind ASCII-Strings. Dieses entspricht zwar nicht der heutigen Parameter- und Datenrepräsentationstechnologie [ASN1], erlaubt aber einfa-

che Fehlersuche, da die meisten Parameter einfach lesbar sind. Das Name-Wert Verfahren hat den Vorteil der einfachen Erweiterbarkeit. Es verbleiben allerdings ein paar Probleme. Zum einen ist keine Semantik an den Namensraum für die Namen der Name/Wertpaare gebunden. Hier ist man auf Konventionen angewiesen. Dieses ist eine direkte Folge der Erweiterbarkeit. Zum anderen ist es etwas umständlich, höhere Strukturen wie Felder und Verbundtypen zu übertragen. Diese Schwächen können jedoch behoben werden.

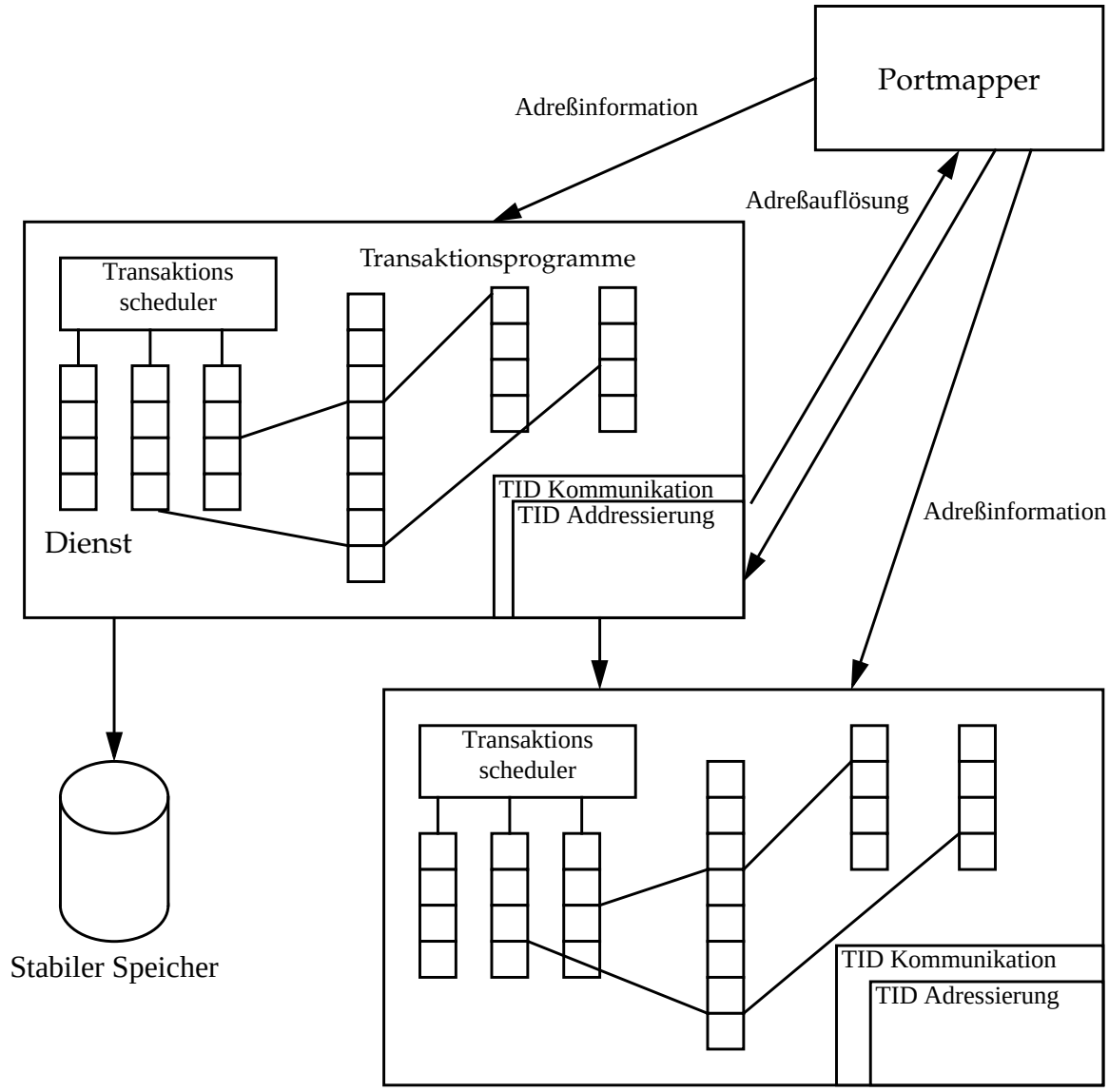


Abb. 3.4 BackStage Transaktionsimplementierung

3.3.7 Betriebsmittelverwaltung

Der oben beschriebene Transaktionsscheduler (vgl. 3.3.2 auf Seite 30) trägt nur die Basismechanismen für den Ablauf einer Transaktion bei. Die Entscheidung, welche Transaktionen wann lauffähig sind, ist von dem Koordinierungscode innerhalb der

Transaktionen abhängig. Viele Entscheidungen über die Lauffähigkeit von Transaktionen sind an die Verfügbarkeit von Betriebsmitteln oder an das Terminieren von anderen (Sub-)Transaktionen gebunden. In dem Falle von BackStage ist die Lauffähigkeit gewöhnlich an die Verfügbarkeit von Datenträgern gebunden. Gewöhnlich gibt es mehr Datenträger als Geräte, die die Datenträger handhaben können. Dieser Engpaß erfordert die Planung der Betriebsmittelzuteilung.

Es gibt viele Strategien, Betriebsmittel zuzuteilen. Auch die Art wie Magnetbänder zugeteilt werden können, unterliegt verschiedenen Optimierungsaspekten.

Normalerweise werden Datenträgerwechsel minimiert. Sie benötigen viel Zeit.
Bei längerfristigen Operationen wird die Sicherung von großen Datenbeständen vorzugsweise auf Zeiten geringer Aktivität (nachts) verlegt.

Es gibt jedoch Situationen, in denen man die Bearbeitungszeiten von bestimmten Aufträgen minimieren möchte. Da die Anforderungen sogar im täglichen Betrieb wechseln, oder sich die Umgebung durch neue Hardware verändert, kann man kaum eine allgemeingültige Betriebsmittelzuteilungstrategie entwickeln.

Dieses Problem kann nur durch die Trennung von Mechanismus und Strategie befriedigend gelöst werden. Aus diesem Grunde wurde ein extern programmierbares Modul zur Betriebsmittelzuteilung entwickelt.

Dieses Modul besteht aus den Komponenten:

- der partiellen Ordnung, die programmierbar ist
- einer Transaktion, die die Betriebsmittel besorgt und zuteilt.

Die partielle Ordnung bestimmt, welche Betriebsmittel besorgt werden und an welche Transaktionen diese Betriebsmittel zugeteilt werden.

Die Art der benötigten Betriebsmittel wird durch die anfordernde Transaktion bestimmt. Sie ist von der jeweiligen Art des Dienstes abhängig. Bei einem Volume-Daemon wären die verwalteten Betriebsmittel:

- die Geräte,
- die Datenträger.

Die partielle Ordnung gibt die Reihenfolge und mögliche Parallelität der Aufträge an. Herkömmliche Systeme lösen diese Schedulingaufgaben durch Warteschlangen mit statischen oder dynamischen Prioritäten und einer festen Strategie.

Das Besondere an der BackStage-Betriebsmittelzuteilung ist, daß die Bewertungsfunktion für die Betriebsmittelzuteilung im laufenden Betrieb geändert werden kann.

Die Betriebsmittelverwaltungskomponente verwaltet Transaktionen, die Betriebsmittel belegen wollen:

- Die Menge der benötigten Betriebsmittel ergibt sich aus der Menge der wartenden Transaktionen, die entsprechend sortiert sind.

- Es werden immer alle benötigten Betriebsmittel auf einmal angefordert.
- Es gibt keine Nachforderungen von Betriebsmitteln. Damit wird vermieden, daß es zu Deadlock-Situationen kommen kann.
- Anfordernde Transaktionen werden in Klassen eingeteilt. Die Klassen werden durch entsprechende Selektionen auf den Attributwerten der Transaktionen beschrieben.
- Für die Klassen gibt es eine partielle Ordnung.
- Die höchsten belegten unvergleichbaren Klassen bilden die Menge der zu befriedigenden Betriebsmittelanforderungen.
- Transaktionen aus den höchsten Klassen, deren Betriebsmittelanforderungen befriedigt werden können, dürfen nach Bindung der Betriebsmittel weiterarbeiten.

Es gibt zwei Arten von Betriebsmitteln. Die erste Art kann automatisch besorgt werden (Datenträgerwechsler, freier Plattenplatz,...).
Die zweite Art erfordert manuelle Intervention (Band einlegen,...). Immer, wenn alle Anforderungen einer Transaktion befriedigt sind, wird die Transaktion fortgesetzt. Ein Neuabgleich zwischen verfügbaren und angeforderten Betriebsmitteln erfolgt bei drei Ereignissen:

- Verfügbarwerden eines Betriebsmittels,
- Änderung der partiellen Ordnung (neue Auftragssortierung),
- Ankunft einer neuen Transaktion (eventuelle Verdrängung anderer Transaktionen).

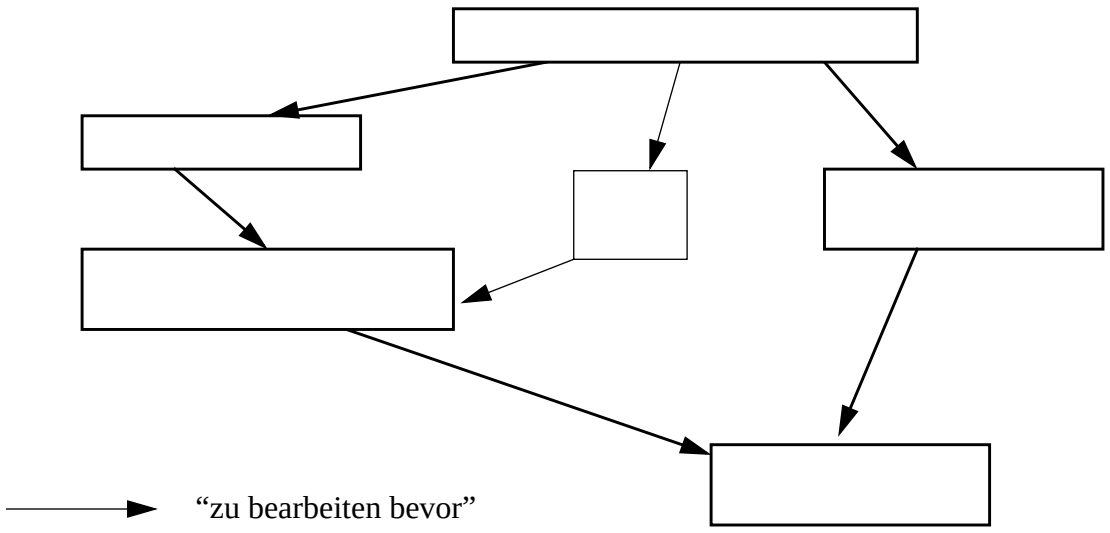


Abb. 3.5 Partielle Ordnung zur flexiblen Auftragsplanung

Die Vorteile der Betriebsmittelzuordnung nach dem oben beschriebenen Verfahren liegen in:

- möglicher Einbeziehung aller Attribute und Systemfunktionen (großer Optimierungsraum),
- guter Anpaßbarkeit an den jeweiligen Betrieb,
- vielen möglichen Strategien (First-Come-First-Served, Shortest-Remaining-Time-First, Most-Important-First,...),
- vollständigem Überblick über die aktuellen und zukünftig benötigten Betriebsmittel,
- vorausschauender Planung,
- möglicher Verdrängung (unterstützt durch das Transaktionskonzept),
- dauerhafter Konfiguration der Strategie.

3.3.8 Authentisierung und Autorisierung

Diese beiden Vorgänge stellen in einem Massenspeichersystem für langfristige Speicherung ein großes Problem dar. Zum einen verwenden die heutigen Betriebssysteme eine Vielzahl von Authentisierungsmechanismen. Zum anderen wird von jeder Institution eine andere Sicherheitsphilosophie und Abrechnungsstrategie verwendet. Es erscheint fast unmöglich, diese Anforderungen vernünftig zu vereinen.

Zugriffsrechte und Authentisierungsinformation in langfristig gespeicherten Datenbeständen unterliegen demselben Problem, wie die gespeicherten Daten selbst. Bei noch so sorgfältiger Organisation von Zugriffsrechten und Benutzerkennungen kommt es unweigerlich zum Veralten dieser Informationen. Ein Projekt wird aufgelöst, der Betrieb wird umorganisiert, oder Mitarbeiter scheiden aus. Die Autorisierungsdaten konsistent zu halten, ist aufwendig. Dieses Problem ist bei heutigen Systemen (besonders bei denen, die auf heutigen de-Facto-Standards basieren) nicht zu lösen. Wenn man nicht zusätzlich zu den numerischen Identifikatoren noch die zeitliche Folge der Vergabe der numerischen Identifikatoren an Benutzer dokumentiert, kommt es bei diesem Verfahren unweigerlich zu Aliasing-Effekten, bei denen dieselbe numerische Kennung für unterschiedliche Benutzer wiederverwendet wird. Demnach ist eine hierarchische Rechteorganisation wohl noch das Bestmögliche. Um überhaupt eine Chance zu haben, müssen rechtetragende Attribute auch nachträglich veränderbar sein. Im Idealfall wird die Rechtekontrolle innerhalb der Archive mit der aktuellen Benutzerverwaltung gekoppelt. Damit wird es dann auch möglich, alte Datenbestände, die ausgeschiedenen Mitarbeitern gehören, zu entdecken und entsprechend zu behandeln (löschen oder weitergeben).

3.3.8.1 Authentisierungsverfahren

Derzeit sind verschiedene Authentisierungsverfahren in Gebrauch. Bei der Authentisierung handelt es sich um Verfahren, um zu verifizieren, ob ein Subjekt die von ihm angegebene Identität besitzt. Dieser Beweis wird häufig mit Hilfe von Geheimnissen durchgeführt, deren Kenntnis verifiziert wird. Als Geheimnisse dienen Passwörter und geheime Schlüssel. Ein besonderes Problem bei diesem Verfahren ist, daß diese Geheimnisse während des Austausches mit dem System möglichst nicht in unbefugte Hände fallen dürfen. Andere Verfahren stützen sich auf unveränderliche, schwer zu reproduzierende Merkmale (Netzhautmuster, Fingerabdrücke). In der Datenverarbeitung werden auch diverse Protokolle verwendet, die diese Authentisierungen ermöglichen sollen. Welche dieser Protokolle eingesetzt werden, hängt von den Verfahrensweisen der Institution ab.

Um die Leistungsfähigkeit in BackStage nicht einzuschränken, ist in BackStage nur vorgesehen, daß die kommunizierenden Systeme das Authentisierungsprotokoll aushandeln können. Dieses geschieht dadurch, daß beim Aufbau einer Kommunikationsverbindung eine gemeinsame Menge von Authentisierungsprotokollen ausgehandelt wird. Diese Protokolle werden dann durchgeführt.

Die Qualität der Authentisierungsverfahren (Unix-Passwort, Kerberos[SNS88][BM91]) ist naturgemäß unterschiedlich. Auch die authentisierten Identitäten unterscheiden sich. Deshalb müssen diese entsprechend qualifiziert verwaltet werden. Backstage übersetzt die ausgehandelten Entitäten mit Hilfe einer Tabelle in eine Backstage-einheitliche Kennung. Diese Dienstleistung wird von einem Authentisierungsdienst erbracht, der BackStage-Identifikationen generiert und kryptographisch absichert. Diese generierte BackStage-Identifikation wird für alle Zugriffsoperationen verwendet.

Da die Identitätsnachweise aber unterschiedlich vertrauenswürdig sind, wird die Backstagekennung mit den verwendeten Verfahren qualifiziert. So können einem Benutzer mehr Rechte eingeräumt werden, wenn er sich mit einem, gemäß lokaler Einschätzung, vertrauenswürdigeren Verfahren authentisiert hat.

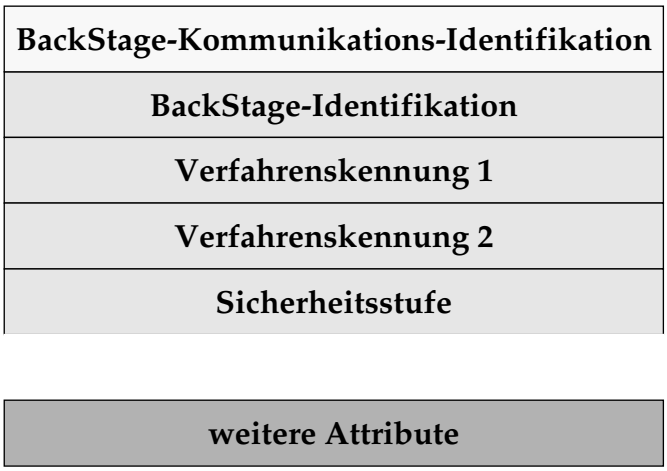


Abb. 3.6 BackStage-Identifikation

In der Authentisierungsstruktur werden zwei Identifikationen gehalten.

Die eine ist die Kommunikationsidentifikation. Sie gibt an, von welchem Subsystem die Authentisierungsinformation übermittelt wurde. Damit ist überprüfbar, ob die richtigen Teilsysteme miteinander kommunizieren.

Die andere ist die BackStage-Identifikation. Sie gibt an, für welches Subjekt der Dienst erbracht werden soll. Diese Identifikation bestimmt die Zugriffsrechte.

Sollen Anpassungen, wie die Änderung der Sicherheitsstufe, an der BackStage-Identifikation vorgenommen werden, kann dieses nur vom Authentisierungsdienst durchgeführt werden. Dieser Dienst kann die Legalität der Anforderung anhand der Kommunikationsidentifikation entscheiden.

Die Absicherung der Identifikationsdaten sollte über ein Public-Key Verfahren geschehen, damit nicht für jede Prüfung der Authentisierungsdienst herangezogen werden muß.

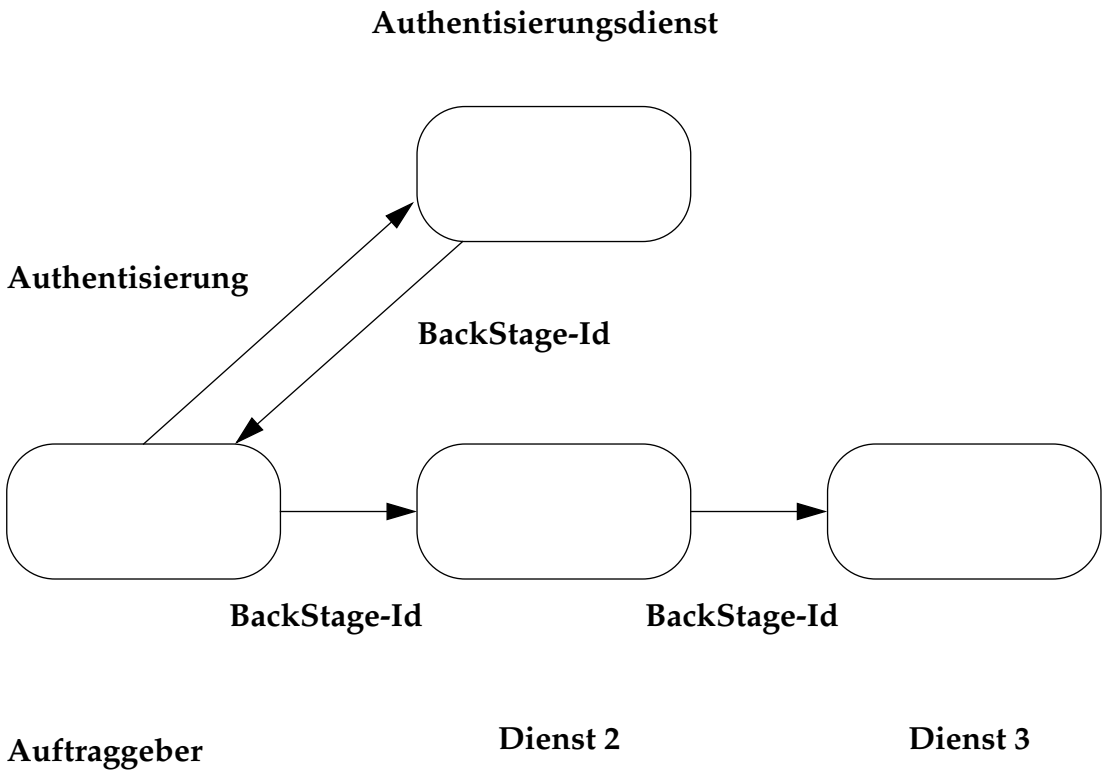


Abb. 3.7 Weiterleitung einer BackStage-Identifikation

3.3.8.2 Authorisierungsmechanismen

Bei den Authorisierungsmechanismen geht es um die Frage, welche Rechte einer Identität eingeräumt werden sollen. Wie schon geschrieben, gibt es unterschiedlich vertrauenswürdige Authentisierungsmechanismen. Diese Tatsache muß auch bei dem Authorisierungsverfahren berücksichtigt werden. Das bedeutet, daß Rechteeinträge mit den entsprechenden Authentisierungsverfahren qualifiziert werden müssen. Damit ist es möglich, die lokalen Verfahrensweisen abzubilden.

BackStage verwendet zur Rechtekontrolle modifizierte Zugriffskontrollisten. Hier werden die eingeräumten Rechte anhand der BackStage-Identifikation und dem verwendeten Authentisierungsverfahren bestimmt. Dieses Verfahren erlaubt auch, die erzwungene Zugriffskontrolle (*mandatory access control*) zu realisieren. Die einzige Forderung dafür ist, daß an die Identifikation die aktuelle Sicherheitseinstufung eingetragen werden kann. Das ist über den Authentisierungsdienst möglich.

Die Zugriffsroutinen können anhand dieser Einstufung den Zugriff bestimmen. Die Sicherheitseinstufungen der BackStage-Datenobjekte sind in Form von Authorisierungsattributen verfügbar. Für diskrete Zugriffskontrollverfahren genügt eine einmal durch den Authentisierungsdienst ausgestellte BackStage-Identifikation.

3.4 BackStage-Komponenten und Abstraktionen

BackStage besteht aus drei Hauptabstraktionsebenen wie sie in Abschnitt 3.2.1 beschrieben sind. Die Eigenschaften der einzelnen Abstraktionsebenen stellen sich wie folgt dar.

3.4.1 Volumemanager

Der Volumemanager verwaltet alle Datenträgeroperationen und übernimmt die Buchführung. Um Daten beliebiger Länge verarbeiten können, baut der Volumemanager die Abstraktion der v-Dateien auf:

> V-Dateien werden zu logischen Gruppen, den Volumes, zusammengefaßt.
>
> Volumes können mit einer Datenträgermenge assoziiert werden. Sie können auch als Ganzes in Form einer Datenträgermenge von einem Volumemanager zu einem anderen übertragen werden.

Weiterhin kann für ein Volume die Art der verwendeten Zugriffskontrollmechanismen spezifiziert werden.

Ein Volumemanager verwaltet alle ihm zugeordneten Volumes und die in ihnen gespeicherten Dateien. Weitere Aufgaben des Volumemanagers sind:

- Gewährleistung von Redundanzanforderungen,
- Datenerhaltungsmaßnahmen,
- Kopieren,
- Kompaktifizieren.

Da der Volumemanager die programmierbare Betriebsmittelverwaltung enthält, können dieses Aktionen als eigenständige Transaktionen innerhalb des normalen Betriebes abgewickelt werden.

Die Strukturierung der Objekte Volume und v-Datei entspricht der in BackStage üblichen offenen Technologie benannter Attributmengen. Da der Volumemanager primär für preiswerte Langzeitspeicherung vorgesehen ist, wird weniger das im Mass Storage Reference Model verwendete Modell der Datei mit wahlfreiem Zugriff verwendet, sondern das der sequentiellen Datei. Diese erlaubt unter Umständen auch noch den wahlfreien Zugriff.

Als Operationen des Volumemanagers sind vorgesehen:

- Erzeugen eines Volumes,
- Löschen eines Volumes,
- Schreiben einer v-Datei,
- Lesen einer v-Datei,

- Setzen von Volume und v-Datei Attributen (auch nachträglich),
- Lesen von Volume und v-Datei Attributen,
- Definition von Datenträgertypen (und ihrer Operationen wie Freispeicherverwaltung oder physischer Charakteristika),
- Definition von Geräten
- Kontrolle (im Sinne von Definition) der partiellen Ordnung der Betriebsmittelverwaltung,
- Liste der anstehenden Aufträge und der manuell zu beschaffenden Betriebsmittel,
- Konfiguration.

Der Volumemanager selbst gliedert sich in mehrere Untereinheiten:

- dem eigentlichen Volumemanager,
- den Gerätesteuereinheiten (device controls),
- der Protokolleinheit für den Massendatentransport (*data-link-provider*).

Alle diese Systeme kommunizieren untereinander mit Hilfe der Transaktionsoperationen.

Die Trennung von Volumemanager und Geräteansteuerung erlaubt es, daß der Volumemanager auch Geräte bedienen kann, die nicht an dem Knoten, an dem der jeweilige Volumemanager realisiert ist, angeschlossen sind.

Weiterhin ergibt sich eine Geräteschnittstelle, die es erlaubt, neue Geräte zu unterstützen. Dieses ist möglich, ohne Änderungen an den Volumemanagern vornehmen zu müssen. Die Geräteansteuerungen müssen nur dem Protokoll genügen.

Durch die Trennung der eigentlichen Geräte und der die Volumes verwaltenden Instanz sind auch optimierte Datenübertragungen möglich. Dieses wird dadurch möglich, daß Verbindungen direkt zwischen der Datenquelle und der Datensenke aufgebaut werden können.

Die Verwaltung der Volumemanageroperationen wird auf einer weiteren dafür vorgesehenen Maschine durchgeführt. Da die Kommunikation mit den Unterkomponenten über das normale BackStage-Transaktionskonzept erfolgt, gelten die üblichen Verfahren für Authentisierung und mögliche Erweiterbarkeit (Aufträge sind Attributmengen).

3.4.1.1 Die Gerätesteuerung

Die Gerätesteuerung bildet die Geräteabstraktion. Neben der Verwaltung einzelner oder einer Gruppe von Geräten eines Typs übernimmt die Gerätesteuerung die Datentransfers zum und vom Gerät, von oder zur Datenquelle / -senke.

Die Basisoperationen der Gerätesteuerung umfassen:

- Belegen,
- Freigeben,
- Positionieren,
- Lesen,
- Schreiben,
- Unterstützte Operationen angeben.

Als schwächste Abstraktion bietet die Gerätesteuerung die des sequentiellen Datenträgers. Sie erlaubt eine dateiweise Positionierung. Diese Abstraktion ist heutzutage von fast jedem Datenträgertyp zu erfüllen.

Da es in BackStage immer möglich ist, das System zu erweitern, kann die Gerätesteuerung auch weitere Informationen über die technischen Möglichkeiten der von ihr unterstützten Geräte liefern. So wird zum Beispiel die Art der möglichen Schreiboperationen angegeben.

Viele Datenträger können nur sequentiell beschrieben werden (Video-Bänder). Neue Daten können nur am Ende des Datenträgers angefügt werden.

Die nächst bessere Art, Daten zu schreiben, erlaubt das Überschreiben einzelner Datenbereiche an jeder Position (reel-tapes). Dieses kann allerdings auf Blockgrenzen eingeschränkt sein.

Auf die Möglichkeit des wahlfreien Überschreibens folgt die Funktionalität des wahlfreien Einfügens (meist nur noch Dateisysteme mit eigener Platzverwaltung).

Durch die Möglichkeit, die unterstützten Zugriffsmethoden abzufragen, ergibt sich eine flexible Integration von neuen Komponenten in das System, ohne alle Systemkomponenten verändern zu müssen.

So können auch neue Datenträgersysteme eingeführt werden und von älteren Volumemanagern verwendet werden, die noch nicht die volle Funktionalität der neuen Speichersysteme ausnutzen. Für eine effektive Ausnutzung der Geräte müssen allerdings auch die entsprechenden Volumemanager angepaßt werden.

Die wesentlichen Unterschiede zwischen den einzelnen Massenspeichersystemen bestehen also einerseits in der Art, wie Daten geschrieben werden können und andererseits darin, wie Datenträger adressiert werden können.

Bei manuellem Betrieb kann nur auf das Einlegen des entsprechenden Datenträgers gewartet werden.

Bei Wechselautomaten kann ein entsprechender Datenträger über eine Datenträgerkennung automatisch angefordert werden. Er wird automatisch in ein Gerät eingelegt und ist dann zugreifbar.

Diese beiden Verfahren werden auch vom Volumemanager unterschieden. Anforderungen für manuell zugreifbare Datenträger verbleiben in der Betriebsmittelverwaltungskomponente als wartende Aufträge. Automatisch zugreifbare Datenträger werden direkt bei der entsprechenden Gerätesteuerung angefordert. Sollte der angeforderte Datenträger dennoch nicht verfügbar sein, so wird diese Fehlerbedingung wie ein manueller Datenträger weiterbearbeitet. Die auf manuelle Intervention wartenden Aufträge mit ihren entsprechenden Geräteanforderungen sind als solche sichtbar, und sie können dem Bedienpersonal angezeigt werden.

Werden diese Aufträge nicht rechtzeitig bearbeitet, so greift der Automatismus für die verwaisten Transaktionen. Die Aufträge werden dann abgebrochen, wenn ihre Lebensdauer überschritten wurde.

Wurde der korrekte Datenträger in das Gerät entweder manuell oder automatisch geladen, so können Lese- und Schreibtransaktionen durchgeführt werden.

Normalerweise wird ein Gerät fest einem Volumemanager zugeordnet. Diese Zuordnung ist aber nicht zwingend, da es sinnvoll sein kann, mehrere Volumemanager mit gemeinsamen Geräten zu betreiben. Aus diesem Grunde ist ein Belegungsprotokoll vorgesehen. Dieses erlaubt, das Gerät zu reservieren.

3.4.1.2 Datentransport

Ein weiteres Subsystem des Volumemanagers ist der Datentransportdienst. Viele Betriebssysteme verfügen über mehrere verschiedene Arten des Datentransports. Diese Dienste sind meist für die verschiedenen Anwendungen optimiert.

Im Bereich Unix sind hier zu nennen:

- gemeinsamer Speicher,
- Pipes,
- reguläre Dateien,
- Geräte,
- Sockets.

Die Anwendung im Bereich BackStage umfaßt zwei Verwendungsformen.
Es ist einerseits die bidirektionale Kommunikation zum Austausch von Aufträgen (Auftragsschnittstelle) und zum anderen der Massendatentransport.
Viele Betriebssysteme bieten hierfür schon die entsprechenden Dienstleistungen an (Datagramm und verbindungsorientierte Dienste mit einstellbaren Verbindungsparametern).

Um innerhalb von Backstage eine konsistente Adressierbarkeit der beteiligten Einheiten zu erreichen, liegt es nahe, auch die möglichen Datenquellen und Datensenken als Transaktionen mit entsprechenden Transaktionsidentifikationen auszubilden. Diese Funktionalität wird von dem BackStage *data-link-provider* zur Verfügung gestellt. Bei

dem *data-link-provider* handelt es sich um ein Programm. Das Programm stellt die Anbindung von BackStage-Transaktionen an das jeweilige Betriebssystem dar.
Von Backstage aus gesehen handelt es sich bei dem *data-link-provider* um eine Transaktion.
Von der Betriebssystemseite her gesehen stellt der *data-link-provider* eine Datenquelle / -senke dar.
Der *data-link-provider* ist in einem Unix System normalerweise ein Anfangs- oder Endstück einer Pipe. Die Generierung einer Datenquelle / -senke innerhalb eines BackStage Systems besteht aus dem einfachen Instantiieren eines Dienstes bei einem Portmapper. Diese Instantiierung erzeugt eine normale BackStage-Transaktionsidentifikation. Sie kann dann als Datenquelle / -senke angegeben werden.
Da die Instantiierung einer Transaktion die Angabe der zu startenden Applikationen beinhaltet, ist hier eine flexible Anbindung an das jeweilige Betriebssystem möglich.
Für eine Datentransporttransaktion gibt es verschiedene Anforderungen an die Dauerhaftigkeit. Je nach Art des verwendeten Dienstes kann die Transaktion nur einmalig und vollständig durchgeführt werden. Dieses gilt besonders für Konstruktionen, die bei erneuter Durchführung neue und andere Daten erzeugen (nicht idempotent). Idempotente Transaktionen können natürlich immer wieder durchgeführt werden. Die Idempotenzüberlegungen beziehen sich hier nicht auf die einzelne Transaktionskennung, die ja einmalig ist, sondern auf die Wiederanlaufeigenschaften im Falle einer Störung. Als Störungen sind Netzpartitionierungen, Anwendungsfehler und Rechnerzusammenbrüche zu betrachten.

Bei Rechnerzusammenbrüchen sind zwei Fälle zu berücksichtigen:

Rechnerzusammenbrüche auf der Seite des Massenspeichers
: Diese Art der Ausfälle kann bei geeigneter Protokollkonstruktion toleriert werden. Es erfordert nur, daß auf der Seite der Datenquelle / -senke die entsprechenden Datenverbindungen aufrechterhalten werden. Daten, die eventuell während eines Netz- oder Knotenzusammenbruchs verloren gehen, können zwischengespeichert werden. Diese Vorgehensweise erlaubt ein Wiederaufsetzen der Transferoperationen ohne die Datenquelle / -senke (bis auf Verzögerungen) zu stören.

Zusammenbrüche auf Seite der Datenquelle / -senke
: Die Idempotenzeigenschaft ist zu berücksichtigen. Hier können nicht idempotente Operationen nicht wieder neu aufgesetzt werden. Dieser Ausfall eines Dienstes stellt sich für das BackStage System in der Form einer abgebrochenen Transaktion (*abort()*) der Datenquelle / -senke dar. Hier zeigt sich auch der Vorteil der einheitlichen Strukturierung aller Systemkomponenten in Form von Transaktionsidentifikationen. Die Fehlerbehandlung wird erheblich vereinfacht.

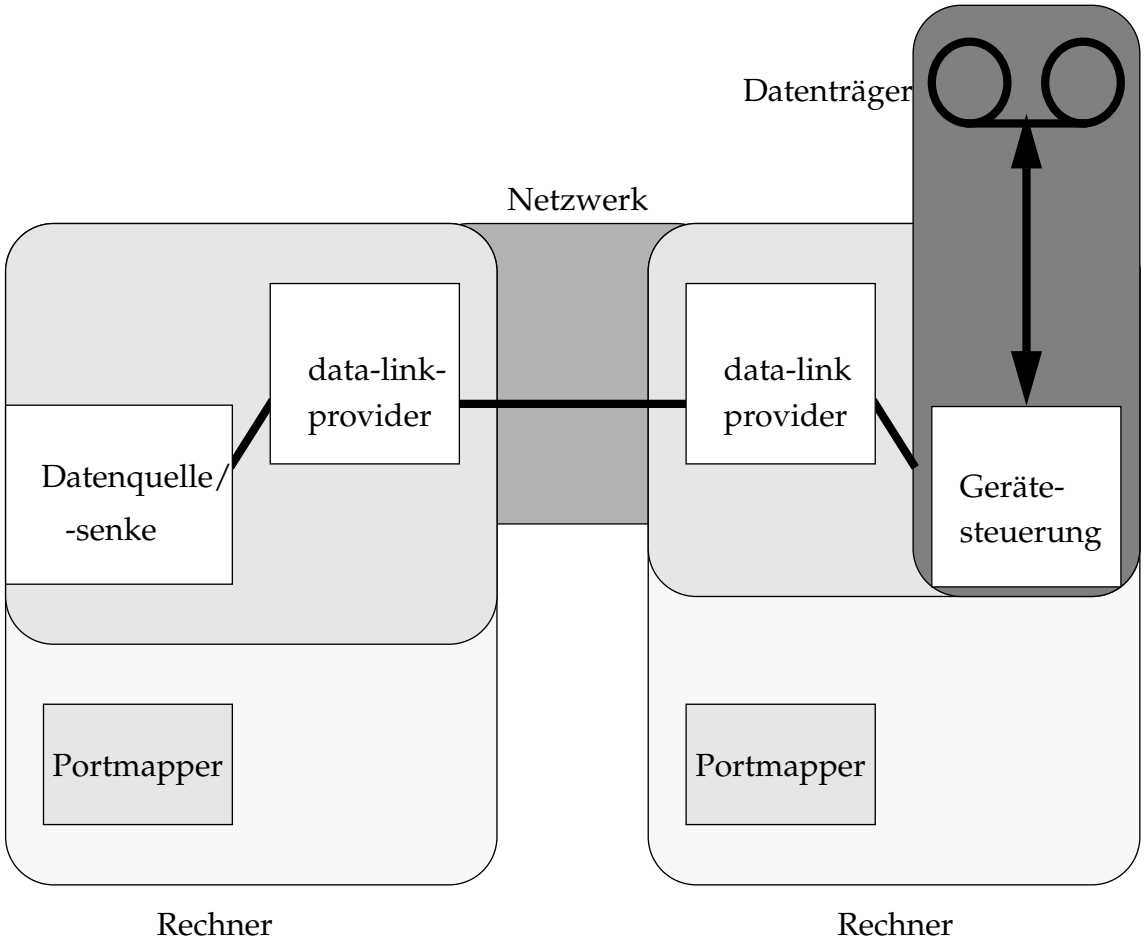


Abb. 3.8 Datentransportausfallbereiche

3.4.1.3 Datenträgerverwaltung

Weil innerhalb des BackStage-Systems verschiedenartigste Datenträger verwaltet werden müssen, liegt die Forderung nach einem flexiblen Adressierungsschema für Massenspeichersysteme nahe. Die Bandbreite der Adressierungsformen der verschiedenen Massenspeichersubsysteme reicht vom einmaligen sequentiellen Beschreiben (CDROM (“wahlfrei lesbar/Blöcke”), Tape (“wieder-beschreibbar/Dateien”) und ROM (“wahlfrei lesbar/Bytes”) über wahlfreien Lese-/Schreibzugriff bis hin zu Assoziativspeichern (CAM). Da es kaum möglich ist, alle diese Adressierungsarten (sequentiell, wahlfrei sowie Blockgranularität, Pfadname und Inhaltskomponenten) mit einem Mechanismus konsistent zu unterstützen, wird als Datenstruktur zur Repräsentation der Adresse ein Vektor von Zeichenketten gewählt. Dieser erlaubt es, die Adressen der meisten heutigen Massenspeichersysteme zu repräsentieren.

Für einfache Adressierungsverfahren, wie sie bei Bandgeräten üblich sind, genügt eine Komponente mit der numerischen Angabe der Position der Datei.

Sind die Geräte in der Lage, weitere Unterstrukturen (zum Beispiel Blöcke) zu adressieren, so kann diese Information in der nächsten Komponente abgelegt werden.

Bei komplexeren Speicherungsformen, wie bei Dateisystemen, können die Pfadkomponenten abgelegt werden. Es gibt also mit dieser Datenstruktur die Möglichkeit, verschiedene Adressen zu repräsentieren. Die Wahl des Adressierungsverfahrens geschieht zwischen Volumedaemon und Gerätesteuerung. Hierbei hat die Gerätesteuerung noch die Übersetzung der Zeichenkettenvektoren in die aktuell vom Gerät verwendeten Adressierungsschemata zu leisten.

Der Nachteil dieser weitgehend ungetypten Adreßinformation liegt darin, daß eine Typüberprüfung bestenfalls nur dynamisch erfolgen kann.

Der Vorteil liegt darin, daß eine große Anzahl verschiedenster Adressierungsverfahren realisiert werden kann. Neue Adressierungsverfahren können leicht mit aufgenommen werden. Für den Volumedaemon stellen diese Adressierungsschemata keine großen Anforderungen.Diese Adressen werden nur zusammen mit den anderen Attributen der gesicherten v-Datei gespeichert.

Das einzige Problem taucht bei der Bestimmung von neuen Adressen zur Speicherung von neuen v-Dateien auf. Die Generierung neuer Adressen kann nicht einfach auf die Gerätesteuerung abgewälzt werden, da diese keinen direkten Zugriff auf die für das jeweilige Volume bestehenden Verzeichnisse und Belegungsinformationen haben.
Die Belegungsplanung muß durch den Volumemanager erfolgen.

Die Belegung stützt sich innerhalb von Backstage auf zwei Modelle:

Das eine Modell ist ein fiktiv lineares Speichermedium, bei dem die Adressierungsinformation mit der aktuellen Platzbelegung korreliert ist. “Größere” Adressen liegen physisch hinter “kleineren” Adressen. Dieses Schema trifft für fast alle magnetischen Datenträger zu. Bei dieser Art der Belegung müssen neu zu vergebende Adressen richtig gewählt werden, da die Adressen unmittelbar mit physischen Positionen korreliert werden.

Bei dem zweiten Modell der Belegung gibt es keine Korrelation zwischen den Adressen und der Platzbelegung. Dieses kommt dadurch zustande, daß sich die eigentliche Platzbelegung dem Volumemanager entzieht, weil diese Verwaltung in der Gerätesteuerung oder im Gerät stattfindet. Hier kann nur noch der Datenträger anhand des noch verfügbaren Speicherplatzes bestimmt werden. Diese Eigenschaft der selbständigen Speicherplatzverwaltung findet man meistens in eigenständigen Subsystemen bei Dateisystemen oder weiteren Volumemanagern.

Der Volumemanager muß also über Belegungs- beziehungsweise Adreßgenerierungsmechanismen verfügen, um die für die entsprechende Gerätesteuerung gültigen Adressen bestimmen zu können. Diese Funktionalität kann wegen der Belegungsinformation, die zu einem ganzen Volume gehört, auch nicht vollständig an die entsprechenden Gerätesteuerungen delegiert werden. Die Menge der zu unterstützenden Generierungsschemata kann aber stark eingeschränkt werden, damit die Gerätesteuerun-

gen entsprechenden Adressumsetzungen durchführen können. Es ist fast immer möglich, das sequentielle Datenträgermodell auf den Datenträgern zu realisieren. Um die für diesen Datenträger optimalen Adressierungsmechanismen zu unterstützen, muß der Volumemanager modifiziert werden. Damit ist eine effizientere Ausnutzung eines Datenträgers möglich.

3.4.1.4 Betriebsmittelverwaltung

Alle Transaktionen, die spezielle Betriebsmittel benötigen (Datenträger, die manuell zur Verfügung gestellt werden), befinden sich im "blockiert"-Zustand in der Betriebsmittelverwaltungskomponente. Hier wartet jede Transaktion auf die von ihr benötigten Betriebsmittel. Sind alle Betriebsmittel für die Transaktion verfügbar, so wird die Transaktion fortgesetzt. Eine dauerhafte Transaktion übernimmt die Auswertung der Betriebsmittelanfragen und macht die Betriebsmittelzuteilung. Zur Bestimmung, welche Betriebsmittelanforderungen vordringlich zu erfüllen sind, dient die im Kapitel 3.3.7 (S. 37) beschriebene partielle Ordnung. Aus der Menge der auf Betriebsmittel wartenden Transaktionen ergeben sich auch die für das Bedienpersonal zu erfüllenden Aufgaben.

3.4.1.5 Operatorschnittstelle

Die Schnittstelle für das Bedienpersonal ergibt sich direkt aus der Menge der auf Betriebsmittel (hauptsächlich Geräte und Datenträger) wartenden Transaktionen. Aus der Liste der wartenden Transaktionen und der angeforderten Betriebsmittel können unmittelbar die notwendigen manuellen Arbeitsgänge bestimmt werden. Weil auch die Daten für später zu befriedigende Transaktionen unter Umständen schon vorliegen, können die Arbeitsgänge optimiert werden. Wird auch noch die Möglichkeit der Beeinflussung der Zuordnung von Transaktionen und ihrer Bearbeitungsklassen gegeben, so können die Anforderungen in die jeweils gewünschte Richtung durch Zusammenfassen aller Aufträge optimiert werden.

Eine weitere wichtige Information für das Bedienpersonal ist der Zustand der verwendeten Betriebsmittel (Zustand der Geräte und Datenträger). Dieser Zustand wird durch drei Einflüsse bestimmt:

- die Aktionen der Transaktionen (automatische Operationen auf den Geräten),
- die Fehlerzustände (defekte Geräte),
- die administrativen Entscheidungen (Wartungen, exklusive Zuordnungen).

Demnach werden den Betriebsmittelverwaltungen Zustandsänderungen von mehreren Quellen übermittelt. Sie werden von den Transaktionen selbst (Belegen und Freigeben sowie den Aktionen), den Geräten (Fehlerbedingungen, Zustandswechsel durch manuelle Intervention) und dem Bedienpersonal (Wartungsanforderungen, Da-

tenträgerwechsel) hervorgerufen. Alle diese Betriebsmittelzustandsänderungen werden durch die Betriebsmittelverwaltung koordiniert und für die Betriebsmittelzuordnung zu den Transaktionen verwendet.

Zusätzlich zu den technisch bedingten Zustandsänderungen gibt es noch administrative Gründe, die ein Eingreifen in den Ablauf der Betriebsmittelzuteilung notwendig machen. So kann es vorkommen, daß bestimmte Aufträge aus technischen oder administrativen Gründen nicht durchgeführt werden können. Hier muß die Möglichkeit des Abbruchs von Transaktionen durch das Bedienpersonal vorgesehen werden. Dieser Abbruch gestaltet sich im allgemeinen als unproblematisch, zumal das gesamte System die Möglichkeit des Abbruchs (*abort*) grundsätzlich vorsieht. Die Behandlung der abgebrochenen Betriebsmittelbelegungen geschieht in der normalen Fehlerbehandlung der Transaktion und führt in der Regel zum Abbruch der gesamten Transaktion.

Ein Problem ist die Zugriffskontrolle. Da hier von dem Auftraggeber eine Dienstleistung von Bedienpersonal gefordert wird, sollte ihm die Möglichkeit des Abbruches aufgrund von administrativen Gegebenheiten eingeräumt werden. Dieses ist einfach durch das Aufnehmen des Bedienpersonals in die Zugriffskontrolliste für die Transaktion zu erreichen. Entscheidungen dieser Art sind natürlich an die jeweils gültigen lokalen Verfahrensweisen gebunden. Deshalb ist die Bindung dieser Rechte an die Zugriffsrechte auf die Transaktionen gerechtfertigt.

Diese Strukturierung der Auftragsbearbeitung erlaubt ein weitgehend automatisches Abarbeiten der Betriebsmittelanforderungen. Durch Intervention kann die der Betriebsmittelzuteilung zugrundeliegende partielle Ordnung modifiziert werden. Ebenso können die Attribute der Transaktionen, die auf Betriebsmittel warten, entsprechend geändert werden. Solche Änderungen führen dazu, daß sich die Position der Transaktion in der derzeit gültigen partiellen Ordnung (durch in der Ordnung bevorzugte Prioritätsattribute) verschiebt.

3.4.1.6 Auftragsbearbeitung

Eine der häufigsten Aktionen des Volumemanagers ist das Lesen oder das Schreiben einer v-Datei.

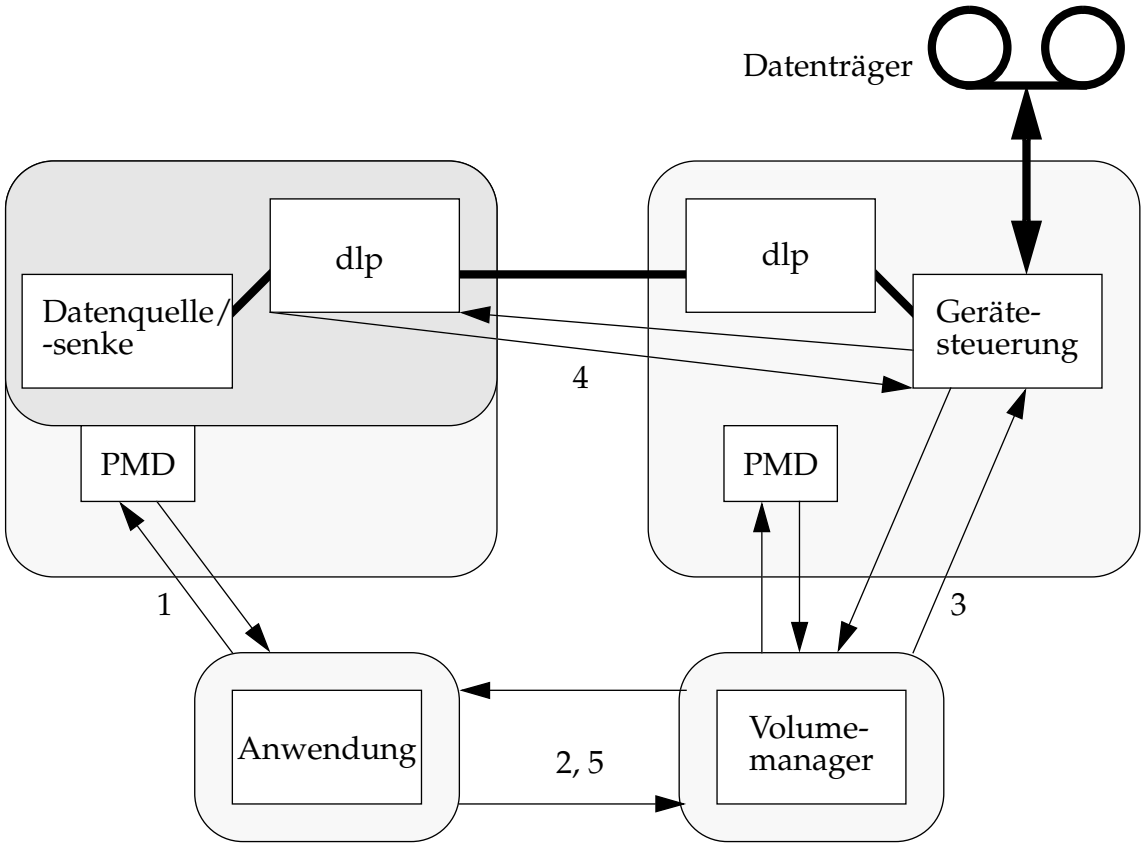


Abb. 3.9 Volumemanager Auftragsschema

Der Ablauf einer solchen Transaktion geschieht folgendermaßen:

(1) Die Anwendung erzeugt auf dem Zielknoten eine Datenquelle/-senke.

(2) Die Anwendung übermittelt den “Schreibe/Lese”-v-Datei Auftrag dem Volumemanager.

(3) Der Volumemanager lokalisiert die benötigten Datenträger und beauftragt die entsprechende Gerätesteuerung, einen Transfer durchzuführen.

(4) Die Gerätesteuerung baut über die Datenprotokolleinheiten eine Datenverbindung zur Datenquelle/-senke auf und führt den Transfer durch.

(5) Nach Erteilung der Aufträge wartet die Anwendung auf die Terminierung des Volumemanagerauftrages und dieser seinerseits auf die Terminierung des Transfers.

3.4.2 Archivmanager

Der Archivmanager stellt bei der langfristigen Datenhaltung die zentrale Komponente dar. Mit dem Archivmanager werden zusammengehörige Datensammlungen, wie sie sich in heutigen Dateisystemen antreffen lassen, verwaltet.

Verfolgt man die Entwicklung eines Dateisystems, so stellt man bei den meisten Datenmengen eine ständige Veränderung fest. Zu bestimmten Zeitpunkten stellt sich auch ein mehr oder weniger konsistenter Zustand der Datenmenge ein. Das ist dann der Fall, wenn eine offizielle Version eines Programmsystems fertiggestellt ist. Zu diesen Zeitpunkten ist es meistens auch wünschenswert, diesen Zustand zur späteren Referenz zu dokumentieren; also zu archivieren.

Meistens erstreckt sich die Lebensdauer solcher Datenmengen aber über mehr als einen konsistenten Ausgabestand hinaus, was zu mehreren "konsistenten" Zuständen einer Datenmenge führt. Daher sollten diese Versionen möglichst auch gemeinsam in einem Archiv gespeichert werden.

Deshalb ist es für ein Archiv geboten, mehrere Versionen einer dateisystemartig strukturierten Datenmenge zu speichern. Ein Archiv beherbergt die Entwicklung eines Programmsystems, einer Datenbasis oder eines Forschungsprojekts. Aufgabe des Archivmanagers ist es also, die Mechanismen zur Speicherung und zeitlichen Dokumentation von zusammengehörigen Datenbeständen aus Dateisystemen auf Abruf bereitzuhalten.

Die Orientierung an den Dateisystemstrukturen ergibt sich aus der Beobachtung, daß sich die meisten heute verbreiteten Dateisysteme mit erträglichem Aufwand in eine gemeinsame Darstellungsform überführen lassen. Die zu archivierenden Daten sind meistens auch von der Dateisystemstruktur abhängig, in der sie abgelegt sind. Sie werden entsprechend der verfügbaren Dateisysteme organisiert. Ein Beispiel mag hier die Namensgebung von Dateien auf MS/DOS basierten Systemen oder die möglichen Namenslängen auf den verschiedenen Unixdateisystemen sein. Es ist nicht sinnvoll, sich von den ursprünglichen Dateisystemstrukturen zu weit zu entfernen. Zumindest die Möglichkeit der weitgehenden Rekonstruktion der ursprünglichen Dateisystemstruktur muß gegeben sein.[1]

Als Ziel sollte die Rekonstruktion von Eigenschaften der normalen Dateisystemschittstelle angestrebt werden.

1. Bestimmte Eigenschaften, wie zum Beispiel die Lokalisation einer Unixdatei auf einer bestimmten Inode wären allerdings eine zu scharfe Forderung. Selbst die meisten Backupsysteme erfüllen sie nicht. Eigenschaften wie Inodenummern gehören auch nicht zu der normalen Dateisystemschnittstelle eines Unixsystems, sondern stellen vielmehr eine Implementationseigenschaft dar. Es gibt allerdings Fälle, wo diese Informationen von den Programmen genutzt werden (Lizenzmechanismen). Jedoch sind solche Programmsysteme schon im normalen Betrieb administrativ schwer zu handhaben.

Hierzu gehören:

- der Namensraum,
- die Zugriffsrechte,
- der Dateninhalt,
- Attribute von Dateien,
  (Zugriffzeiten, Modifikationszeiten).

3.4.2.1 Archivmodell

Nun sollen die Abstraktionen eines dateisystemorientierten Archivs vorgestellt werden. Wichtig für das Archivmodell ist die Darstellung der Dateisystemobjekte. Zwei Aspekte sind zu berücksichtigen:

- Eine möglichst flexible Darstellung der Dateisystemobjekte sollte gewählt werden.
- Der Namensraum sollte möglichst in der Lage sein, die derzeit existierenden Dateisysteme abbilden zu können.

3.4.2.1.1 Objekte als Attributmengen

Da ein Archiv für längerfristige Datenspeicherung nicht extrem dateisystemspezifisch sein darf, muß eine Darstellungsform gewählt werden, die es erlaubt, möglichst viele der derzeitigen und hoffentlich auch die meisten der zukünftigen Dateisysteme aufnehmen zu können. Die Minimalanforderung ist die Darstellung von Dateien (unstrukturierte Byteströme) mit einer begrenzten Menge von Dateiattributen[2] (Länge, Zugriffszeit, Modifikationszeit, belegte Blöcke, Eigner, Gruppe).

Allerdings sind nicht alle Dateisystemattribute bei der Archivierung unproblematisch. Schon bei der Darstellung von normalen Unixdateisystemdateien ergeben sich Probleme bei der Abbildung bestimmter Attribute und Eigenschaften.
Attribute wie die Gerätekennung oder die Inodenummer gehören zwar zu den vom Betriebssystem angebotenen Dateiattributen, lassen sich aber schwer in einem Archiv abbilden, wenn man das Ziel der exakten Rekonstruktion verfolgt. Das Attribut der Gerätekennung ist von der aktuellen Plattenorganisation sowie von der Betriebssystemversion abhängig. Dieses ist auch darin begründet, daß die Gerätekennung kein eigentliches Dateiattribut, sondern vielmehr ein Dateisystemattribut ist. Die Inodenummer hängt vom Belegungsalgorithmus ab und damit von der Vorgeschichte der Dateigenerierung auf dem entsprechenden Dateisystem.
Attribute wie Gerätekennung und Inodenummer können also nur indirekt verwendet werden. In diesem Beispiel können diese Attribute dazu verwendet werden, ein weiteres unixspezifisches Attribut, den Verweisreferenzzähler (*link count*), näher zu be-

2. Inhalt der *stat(2)* (Berkeley Fast File System) Attributstruktur

stimmen. Innerhalb von Unix Dateisystemen ist es möglich, eine Datei unter verschiedenen Namen anzusprechen.Dieselbe Datei ist unter verschiedenen Namen innerhalb des Dateisystems zugreifbar. Solange mindestens ein Name existiert, wird auch die Datei gespeichert. Die Anzahl der Namen einer Datei ist im Verweiszähler hinterlegt. Demnach gibt es im Unixdateisystem für ein und dieselbe Daten-/Attributmenge mehrere Namen. Welche Namen sich auf ein Dateisystemobjekt (Unixdatei) beziehen, läßt sich nur indirekt über den Vergleich der Gerätekennung und der Inodenummer[3] bestimmen. Die vollständige Menge der Namen einer Datei ist dann bestimmt, wenn man so viele Namen gefunden hat, wie der Verweiszähler angibt. Dieses ist ein relativ aufwendiges Verfahren, zeigt aber die Probleme, die bei der Abbildung von Dateisystemsemantiken auf Archive entstehen. Insbesondere ergeben sich Probleme, wenn zwischen zwei Archivierungsoperationen ganze Dateibäume aus Platzgründen verlagert werden. Dann wird eine nachträgliche Zuordnung von Namen zu den eigentlichen Dateisystemobjekten problematisch. Im allgemeinen können nur die Attribute restauriert werden, deren Manipulation von dem jeweiligen Betriebssystem vorgesehen ist.

Trotz dieser nicht immer ganz einfachen Abbildung von Dateisystemattributen in sinnvolle Archivattribute, stellt die Sichtweise von Dateisystemobjekten als Mengen von Attributen den für längerfristige Datenspeicherung aussichtsreichsten Ansatz. Hier werden keine Annahmen über die eigentliche Objektstrukturierung gemacht.
Innerhalb des BackStage-Archivierungskonzepts werden Dateisystemobjekte als Attributmengen betrachtet. Die Attribute enthalten einerseits die eigentlichen Nutzdaten und andererseits die in dem jeweiligen Dateisystem verwendeten Dateisystemattribute.

Attribute sind Paare aus Namen und Wert. Der Wert ist eine Zeichenfolge unbeschränkter Länge. Der Namensraum für die Attributnamen ist hierarchisch organisiert, um es Anwendungen zu erleichtern, die von ihnen verwendeten Attribute eindeutig und ohne Namenskonflikte zu lokalisieren. Eine spezielle Dateisystemimplementation stellt in diesem Sinne auch nur eine Anwendung dar.

Die hierarchische Organisation des Attributnamensraumes erlaubt auch per Konvention die Dokumentation der Semantiken der Attribute. Durch die hierarchische Organisation ist es auch möglich, zusätzliche Attribute zu den Dateisystemobjekten abzulegen. Diese zusätzlichen Attribute können weitere Hinweise zur Verwendung der entsprechenden Daten geben (die Lebensdauer, Hinweise über die Datendarstellung).

Weiterhin ergibt sich die Möglichkeit, Attribute, die mit der Verwaltung der Dateisystemobjekte zusammenhängen, abzuspeichern. Sie erlaubt auch, die Daten, die für die Archivierung relevant sind, abzulegen.

3. Inodenummern sind nur pro Dateisystem (Gerät) eindeutig (Index in die die Datei beschreibende Attributstruktur).

Insgesamt ist also die Darstellung von Dateisystemobjekten mit Hilfe von Attributmengen nur der Mechanismus. Es werden keine Aussagen über den Dateninhalt gemacht. Die Verwendung einer gemeinsamen Darstellung erlaubt es jedoch, verschiedene Dateisystemobjekte zu speichern und zu verwalten. Weiterhin besteht durch die Speicherung von Zusatzinformationen die Möglichkeit, die Interpretierbarkeit der Daten über längere Zeit zu ermöglichen.

Die Verwaltung läßt sich dadurch vereinfachen, daß die Mindestarchivierungszeit und die benötigten Anwendungen spezifiziert werden. Weiterhin erlauben Datenspezifikationen auch eine spätere Interpretierbarkeit. Mit diesen zusätzlichen Attributen wird die Verwaltung von Archiven entscheidend verbessert, da das Archiv nicht nur eine rein chronologische Dokumentation von Dateisystemzuständen darstellt, sondern auch eine eigenständige Bearbeitung der Archivdaten anhand der zusätzlichen Attribute ermöglicht.

Entsprechend der verfügbaren Zusatzinformationen wäre es sogar möglich, weitergehende Informationssysteme (World Wide Web, Gopher und ftp) automatisch mit Daten zu versorgen.

Die Bereitstellung der zusätzlichen Attribute über die von dem jeweiligen Dateisystem verwendeten Attribute hinaus, stellt allerdings noch ein Problem dar. Bis jetzt gibt es nur wenige Verfahren, die aus allgemeinen Rohdaten sinnvolle zusätzliche Aussagen gewinnen können. Als Beispiele können hier die Gewinnung von Indexdateien für die Volltextsuche oder die Bestimmung des Dateityps (Unix *file* Kommando) genannt werden. Grundsätzlich gilt: Nur solche Informationen sind zusätzlich ablegbar, deren spätere Verwendung absehbar ist. So kann man nicht erwarten, daß Bilddaten auch eine textuelle Beschreibung besitzen, um nach bestimmten Objekten suchen zu können [Mey91].

Ein weiterer Vorteil in der Betrachtung von Dateisystemobjekten als Attributmengen ist die Möglichkeit der Darstellung einer Datenmenge in verschiedenen Repräsentationen. Sie erlaubt eine kontinuierliche Anpassung der Daten an die sie verarbeitenden Anwendungen. Das gilt für neuere Softwareversionen mit neuer Funktionalität.

3.4.2.1.2 Hierarchischer Namensraum für Dateisystemobjekte

Nachdem es möglich ist, Dateisystemobjekte als Attributmengen darzustellen, stellt sich nun die Frage der Adressierung einzelner Attributmengen.

Die übliche Vorgehensweise ist die Trennung von Namensraum und Speicherung. Dieser Ansatz erlaubt eine saubere Strukturierung der Software und befreit von den Beschränkungen durch fest vorgegebene Namensräume.

Allerdings kommt es hier gerade bei Dateisystemdaten zur Trennung von den Daten und ihrer Strukturierung innerhalb des Namensraumes des Dateisystems. Es sollte das Ziel der Archivierung sein, gerade die Beziehung zwischen den Daten und ihrem Namensraum aufrechtzuerhalten. Das läßt sich zwar auch mit einer strengen Trennung

von Speicherung und Namensraum erfüllen. Es birgt aber die Gefahr, daß die ursprünglichen Namenskomponenten von den von ihnen beschriebenen Daten getrennt werden. Damit wird eine Rekonstruktion der Daten-/Namensraumsrelation erschwert. Dieses läuft aber der Forderung nach möglichst vollständiger Rekonstruktion eine Archivinhalts zuwider. Da aber auch die Strukturierung der Dateien innerhalb des Dateisystemnamensraums eine Bedeutung hat (Quelldateiverzeichnisse oder Anwendungsinstallationen), ist eine strikte Trennung von Nutzdaten und Namensraum nicht sinnvoll. Archive haben nur einen geringen Wert, wenn Daten- oder Namensrauminformationen nicht rekonstruiert werden können. Ziel sollte es sein, die auf einem Datenträger gespeicherten Informationen (meist eine Teilmenge aus einem Archiv) vollständig (Daten- und Namensraumkomponenten) rekonstruieren zu können.

Für die BackStage Archivierung muß ein Namensraumkonzept vorgesehen werden. Dieses wird schon bei der Speicherung der Daten benötigt. Hier verfolgt BackStage ein von den gewöhnlichen Mechanismen, wie sie auch im IEEE Mass Storage Reference Model und dessen Nachfolgern vorgeschlagen werden, abweichenden Ansatz. Die Aufnahme eines Namensraumes in die Struktur des Archivs verbietet allerdings in keiner Weise die Verwendung von alternativen Namensräumen (wie semihierarschische Strukturen [CS92]). Sie stellt vielmehr den elementaren Namensraum zur Benennung von Archivobjekten dar. Die Aufgabe eines Namensraumes ist die Übersetzung von einer Bezeichnung für ein Objekt in eine andere Bezeichnung. Handelt es sich bei der Bezeichnung nach der Übersetzung um einen Primärnamen, so ist ein Zugriff auf das entsprechende Datenobjekt möglich. Primärnamen stellen somit nur die elementaren Angaben dar, die für den Zugriff auf das Datenobjekt benötigt werden. Demnach kann es aufgrund der durch die Software aufgebauten Abstraktionen viele Ebenen von Namensübersetzungen geben. Aus diesem Grunde stellt es auch keine Einschränkung dar, innerhalb von BackStage ein bestimmtes Namensschema vorzugeben.

Es ist jederzeit möglich, zusätzlich zum primären Namensschema weitere Namensschemata zu verwenden. Die Daten der zusätzlichen Namenschemata werden dann allerdings nicht notwendigerweise zusammen mit den Datenobjekten gespeichert. Diese Einschränkung ist aber keine Verschlechterung gegenüber den herkömmlichen Strukturierungsmechanismen.

Aufgrund des Einsatzgebietes der BackStage–Archive für dateisystemartige Strukturen bietet sich für die Archive der heute bei den Dateisystemen verbreitete hierarchische Namensraum an. Backstage Archive verwenden als Namen Vektoren von Zeichenfolgen von prinzipiell unbegrenzter Länge. Für die Namensverwaltung ist es unerheblich, welche Interpretation diese Zeichenfolgen haben (ASCII, ISO8859-x oder andere Verfahren zur Interpretation von Binärdaten als Zeichensätze). Die einzige Strukturierungskomponente der Namen ist ein Vektor von Zeichenfolgen. Diese Struktur erlaubt eine weitgehende Abbildung der Namensräume heutiger Dateisysteme.

Die Interpretation der Namen innerhalb eines Archivs und ihrer Darstellung ist den Anwendungen überlassen. Um jedoch auch eine Interpretierbarkeit eines Archivs über die Lebensdauer einer Anwendung hinaus zu ermöglichen, ist es sinnvoll, die Art der Interpretation (das Datenformat) der Namensraumkomponenten als Attribut des Archivs zu hinterlegen. Häufig ergibt sich die Art der Interpretation aus der Art des archivierten Dateisystems (DOS, Unix, VMS oder andere). Dieses wird innerhalb eines Archives als Archivattribut hinterlegt. Mit Hilfe dieses Attributes ist es dann möglich, auf die korrekte Interpretation der Namenskomponenten zu schließen.

Die BackStage Archive stellen durch ihren hierarchisch organisierten Primärnamensraum die Basis zur Verwaltung hierarchisch organisierter Dateisysteme dar. Eine Interpretation wird nicht vorgegeben. Mit Interpretation sind hier nicht nur die Darstellung der Namensraumkomponenten gemeint, sondern auch alle zusätzlichen Semantiken, die in heutigen Dateisystemen über den Namensraum realisiert werden. Hierzu gehören auch die Zugriffskontrolle über den Namensraum wie sie in Unix durch Berechtigungen auf Verzeichnissen realisiert wird.

Zusätzlich zur Bereitstellung der Infrastruktur zur Speicherung werden auch Komponenten benötigt, die eine entsprechende Interpretation erlauben. Diese Komponenten übernehmen die Abbildung der Archivstruktur in die jeweilige Interpretation des archivierten Dateisystems. Durch das Attributkonzept können genügend Informationen hinterlegt werden, um eine spätere Interpretation (idealerweise auch ohne die ursprüngliche Anwendung) zu ermöglichen. Von der Nutzung dieser Mechanismen hängt es ab, inwieweit die Daten eines Archivs auch dann noch nutzbar sind, wenn die ursprüngliche Anwendung, die die Archivdaten erstellt hat, nicht mehr vorhanden ist.

Durch die Abbildung der verschiedenen Dateisysteme in die einheitliche Repräsentation von Attributmengen innerhalb eines hierarchischen Namensraumes ist es auch möglich, von verschiedenen Systemen auf die Daten zuzugreifen. Hierbei sind unter Umständen viele Anpassungen hinsichtlich der Darstellung von Namensraumkomponenten, Zugriffsrechten und Datenformaten zu beachten. Dieses ist umso aufwendiger, je leistungsfähiger das ursprüngliche Dateisystem war. Als Beispiel mag hier die Abbildung von “langen” Unix Dateinamen in die 8+3 Form bei DOS in *pcnfs* dienen. Solche Abbildungen sind nur unter Einschränkungen möglich. Dennoch wird gern in der Praxis von solchen Möglichkeiten Gebrauch gemacht, um die Vorteile der gemeinsamen Datenhaltung, größere Datenkonsistenz[4] zwischen den Rechnern und zentralem Backup der zentralen NFS-Server, nutzen zu können. Wegen der vielfältigen Systeme und der ständigen Weiterentwicklungen auf dem Sektor der Betriebssysteme wird man auch in Zukunft auf diese Abbildungen zurückgreifen müssen.

4. NFS verfügt über kein vollständiges Cachekohärenzprotokoll

3.4.2.2 Archivverwaltung

Die eigentliche Aufgabe des BackStage–Archivs ist die Speicherung von logischen Kopien von Teilmengen von Dateisysteminhalten über die Zeit. Demnach ist zusätzlich zur Verwaltung von Attributen der entsprechenden Dateisystemobjekte auch die Verwaltung der zeitlichen Entwicklung des Dateisystems nötigt. Sie stellt die zentrale Aufgabe des Archivs dar.

Innerhalb eines Dateisystems können zwei wesentliche Veränderungen vorgenommen werden. Einerseits ergeben sich durch Löschen und Erzeugen von Dateisystemobjekten Veränderungen innerhalb des Namensraums des Dateisystems. Andererseits können die gespeicherten Dateisystemobjekte selbst verändert werden. Beide Veränderungen müssen erfaßt und dokumentiert werden. Die Veränderungen an den Dateisystemobjekten wirken auf die sie darstellenden Attribute. Veränderungen innerhalb des Namensraumes lassen sich durch Vergleich mit dem vorherigen Zustand des Namensraumes feststellen. Bei diesen recht umfangreichen Vergleichen sind immer dynamische Effekte (Veränderungen während der Sicherungsphase) zu berücksichtigen. Um die Vergleiche zuverlässig durchführen zu können, ist es nötig, daß keine Veränderungen an den zu vergleichenden Archiven oder Dateisystemen auftreten. Dieses ist zwar selbstverständlich, ist aber in der Praxis selten aufrecht zu erhalten. Das gilt besonders dann, wenn die zu vergleichenden Datenmengen sehr umfangreich sind. Rechensysteme, die heutzutage in der Forschung eingesetzt werden, haben nicht selten Plattenkapazitäten von mehreren zig Gigabytes. Das Feststellen der Veränderungen im Dateisystem gegenüber dem Archiv und das Sichern des aktuellen Dateisystemzustands sind die beiden Bestandteile einer Archivaktualisierung. Ein Abgleich von Dateisystemen dieser Größe mit den Archiven, deren Daten meist auf langsameren, aber preiswerteren Datenträgern hinterlegt sind, ist aufgrund der gewaltigen Datenmengen und der beschränkten Transferkapazität der Geräte eine zeitaufwendige Operation. So benötigt allein die Datensicherung einer mittleren Installation nicht selten mehrere Stunden. Eine Einschränkung des Zugriffs während der Zeiten des Abgleich zwischen Dateisystem und Archiv kann nicht immer hingenommen werden.

Um die Beeinträchtigung des normalen Betriebes durch Archivierungsaufgaben und Datensicherungsmaßnahmen auf ein Minimum zu beschränken, werden verschiedene Optimierungen durchgeführt.
Die einfachste Archivierungsmethode wäre die komplette Sicherung des Dateisystemzustandes bei der Archivierung. Das Bestimmen der Veränderungen kann dann jederzeit nachträglich geschehen. Dieser relativ einfache Vorgang bei der Aktualisierung des Archivzustandes wird allerdings bei großen zu archivierenden Mengen mit viel Aufwand und Datentransferleistung sowie Speicherkapazität erkauft. Dieser Aufwand schlägt sich in Datenträgerkosten und Laufzeiten nieder. Außerdem ist der Dokumentationsschritt der Veränderungen nachträglich zu erbringen.
Normalerweise werden in einem Dateisystem nur geringe Veränderungen durchgeführt (Hinzufügen/Löschen neuer Dateisystemobjekte, Modifikation einiger Dateien).

Führt man sich diese Tatsache vor Augen, so wird auch eine wirkungsvolle Optimierung des Archivierungsvorganges offensichtlich. Um den neuen Dateisystemzustand rekonstruieren zu können, ist es ausreichend, nur die Veränderungen zu dokumentieren. Dieses Verfahren ist sehr verbreitet bei der Datensicherung (vollständige und inkrementelle Datensicherungen) und Versionsverwaltung (SCCS und RCS). Unter der Voraussetzung, daß einmal gesicherte Daten immer zugreifbar sind, ist diese Sicherung der Veränderungen ein sehr effizientes Verfahren. Probleme können allerdings bei der Rekonstruktion eines bestimmten Zustandes auftreten. Man muß, ausgehend von einer vollständigen Sicherung, alle Änderungen bis zu dem gewünschten Zustand nachvollziehen. Dieses kann im Einzelfall sehr aufwendig sein. Ebenso steigt der Aufwand umso mehr an, je mehr Archivierungsläufe gemacht werden. Es vergrößert sich die Distanz zwischen der ursprünglichen vollständigen Sicherung und dem aktuellen Zustand. Bei dem inkrementellen Verfahren muß ein Vergleich mit der vorhergehenden Version durchgeführt werden. Die Vorversion ist aus dem initialen Zustand mit Hilfe aller inzwischen erfolgten Änderungen aufzubauen. Dieses Verfahren ist also zwar sehr günstig für den belegten Speicherplatz, erfordert aber dennoch einen hohen Datentransferaufwand und viel Rechenzeit. Auch ist es nicht sinnvoll, jedesmal den gesamten Datenbestand nach etwaigen Änderungen zu durchsuchen. Normalerweise ergeben sich relativ wenig Änderungen.

Ein weiteres brauchbares Verfahren für die Archivierung kann die Eigenschaften der zu archivierenden Dateisysteme nutzen. So entstehen zum Beispiel beim Schreiben neuer Dateien in VMS oder NOS neue Dateisystemobjekte. Diese neuen Inkarnationen der Daten lassen sich unmittelbar als Hinweise für eine neue Version nutzen. Ebenso werden bei dem Unix-Betriebssystem die Zugriffs- und Modifikationszeiten der Dateisystemobjekte erfaßt. Setzt man voraus, daß diese Mechanismen zuverlässig sind, so kann man diese Attribute dazu verwenden, Unterschiede zwischen dem Archiv und dem Dateisystem festzustellen. Diese Vergleiche sind auch bei großen Dateisystemen einigermaßen schnell durchzuführen. Voraussetzung ist allerdings, daß es Attribute gibt, die eine Modifikation zuverlässig dokumentieren. Sind diese Attribute aufgrund der Dateisystemstruktur nicht verfügbar, so bleibt nur das Verfahren der vollständigen Sicherung aller zu archivierenden Daten.

3.4.2.3 Schnittstellen von verbreiteten Archivierungssystemen

Der Vorgang der Archivierung ist einerseits durch die Dokumentation eines bestimmten Dateisystemzustands und andererseits durch die Auslagerung von Daten auf langsamere und damit preiswertere Massenspeicher charakterisiert. Verfahren, die einen potentiell unendlich großen Massenspeicher durch Migrationsmechanismen realisieren, sind zwar sehr verbreitet, (Unitree, EMASS,...), erlauben aber selten die Dokumentation eines ausgezeichneten Zustandes oder die Speicherung bestimmter über die eigentliche Archivierung der Rohdaten hinausgehender Informationen. Weiterhin leiden diese Systeme darunter, daß sie bestimmten Einschränkungen hinsichtlich der möglichen Operationen unterliegen.

Die heutigen Systeme sind darauf ausgelegt, bestimmte, eingeschränkte Dateisystemsemantiken zu unterstützen. Sie stellen häufig die primären Dateisystemschnittstellen wie *AFS*, *NFS* und *ftp* zur Verfügung. Diese Einschränkungen liegen vermutlich auch an den zur Zeit noch am unmittelbaren Betrieb orientierten Notwendigkeiten. Sie sind weniger auf längerfristige Datenhaltung, sondern mehr auf die Bereitstellung der jetzigen Dateisystemfunktion mit großen Kapazitäten ausgerichtet. Für die Handhabung großer Datenmengen stehen jedoch noch kaum Verfahren zur Verfügung.

3.4.2.4 Attribute innerhalb eines Archivs

Archive sollen die zeitliche Entwicklung eines Dateisystemteilbereichs dokumentieren und die Rekonstruktion eines Dateisystemzustandes zu einem bestimmten Punkt unterstützen. Wichtig für diese Operationen sind für den Anwender nicht die eigentlichen Daten, sondern der Überblick über die zeitliche Entwicklung der Dateisystemobjekte. Diese Unterscheidung zwischen “wichtigen” und “unwichtigen” Daten zeigt, daß ein Archiv eine besondere Form der Dateisystemdaten verwaltet. Es sind bis auf wenigen Ausnahmen die Daten, die die eigentlichen Datenmengen beschreiben.

Hierzu gehören:

- Informationen über die Modifikationszeiten,
- Datenmengen,
- Datenformate,
- Zugriffsrechte,
- weitergehende Informationen.

Letztere geben an, welche Programme die archivierten Daten verarbeiten können, oder wie lange die Daten zu speichern sind. Alle diese Informationen werden benötigt, um Entscheidungen zu treffen, ob gewisse Teile des Archivs wieder in ein Dateisystem zur Nutzung überführt werden sollen oder ob die archivierten Daten gelöscht werden können. Diese, die eigentlichen Daten beschreibenden Daten (Metadaten), sind die von einem Archiv zu verwaltenden Daten. Nun gibt es aus der Gesamtmenge des Dateisystems nur eine Teilmenge von Daten, die für die Verwaltung innerhalb eines Archives sinnvoll nutzbar sind. Eine Verwaltung aller in einem Dateisystem vorhandenen Daten über einen längeren Zeitraum ist aufgrund von räumlichen und finanziellen Beschränkungen nicht durchführbar.

Um bestimmte Teilmengen der Attribute der Dateisystemobjekte leicht zugreifbar zu halten, werden diese auf Direktzugriffsspeichern hinterlegt. Diese Metadaten werden benötigt, um zu entscheiden, ob gelöscht oder restauriert werden soll.

Die große Menge der Attribute der Dateisystemobjekte läßt sich in vier Teilmengen einteilen. Diese Teilmengen sind:

- *versionsbestimmenden* Attribute,

- die *informativen* Attribute,
- die Archivattribute,
- die sonstigen Attribute.

Ein Dateisystemobjekt liefert normalerweise alle Attribute bis auf die Archivattribute, welche vom Archivsystem generiert und zur Verfügung gestellt werden. Diese Einteilung der Attribute ist nicht fest für die Lebensdauer des Archivs vorgegeben. So können sich die Anforderungen an die Verfügbarkeit bestimmter Attribute in der Archivdatenbasis (versionsbestimmende und informative Attribute) ändern. Solche Änderungen lassen sich jederzeit durchführen. Während die Löschung von Attributen aus der Archivdatenbasis direkt in der Archivdatenbasis geschieht, benötigt eine Erweiterung der Menge der in der Archivdatenbasis verfügbaren Attribute ein Einlesen des archivierten Datenbestandes. Dieser Vorgang ist mit dem Vorgang der Rekonstruktion der Archivdatenbasis aus dem Archivdatenbestand identisch.

3.4.2.4.1 Versionsbestimmende Attribute

Attribute, die es erlauben, auf Veränderungen an dem eigentlichen Dateisystemobjekt zu schließen, werden als *versionsbestimmende* Attribute bezeichnet. Versionsbestimmende Attribute stehen also in Relation zu anderen Attributen. Diese Modifikationsrelation erlaubt es im allgemeinen, den Archivierungsaufwand zu reduzieren, da nur geänderte Dateisystemobjekte wirklich abgesichert werden müssen. Durch die Einführung der versionsbestimmenden Attributen wird aber auch keine Einschränkung vorgenommen. Am Beispiel von Unix werden folgende Attribute als versionsbestimmend angesehen:

- Dateilänge (-> Datenveränderung),
- Modifikationszeit der Datei (-> Datenveränderung),
- Modifikationszeit der Inodeinformation (-> Attributveränderung).

Bei Veränderungen dieser Attribute muß von einer Veränderung des Dateisystemobjektes ausgegangen werden.
Eine Besonderheit stellt eine Veränderung der Modifikationszeit der Inode dar. Dieses Attribut wird immer dann aktualisiert, wenn Inodeinformationen verändert werden. Nun stellt die Inode innerhalb eines Unixsystems im wesentlichen die Verwaltungsinformationen (Metainformationen) für die jeweilige Datei dar. Hier sind auch alle Attribute einer Unixdatei bis auf den eigentlichen Datenanteil abgelegt. Die Modifikationszeit der Inodeinformation gibt demnach an, wann zuletzt irgendein Attribut der Unixdatei verändert wurde. Bei Änderungen von Attributen von geringem Datenumfang, wie es häufig bei Attributen der Fall ist, die Metainformationen darstellen, kann es sinnvoller erscheinen, nur diese Attribute zu sichern. Eine solche Vorgehensweise würde aber wieder einem Gesamtvergleich aller Attribute nahekommen. Zumindest bei großen Attributwerten sollte dieses in Hinblick auf eine effiziente Archivierung vermieden werden.

Eine Archivierung einzelner Attribute von Dateisystemobjekten ist nur im Einzelfall sinnvoll. Hier wird einerseits das Prinzip der möglichen vollständigen Rekonstruktion eines Dateisystemobjekts durchbrochen. Andererseits wird durch die Beschreibung partieller Dateisystemobjekte innerhalb des Archivs der Verwaltungsaufwand erhöht. Es ist nicht Aufgabe eines Archivs, alle Aspekte eines Dateisystems zu modellieren. Die Aussage dieser Attribute innerhalb eines Archivs ist nur von eingeschränktem Wert.

3.4.2.4.2 Informative Attribute

Die informativen Attribute stellen diejenigen Informationen dar, die so wichtig sind, daß sie von dem Benutzer herangezogen werden, um Entscheidungen über die innerhalb eines Archivs gespeicherten Daten zu treffen. Durch die mögliche Verwendung der informativen Attribute ist eine Speicherung innerhalb der direkt zugreifbaren Archivdatenbasis gerechtfertigt. Da eine solche Speicherung auch Systemressourcen (Speicherplatz) auf relativ teuren Direktzugriffsmedien benötigt, ist meistens ein Kompromiß zu schließen. Dieser wird zwischen den informativen Attributen und dem dafür benötigen Speicherplatz gefunden. Gerade bei großen Archiven können die Archivdatenbasen beträchtliche Größe erreichen. Je mehr Attribute zur Verfügung stehen, desto mehr Aussagen sind prinzipiell über die archivierten Daten zu machen. Zu den informativen Attributen gehören:

- Informationen über die letzte Zugriffszeit vor der Archivierung,
- Anzahl der belegten Dateisystemblöcke,
- andere für die eigentliche Archivierung sekundäre Dateisystemobjektattribute.

3.4.2.4.3 Archivattribute

Die Archivattribute beschreiben Eigenschaften der Dateisystemobjekte in Bezug auf das Archiv. Hierzu gehören Informationen wie die Versionsnummer eines Dateisystemobjekts oder auch die Verweise, wo dieses Dateisystemobjekt gesichert wurde. Die Archivattribute werden vom Archivsystem generiert und verwaltet.

3.4.2.4.4 Sonstige Attribute

Unter den sonstigen Attributen werden alle Attribute verstanden, die nicht zu den versionsbestimmenden oder informativen Attributen gerechnet werden, also in der Datenbasis des Archivsystem für den Direktzugriff geführt werden. Solche Attribute werden zwar immer mit dem Dateisystemobjekt zusammen abgespeichert. Einige dieser Attribute sind notwendig, um das Dateisystemobjekt vollständig rekonstruieren zu können (die Liste der belegten Datenblöcke bei partiell belegten Dateien innerhalb eines Unixdateisystems).

3.4.2.5 Archivstruktur

Ein Archiv baut sich aus mehreren Dateisystemzuständen auf. Die Veränderungen der Dateisystemzustände werden durch besondere Archivattribute als verschiedene Versionen markiert. Die primäre Identifikation eines Dateisystemobjektes ist sein Name innerhalb eines hierarchischen Namensraumes. Andere Identifikationsmechanismen, wie die Verwaltung aller Dateisystemobjektattribute innerhalb einer relationalen Datenbank, wären zwar denkbar, stellen aber nicht die Hauptanwendung der BackStage Archive dar. Sie sind somit nur sekundäre Namensräume.

Die Namen innerhalb eines BackStage Archivs bezeichnen aufgrund der zeitlichen Entwicklung eines Archivs nicht immer eindeutig ein Dateisystemobjekt. Jede Veränderung des Dateiinhalts eines Dateisystemobjekts führt zu einem neuen Archivobjekt unter demselben Namen innerhalb des Namensraumes des Dateisystems und damit auch innerhalb des Namensraumes des Archivs. Um diese verschiedenen Instanzen des Dateisystemobjekts innerhalb des Archivs handhaben zu können, werden diesen Instanzen *Versionen* zugeordnet. Diese *Versionen* erlauben eine eindeutige Identifizierung des archivierten Dateisystemobjekte. Die *Versionen* beziehen sich nur auf die bei dem Archivierungsläufen beobachteten Veränderungen. Sie haben nur indirekten Bezug zu den innerhalb eines Dateisystems auftretenden Versionen. Es ist vorstellbar, Dateisysteme dahingehend zu erweitern, daß alle *Versionen* archiviert werden können. VMS erlaubt die Speicherung mehrerer Versionen eines Dateisystemobjekts. Neben der Kennzeichnung unterschiedlicher Dateisystemobjektinhalte ist auch die Dokumentation von Löschungen eines Dateisystemobjektes von Bedeutung. Diese Information wird benötigt, um eine vollständige Rekonstruktion eines Dateisystemzustandes zu einem bestimmten Archivierungszeitpunkt zu ermöglichen.

3.4.2.6 Versionsverwaltung von Dateisystemobjekten

Eine Version innerhalb eines Backstage-Archivs hat zwei Bestandteile:

- Inkarnationszähler,
- Versionsnummer.

Diese Zweiteilung ist nötig, um Mehrdeutigkeiten zu vermeiden. Diese Mehrdeutigkeiten entstehen dadurch, daß auf alle innerhalb eines BackStage-Archivs gespeicherten Dateisystemobjekte ausschließlich über den Namen zugegriffen wird. Ein Dateisystemobjekt kann aber verschiedene Zustände haben. Diese Zustände sind bestimmt vom Typ des Dateisystemobjektes. Immer, wenn sich der Typ ändert, dann wird der Inkarnationszähler erhöht. Dieses passiert auch, wenn ein Dateisystemobjekt als gelöscht markiert wird. Immer, wenn sich der Inhalt eines Dateisystemobjekts ändert, erkennbar an den Veränderungen an einem *versionsbestimmenden* Attribut, dann wird die Versionsnummer erhöht.

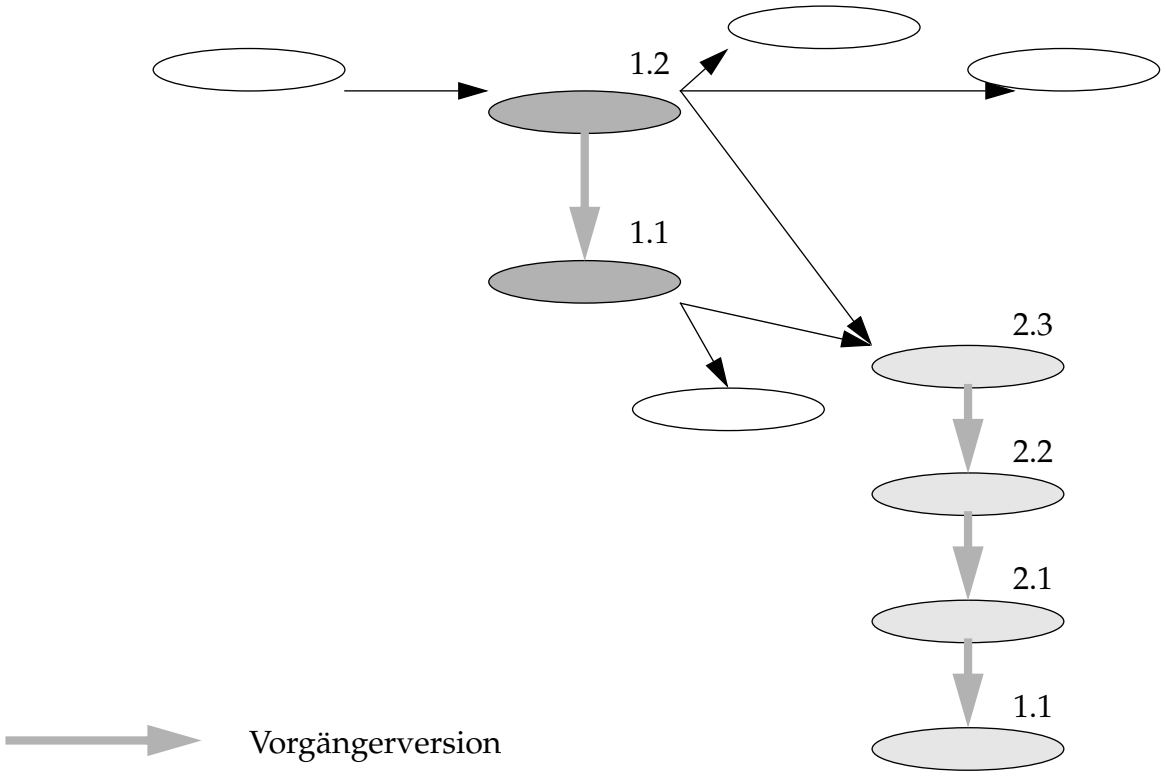


Abb. 3.10 Versionstiefer Dateibaum

3.4.2.7 Archivmanager - Struktur

Ein Archiv wird durch einen Archivmanager realisiert. Dieser Prozeß übernimmt die Verwaltung aller Archivoperationen, sowie die Realisierung der für Archive beschriebenen Operationen. Hauptaufgaben sind die Speicherung der Archivdatenbasis und die Durchführung der eigentlichen Archivierungs- und Restaurierungsoperationen. Der Archivmanager untergliedert sich:

- Die Archivdatenbasis übernimmt die Verwaltung der versionstiefen Dateibäume.
- Die Archivverwaltung stellt die Transaktionssteuerung der Archivoperationen dar.
- Die eigentliche Archivierungsarbeit, das Packen und Entpacken der eigentlichen Dateisystemdaten, wird von dem Packer übernommen.

Der Packer ist gewöhnlich eine ausgelagerte Komponente und wird lokal auf dem Rechner, zu dem das zu bearbeitende Dateisystem gehört, ausgeführt. Der Packer selbst muß den Schnittstellen zweier Systeme genügen. Die Daten des zu archivierenden Dateisystems müssen möglichst exakt rekonstruierbar sein. Aus diesem Grunde muß der Packer die entsprechenden Dateisystemsemantiken weitestgehend darstellen und rekonstruieren können. Da der Packer aber auch Bestandteil des Archivsystems ist, muß die archivierte Dateisystemstruktur aber auch dem Archivmanager in der ab-

strakten Form der Attributmengen übermittelt werden. Wegen dieser beiden Schnittstellen stellt der Packer das Bindeglied zwischen der abstrakten BackStage Archivwelt und den konkreten Dateisystemimplementierungen dar.

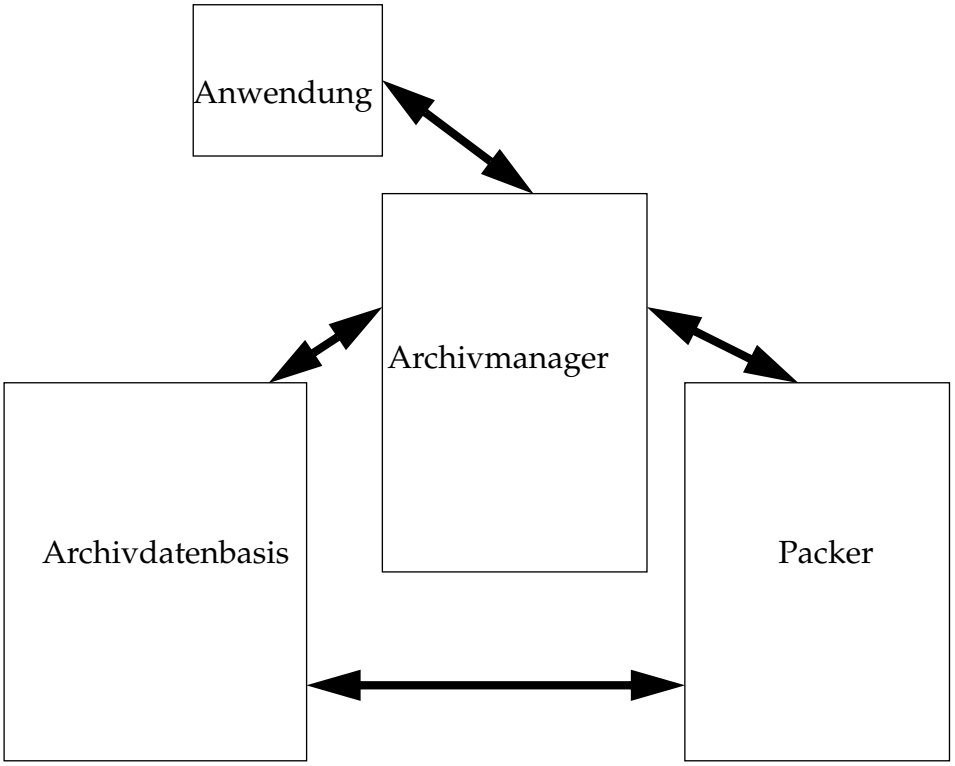


Abb. 3.11 Archivmanagerstruktur

3.4.2.7.1 Packer/Entpacker

Der Packer des BackStage Systems stellt die Schnittstelle zwischen den jeweiligen Dateisystemen und dem BackStage System dar. Er hat die Aufgabe, Dateisystemdaten einerseits in BackStage zu einer einheitlichen (zumindest dokumentierten) Form zu bündeln. Andererseits müssen durch den Packer gebündelte Dateisystemdaten wieder in dem entsprechenden Dateisystem rekonstruierbar sein.

Gesteuert wird der Packer über die Schnittstelle zum Archivsystem. Wegen der großen Abhängigkeit des Packers von dem zugrundeliegenden Dateisystem ist der Packer primär als Bestandteil der Dateisystemimplementierung zu betrachten. Er leistet für den Archivmanager neben der eigentlichen Pack- und Entpackoperation auch die Umwandlung der Dateisystemstrukturen in die dateisystemunabhängige Objektsicht.

3.4.2.7.2 Packen und Entpacken

Hauptaufgabe des BackStage Systems ist es, Daten in einer Form zu speichern, die es erlaubt, die Daten langfristig interpretieren zu können. Die Interpretierbarkeitsanforderung führt dazu, daß der Packer ein einheitliches Datenformat schreibt, das an Objekte, bestehend aus Attributmengen, angelehnt ist. Ein Packer baut also den sequen-

tiellen strukturieren Datenstrom als Folge von Attributmengen auf. Auf dieser Ebene ist es möglich, den Datenstrom zu interpretieren. Packer und Entpacker stellen sehr dateisystemspezifische Softwarekonstrukte dar. Dennoch sollten sie alle einer einheitlichen Grundstruktur folgen. Dadurch wird es möglich, die von einem Packer erzeugten Datenströme bis zum Dateisystemobjektlevel hin zu interpretieren.

Ob ein bestimmter Entpacker dann in der Lage ist, nicht von ihm selbst erzeugte Dateisystemobjekte in dem jeweiligen Dateisystem zu rekonstruieren, hängt von der Mächtigkeit des Dateisystems und der Funktionalität des Entpackers ab. In jedem Fall muß ein Entpacker in der Lage sein, die von dem dazugehörigen Packer erzeugten externen Repräsentationen der Dateisystemobjekte im Dateisystem wieder rekonstruieren zu können.

Die Standardisierung des sequentiellen Datenstroms bis auf Dateisystemobjektebene soll es erleichtern, auch systemübergreifenden Datenaustauch vorzunehmen. Diese Forderung wird nicht nur aus Gründen der möglichen Interoperabilität verschiedener Systeme erhoben, sondern auch zur Sicherstellung einer möglichen späteren Interpretation durch neuere Systeme. Weiterhin soll das Datenformat des sequentiellen Dateistroms so gewählt werden, daß ein Überlesen nicht interpretierter Daten zuverlässig möglich ist. Eine weitere wünschenswerte Eigenschaft wäre, wenn die Daten innerhalb der Dateisystemobjekte so abgelegt wären, daß die Dateisystemobjektgrenzen innerhalb des sequentiellen Datenstroms auffindbar sind (Synchronisation und Datenkapselung). Die Realisierung dieser Forderungen würde es erlauben, Dateisystemobjekte aus einem partiellen Datenstrom zu extrahieren. Dieses würde die Verlustmenge begrenzen können. Eine solche Kodierung hätte allerdings einen Aufwand. Alle Nutzdaten müßten noch einmal kodiert werden. Andererseits wird hierdurch eine saubere Trennung zwischen Metadaten und Nutzdaten innerhalb des sequentiellen Datenstroms erreicht. Für die Kodierung bieten sich hier viele der bekannten Verfahren an. Durch die Trennung von Nutz- und Metadaten und Strukturierung der Dateisystemobjekte auf Attributebene wird erreicht, daß alle Entpacker ein Datenformat interpretieren können.

Es wird nicht möglich sein, alle vorkommenden Dateisystemobjekte zu rekonstruieren. Dieses stellt aber keine wesentliche Einschränkung dar, da die lokal verwendeten Dateisystemobjekte in jedem Falle rekonstruiert werden können. Alle anderen Dateisystemobjekte sind nur dann rekonstruierbar, wenn eine entsprechende Abbildung in die lokal vorhandene Dateisystemwelt stattgefunden hat.

Nicht unterstützte Dateisystemattribute sind erkennbar. Deshalb ist es auch möglich, das System dahingehend zu erweitern, und nachträglich auch fremde Dateisystemobjekte zu unterstützen, sobald eine geeignete Abbildung gefunden ist. Die Aufgabe des Packers ist die Überführung einer Dateisystemstruktur in einen sequentiellen Datenstrom. Nutz- und Metadaten werden voneinander durch geeignete Kodierung getrennt. Der Entpacker überführt einen sequentiellen Datenstrom in eine Dateisystemstruktur.

Die Packer können im allgemeinen nicht jede mögliche Attributmengenkombination erstellen, sondern nur die für ihr Dateisystem relevanten Attributmengen. Ebenso können die Entpacker meistens nicht alle Attributmengen in eine lokale Dateisystemrepräsentation überführen. Es werden aber alle im lokalen Dateisystem vorkommenden Dateisystemobjekte unterstützt. Darüber hinaus ist es möglich, weitere Abbildungen von Attributmengen durchzuführen, sofern es dafür sinnvolle Darstellung innerhalb des lokalen Dateisystems gibt

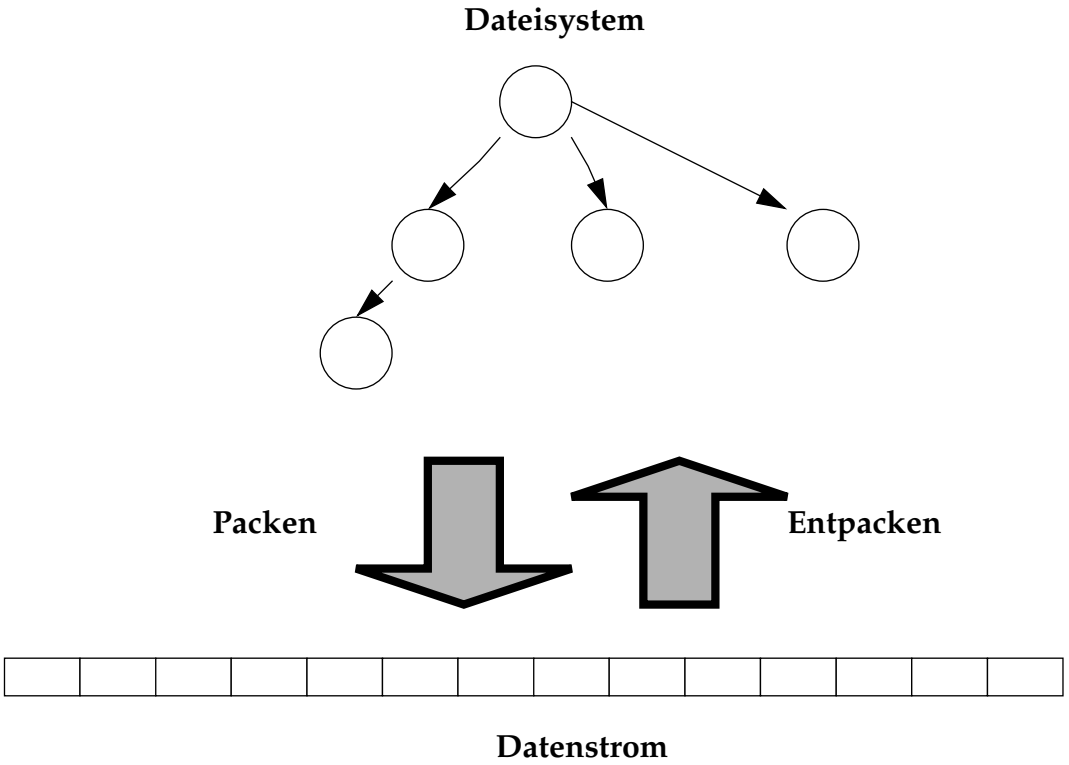


Abb. 3.12 Abbildung eines Dateisystems in einem sequentiellen Datenstrom

3.4.2.7.3 Archivdatenbasis

Die Datenbasis des Archivmanagers übernimmt die Verwaltung die Archivdaten, wie sie von den Packern erzeugt werden. Die Struktur der Datenbasis folgt dem für Archive beschriebenen Attributmengenkonzept.

Zusätzlich zu der reinen Verwaltung der versionstiefen Dateibäume unterstützt die Archivdatenbasis die transaktionsorientierte Aktualisierung der Archivdatenbasis. Die Archive können nur zu diskreten Zeitpunkten in Schritten aktualisiert werden. Wegen der Größe der Archive erstreckt sich ein Archivierungslauf häufig über eine längere Zeitspanne. Da der Archivierungslauf nach längerer Zeit abbrechen kann und die Archivdatenbasis auch die Transaktionssemantik des Archivierungslaufs unterstützen muß, werden die Archivdatenbasen in sogenannten *write-transactions* aktualisiert. Alle Einträge während einer solchen Transaktion werden miteinander verkettet, um ein späteres Aufräumen beim Abbruch der Transaktion zu ermöglichen.

Wird auf die Archivdatenbasis lesenderweise zugegriffen, so sind nur alle schon abgeschlossenen Datenbestände sichtbar, sowie die Daten zu den entsprechenden *write-transactions*. Es kann zu einer Zeit nur eine *write-transaction* geben. Dementsprechend einfach gestaltet sich die *abort*-Operation.

Die Struktur der Archivdatenbasis setzt sich aus folgenden Elementen zusammen:

- Archivverwaltungsinformation,
- *write-transaction* Daten,
- Archivattribute,
- Knoten des versions-tiefen Baums.

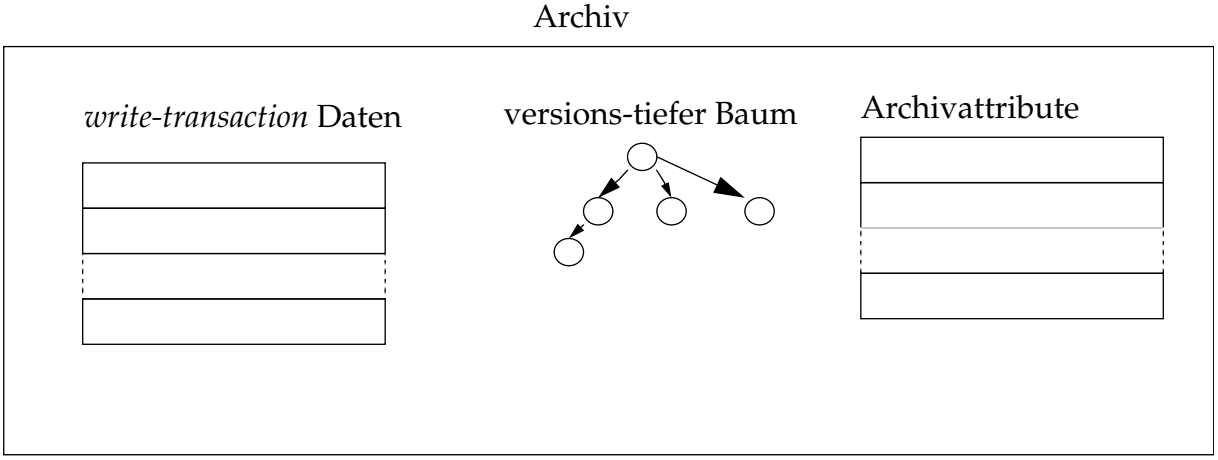


Abb. 3.13 Archivbestandteile

Als Operationen des Archivmanagers sind vorgesehen:

- Erzeugen eines Archivs,
- Löschen eines Archivs,
- Listen der vorhandenen Archive,
- Lesen der Archivattribute,
- Schreiben der Archivattribute,
- Durchführen einer *write-transaction*,
- Löschen einer *write-transaction*,
- Listen aller abgeschlossenen *write-transactionen*,
- Lesen der Attribute einer *write-transaction*,
- Durchführen eine Restaurierungs-Transaktion,
- Konfiguration.

3.4.2.8 Ablauf einer *write-transaction*

Eine *write-transaction* erhält zur Durchführung folgende Parameter. Die Transaktions-Identifikation des Knotens, auf dem das zu sichernde Dateisystem liegt, eine Dateiliste der zu sichernden Dateien. Alle anderen Daten werden normalerweise aus den Archivattributen bestimmt, können aber beim Auftrag angegeben werden, um so die Archivattribute zu übergehen. Während der Sicherung werden die Dateien vom Packer zusammengetragen und auf entsprechende *v-Dateien* im Volumesystem gesichert. Jede dieser *v-Dateien* entspricht im Archiv einer *location.* Um die einzelnen erzeugten, gepackten Dateien in erträglicher Größe zu halten, können mehrere *locations* pro Lauf erzeugt werden. Die Dateiliste gibt an, welche Dateien wie gesichert werden sollen. Zu jedem Dateinamen wird angegeben, mit welchem Redundanzgrad das entsprechende Dateisystemobjekt gesichert werden soll. Ein Redundanzgrad von 0 bedeutet, daß die Datei nur als vorhanden markiert wird, aber die Daten werden nicht gesichert. Ein Redundanzgrad von *unendlich* definiert, daß die Datei immer gesichert werden muß. Eine Datei wird immer dann gesichert, wenn sich ihre *versionsbestimmenden* Attribute verändern. Der Redundanzgrad bestimmt auf wievielen Locations die Datei mindesten gespeichert sein muß. Wenn mehr *write-transactions* vorhanden sind, als benötigt werden, um die entsprechenden Redundanzanforderungen zu erfüllen, dann können ganze *write-transaction*s mit ihren dazugehörigen *locations* gelöscht werden. Im Falle einer Restauration können die zusätzlichen *locations* einer Dateiversion verwendet werden, wenn eine *Location (= v-Datei)* mal nicht mehr zugreifbar sein sollte.

Der Ablauf einer Archiv *write-transaction* gestaltet sich folgendermaßen:

(1) Erteilung des Auftrags mit Transaktionsidentifikation des Packers des zu sichernden Dateisystem und der vollständigen Dateiliste.

(2) Bestimmung der zu schreibenden *v-Dateien* bei dem Volume-Manager, der dem Archiv zugeordnet ist.

(3) Sichern eines Segments des Dateisystems über den Volume-Manager mit dem Packer als Datenquelle (unter Umständen für mehrere *v-Dateien*).

(4) Transfer eines Segments der Dateisystemsicherung vom Packer auf die *v-Datei.*

(5) Kommunikation zwischen Packer und Archivmanager zur Bestimmung des Sicherungumfangs.

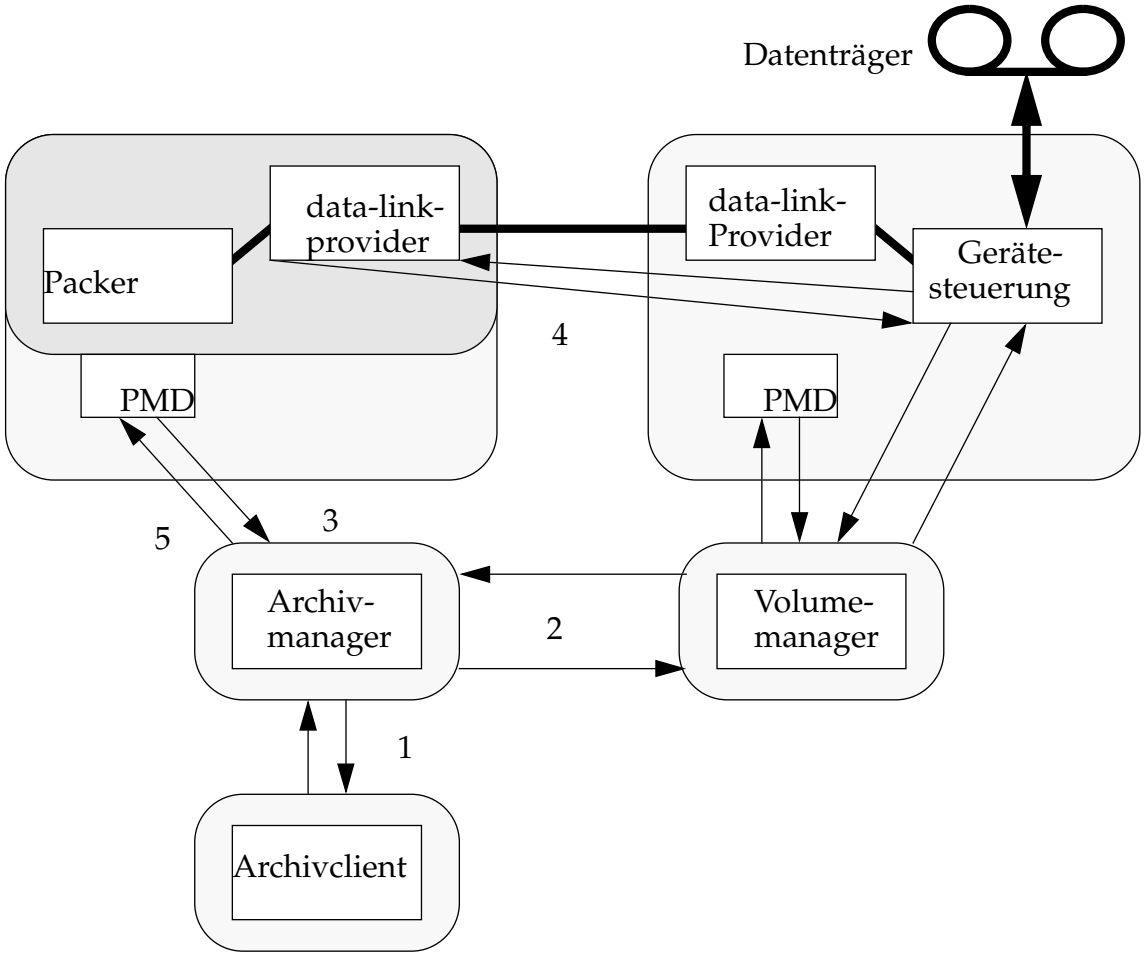


Abb. 3.14 Archivmanager Auftragsschema

3.4.2.9 Zugriffskontrolle

Da die Daten innerhalb eines Archivs von einem Dateisystem abgeleitet werden, das auch über Zugriffskontrollmechanismen verfügt, müssen diese Kontrollen auch innerhalb eines Archivs nachgebildet werden. Die BackStage-Archive verfügen über keine eigentliche Zugriffskontrolle. Sie verwenden vielmehr die Archivattribute, um das geeignete Zugriffskontrollverfahren (Unix-Rechte, Sicherheitsstufen, Zugriffskontrollisten) auszuwählen. Das entsprechende Verfahren wird mit einem Archivattribut ausgewählt. Damit die Zugriffskontrolle durchgeführt werden kann, müssen alle vom Dateisystem für die Zugriffskontrolle vorgesehenen Attribute als versionsbestimmende Attribute mit abgespeichert werden. Beim Zugriff auf das Archiv kann dann das gewählte Zugriffskontrollverfahren die Zugriffsberechtigungen anhand der BackStage-Identifikation und den verschiedenen angegebenen Authentisierungsdaten überprüfen. Mit diesem Verfahren ist gewährleistet, daß die Zugriffssemantiken der entsprechenden Dateisysteme erhalten bleiben. Das bedeutet, daß die Anwender mit dem Archivmanager kommunizieren können, um die Datenrestauration anzustoßen. Die Datenrestauration kann dann automatisch erfolgen.

3.4.2.10 Höhere Archivfunktionen

Neben den Basisarchivfunktionen gibt es noch weitere Operationen, die mit einem Archivmanager durchführbar sind. Um Archive transportieren zu können, müssen diese in eine austauschbare Form umgewandelt werden. Hierzu werden die Datenbestände so im Packerformat zusammenkopiert, daß alle Versionen der Datenbestände eines Archives, sowie eine Kopie der Archivattribute und -metadaten, enthalten sind. Der empfangende Archivmanager kann dann anhand der Packerdaten und der Archivattributen wieder die Datenbasis rekonstruieren.

Der Zugriff von Systemen, die nicht über die Dateisysteme verfügen, die in den Archiven gespeichert sind, ist ebenfalls möglich. Es kann immer auf die einzelnen Attribute zugegriffen werden. Die Abbildung von Archiven mit fremden zugrundeliegenden Dateisystemen ist dann Aufgabe der entsprechenden Schnittstellenprogramme. Diese Abbildungen können für alle oder auch nur für einen Teil der im Archiv verzeichneten Dateisystemobjekte existieren. Eine mögliche Abbildung ist die der Textdatei. Zumindest bei älteren Textdateien ist der Zeichensatzvorrat bekannt und kann mit wenigen Einschränkungen in die lokale Repräsentation überführt werden.

Die vielleicht wesentlichste Eigenschaft der BackStage-Archive liegt in den Attributen, die zu jedem Dateisystemobjekt gespeichert werden können. Hier können genügend Zusatzinformationen hinterlegt werden, um komplexere Aussagen über die im Archiv gespeicherten Daten zu machen. Informationen, die angeben, welche Anwendungen zur Verarbeitung der Daten benötigt werden, stellen hier wohl den wichtigsten Anteil dar. Hiermit läßt sich bestimmen, ob Daten noch verarbeitbar sind und was zu deren Verarbeitung an Anwendungen benötigt wird. Die Anwendungen selbst könnten dann entsprechende Umgebungsanforderungen definieren, die es erlauben, Rückschlüsse auf die einzusetzende Hardware zu ziehen. In letzter Konsequenz sind dann Aussagen über die Interpretierbarkeit von Daten innerhalb eines Archivs zu machen.

3.4.3 Backupmanager

Der BackStage-Backupmanager nutzt zur Durchführung seiner Aufgaben den Archivmanager und damit indirekt den Volume-Manager. Die Struktur des Backup-Manager entspricht der Grundstruktur aller BackStage-Komponenten. Die Aufgabe des Backup-Managers ist es nicht, während des laufenden Betriebs vollständig konsistente Sicherungen der Dateisysteme durchzuführen. Diese Forderung zu erfüllen, bedürfte starker Modifikationen innerhalb der Dateisysteme der betroffenen Rechner. Eine vollständig konsistente Sicherung würde erfordern, daß alle Dateisystemoperationen abgeschlossen sind. Die Sicherung umfaßt dann den Verzeichniszustand und die Daten in der Version, wie sie zu dem Sicherungszeitpunkt gültig sind. Eine Sicherung des Verzeichniszustandes ist noch innerhalb kurzer Zeiträume machbar, die Sicherung der Daten ist aber schon eine längerfristige Operation. Lösungsansätze zu diesem Problem wurden schon von der Firma Sun in ihrem *BackupCopilot* Produkt [Kol91] vorgestellt.

Hier wird der Dateisystemzustand eingefroren, solange die Verzeichnissicherung stattfindet. Eine Modifikation der Datenbereiche wird erst wieder zugelassen, wenn die Datenbereiche gesichert wurden. Dieses Verfahren erlaubt eine vollständige Sicherung während des Betriebes mit tolerierbaren Einschränkungen. Da diese Modifikationen aber für jedes Dateisystem gemacht werden müßte, wird dieser Ansatz im Allgemeinen von BackStage nicht verfolgt. Die mit dem BackStage-System erstellten Sicherungen sind immer noch nutzbar, zumal der größte Teil der Daten sich nur langsam verändert. Ein wesentlich besseres Verfahren zur Sicherung von Dateisystemzuständen wäre, wenn die Basisdateisysteme schon ein Versionskonzept unterstützen würden. Die Hauptaufgabe des BackStage-Backups wird auch primär in der Sicherung von Datenbeständen in einer vernetzten Umgebung gesehen.

Jedes automatische Backup-Verfahren hat das Problem, daß die Backup-Aktivität mit den laufenden Anwendungen abgestimmt werden muß. Nur die Anwendung selbst kann eine Aussage darüber machen, ob ihr im Dateisystem hinterlegter Datenbestand auch einem konsistenten Zutand entspricht. Diese Aussage wird bei konventionellen Backup-Verfahren dadurch erzwungen, daß die Systeme zum Backupzeitpunkt in den Einzelbenutzerbetrieb geschaltet werden. Damit ist gewährleistet, daß alle Anwendungen terminiert sind. Dabei wird davon ausgegangen, daß die Datenbestände der Anwendungen auch konsistent sind. Weil die Entwicklung sich aber immer mehr in Richtung 24 Stunden Dauerbetrieb bewegt, können diese Einzelbenutzerphasen nicht bei diesen Systemen eingesetzt werden. Andere Sicherungschemata müssen hier entwickelt werden. Ein mögliches Schema wäre, daß die Anwendung den für sie relevanten Sicherungspunkt erzeugt. Diese Datenbestände könnten dann mit dem normalen Backupverfahren gesichert werden. Die Anwendung könnte im Notfall von einem definierten Sicherungpunkt wieder aufsetzen.

3.4.3.1 Namensräume und Abbildungen

Wie schon erwähnt, ist BackStage-Backup primär für netzwerkweites Backup vorgesehen. Da das Backupsystem auch dem Benutzer zugänglich sein soll, muß es auch die Sichtweise des Benutzers unterstützen. Diese Sichtweise ist in vernetzten Systemen erheblich komplexer als bei Einzelsystemen. So gibt es zusätzlich zu den auf den einzelnen Systemen vorhandenen Dateisystemen auch noch über Netzwerk (NFS, AFS) verfügbare Dateisysteme. Idealerweise braucht sich ein Benutzer nicht darum zu kümmern, wo sich seine Daten befinden. Der Benutzer operiert allein mit Pfadnamen. Diese Pfadnamen können auf jedem Rechner unterschiedlich gestaltet sein, auch wenn sie letztendlich auf dasselbe Dateisystemobjekt zeigen. Die Abbildung des lokalen Namensraums eines Rechners auf die entsprechenden Dateisysteme geschieht in einer Art *mount*-Operation. Auch Systeme, die nicht unixartig strukturiert sind, besitzen äquivalente Operationen. Im wesentlichen wird unter einem Präfixpfad ein Dateisystem angebunden. Das Dateisystem kann dabei entweder lokal oder über das Netzwerk erreichbar sein. Der Namensraum wird gewöhnlich anhand von rechnerlokalen

Konfigurationsdaten aufgebaut. Fortschrittlichere Systeme verfügen über automatische Mechanismen, die die Dateisysteme bei Bedarf und anhand von netzwerkweit verfügbaren Tabellen einbinden.

Um nun in der Lage zu sein, dem Benutzer auch im Backup-Fall die vom vernetzten System gewohnte Sicht zu bieten, benötigt man die zeitliche Entwicklung der Anbindungsinformationen eines jeden beteiligten Rechners. Mit diesen Informationen ist es dann möglich, die eigentlichen Dateisysteme, die gesichert werden müssen, aufzufinden. Es brauchen nur die eigentlichen Dateisysteme und die jeweils gültigen Abbildungen der Netzwerksicht auf die Dateisysteme gesichert werden.

Bei einem Restaurationsauftrag kann dann anhand der Parameter Rechner, Zeitraum und verwendeter Pfad die korrekte Abbildung bestimmt werden und die entsprechende Sicherung des oder der betroffenen Dateisysteme aufgefunden werden.

Die Namensraumabbildungen werden umso aufwendiger, je uneinheitlicher die Struktur des vernetzten Systems ist. Wird jeder einzelne Rechner für sich alleine verwaltet, so kann die Namensraumabbildung für jeden Rechner unterschiedlich sein. Existieren jedoch Mechanismen zur zentralisierten Verwaltung ganzer Rechnergruppen, so daß auch die Namensraumabbildung auf diesen Rechnern gleichartig geschieht, so ist die Backup-Verwaltung erheblich vereinfacht. Es bleiben immer noch die Probleme, daß eventuell nicht alle Versionen der Namensraumabbildung erfaßt werden, da die Backup-Operation nur zu bestimmten Zeitpunkten läuft und damit gewöhnlich nur zeitliche Ausschnitte der Dateisystemzustände sichern kann. Für den normalen Betrieb ist diese Art der Sicherung aber ausreichend.

3.4.3.2 Ablauf und Archivmanager-Interaktion

Backup ist eine regelmäßige automatisierte Sicherung von Dateisystemen zum Schutz vor unerwarteten Verlusten durch technische Probleme. Backup wird meistens für eine administrativ zusammenhängende Menge von Rechnern durchgeführt. Dadurch läßt sich auch begründen, warum die Aktivität zentralisiert durchgeführt wird. Der Ablauf eines Backuplaufs hat folgende Struktur:

- Zu sichernde Bereiche werden bestimmt.
- Die jeweiligen Dateilisten für die Sicherung werden erzeugt.
- Die Archivierungsaufträge werden vergeben.
- Die Ergebnisse der Archivierungsaufträge werden ausgewertet.

Dieser Ablauf wiederholt sich periodisch. Dabei kann es vorkommen, daß bestimmte Teilbereiche nicht vollständig durchführbar sind, weil die beteiligten Rechner oder Netze nicht betriebsbereit sind. Diese Ausfälle werden durch das Transaktionssystem in BackStage abgefangen. Es fällt auf, daß das BackupSystem Archivierungsaufträge erstellt. Obwohl für Backup normalerweise sehr spezialisierte Werkzeuge verwendet werden, nutzt BackStage das eigene Archivsystem. Das Archivsystem enthält alle we-

sentlichen Mechanismen, um die Anforderungen, die an ein Backupsystem zu stellen sind, zu erfüllen. Die Archive erlauben vollständige und inkrementelle Sicherungen. Im normalen Betrieb werden inkrementelle Sicherungen, die nur die geänderten Datenbestände sichern, bevorzugt, da der Zeit- und Platzaufwand erheblich geringer ist. Diese Vorgehensweise wird von den Archiven dadurch unterstützt, daß sie unterschiedliche Versionen einer Datei immer sichern. Weil die Archive auch die Namensraumgeschichte durch die Speicherung einer Löschungsinformation unterstützen, ist es immer möglich, den Dateisystemzustand zu einem bestimmten Zeitpunkt zu rekonstruieren. Mit dem Redundanzmechanismus kann man auch bestimmen, wann alte Sicherungen (entsprechend einer *write-transaction* im Archiv) gelöscht werden können, ohne daß die vollständige Sicherung gefährdet ist. Archive erlauben eine vollständige Sicht auf die gesicherten Dateisysteme, auch wenn sie aus vollständigen und inkrementellen Sicherungsläufen zusammengesetzt sind.

Die Reihenfolge, wie welche Dateisysteme gesichert werden sollen, hängt von den lokalen Anforderungen ab. Der Ablauf ist, wie bei anderen gängigen Systemen auch, tabellengesteuert. Der Ablauf selbst wird mit Hilfe des Transaktionssystems durchgeführt. Die Reihenfolgeplanung geschieht über die Vorgabe der partiellen Ordnung für den Transaktionsablauf.

3.4.3.3 Datenrestauration

Die Datenrestauration aus dem Backupsystem geschieht mit Hilfe des Archivmanagers. Das Backupsystem hat lediglich die Aufgabe, die Zuordnung des gewünschen Pfades von einem Rechner aus gesehen zu einem bestimmten Zeitpunk oder Zeitraum auf das betroffene Dateisystem zu leisten. Ist das Dateisystem bestimmt, so kann aus dem Archiv für dieses Dateisystem mit Hilfe des Archivmanagers die Datenrestauration erfolgen.

3.5 Zusammenfassung

In diesem Kapitel wurden die Basismechanismen und die Struktur des BackStage Systems beschrieben. Es wurde gezeigt, daß es mit wenigen mächtigen Konzepten möglich ist, eine leistungsfähige zukunftssichere Archivierungs- und Backupsoftware zu erstellen. Die wesenlichen Konzepte sind:

- Transaktionskonzept für Auftragsbearbeitung,
- Erweiterbare Aufrufschnittstelle,
- Flexible Betriebsmittelplanung,
- Modulare Authentisierungsmechanismen.

BackStage gliedert sich in drei Hauptkomponenten:

- Volume-Manager,
- Archiv-Manager,
- Backup-Manager.

Diese Softwaresysteme bauen aufeinander auf und kommunizieren untereinander mit Hilfe des BackStage-Transaktionskonzepts. Die Trennung der verschiedenen Ebenen erlaubt es, einzelne Komponenten zu ersetzen, solange die BackStage-Transaktionsschnittstelle erhalten wird. Damit ist es möglich, auch Systeme zu verwenden, wie sie von der IEEE Mass Storage Gruppe modelliert werden. Diese Systeme eignen sich besonders für die Ebene des Volume-Managers.

Die Darstellung eines Archivs als Menge von Attributmengen erlaubt es im Zusammenhang mit dem hierarchischen Namensraum, alle heutigen Dateisysteme abzubilden. Die Verfolgung der zeitlichen Entwicklung eines Dateisystemzustandes wird durch die Archivattribute mit Inkarnationsnummer und Versionsnummer ermöglicht.

Insgesamt zeigt das BackStage Projekt, daß eine leistungsfähige Archivierungssoftware heutzutage durchaus realisierbar ist. Es kann auch eine Form gefunden werden, die es erlaubt, zukünftige Entwicklungen im Datenhaltungsbereich zu unterstützen.

4 Dateisystemerweiterungen

Wie schon in den vorhergehenden Kapiteln dargelegt, reichen die von den verbreiteten Dateisystemen angebotenen Attribute selten aus, um längerfristige Datenhaltung zu unterstützen. Im folgenden Kapitel wird eine `Erweiterung` des Unix-Dateisystems beschrieben, das ein Speichern von `zusätzlichen Attributen` erlaubt. Die dafür notwendigen Änderungen haben sich als minimal herausgestellt und lassen sich auch auf andere Dateisystemimplementierungen übertragen.

Die Problematik der erweiterbaren Dateisysteme wird schon in der Arbeit von Jeffrey C. Mogul[Mog86] erörtert. Dort werden Dateiobjekte um Properties erweitert. In dieser Arbeit wird auch ein entsprechendes Dateisystem beschrieben.

Der folgende Text beschreibt eine einfache Erweiterung des existierenden Unixdateisystems gekoppelt mit `Kernstrategien`, die es erlauben, den Zugriff auf die Dateien unter Verwendung der Attribute zu kontrollieren.

4.1 Attributierte Dateien

Bei dem Unix-Dateisystem handelt es sich um ein hierarchisches Dateisystem. Es gibt Inhaltsverzeichnisse, Dateien, Gerätedateien und symbolische Verweise als "Objekte" im Dateisystem. Eine normale Datei ist eine uninterpretierte Sequenz von Bytes. Zu Dateien werden eine Reihe von Eigenschaften verwaltet, die mit dem *stat()* Systemaufruf[Bac86] ausgelesen werden können. Das sind:

- Eigner,
- Gruppe,
- Index des letzten Bytes,
- Anzahl belegter Blöcke,
- Gerätekennung,
- Zugriffsrechte,
- Typ.

Diese Eigenschaften werden in einer Struktur, der *Inode*, hinterlegt. Diese Struktur besitzt eine feste Größe und ist nicht elementar änderbar.

Häufig ist es wünschenswert, die Liste der Eigenschaften um die Versionsnummer oder des Datums der letzten Sicherung zu erweitern. Dieses ist im normalen Unix-System nicht möglich, außer, man führt Namenskonventionen ein. Die Erweiterung um Attribute soll es ermöglichen, die beschränkte Anzahl von Dateieigenschaften zu vergrößern. Unter Attributen soll ein Paar (Name, Wert) verstanden werden.

Im Rahmen einer Vorlesung/Übung wurde das UNIX-Dateisystem um die Vergabe von Attributen an die Datei erweitert.

Die Erweiterung umfaßt folgende Eigenschaften:

- Es gibt potentiell beliebig viele Attribute.
- Es werden alle Dateisystemoperationen und Dateitypen unterstützt.
- Jedes Attribut verhält sich wie eine normale Datei.
- Attribute sind nicht im Namensraum des Dateisystems verzeichnet, aber sie sind über die Datei, an die sie gebunden sind, erreichbar.

Zweck der Erweiterung der Dateien um frei definierbare Attribute ist es, die Mechanismen zu schaffen, die es erlauben, die erweiterten Archivsemantiken, wie in den vorherigen Kapiteln beschrieben, zu unterstützen.

Die Erweiterung der Dateisysteme um frei definierbare Attribute wurde auch in einem Projekt zur Entwicklung von neuen Softwareentwicklungswerkzeugen durchgeführt [Lam91]. Die daraus gewonnenen Erfahrungen zeigen die Mächtigkeit dieses Ansatzes.

4.2 Kernstrategien

Die Kernstrategien sind eine allgemeine Anwendung der attributierten Dateien. Bei Kernstrategien geht es um ein Verfahren, das schon in Hydra [WLH81] realisiert wurde: Die Trennung zwischen Mechanismus und Strategie. Das Unix-Dateisystem enthält viele vorgegebene Strategien, dazu gehören die Zugriffsrechteüberprüfung oder das Konzept des Dateityps.

Mit Kernstrategien wird es also möglich sein, alle Dateisystemoperationen unter Verwendung der bestehenden Dateisystemoperationen neu zu definieren. Man wird an jede Datei eine Menge von Strategien binden können. Diese Strategien erlauben es, weitere benötigte Dateisystemeigenschaften einzuführen.

Mit Hilfe der attributierten Dateien existiert ein Basismechanismus, mit dem man beliebige Strategien verwirklichen kann. So ist es beispielsweise möglich, eine andere Strategie für Zugriffsrechte installieren (*Frozen Files[Kar90] / AKBP-II 90).* Voraussetzung hierfür ist natürlich, daß die entsprechenden Attribute einer Datei nicht ohne weiteres von einem Benutzer änderbar sind.

Wesentliche Kriterien von Kernstrategien sind:

- Potentiell soll jede Dateisystemoperation überlagert werden können.
- Strategien können die Dateiattribute nutzen.
- Strategien sind an Dateien bindbar.

4.2.1 Ordnung von Kernstrategien

In der derzeitigen Implementation der attributierten Dateien wird die Ordnung der Überlagerung der Operationen durch einen Stapel (*stack*) vorgegeben. Bei Strategien, die disjunkte Operationen überlagern, hat die Wahl des Ordnungsmechanismusses keinen Einfluß auf das Ergebnis der Überlagerung. Bei Strategien, die dieselben Operationen überlagern, hängt das Endergebnis der Operation von der Reihenfolge der Überlagerung ab. Es sind die Operationen im allgemeinen nicht kommutativ.
Für spätere Anwendungen mag ein anderer Ordnungsmechanismus eventuell geeigneter sein.

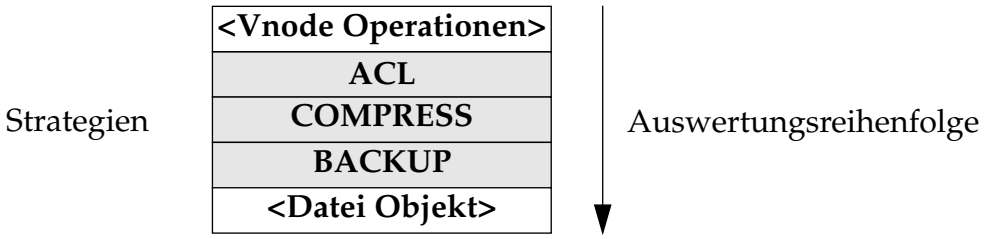


Abb. 4.1 Dateisystemobjekt mit angebundenen Strategien

4.2.2 Realisierung attributierter Dateien

Um die attributierten Dateien zu realisieren, mußten die Dateisystemimplementationen um den Dateityp der attributierten Datei erweitert werden. Diese "Datei" besteht aus zwei Teilen:

- der eigentlichen Datei,
- dem Inhaltsverzeichnis, in dem die Attributdateien verzeichnet sind.

.

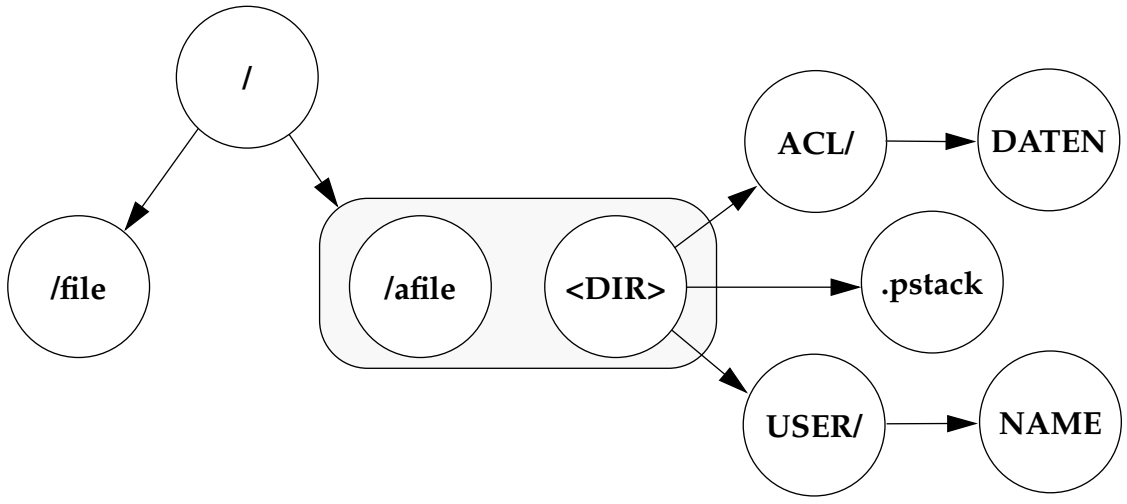


Abb. 4.2 Schema einer attributierten Datei

Der Zugriff auf die die Attribute realisierenden Dateien geschieht über neue Systemaufrufe oder implizit durch die mit den Attributen verknüpften Kernstrategien. Weil die Attribute als Dateien realisiert sind und in einem der Datei zugeordneten Verzeichnis hinterlegt sind, ist die mögliche Anzahl von Attributen für eine Datei nur durch die Beschränkungen des zugrundeliegenden Dateisystems begrenzt.

4.2.3 Änderungen am Unix-Kern

Es waren einige Änderungen in verschiedenen Teilen des Systems zu machen. Als Basis für die prototypische Implementierung diente SunOS 4.1.3. Innerhalb der Dateisystemimplementierung von SunOS4.1.3 gibt es eine Schnittstelle, die als *vnode*-Schnittstelle bezeichnet wird. Die *vnode*-Schnittstelle abstrahiert von der aktuellen Dateisystemimplementierung. Durch die *vnode*-Schnittstelle ist es möglich, innerhalb eines Systems verschiedene Dateisystemimplementierungen einzusetzen. Innerhalb von SunOS werden verschiedene Dateisystemtypen eingesetzt. Hierzu gehören:

- *ufs* (lokales Berkeley Fast File System),
- *nfs* (Network File System),
- *lofs* (Loopback File System),
- *specfs* (Gerätedateien),
- *tmpfs* (Dateisystem im virtuellen Speicher),
- *pcfs* (MSDOS Dateisystem).

Um eine möglichst homogene Erweiterung für Dateien mit zusätzlichen Attributen zu erhalten, wurden die 29 Operationen der *vnode*-Schnittstelle um drei zusätzliche Operationen ergänzt. Diese zusätzlichen Operationen erlauben das Eintragen von zusätzlichen Attributen bei Dateisystemobjekten. Die Erweiterungen der Dateisysteme um diese Operationen für zusätzliche Attribute sind geringfügig, da schon alle benötigten Mechanismen zur Verwaltung und Darstellung von Dateien existieren. Das Dateisystem muß nur um die drei Operationen zur Attributverwaltung und um den Typ der attributierten Datei erweitert werden. Es ist zu erwarten, daß auch in anderen Betriebssystemen diese Erweiterungen mit vertretbarem Aufwand durchführbar sind. Je modularer die entsprechenden Betriebssysteme gegliedert sind, desto leichter wird eine entsprechende Erweiterung ausfallen. Einfache Implementierungen können sich auf die vorhanden Mechanismen zur Dateiverwaltung stützen. Für eine effiziente Implementierung kann sich ein höherer Aufwand lohnen, da mit vielen kurzen Attributen zu rechnen ist. Für diese kurzen Attribute lohnt es sich sicher nicht, eine vollständige Datei zu verwalten. Bei einer solchen optimierten Realisierung der attributierten Dateien ist ein entsprechender Aufwand für die Realisierung der kurzen Attributwerte nötig.

4.2.3.1 Neue *vnode*-Operationen

Um attributierte Dateien realisieren zu können, muß ein Dateisystemobjekt mit den entsprechenden Attributen assoziiert werden können. Wegen der nicht festgelegten Anzahl von Attributen bietet sich hier ein Verzeichnis als Dateisystemstruktur an. Die entsprechende *vnode*-Operation VOP_ATFDIR liefert, sofern vorhanden, zu einer *vnode* das dazugehörige Verzeichnisobjekt für die Attribute. In diesem Verzeichnisobjekt können dann die entsprechenden Attribute hinterlegt werden.
Attributierte Dateien werden mit der VOP_ATFMAKE-Operation erstellt. Hierbei wird ein Attributverzeichnis an die entsprechende *vnode* gekoppelt und in der physischen Repräsentation des Dateisystems hinterlegt. Wird das Attributverzeichnis nicht mehr benötigt, so kann die Assoziation des Dateisystemobjekts mit seinem Attributverzeichnis mit der VOP_ATFDESTROY Operation wieder aufgehoben werden.
Diese drei eben beschriebenen Operationen erlauben die Verwaltung von zusätzlichen Attributen eines Dateisystemobjektes.

Neben der Verwaltung der zusätzlichen Attribute muß das Dateisystem eine attributierte Datei an der *vnode*-Schnittstelle als solche kenntlich machen, damit es möglich ist, diese Datei entsprechend mit den Kernstrategien zu behandeln.

4.2.3.2 Aufbau der Abstraktion der attributierten Datei

Bei den attributierten Dateien handelt es sich um eine allgemeine Dateisystemabstraktion. Die attributierten Dateien sind prinzipiell für jedes Dateisystem realisierbar. Demnach stellt die Realisierung der attributierten Datei, softwaretechnisch gesehen, eine Schicht über dem eigentlichen Dateisystem dar. Die Einbindung der Funktionalität der attributierten Dateien in die Systemaufrufschnittstelle geschieht über die Realisierung der Kernstrategien. Die Kernstrategien übernehmen die Bedienung der entsprechenden *vnode*-Operationen.

4.2.3.3 Kernstrategie-Systemaufrufe

Die Realisierung der Kernstrategien bringt vier neue Systemaufrufe mit sich, um die Verwendung der Kernstrategien zu ermöglichen. Die ersten drei Aufrufe stellen die Basisfunktionalität dar:

- Mit *pol_push(**Datei**, **Strategie**, **Schalter***) wird eine weitere Strategie an das angegebene Dateiobjekt gekoppelt.
- Die Operation *pol_pop(**Datei**, **Strategie**, **Schalter***) macht die vorherige Operation wieder rückgängig.
- Die an ein Dateisystemobjekt gekoppelten Strategien lassen sich aus einer Datei lesen, die mit der Operation *pol_open(**Datei**)* eröffnet wird.
- Die Operation *pol_ctl(**Datei**, **Strategie**, **Kommando**, **Argument**)* erlaubt es, strategiespezifische Operationen anzustoßen.

4.2.3.4 Strategieimplementierungen

Es wurden mehrere verschiedene Strategien realisiert, um die Mächtigkeit der Erweiterungen zu überprüfen. Zu den realisierten Strategien gehören:

- Benutzerattribute,
- Zugriffkontrollisten,
- TAR-Dateisystem,
- Bindung von Programmen an Daten
- versionsbehaftete Dateien.

Die verschiedenen Strategien benötigen alle ihre eigenen Attributmengen. Damit keine Kollisionen auftreten, werden die Attribute der einzelnen Strategien in eigenen Verzeichnissen unterhalb des mit der *vnode* assoziierten Attribut-Verzeichnisses abgelegt.

O*pen* und *unlink*-Operationen wurden zur Verfügung gestellt, um die benutzerdefinierten Attribute zu realisieren. Die eigentliche Dateibehandlung wird mit den normalen Mechanismen über den Dateideskriptor durchgeführt. Weil die Attribute nicht zum normalen Namensraum des Dateisystems gehören, müssen deswegen die *attr_open-* und *attr_unlink*-Operationen angeboten werden. Die Strategie für die benutzerdefinierten Attribute überlagert keine Dateioperationen. Sie stellt nur den Zugriff auf die benutzerdefinierten Attribute dar.

Die Realisierung der Zugriffskontrollisten gestaltete sich ebenfalls recht einfach. Es wurden die Operation zum Lesen, Setzen und Löschen von Zugriffskontrollisteneinträgen bereitgestellt. Die Zugriffsüberprüfungsroutine VOP_ACCESS wurde überlagert und wertet die Zugriffskontrollisteninformation aus. Die entsprechenden Zugriffskontrollisten werden als Attribute gespeichert.

Das TAR-Dateisystem stellt eine Datei im TAR-Format als Dateisystembaum dar. Die in der TAR-Datei befindlichen Dateien verhalten sich wie normale Dateien. Allerdings wird Schreibzugriff auf die Dateien nicht erlaubt. Diese Strategie simuliert demnach ein eigenes Dateisystem, dessen Daten in einer TAR-Datei verzeichnet sind.

Bei der Bindung von Programmen an Daten wurde der *exec*-Systemaufruf des Kerns dahingehend erweitert, daß ein über Attribute eingestelltes Programm beim Ausführen einer Datei aufgerufen wird. Diese Funktionalität entspricht in etwa dem Doppelklicken auf Datendateien in MacOS. Dieses führt zur Ausführung der mit der Datei assoziierten Anwendung.

Die Strategie der versionsbehafteten Dateien erlaubt es, bei jedem Öffnen einer Datei zum Schreiben eine neue Version dieser Datei anzulegen. Weitere Attribute der Versionspolicy geben an, wieviele Versionen gehalten werden dürfen.

Für die Unterstützung der attributierten Dateien mußten auch einige Systemprogramme erweitert werden. Das *fsck* -Programm ist vor allem betroffen. Es muß Dateisysteme mit attributierten Dateien unterstützen können.

Um den Umgang mit attributierten Dateien zu erleichtern, wurde das *ls*-Programm erweitert. Es muß attributierte Dateien markieren können. Es ist möglich, die mit den attributierten Dateien assoziierten Strategien auszugeben. Falls ein spezielles Ausgabeprogramm dafür vorhanden ist, werden die strategiespezifischen Informationen ausgegeben. Letztere Funktion wurde so implementiert, daß das *ls*-Programm strategiespezifische *ls*-Programme aufruft.

Insgesamt entspricht der Implementierungsaufwand für die zusätzlichen Strategien in etwa der Funktionalität der Strategien. Auch war die Trennung der Strategien von dem restlichen Kerncode recht deutlich. Sie mußte nur dann durchbrochen werden, wenn Eingriffe in Bereichen vorgenommen wurden, die nicht unmittelbar zur Dateisystemschnittstelle gehörten. Dieses war bei dem *exec*-Systemaufruf notwendig.

4.3 Besonderheiten bei der Implementierung

Während der Implementierung traten einige kleinere Besonderheiten auf.
Die unangenehmste Eigenschaft war, daß alle für Attribute erzeugten Verzeichnisse und Dateien natürlich den Quota- und Rechtebeschränkungen unterlagen. Diese Verhaltenweise ist an sich nicht weiter ungewöhnlich, hat aber doch dazu geführt, daß die *chown*-Operation jetzt rekursiv auch den Eigner aller Attribute ändern muß. Hier wäre eine von den normalen Dateisystemobjekten getrennte Implementierung für die Attribute wahrscheinlich problemloser gewesen. Eine solche Implementierung ist aber in der kurzen Projektzeit nicht durchführbar gewesen.

Bei der Realisierung der Zugriffskontrollisten ist aufgefallen, daß in den zugrundeliegenden Dateisystemen bei der Rechteüberprüfung manchmal Abkürzungen verwendet wurden, um Zugriffsrechte auf Verzeichnisse überprüfen zu können. Hier waren einige Korrekturarbeiten notwendig. Es war jedoch nur durch den noch nicht vollständig modularisierten Zustand der Softwarebasis begründet.

Alles in allem ließen sich die Erweiterungen mit vertretbarem Aufwand durchführen. Die neuen Kernstrategiemodule erreichen einen recht hohen Modularitätsgrad und damit eine gute Abkapselung gegenüber dem restlichen Unixsystem.

Eine nicht prototypische Implementierung der attributierten Dateien sollte einen anderen Ansatz zur Realisierung der Attributdateien wählen. Viele der Attributdateien sind sehr klein. Sie benötigen nicht die vollständige Dateisemantik in der Implementierung. Auch die Realisierung der Verzeichnisstruktur für Attribute muß sich nicht unbedingt auf die normalen Verzeichnisstrukturen des Dateisystems stützen.

4.4 Graphische Sichtweise von "Attributierten Dateien und Kernstrategien"

Die Verwendung von attributierten Dateien im Zusammenhang mit Kernstrategien verändert die Reihenfolge der Abarbeitung der Abstraktionen innerhalb des Betriebssystemkerns geringfügig. Im wesentlichen bleibt die Hierarchie *Systemaufruf->Systemaufrufimplementierung->Vnode-Schnittstelle->Dateisystemimplementierung* bestehen.

Abb. 4.3 Standardaufrufhierarchie in SunOS4.1.3

Die attributierten Dateien und Kernstrategien greifen an der *vnode*-Schnittstelle an. Sie iterieren den Aufruf über die *vnode*-Schnittstelle für alle auf den für dieses Dateisystemobjekt angegebenen Strategien. Hierdurch wird der gewünschte Überlagerungseffekt erreicht.

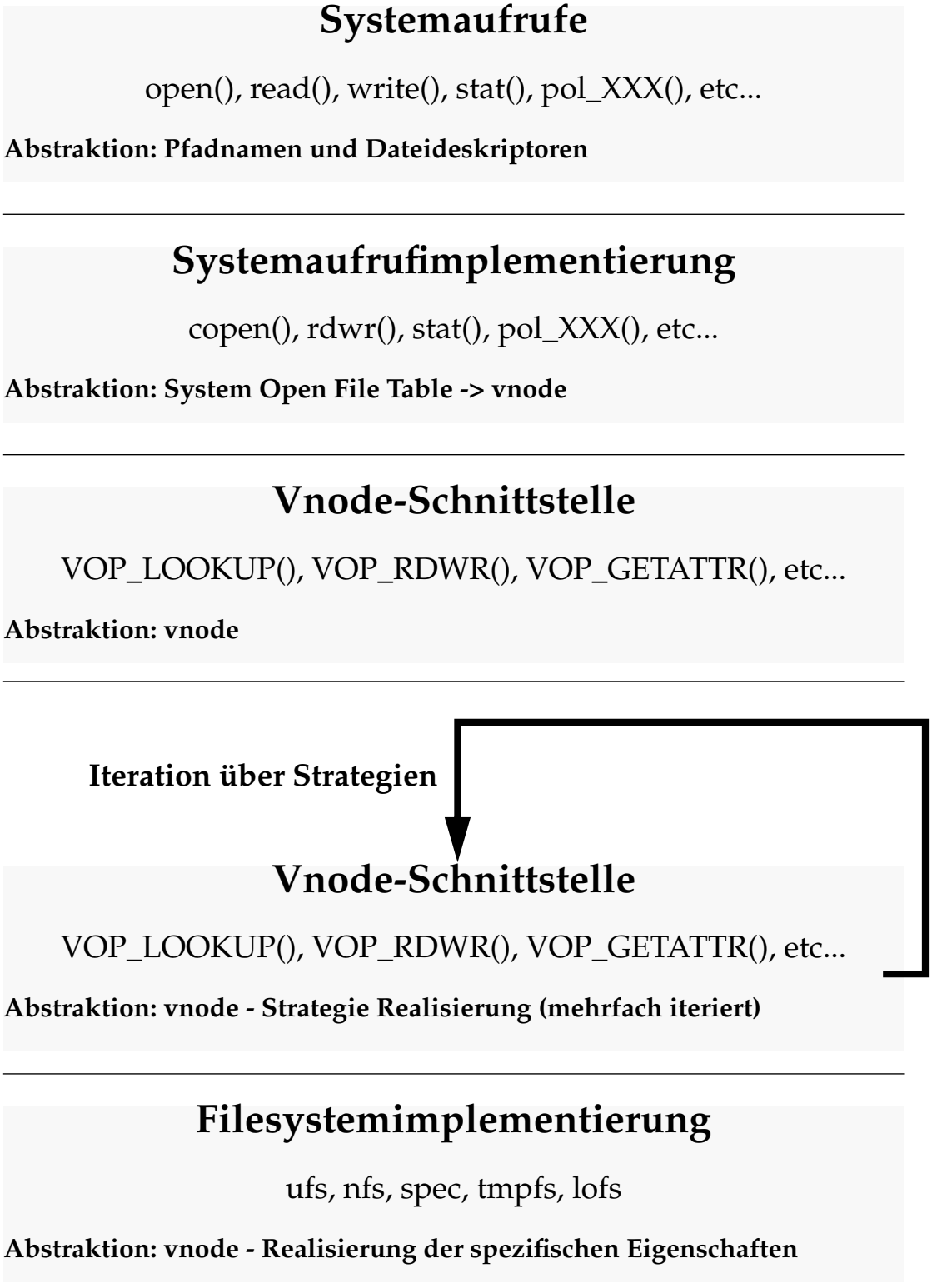


Abb. 4.4 Aufrufhierarchie der Kernstrategien einer attributierten Datei

4.5 Umstrukturierung des Dateisystems

Wenn man die attributierten Dateien und Kernstrategien weiter betrachtet, so stellt sich die Frage nach einer Umstrukturierung des Unix Dateisystems.

Mit Hilfe der Strategien lassen sich alle Dateitypen, die zur Zeit im Unix Dateisystem bekannt sind, sowie viele zukünftige Dateitypen, realisieren. Die einzigen Basistypen sind das Inhaltsverzeichnis und die einfache Datei. Diese Art der Erweiterung findet man auch in dem Vererbungskonzept moderner Programmiersprachen.

4.6 Mögliche Strategien

Folgende Tabelle zeigt existierende und mögliche zukünftige Strategien auf. Es gibt sicher noch mehr Anwendungen, besonders im Bereich der Datentypen für Multimedia und längerfristiger Datenhaltung.

Funktion	Name	Prototyp
Zusätzliche Dateiattribute “KOMMENTAR, AUTOR, ...”	USER	existiert
Zugriffskontrollisten	ACL	existiert
Unterstützung der Datensicherung	BACKUP	-
Repräsentation (tar, cpio Dateien als Dateibäume)	TAR, CPIO	existiert
Verbesserte Gerätebeschreibung	DEVICE	-
Statistiken	STAT	-
Implizite Netzwerkverbindungen	NETLINK	-
Komprimierte Speicherung	COMPRESS	-
Verschlüsselte Speicherung	CRYPT	-
Kontextabhängige Dateien (Architektur, Rechner, Konfiguration)	CONTEXT	-
Prozeßkommunikation (z.B. implizites *fork()/exec()*)	EXEC	existiert
Höhere Zugriffsmechanismen (ISAM, RECORD)	ISAM, RECORD	-
Versionskontrolle	VERSION	existiert

Tab. 4.1 Mögliche Kernstrategien auf der Basis von attributierten Dateien

4.6.1 Realisierung attributierter Dateien und Kernstrategien

Die Implementierung von attributierten Dateien und Kernstrategien wurde in zwei zweiwöchigen Intensivkursen mit je 16 Studenten durchgeführt. Obwohl diese Studenten noch keine Erfahrung mit der Realisierung von Betriebssystemsoftware hatten, war es möglich, dieses Projekt erfolgreich durchzuführen. Neben der Implementierung der Basismechanismen sind auch zahlreiche Kernstrategien entstanden.

4.6.2 *vnode*-Datenstruktur

Die *vnode*-Datenstruktur stellt eine abstrakte Unixdatei dar. Dementsprechend fallen auch die möglichen Operationen und Ausprägungen aus. So gibt es bei den Dateitypen nur die normalen Unixdateien, Inhaltsverzeichnisse, Gerätedateien, symbolische Verweise, Sockets und Pipes.
Die *vnode*-Datenstruktur enthält nur Informationen, die zur Darstellung einer abstrakten, allgemeinen Unixdatei nötig sind.

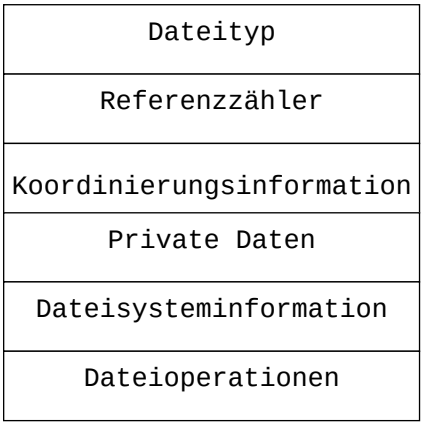


Abb. 4.5 Die *vnode* als abstrakter Dateityp

4.6.3 Datenfelder der *vnode*-Struktur

Die Datenfelder der *vnode*-Struktur enthalten nur die abstrakten Bestandteile zur Beschreibung einer allgemeinen Unix-Datei. Innerhalb der *vnode* gibt es 6 wesentliche Informationsfelder. Der Dateityp (VNON, VREG, VDIR, VBLK, VCHR, VLNK, VSOCK, VBAD, VFIFO) gibt die Art der Datei an. Die Typkennung wird zur Unterscheidung einiger weniger Spezialfälle herangezogen. Zu diesen Spezialfällen gehören Pfadnamenübersetzungen (VDIR und VLNK), spezielle Gerätedateien (VCHR und VBLK) und benannte interne Datenstrukturen (VFIFO und VSOCK). Diese Unterscheidungen wären nicht mehr nötig, wenn entsprechende Strategien verwendet würden. So ließen sich die Typen VCHR, VBLK, VFIFO und VSOCK mit einer allgemeinen Gerätestrategie sauber abbilden. Die Überprüfung, ob es sich um ein Verzeichnis handelt, könnte man sich sparen, wenn die VOP_LOOKUP-Operation auf allen Dateisystemobjekten erlaubt

wäre. Dateisystemobjekte ohne Unterstruktur könnten dann die VOP_LOOKUP Operation verweigern.
Der Referenzzähler wird verwendet, um zu bestimmen, wie lange die mit einer *vnode* assoziierten Daten gültig sind.
Koordinierungsinformationen werden für die Steuerung des Dateizugriffs verwendet.
Unter den privaten Daten werden die *vnode*-spezifischen Daten hinterlegt. Diese Daten werden nur von den dateisystemspezifischen Routinen interpretiert.
Der Zugriff auf die dateisystemspezifischen Operationen erfolgt über den Zeiger auf die Dateioperationen.

4.7 Interne Datenstrukturen

Attributierte Dateien sind als *atf_vnodes* realisiert. Jede *atf_vnode* hat einen Datenanteil, der auf drei Komponenten verweist. Zwei davon sind weitere *vnodes*. Es sind:

- die eigentliche Datei-*vnode,*
- die dazugehörige Verzeichnis-*vnode*.

Die dritte Komponente ist der Zeiger auf die Stapelelemente für die Kernstrategien. Als Operationen der *atf_vnode* sind die Operationen für attributierte Dateien mit Kernstrategien eingetragen. Diese Operationen arbeiten den Kernstrategienstapel ab, sofern hier eine Kernstrategie die entsprechende Operation überlagert hat. Die Realisierung der Kernstrategien stellt eine eigene kleine Dateisystemimplementierung dar. Das Besondere an dieser Dateisystemimplementierung ist, daß sie keine eigenen Dateisystemobjekte besitzt, sondern sich anderer Dateisysteme bedient. Insofern können alle Dateisysteme, die die Attributschnittstelle unterstützen, von den Kernstrategien profitieren. Das Dateisystem für die Kernstrategien liegt oberhalb der Basisdateisysteme.

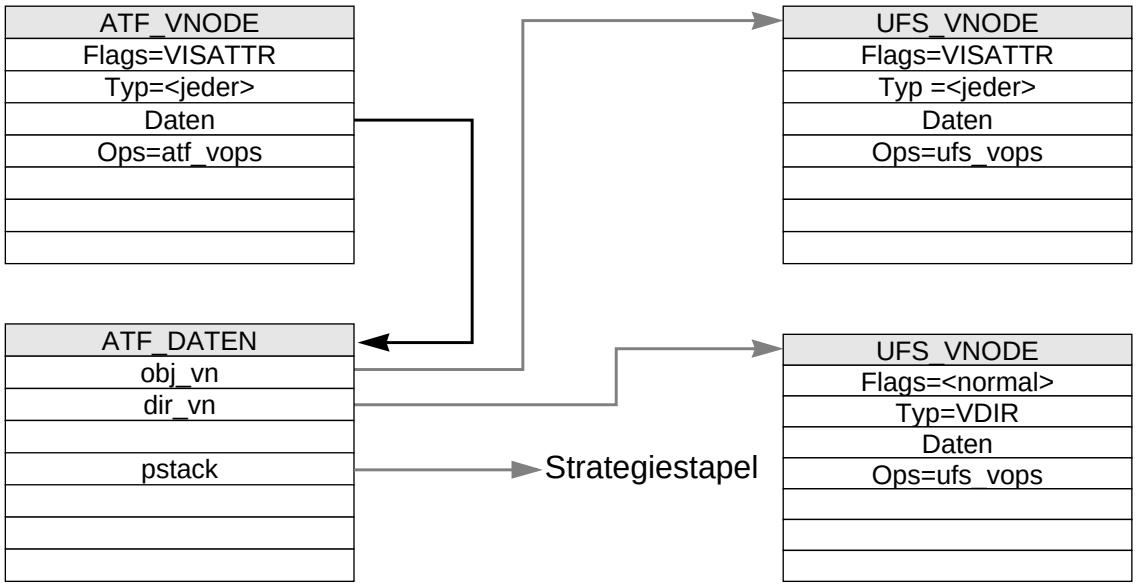


Abb. 4.6 Schema der Datenstrukturen für eine *atf_vnode*

Die in der *atf_vnode* referenzierten *vnodes* verweisen auf die entsprechenden Dateisystemobjekte innerhalb ihres Dateisystems. Diese müssen innerhalb des Dateisystems als zusammengehörig gekennzeichnet sein, damit sie zu der gewünschten *atf_vnode* aggregiert werden können. Im Falles der prototypischen *ufs*-Dateisystemimplementierung geschieht dieses durch eine Kennung im Modusfeld und durch einen Verweis auf die korrespondierende Verzeichnis-*inode.*Die Datei, die die symbolischen Namen für die Kernstrategien enthält (.pstack), wird direkt in dem der attributierten Datei zugeordneten Verzeichnis hinterlegt. Attribute der eigentlichen Kernstrategien werden in Verzeichnissen hinterlegt, die den Namen der jeweiligen Kernstrategie tragen.

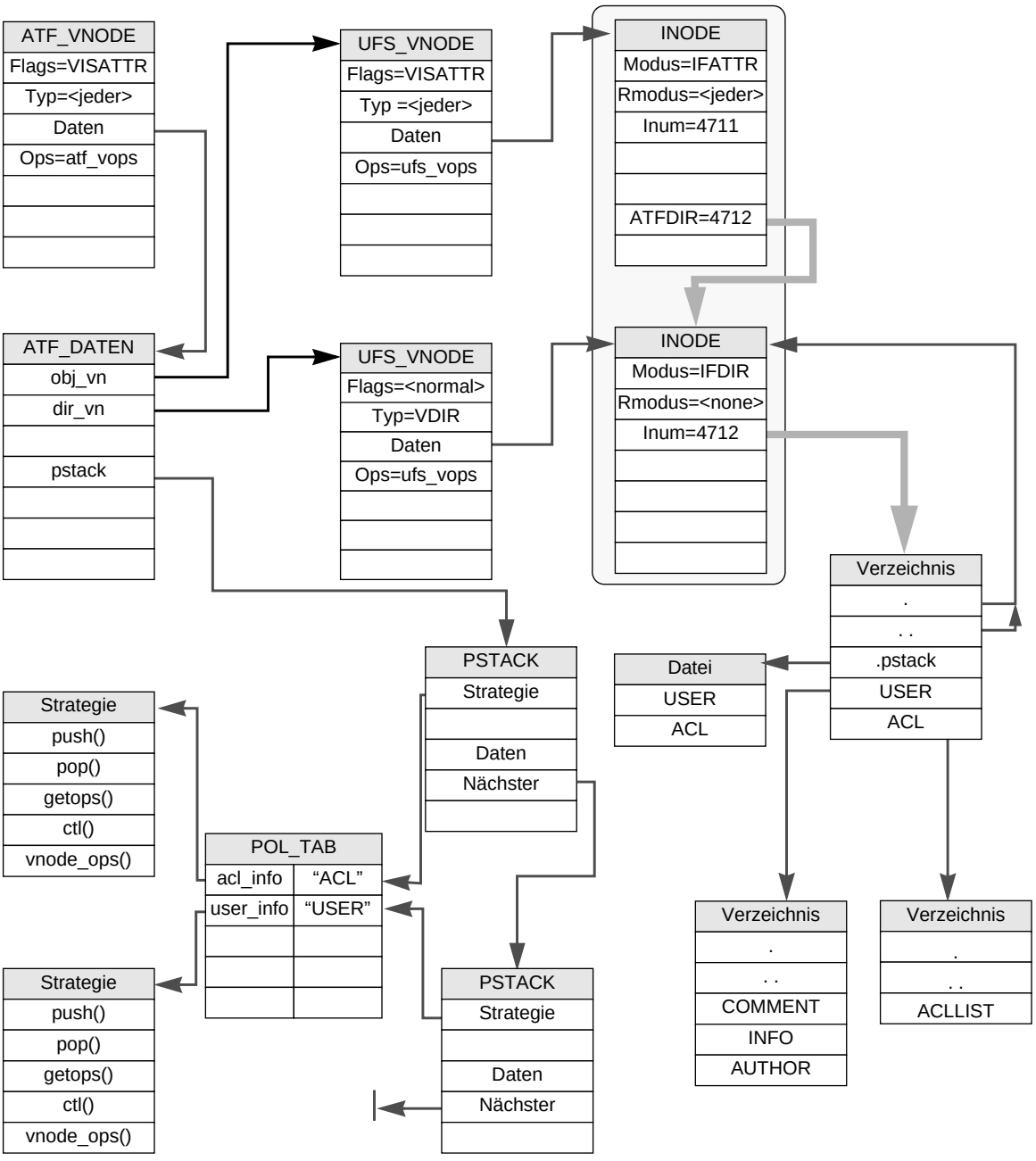


Abb. 4.7 Schema der Datenstrukturen für attributierte Dateien mit Kernstrategien

4.8 Aufbau einer *atf_vnode*

Zunächst ist es wichtig, daß die Abstraktion der attributierten Datei aufgebaut wird, sobald eine solche innerhalb eines Dateisystems angetroffen wird. In der *vnode*-Schnittstelle gibt es genau zwei Operationen (VOP_CREATE, VOP_LOOKUP), die eine Pfadnamenskomponente in eine *vnode*-Referenz verwandeln. An dieser Stelle muß überprüft werden, ob es sich bei der *vnode*-Referenz um die eines attributierten Dateisystemobjekts handelt. Ist dieses der Fall, so wird eine neue *vnode*, die *atf_vnode*, aufgebaut. Sie ermöglicht die Realisierung der Kernstrategien. Für reine attributierte Dateien wäre dieses Vorgehen nicht notwendig, aber die zusätzlichen Kernstrategien erlauben größere Einsatzmöglichkeiten des Konzeptes der attributierten Dateien.

Immer, wenn eine *vnode* einer attributierten Datei aufgefunden wird, muß die entsprechende *vnode*, die die Kernstrategien realisiert, zurückgeliefert werden. Die Operationen der *atf_vnode* führen die *vnode*-Operationen entsprechend dem angegebenen Kernstrategienstapel aus. Um nicht die existierende *vnode*-Struktur für den Prototyp erweitern zu müssen, wird die zu einer *vnode* gehörige *atf_vnode* mit einer Hash-Tabelle aufgesucht. Weil die Übersetzung eines Pfadnamens immer zuerst die *vnode* der attributierten Datei liefert, aber die entsprechende Abstraktion gewährleistet sein muß, ist diese Übersetzung immer notwendig.

4.9 Aufbau einer attributierten Datei

Eine attributierte Datei erzeugt man, indem man sie mit der *pol_push()*-Operation an eine Kernstrategie bindet. Zu diesem Zeitpunkt werden alle damit verbundenen Datenstrukturen (zusätzliches Verzeichnis, Datei mit die Liste der Strategien) angelegt. Haben andere Prozesse die Datei schon vorher eröffnet, so arbeiten sie auf der ursprünglichen Referenz auf die Datei und stellen damit keine Veränderungen fest. Alle Prozesse, die später auf die attributierte Datei zugreifen, finden wegen der Übersetzung der attributierten *vnode* in eine *atf_vnode* die attributierte Datei vor.

4.10 Abbau einer *atf_vnode*

Die Deaktivierung einer *atf_vnode* kann auf zwei Arten geschehen. Entweder gibt es keine Referenzen mehr auf die *atf_vnode*; aber es sind noch Strategien angebunden. In diesem Falle werden einfach alle Datenstrukturen wieder freigegeben. Oder die Freigabe entsteht durch das Entfernen der letzten Strategie mit der *pol_pop()*-Operation. Hier werden die Datenstrukturen soweit aufgeräumt, bis ein leerer Stapel an die *atf_vnode* angebunden ist. Die *atf_vnode* kann aber erst dann freigegeben werden, wenn keine Referenzen mehr auf sie existieren. Die Operationen verhalten sich in diesem Falle wie bei einer normalen, nicht attributierten Datei.

4.11 Strategieoperationen

Die Operationen der Strategien gliedern sich: Einerseits sind es die Dateisystemoperationen. Diese werden bei jedem normalen Dateizugriff ausgeführt und die Reihenfolge wird durch den Stapel bestimmt. Andererseits gibt es die Strategieoperationen, die direkt von der Systemaufrufebene mit der *pol_ctl()*-Operation angestoßen werden. Der *pol_ctl()*-Aufruf stellt die Schnittstelle zwischen Anwendung und Strategie dar. Hierüber werden zusätzliche Daten an die entsprechende Strategie übergeben oder von ihr abgefragt. Bei den benutzerdefinierten Attributen werden die Operationen *Öffnen* und *Löschen* über diese Schnittstelle durchgeführt.

4.12 Koordinierung der Strategieoperationen

Die Operationen *pol_push()* und *pol_pop()* verdienen besondere Beachtung. Da diese Operationen den Stapel einer attributierten Datei verändern, müssen sie unter gegenseitigem Ausschluß gegenüber sich selbst und den anderen Dateisystemoperationen laufen. Diese Eigenschaft wird durch eine exklusive Sperre gewährleistet. Alle anderen Operationen belegen nur eine gemeinsame Sperre. Die exklusive Sperre ist bevorzugt, um ein Aushungern der entsprechenden Operationen zu vermeiden. In den dateisystemabhängigen Operationen kann es auch zu Unterbrechungen kommen, die einen Neustart des Systemaufrufs zur Folge haben. In diesem Falle muß dafür gesorgt werden, daß gesetzte Sperren aufgehoben werden. Zu Unterbrechungen kann es insbesondere bei Operationen in Gerätetreibern (Terminaltreiber) kommen.

4.13 Zusammenfassung

In diesem Kapitel wurde die Erweiterung des Unix-Dateisystems um die Komponenten *attributierte Dateien* und *Kernstrategien* vorgestellt. Der Mechanismus der attributierten Datei erlaubt es, an alle Dateisystemobjekte Attribute anzubinden. Dieser stellt eine Basis dar, mit der man genügend Zusatzinformationen mit den Daten speichern kann, um eine längerfristige Speicherung zu ermöglichen. Diese Daten können auch für andere Zwecke (Beschreibungen für Suchsysteme oder zur Unterstützung von Softwareentwicklungssystemen) genutzt werden. Der Aufwand für die Realisierung von attributierten Dateien in einem Dateisystem ist minimal. Alle für die Realisierung notwendigen Basismechanismen sind in einem Dateisystem schon vorhanden. Es ist sehr wahrscheinlich, daß auch Dateisysteme von anderen Betriebssystemen ähnlich einfach zu erweitern sind.

Mit den Kernstrategien wird ein erweiterbares Konzept den Dateisystemen hinzugefügt. Die Kernstrategien sind auf jedem Dateisystem anwendbar, das die attributierten Dateien unterstützt. Mit den Kernstrategien können die Semantiken der vorhandenen

Dateisysteme einfach erweitert werden. Das geschieht ohne daß die Funktionalität in alle Dateisysteme eingebunden werden muß. Die entsprechenden Erweiterungen sind in diesem Kapitel beschrieben worden.

Nimmt man die Funktionalität der attributierten Dateien und der Kernstrategien, so geben sich auch Ansätze zur Vereinheitlichung des Unix-Spezialdateikonzepts. Durch die Kernstrategien ließen sich alle Sonderkonstrukte mit Ausnahme der regulären Datei und des Verzeichnisses abbilden. Die für diese Dateitypen benötigten Parameter können dann in den entsprechenden Attributen gesichert werden.

Eine Realisierung der attributierten Dateien innerhalb des Betriebssystemkerns hat den Vorteil, daß der Zugriff über die Kernstrategien kontrolliert werden kann und so die Integrität der Dateiobjekte besser als bei einer Bibliothekslösung gewahrt werden kann[Lam91].

Die Realisierung des Prototyps hat gezeigt:

- Attributierte Dateien sind einfach realisierbar.
- Kernstrategien können die Funktionalität aller Dateisysteme erweitern.
- Spezialdateien in Unix können mit attributierten Dateien und Kernstrategien ersetzt werden.
- Attributierte Dateien stellen eine sinnvolle Ergänzung der Dateisysteme dar. Sie ermöglichen die Speicherung von Zusatzinformationen.

Bemerkenswert ist, daß nach über 8 Jahren Entwicklung[Mog86] im Bereich der Unix-Dateisysteme noch kein größerer Hersteller erweiterbare Dateisysteme zur Verfügung gestellt hat.

5 Langfristige Archivierung

In den vorherigen Kapiteln wurde am Beispiel von BackStage beschrieben, wie ein offenes, zukunftssicheres System aufgebaut sein kann. BackStage trägt mit dem vereinfachten Attributmengenmodell der heterogenen Struktur der heute verfügbaren Dateisysteme Rechnung. Bei BackStage wurden, wie auch bei anderen Speicherungssystemen, keine Aussagen über die gespeicherten Daten gemacht. Eine wesentliche Aussage ist jedoch, ob die gespeicherten Daten überhaupt noch für den Anwender in der gespeicherten Form nutzbar sind. Die folgenden Kapitel werden sich mit dieser Fragestellung beschäftigen.
Nach einer Analyse der Verwendung von `Daten` und einer entsprechenden `Klassifikation` werden die Softwaresysteme betrachtet. Letztere unterliegen auch zeitlichen Entwicklungsprozessen. Aus der Beobachtung heraus, daß die `Entwicklungszyklen` der Daten sich nicht mit denen der Softwaresysteme decken, werden Aussagen für die `Interpretierbarkeit` von Daten gewonnen. Die Interpretierbarkeit, damit auch der `Aufbewahrungswert`, ist vom `Abstraktionsgrad` der Daten abhängig. Um eine sinnvolle Speicherung in Archiven zu erreichen, werden zwei Relationen angegeben. Sie ermöglichen es, auf die Interpretierbarkeit und `Generierbarkeit` der Daten zu schließen. Mit diesen Relationen kann man bestimmen, wann es sich nicht mehr lohnt Daten zu speichern oder wann diese gefährdet sind uninterpretierbar zu werden.

5.1 Problematik

Die eigentlichen Probleme der langfristigen Archivierung erwachsen aus der ständigen Entwicklung der Datenverarbeitungstechnologie. Archivierte Daten sind statisch und machen diese Entwicklung im allgemeinen nicht mit. Deshalb muß ein System für längerfristige Speicherung so strukturiert sein, daß es die weitergehende Entwicklung mitzutragen kann. Die Archivierung muß möglichst lange die Nutzbarkeit der gespeicherten Daten unterstützen.

5.2 Datenklassifikationen

Nicht alle in einem Datenverarbeitungssystem gespeicherten Daten haben die gleichen Eigenschaften. Aufgrund der unterschiedlichen Verwendung der Daten kann man sie in verschiedene Klassen einteilen.
Aus der Sicht der langfristigen Datenspeicherung sind die Kriterien Lebensdauer und Interpretierbarkeit wesentliche Strukturierungsmittel.
Die Lebensdauer bestimmt, wie lange Daten mindestens gültig sind. In den folgenden Abschnitten wird auf diese beiden Klassifikationsschemata genauer eingegangen. Die Interpretierbarkeit erlaubt eine Abschätzung, wie lange die Daten maximal nutzbar sein können.

5.2.1 Lebensdauer von Daten

Die Lebensdauer von Daten wird von der eigentlichen Verwendung der Daten bestimmt. Die Lebensdauer kann durch technische Gegebenheiten oder durch externe Anforderungen bestimmt werden.

Die Datenverarbeitungsprozesse definieren die Lebensdauer der Daten. Jeder Datensatz muß so lange verfügbar sein, wie er vom Datenverarbeitungssystem benötigt wird. Wenn alle Verarbeitungsschritte durchgeführt wurden, können die Daten sofort gelöscht werden. Daten dieser Art rechnet man zu der Klasse der *transienten* Daten. Diese Daten werden nur temporär zur Verarbeitung generiert. Sie sind mit Abschluß der entsprechenden Verarbeitungsschritte löschbar.

In der nächsten Klasse der Daten werden die Daten als *kurzfristige* Daten bezeichnet. Hierbei handelt es sich meistens um Datenbestände mit administrativem Charakter. Diese Daten enthalten Konfigurationsinformationen, die zum Rechenbetrieb in der jeweiligen Installation benötigt werden. Sie können Notizen von Anwendern mit kurzlebigem Inhalt ablegen.

Als *mittelfristige* Daten bezeichnet man, was zu einer Produktentwicklung führt. Es sind die Datenbestände, die den eigentlichen Sinn des Betriebes der Rechensysteme ausmachen.

In der letzten Klasse findet man die *langfristigen* Daten. Hierbei handelt es sich um die Hauptergebnisse des Produktionsbetriebes. Es sind die gesammelten Erfahrungen des Betriebes oder nur Daten, die aufgrund externer Anforderungen, wie gesetzlichen Vorgaben, eine lange Aufbewahrungszeit haben. Die externen Vorgaben definieren meistens noch wesentlich schärfere Bedingungen für die Aufbewahrung von Daten. Zu den externen Anforderungen sind auch solche zu rechnen, die sich aus persönlichen Einschätzungen der Anwender ergeben. Daten, die für sich betrachtet, nur eine geringe Lebensdauer hätten, können dennoch aus der Sicht des Anwenders einen solchen Wert darstellen. Dementsprechend ist dann eine längere Speicherung durchzuführen.

Für die verschiedenen Klassen der Datenlebensdauern lassen sich grobe Schätzwerte angeben. Diese Werte können von Einsatzgebiet zu Einsatzgebiet stark differieren.

Klasse	Verwendung	Lebensdauer
transient	aktuelle Anwendung	< 1 Tag
kurzfristig	Rechnerbetrieb - Notizen	1 Tag - 1 Jahr
mittelfristig	Produktionsdaten, Ergebnisse	1 - 5 Jahre
langfristig	Dokumentation, Verträge, Daten mit gesetzlichen Anforderungen	> 5 Jahre

Tab. 5.1 Datenklassifikation nach Lebensdauern

Diese Datenklassifikation wirkt sich auf die Art der Datensicherung aus. Für die entsprechenden Lebensdauerklassen lassen sich folgende Strategien angeben:

Klasse	Sicherungstrategie
transient	keine
kurzfristig	Backup
mittelfristig	Archivierung
langfristig	Langzeitarchiv mit mehreren Kopien

Tab. 5.2 Sicherungsverfahren für Daten

5.2.2 Softwarelebensdauern

Die Lebensdauer von Software ist ein weiterer Kernbereich. Er muß bei der Betrachtung von langfristigen Datenspeicherungsverfahren berücksichtigt werden. Nun ist eine stetige Weiterentwicklung der Software in verschiedenen Bereichen der Datenverarbeitung zu beobachten. Die Neuentwicklungen berücksichtigen diese Strukturierung.

Die Entwicklung findet in den drei Bereichen Hardware, Betriebssystem und Anwendungssoftware statt. Entsprechend der schnellen Entwicklung in den Bereichen Hardware und Anwendungen ergeben sich kurze Softwarelebensdauern. Die schnelle Entwicklung im Bereich der Hardware von etwa 1-2 Jahren pro Generation hat auch Auswirkungen auf die Lebensdauer von Betriebssystemen und Anwendungen. Das trifft auch zu, wenn die Hardwareentwicklungen die Kompatibilität bewahren. Dadurch, daß zunehmend auch in verbreiteten CISC-(Complex Instruction Set Computer) Prozessoren (Intel Pentium) RISC-(Reduced Instruction Set Computer) Kerne verwendet werden, kommt es auch in diesem Bereich dazu, daß die Anwendungen und Betriebssysteme an die neue Hardwarearchitektur angepaßt werden müssen. Damit kann der Anwender auch die Leistungsverbesserungen nutzen. Im Bereich der RISC-Prozessoren existiert sogar die Abhängigkeit der Betriebssysteme und Anwendungen von Beginn an.

Die kürzeren Lebensdauern sind bei den Anwendungen selbst zu beobachten. Erfolgreiche Anwendungspakete haben eine Lebensdauer von etwa 5 - 10 Jahren, bevor sie durch Konkurrenz- oder Nachfolgeprodukten ersetzt werden. Selbstverständlich kommen während der Lebensdauer mehrere Versionen des Softwareproduktes auf den Markt.

Die längsten Entwicklungszyklen sind bei den Betriebssystemen zu verzeichnen. Das ist einerseits durch den immensen Aufwand bei der Betriebssystementwicklung bedingt und andererseits dadurch, daß die Betriebssysteme, wenn sie erfolgreich sind,

eine breite Installationsbasis erreichen. Sie zwingen den Betriebssystemhersteller gewissermaßen zur Unterstützung seines Softwareproduktes. Zusammenfassend ergibt sich etwa folgende Einteilung für die Softwarelebensdauern.

Softwareklasse	Lebensdauer
Architektur	1 - 2 Jahre
Betriebssystem	10 - 20 Jahre
Anwendung	5 Jahre

Tab. 5.3 Lebensdauern von Softwarepaketen

5.2.3 Interpretierbarkeit

Daten sind dann interpretierbar, wenn sie von den aktuell vorhandenden Anwendungen verarbeitet werden können. Die Verarbeitung muß über einfache Kopiervorgänge hinausgehen.

Die Interpretierbarkeit von Daten ist von verschiedenen Faktoren abhängig. Im Idealfall sind die Daten von der Anwendung nutzbar, die diese Datenbestände auch erzeugt hat. Durch die Speicherung von Datenbeständen kommt es zu einer Trennung von Anwendung und Daten. Mit der Zeit entwickelt sich die Hardware, die Anwendung oder das Betriebssystem weiter. Die Daten machen üblicherweise diesen Entwicklungsprozeß nicht mit. Hierdurch entsteht eine semantische Lücke. Sie wird umso größer, je mehr sich Anwendung und Datenbestand voneinander entfernen.

Im günstigsten Falle ist die Anwendung in der Lage, alte Datenformate zu lesen und korrekt zu interpretieren. Da aber die Entwicklung der Anwendung voranschreitet, können häufig nicht alle Eigenschaften des Datenformates vollständig korrekt übernommen werden. Hier ist also mit Einschränkungen zu rechnen. Wie erfolgreich solche Konvertierungen sind, hängt also vornehmlich von der Struktur der Anwendung und ihrer Kompatibilitätssoftware ab.

Mit der Dauer kann es passieren, daß der ursprüngliche Datenbestand nicht mehr vollständig rekonstruierbar ist. Dann ist die Grenze der Interpretierbarkeit erreicht. Es macht wenig Sinn, die Daten weiter aufzubewahren. Aus diesem Grunde stellt die Interpretierbarkeit ein Maß für die maximale Lebensdauer eines Datenbestandes dar.

Es ist teilweise möglich, Datenbestände über längere Zeit interpretierbar zu halten. Das geschieht, indem die Konvertierungen eventuell auch als menschliche Eingriffe durchgeführt werden. Damit dieses möglich ist, muß allerdings bekannt sein, welche Datenbestände von solchen Konvertierungen betroffen sind und ob die Konvertierungen notwendig sind. Konvertierungen im Datenbestand sind immer dann notwendig, wenn von einem Softwaresystem eine neue Version erscheint, die ein neues Datenfor-

mat verwendet. Konvertierungen können auch durch einen allgemeinen Softwarewechsel ausgelöst werden, wenn andere Datenformate favorisiert werden (Zeichensätze).

5.2.4 Verhältnis Interpretierbarkeit und Lebensdauer

Durch die Abhängigkeit von Interpretierbarkeit und Lebensdauer kann es vorkommen, daß die gewünschte Lebensdauer die nach der Interpretierbarkeit mögliche Lebensdauer überschreitet. Dieser Effekt tritt immer dann ein, wenn die Daten in einer nicht den Lebensdaueranforderungen entsprechenden Form abgelegt sind. Das Datenformat muß in einem solchen Falle an die Lebensdaueranforderungen angepaßt werden. Dies kann einerseits durch die Verwendung längerfristig gültiger Datenformate geschehen oder ander andererseits durch mehrmalige Konvertierung der Daten erfolgen. Es ist heute schwer vorherzusagen, welches Datenformat sich längerfristig durchsetzen wird. Dieses hat zur Folge, daß längerfristig zu speciherende Daten mehrfach innerhalb ihrer Lebensdauer konvertiert werden müssen.

5.2.5 Abstraktion

Bei einer längerfristigen Speicherung fällt auf, daß der Aufwand umso höher ist, je mehr Abhängigkeiten der Daten von der Umgebung bestehen. Mit Umgebung ist das Umfeld gemeint, in dem die Daten interpretiert werden.

Je mehr die Daten von konkreten Umgebungen abstrahieren, desto leichter wird es, die Daten längerfristig zu speichern. Bei Datenbeständen, die aus anderen Datenbeständen generiert werden können, verfügen die Vorgängerdatenbestände über ein höheres Abstraktionsniveau. Hier werden abstraktere Beschreibungen, wie sie Programmquellen oder Dokumente in Textverarbeitungssystemen darstellen, durch den Umwandlungsprozeß in konkretere Formen (ausführbare Programme, Druckerdaten) überführt.

Gewöhnlich braucht man nur die abstraktere Form des Datenbestandes zu sichern. Es muß jedoch gewährleistet sein, daß die Umwandlungsfunktionalität über den Speicherungszeitraum erhalten bleibt. Die Gewährleistung dieser Umwandlungsfunktionalität ist aber mit Abhängigkeiten behaftet. Die Umwandlung geschieht meistens mit Hilfe von Programmsystemen. Hier treten die entsprechenden Abhängigkeiten von Betriebssystem und Hardware auf.

Das Abstraktionsprinzip ist mit dem Aufbewahrungswert und der Interpretierbarkeit verknüpft.

Es ergibt sich folgendes Bild:

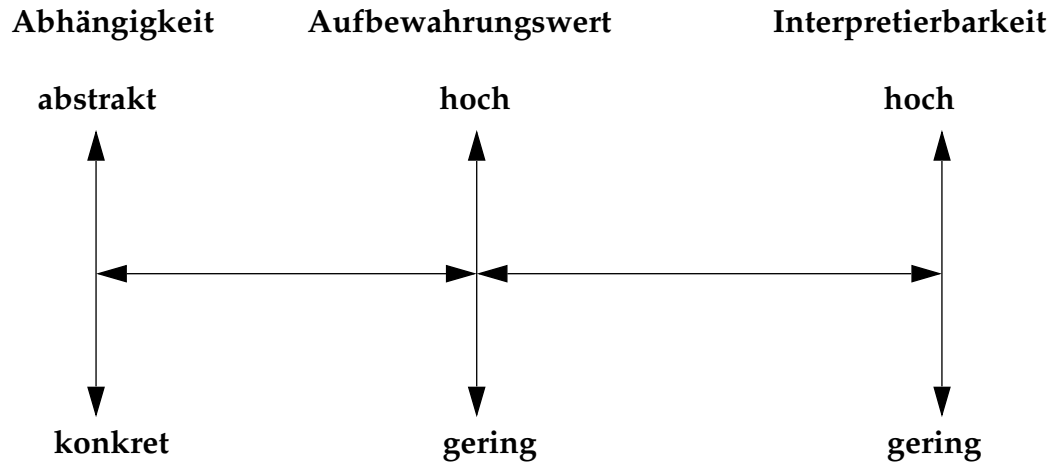


Abb. 5.1 Relation von Datenabstraktionsgrad und Langzeitspeicherung

- Die Interpretierbarkeit der Daten bleibt um so länger erhalten, je unabhängiger die Daten von der jeweiligen Umgebung sind.
- Je höher die langfristige Interpretierbarkeit ist, desto eher ist eine längerfristige Datenspeicherung gerechtfertigt.

5.2.6 Datenabhängigkeiten

Daten werden auf zwei Arten verarbeitet. Entweder werden sie durch den Prozessor oder eine Anwendung interpretiert, oder sie werden einfach als anonyme Datenmengen behandelt. Beide Verwendungsarten haben eigene Kriterien.

Bei der Verwendung als Datenmenge ist das Wesentlichste, daß diese Daten aus anderen generiert werden können. Ist dieses der Fall, so kann man die Daten ohne großen Verlust löschen. Generierbar heißt in diesem Zusammenhang, daß alle benötigten Programme, Daten und Lizenzen zur Generierung verfügbar sind.

Werden die Daten kopiert, so ist die Abhängigkeit von anderen Datenbeständen von Interesse. Nur die Kopie von vollständigen Datenmengen ist eine sinnvolle Operation. Die Abhängigkeit von anderen Datenbeständen ist dann wichtig, wenn die Daten interpretiert werden. Es können während der Interpretation der Daten Referenzen auf andere Datenbestände (Bibliotheken, Dateien) oder auch Dienstleistungen (Namensdienste, Lizenzserver) entstehen.

Damit die Interpretation der Daten erfolgreich durchgeführt werden kann, sind Datenbestände und Dienstleistungen bereitzustellen, sofern sie unverzichtbar für die Interpretation der Daten sind. Dieser Sacherverhalt wird durch die “Benötigt”-Relation beschrieben.

Der Abschluß unter der "wird benötigt"-Relation gibt die Menge der zur Verfügung zu stellenden Daten und Dienstleistungen an, die für die erfolgreiche Interpretation wesentlich sind. Können diese Daten und Dienstleistungen nicht zur Verfügung gestellt werden, ist die Interpretierbarkeit der jeweiligen Daten gefährdet. Graphisch stellen sich die Abhängigkeitsrelationen wie folgt dar:

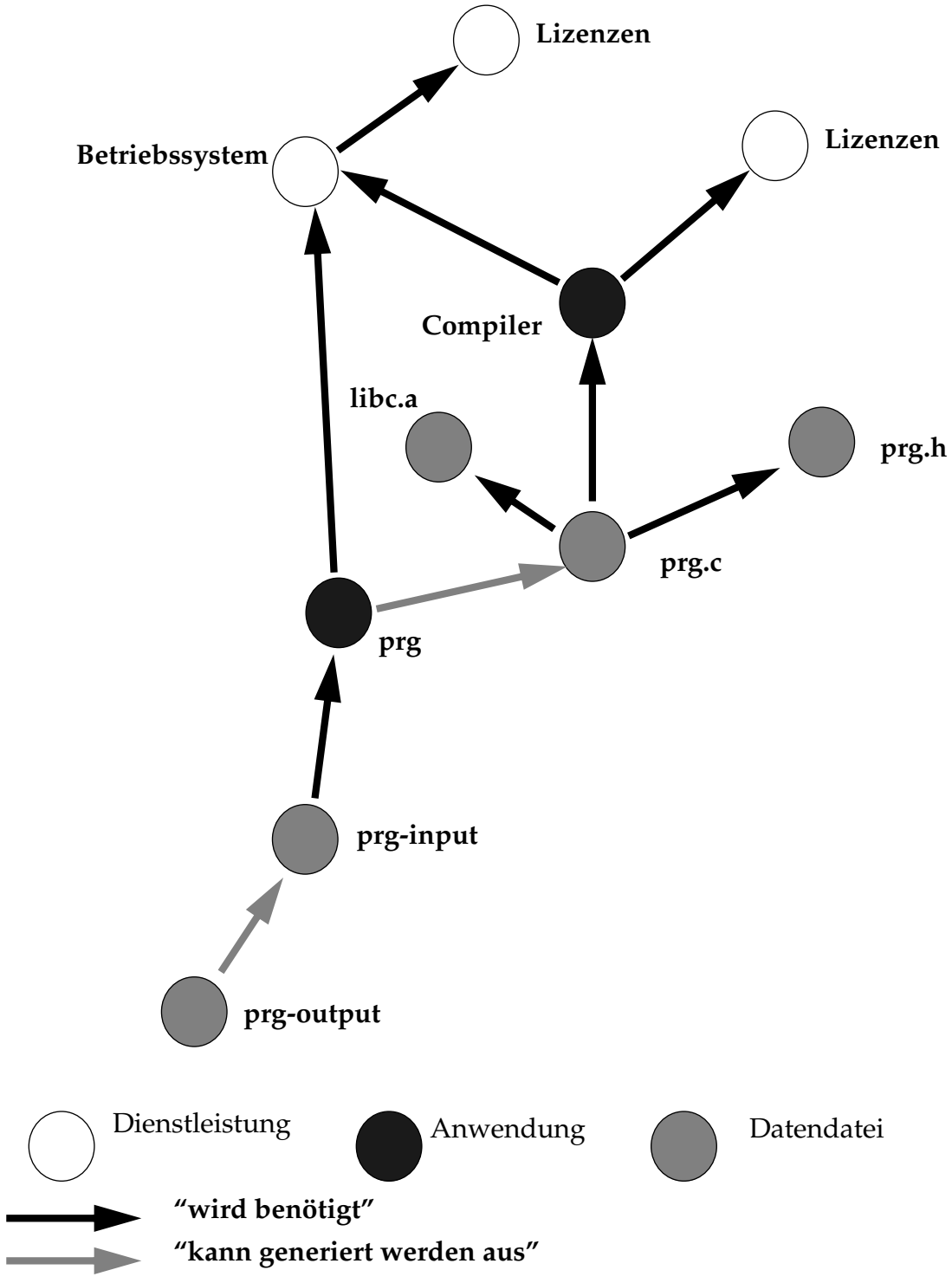


Abb. 5.2 Abhängigkeitsgraph für interpretierte Daten

Um eine Auswertung dieser Relationen durchführen zu können, müssen unter Umständen zu den eigentlichen Datenobjekten noch Zusatzinformationen im Sinne von "wird bereitgestellt" gespeichert werden. Die Information "wird bereitgestellt" erlaubt es, aufwärtskompatible Programme zu klassifizieren. Es muß aber erkannt werden, welche Funktionalitäten von diesen Programmen abgedeckt werden. Mit der "wird bereitgestellt"-Information können dann alle Datenobjekte für eine "wird benötigt"-Relation gefunden werden.

Zusammenfassend ergibt sich für die beiden Verwendungsarten von Daten:

- Nicht interpretierte Daten benötigen eine Anwendung oder Umgebung, die sie interpretiert.
- Interpretierende Anwendungen und Umgebungen müssen spezifizieren, unter welchen Bedingungen eine ordnungsgemäße Interpretation möglich ist.
- Zu interpretierende Daten müssen spezifizieren, welche Umgebung oder Anwendung vorausgesetzt wird.
- Die transitive Hülle der oberen Beziehungen definiert die Voraussetzungen unter denen Daten interpretierbar sind.
- Daten, die keine sie interpretierende Anwendung, Umgebung oder Nutzer haben, brauchen nicht gespeichert werden.

Diese Relationen beziehen sich nicht nur auf einzelne Dateisystemobjekte, sondern auch auf Gruppen davon. Die Gruppenbildung geschieht meistens in Anlehnung an die abgeschlossenen Software-Produkte. Die Anforderungen einer "wird benötigt"-Relationen müssen nicht unbedingt von einem Dateisystemobjekt spezifiziert werden. Sie können auch Anforderungen an das Datenverarbeitungssystem durch den Betrieb sein.

5.2.7 Kostenaspekte

Langfristige Datenspeicherung ist mit Kosten verbunden. Neben den unmittelbaren Kosten der Datenträger, der Pflege der Datenträger und Geräte gibt es auch noch Aufwände, die durch die Speicherung der einzelnen Daten selbst erzeugt werden.
Wie dargelegt, muß man nicht unbedingt alle Ausprägungen von Daten speichern, wenn man über die entsprechenden Umwandlungswerkzeuge verfügt, um Daten einer höheren Abstraktionsstufe in eine niedrigere umzuwandeln. Auch erlauben Konvertierprogramme die Verarbeitung von älteren Datenformaten. Solche Umwandlungen sind allerdings mit dynamischem Aufwand und Kosten verbunden. Je komplizierter die Umwandlungen selbst sind, desto höher ist der dynamische Aufwand, um eine adäquate Repräsentation der Daten zu erzeugen. Es kann demnach sinnvoll sein, weitere aktuelle Repräsentationen der Daten mit einer niedrigeren Abstraktionsstufe zu

erzeugen (Caching). Da Umwandlungen sehr aufwendig werden können, ist es möglich, daß die Kosten für die Darstellung der gewünschten Daten den eigentlichen Wert, der selten exakt zu bestimmen ist, übersteigen.

5.3 Archivformen

Heutige Archivsysteme nehmen immer mehr Daten in sich auf.
Dateisysteme, die auf diese Art gespeichert werden, erzeugen ständig wachsende Archive. Beim Durchsuchen der Archive wird die Entwicklungsgeschichte des Dateisystems sichtbar. Es gibt aber keine Mechanismen, die Archive auf die wesentlichen Daten zu reduzieren.
Wie dargelegt, werden Daten mit der Zeit immer weniger interpretierbar und verlieren damit an Wert. Diese Betrachtung ist zwar in jedem Falle subjektiv, aber zumindest für Dateien, bei denen die Interpretierbarkeit von technischen Gegebenheiten abhängt (Anwendungen), ist die Aussage relativ zuverlässig. Diese Art der Archivverdichtung wird bei heutigen Archiven nicht unterstützt. Die notwendigen Informationen fehlen.

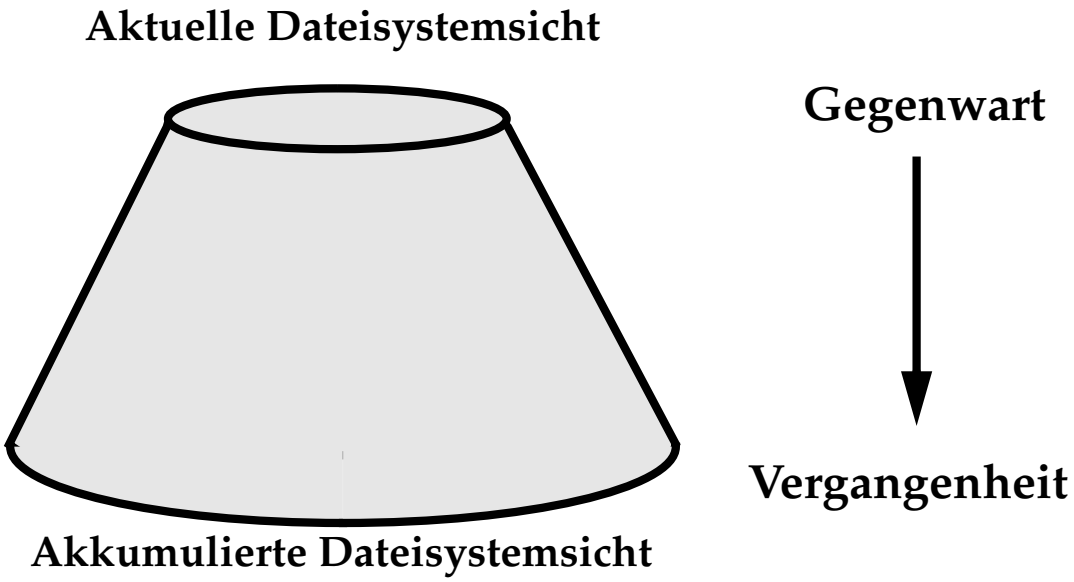


Abb. 5.3 Heutige Archivstruktur (Akkumulierende Archive)

Ziel für Archive und Dateisysteme der Zukunft sollte sein, auch die Interpretierbarkeitsaussagen, sowie die Lebensdauerangaben, zu unterstützen. Werden die Daten für Interpretierbarkeit und Lebensdauer noch mitverwaltet, so ist es nicht nur möglich, verdichtende Archive und Dateisysteme zu erstellen, sondern auch den Wechsel einer Umgebung durch den Einsatz neuer Hardware zu erleichtern. Wenn die wichtigen Daten mit den entsprechenden Umgebungsangaben versehen sind, ist es möglich, das Anforderungsprofil der neuen Umgebung zu bestimmen. Aussagen, welche Softwareprodukte und Datenformate unterstützt werden müssen, sind dann aus den Dateisystem- und Archivdaten ableitbar. Bei den “wichtigen” Daten handelt es sich um Daten

der mittel- und langfristigen Speicherung. Dabei zeigt sich, daß es bei der Archivierung über größere Zeiträume nicht ausreicht, den aktuellen Dateisystemzustand zu sichern. Es ist vielmehr notwendig, zu jeder Datenmenge, die längerfristig gespeichert werden soll, ausreichend Informationen zu liefern. Diese Mehrarbeit scheint aufwendiger, als sie bei der Realisierung ist. Daten, die längerfristig gespeichert werden müssen, sind vorwiegend Korrespondenz, Dokumente und Projektdaten. Von den meisten dieser Daten ist bekannt, wie und wo sie abgelegt werden und mit welcher Anwendung sie bearbeitet werden. Es sind also einfache Heuristiken möglich, um eine automatisierte Archivierung dieser Datenbestände zu erwirken. Durch Verwendung eines attributierten Dateisystems kann dieses noch erleichtert werden. Durch die konsequente Verwendung von zusätzlichen Attributen bei den Dateisystemobjekten könnte man Dienste wie Suchsysteme (*Archie*), Auskunftssysteme (*WWW*) unterstützen. Die Integration in diese Dienste würde durch die entsprechenden Attribute ermöglicht werden.

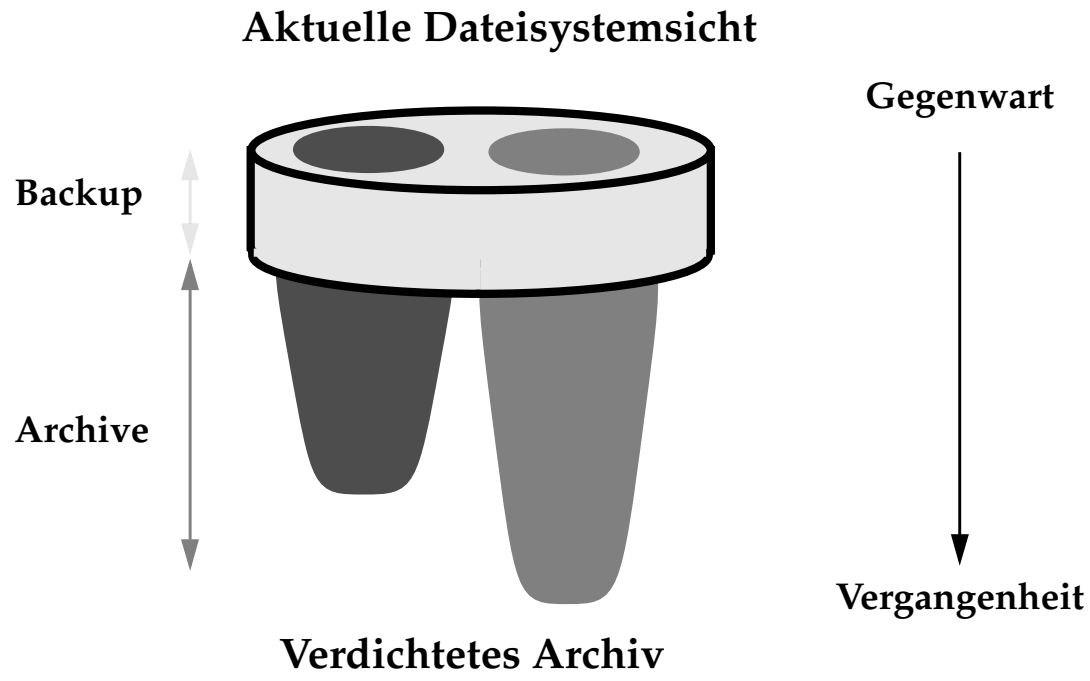


Abb. 5.4 Zukünftige Archivstruktur (Verdichtende Archive)

5.4 Datenhierarchien

Mit den Mechanismen der attributierten Dateien ist es nun möglich, auch komplexere Datenabhängigkeiten zu dokumentieren. Die Abhängigkeitsrelationen, die zwischen den einzelnen Daten verschiedener Abstraktionsstufen bestehen, lassen sich ausnutzen. Man kann die Beziehungen zwischen den einzelnen Dateien beschreiben.

Als Beispiel soll hier die Entwicklung eines Projekts mit Hilfe eines CAD Systems dienen. Bei der Verwendung eines jeden komplexeren Werkzeugs entsteht eine Vielzahl von Dateien, die aus anderen abstrakteren Repräsentationen generiert werden. Die verschiedenen Ausgabeformate wie Druckaufbereitung, Bitmaps und Zwischencode für weitere Systeme können genannt werden. Daten dieser Art sind zwar für sich selbst gesehen nutzbar, aber eine Modifikation ist nicht angeraten, denn Änderungen auf einem niedrigeren Abstraktionsniveau lassen sich selten automatisch in die höheren Schichten integrieren. Das Verbot der Modifikation ergibt sich immer dann, wenn die Generierung automatisch aus den Vorgängerdateien zu vollziehen ist.
Bei Datenbeständen, die durch Anwender gepflegt werden müssen, ist dagegen eine Modifikation erlaubt. Es handelt sich hier idealerweise um eine Verfeinerung der Spezifikation. Dennoch ergibt sich das Konsistenzproblem mit noch höheren Abstraktionsebenen wie der informellen Spezifikation des Systems.

Diese Konsistenzprobleme müssen allerdings auch manuell aufgelöst werden. Ein solcher Implementierungsschritt erfolgt manuell und ist damit ein kreativer Prozeß. Trotzdem ist es möglich, eine Unterstützung für diese Abhängigkeit anzugeben, da man bei jedem Dateisystemobjekt bestimmen kann, von welchen weiteren Einflüssen es abhängt. Bei solchen Beziehungen wirkt die “wird benötigt”-Relation. Mit dieser Relation kann man bei Änderungen an Daten auf einer niedrigeren Abstraktionsebene auf die damit verknüpften Daten der höheren Abstraktionsebene verweisen. Dieses erleichtert die manuelle Pflege von diesen Abhängigkeiten. Man kann auch nachträglich feststellen, wenn Daten einer höheren Abstraktionsebene geändert wurden, und ob alle davon abhängigen Daten niedriger Abstraktionsebenen zumindest auf Konsistenz überprüft wurden. Mit den attributierten Dateien wäre eine rudimentäre Unterstützung dieser Beziehungen möglich. Diese gilt besonders für Abhängigkeiten, die nicht von Anwendungen selbst verwaltet werden (Relation zwischen Projektantrag, Projektspezifikation und Projektimplementierung).

Die Verwaltung der Spezifikations-Implementierungs-Relation sollte Bestandteil von heutigen CASE-Werkzeugen sein. Diese Werkzeuge verwalten viele Daten unterschiedlicher Abstraktionsebenen. Für ein gutes CASE-Werkzeug wäre eine Arbeitsweise, wie sie durch die “wird benötigt” und “kann generiert werden aus” Relationen nahegelegt wird, sinnvoll. Durch Unterstützung der attributierten Dateien ließe sich die Funktionalität auch auf Dateisysteme übertragen. Ebenso ließen sich Dateisysteminhalte dann auch mit Mechanismen der CASE-Werkzeuge verwalten. Hier könnten sich Synergieeffekte zwischen CASE-Methoden und Dateisystemverwaltung ergeben. Ein vielversprechender Ansatz zur Integration von Dateisystemobjekten und Softwareentwicklungsumgebungen zeigt sich bei objektorientierter Interpretation von Dateisystemobjekten in [Mah91]. Die in diesem System aufgezeigten Möglichkeiten zeigen deutlich die Mächtigkeit der attributierten Dateien und den darauf aufsetzenden Werkzeugen.

Es ist nicht Aufgabe des Attributmodells, alle mit solchen Funktionen verbundenen Arbeitsweisen, zu realisieren. Es stellt Mechanismen zur Verfügung, um die für die Bestimmung der Abhängigkeiten notwendigen Metadaten zu hinterlegen. Mit geeigneten Anwendungen ist man dann in der Lage, automatisch jede unterstützte Datenrepräsentation der verschiedenen Dateisystemobjekte zu generieren. Ebenso kann die "generierbar aus"-Relation bestimmt werden. Mit den Angaben für die erwarteten Umgebungsparameter lassen sich auch die Nutzbarkeitsaussagen machen.

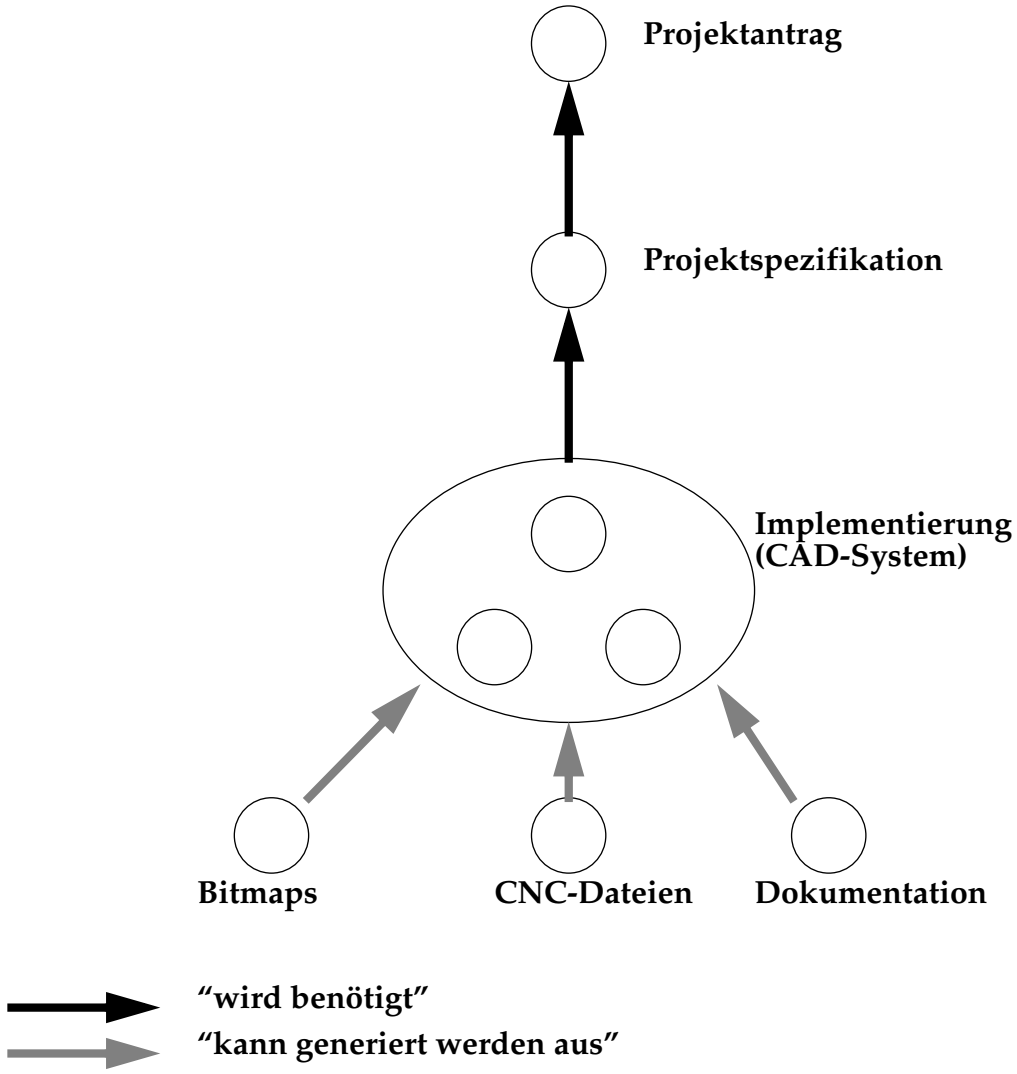


Abb. 5.5 Dateiabhängigkeiten

5.5 Zusammenfassung

In diesem Kapitel wurden die Aspekte der langfristigen Datenhaltung beleuchtet. Das Basisproblem ist die unterschiedliche Lebensdauer von Daten und die sie interpretierenden Anwendungen oder Nutzern. Da die technische Interpretation nicht immer so-

lange gewährleistet werden kann, wie die Daten interpretierbar sein müssen, sind besondere Vorkehrungen zu treffen. Damit die Daten auch entsprechend behandelt werden können, sind vornehmlich diese Anforderungen zu dokumentieren. Diese Dokumentationspflicht ist mit Hilfe der freien Attributierung der Daten zu erfüllen.
Da Daten umgebungsabhängig sind, sind sie nur unter bestimmten Bedingungen interpretierbar. Dieses wird durch die "wird benötigt"-Relation dargestellt. Sie beschreibt, unter welchen Bedingungen die Daten interpretierbar sind. Man kann jetzt zwei weitere Schlußfolgerungen ziehen:

- die Daten sind löschbar, weil sie nicht mehr interpretierbar sind,
- um die Daten zu interpretieren, müssen bestimmte Bedingungen erfüllt sein.

Diese Charakteristika erleichtern es, die Auswirkungen von Umgebungsveränderungen auf den Datenhaushalt abzuschätzen.

Mit der "kann generiert werden aus"-Relation lassen sich die Datenhierarchien dokumentieren. Wenn automatische Generierungsschritte beteiligt sind, so sind die davon abhängigen Daten löschbar. Mit dieser Relation läßt sich also das Minimum an zu haltenden Daten bestimmen.

Unter Anwendung der beiden oben definierten Relationen ist es möglich, Archive und Dateisysteme zu betreiben. Sie erlauben es aktuelle, nutzbare Daten zu verwalten und zu pflegen. Die Realisierung dieses Verfahrens wäre eine Qualitätsverbesserung in der Datenhaltung.

6 Zusammenfassung

In dieser Arbeit wurden die aktuellen Ansätze zur Massendatenspeicherung aufgezeigt. Es wurde dargelegt, daß diese Ansätze nur ein Anfang zur Lösung der Probleme der Massendatenspeicherung sein können.

Das BackStage-Modell zeigt exemplarisch die für eine zukünftige, längerfristige Datenspeicherung benötigten Konzepte:

- Das BackStage-Modell nutzt konsequent die Netzwerktechnologie.
- Die Verwendung von knotenübergreifenden Transaktionen erlaubt eine sichere Abarbeitung der Aufträge mit definierten Fehlerbehandlungsverfahren.
- Die Kommunikation für die Abarbeitung von Transaktionen wurde auf ein Minimum bei der Auftragserteilung und Ergebnisübermittlung beschränkt.
- Datentransfers werden direkt zwischen den beteiligten Knoten durchgeführt.
- Die Reduzierung der Kommunikationsphasen erlaubt eine weitgehende Toleranz gegenüber Netzwerkpartitionierung und Rechnerzusammenbrüchen.
- Dem Aspekt der Nebenläufigkeit wurde Rechnung getragen. Es ist erlaubt, mehrere Untertransaktionen anzustoßen, bevor die Ergebnisse eingefordert werden.
- Wegen der vielfältigen Authentisierungs- und Authorisierungsmechanismen sind die Netzwerkverbindungen so gestaltet, daß beim Aufbau einer Assoziation das Authentisierungsprotokoll ausgehandelt und durchgeführt wird. Diese Authentisierungsinformation kann dann von den Autorisierungsmodulen ausgewertet werden. Es ist aber immer noch ein ungelöstes Problem, wie mit der Veralterung der Autorisierungsinformation umgegangen werden muß.
- Um den wechselnden Anforderungen des Betriebs gerecht zu werden, sind alle BackStage-Komponenten mit einer konfigurierbaren Betriebmittelverwaltung ausgestattet. Sie erlauben flexible Kontrolle der Auftragsbearbeitung.

Um eine fortschreitende Interpretation der Daten zu ermöglichen, müssen längerfristig zu speichernde Daten mit zusätzlichen Informationen versorgt werden:

- Zu diesen gehören die attributierten Dateisysteme, die eine Verwaltung von Zusatzinformationen zu den eigentlichen Daten erlauben. Diese Zusatzinformationen können dazu benutzt werden, die Archivierung zu unterstützen, Nutzinformationen zu hinterlegen und Datenabhängigkeiten zu dokumentieren.
- Es gibt wenig Hoffnung, daß Standards das Datenrepräsentationsproblem dauerhaft lösen werden. Vielmehr müssen längerfristige Speichersysteme in der Lage sein, die Vielfalt an Darstellungen und existierenden Standards zu unterstützen. Zumindest müssen sie erlauben, die gespeicherten Daten zu qualifizieren.

Weiter wurden Aussagen über Interpretierbarkeit und Lebensdauer gemacht. Sie zeigen auf, daß es möglich ist, einen Archivdatenbestand kontrolliert zu verkleinern. Daten, die nicht mehr verwendbar sind oder nicht mehr benötigt werden, können gelöscht werden. Dadurch wird die Dateninflation eingeschränkt.

Mit der Dokumentation der Datenabhängigkeiten ist es nicht nur möglich, die Dateninflation einzuschränken, sondern Änderungen können auch besser durchgeführt werden, selbst wenn die Generierung von abhängigen Dateien nicht immer automatisch erfolgen kann.

Eine Integration der attributierten Dateien in Verbindung mit den “wird benötigt”- und “kann generiert werden aus”-Relationen kann die Datenhaltung und Pflege wesentlich verbessern. Diese Relationen können dazu beitragen, den Datenbestand klein zu halten. Bei Erweiterungen und Anpassungen des Systems an neue Entwicklungen kann bestimmt werden, ob alle relevanten Datenbestände nutzbar bleiben.

7 Ausblick

Die in dieser Arbeit dargestellten Überlegungen zur Archivierung von dateisystemartig organisierten Datenbeständen können nur ein Anfang sein. Wünschenswert wäre es, erweiterte Attribute für Dateien in bestehende Dateisysteme zu integrieren. Ebenso ist die Frage zu klären, ob bestimmte Attribute automatisch bestimmt werden können.

In wieweit Mechanismen zur Vererbung von (Meta-)Informationen im Einsatz sinnvoll sind, ist noch unklar. In letzter Konsequenz würden attributierte Dateisysteme mit Mechanismen, die die verschiedenen Attribute auswerten und verwalten, eine Datenbasis darstellen. Diese Datenbasis enthielte viele unstrukturierte Datenobjekte, die über die Attribute miteinander in Beziehung gesetzt werden könnten. Es fände hier eine wesentlich engere Koppelung von Daten und Anwendungen statt, als sie bei heutigen Systemen vorzufinden ist, dynamische objektorientierte Systeme einmal ausgenommen.

Ein großer Vorteil wäre es, wenn aufgrund der Attributstruktur verdichtende Archive aufgebaut werden würden. Fragen hinsichtlich der Interpretierbarkeitsaussagen und Lebensdauerangaben wären noch zu klären.

Attributierte Dateisysteme und Archive erlauben auch neue Ansätze zur Klärung der Datenrepräsentationsfrage. Durch die Dokumentation der Abhängigkeiten und möglichen Konvertierungen könnten Dateisysteme und Archive geschaffen werden, die es erlauben, die Daten immer so zu präsentieren, wie die Anwendung sie erwartet. Hier wäre es möglich, eine neue Sicht auf die Datenhaltung zu entwickeln. Erste Ansätze hierzu sind spezielle Dateisysteme, die Kompression und Verschlüsselung erlauben. Diese dürften aber keine Einzellösungen bleiben, sondern sollten grundsätzlich als Problem der Datenrepräsentation aufgefaßt werden.

Mit der fortschreitenden Vernetzung und dem wachsenden Kommunikationsbedarf (Datenautobahn) wären Überlegungen zur Integration verschiedenster Informationsdienste mit den Datenhaltungssystemen sinnvoll. Hier könnte man Arbeitsgänge sparen, wenn die Informationsdienste leichteren Zugriff auf die Daten in Datenhaltungssystemen hätten. Bei den Datenhaltungssystemen wären nur geringe zusätzliche Informationen für die Informationssysteme zu speichern. Die Informationssysteme wären dann in der Lage, anhand der Attribute und der entprechenden Dokumente, die Informationen aufzubereiten.

Für den Betrieb von Rechenanlagen wären die Informationen über Daten- und Diensteabhängigkeiten von erheblichem Wert. Sie würden die Planung von Umstellungen und Erweiterungen sehr vereinfachen. Auch wären die Effekte abschätzbar, die auftreten, wenn ein Softwareprodukt nicht mehr weitergepflegt wird. Es könnten rechtzeitig entsprechende Datenkonvertierungen vorgenommen werden, ohne daß man feststellen müßte, daß die Daten zwar gesichert sind, aber nicht mehr nutzbar.

Zukünftig sollte die Devise nicht mehr

> “Es ist sicher irgendwo im Archiv, zumindest Teile davon.”

lauten, sonder vielmehr

> “Wenn es noch nutzbar ist, dann ist es vollständig im Archiv gespeichert.”

8 Literatur

App86 Apple Computer, “Inside Macintosh Volume IV”; Addison-Wesley Publishing Company, Inc.,Reading, Massachusetts, 1986, pp. 87 - 183

ASN1 “Information Processing Systems – Open Systems Interconnection – Specification of Abstract Syntax Notation One (ASN.1)”, International Standard 8824 (1987), International Organisation for Standardization and International Electrotechnical Commitee

Bac86 M. Bach, ,The Design of the UNIX Operating System”, Prentice-Hall Software Series, 1986, Englewood Cliffs, New Jersey

BLP85 E. Balkovich, S. R. Lerman und R. P. Parmelee, “Computing in Higher Education: The Athena Experience”, Communications of the ACM 28(11), pp 1214-1224, ACM (November 1985)

BM91 Steven M. Bellovin, Micheal Merrit, “Limitations of the Kerberos Authtication System”; (Proc. of the Fifth Large Installation Systems Administration Conference, San Diego, CA, USA, 30. 9. 1991 - 3. 10. 1991); USENIX,Berkeley, CA; 1991; pp. 253-267

Bro88 J. Brown : “The CTSS/POSIX Project”; Proc. of the Workshop on UNIX and Supercomputers,Pittsburgh, PA,26-27.9.1988; USENIX,Berkeley, CA; 1988; S.33-34

CGR90 G. A. Champine , D. E. Geer , W. N. Ruh : “Project Athena as a Distributed Computer System”; IEEE Comp. 23(9); IEEE Comp. Soc.,Wash., DC; 9.1990; S.40-51

CM90 Sam Coleman, Steve Miller, Ed., “Mass Storage System Reference Model”, Version 4, May 1990

CS92 Michael McClennen, Stuart Sechrest, “Introducing Multi-structured File Naming into Unix”, Proceedings of the USENIX File Systems Workshop, USENIX, Berkeley, 1992, pp. 151-152

Del88 C. A. DellaFera, “The Zephyr notification service”, Usenix Association Winter Conference Proceedings, pp. 213-220, February 1988

Enq91 Jim Enquist, “A Database for Unix Backup”; (Proc. of the Fifth Large Installation Systems Administration Conference, San Diego, CA, USA, 30. 9. 1991 - 3. 10. 1991); USENIX,Berkeley, CA; 1991; pp. 89-96

GR93 Jim Gray, Andreas Reuter, “Transaction Processing”, San Mateo, Calif. Kaufmann 1993

IEEE94 IEEE Storage System Standards Working Group (Project 1244), Reference Model for Open Storage Systems Interconnection", IEEE New York, USA, September 8, 1994, URL: ftp://swedishchef.lerc.nasa.gov/mass_store/ossiv5.ps{1,2,3}

ISO87 "Information processing -- 8-bit single-byte coded graphic character sets, ISO 8859", International Organization for Standardization, Genf, 1987

ISO91 "Information Technology -- ISO 7-bit coded character set for information interchange, ISO/IEC 646", International Organization for Standardization, Genf, 1991.

ISO93 "Information technology -- Universal Multiple-Octet Coded Character Set (UCS) -- Part 1: Architecture and Basic Multilingual Plane, ISO/IEC 10646-1", International Organization for Standardization, Genf, 1993.

Kar90 Frank Kardel, "Frozen Files", USENIX Security Workshop II Portland OR, USENIX, Berkeley, CA,USA, pp. 83 - 86

Kle92 Jürgen Kleinöder, "Objekt- und Speicherverwaltung in einer offenen, objektorientierten Betriebssystemarchitektur", Dissertation, Arbeitsberichte des IMMD, Universität Erlangen-Nürnberg: IMMD, 1992.

Kol91 Rob Kolstad, "A Next Step in Backup and Restore Technology"; (Proc. of the Fifth Large Installation Systems Administration Conference, San Diego, CA, USA, 30. 9. 1991 - 3. 10. 1991); USENIX,Berkeley, CA; 1991; pp. 73-80

Lam91 Andreas Lampen, "Advancing Files to Attributed Software Objects"; (Proc. of the USENIX Winter 1991 Techn. Conf., Dallas , TX, USA,21-25. 1. 1991); USENIX,Berkeley, CA; 1991; pp. 219-229

Leg93 Legacy-Applications: "Common Object Request Broker Architecture (CORBA) und Application Control Architecture (ACA)"; DECtec, Techn. Inform. d. Digital Equipment GmbH(3-4); Digital Equipment,München; 1993

Lis87 Barbara Liskov, "Argus Reference Manual", Cambridge, Mass. 1987

Mah91 Axel Mahler, "Organizing Tools in a Uniform Environment Framework"; (Proc. of the USENIX Winter 1991 Techn. Conf., Dallas , TX, USA,21-25. 1. 1991); USENIX,Berkeley, CA; 1991; pp. 231-242

Mey91 Klaus Meyer-Wegener, "Multimedia-Datenbanken", Teubner, Stuttgart, 1991

Mil94 Dave L. Mills, "Improved Algorithms for Synchronizing Computer Network Clocks", Proceedings of SIGCOMM94, Vol. 24, No. 4, October 1994, pp. 317 - 327

Mog86 Jeffrey C. Mogul, “Representing Information about Files”, PhD Thesis, Department of Computer Science, Stanford University, Report No. STAN-CS-86-1103, Stanford California, USA (Mar. 1986)

Mor86 J. Morris, M. Satyanarayanan, M. H. Conner, J. H. Howard, D. S. Rosenthal und F. D. Smith, “Andrew: a distributed personal computing environment”, Communications of the ACM, Vol. 29, No. 3, 1986

Mul89 Sape Mullender, Ed., “Distributed Systems”, ACM Press, New York, 1989

Nel81 B. J. Nelson, “Remote Procedure Call”; PhD Thesis, Department of Computer Science, Carnegy Mellon University, Report No CMU-CS-81-119; 1981

Osf94 Open Software Foundation, “Introduction to OSF/DCE”, Cambridge, USA, 1994

RGL88 M. A. Rosenstein , D. E. jr. Geer , P. J. Levine : “The Athena Service Management System”; (Proc. of the USENIX Winter 1988 Techn. Conf.,Dallas, TX,9-12.2.1988); USENIX,Berkeley, CA; 1988; S.203-212

SGK85 R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh und B. Lyon, “Design and Implementation of the Sun Network Filesystem”, Usenix Conference Proceedings Summer 1985

Shu91 Steve Shumway, “Issues in On-line Backup”; Proc. of the Fifth Large Installation Systems Administration Conference, San Diego, CA, USA, 30. 9. 1991 - 3. 10. 1991; USENIX,Berkeley, CA; 1991; pp. 81-88

Shu91a Steve Shumway, “A Distributed Operator Interaction System”, Proc. of the Fifth Large Installation Systems Administration Conference, San Diego, CA, USA, 30. 9. 1991 - 3. 10. 1991; USENIX,Berkeley, CA; 1991; pp. 97-104

SNS88 J. G. Steiner , C. Neuman , J. I. Schiller : “Kerberos: An Authentication Service for Open Network Systems”; Proc. of the USENIX Winter 1988 Techn. Conf.,Dallas, TX,9-12.2.1988; USENIX,Berkeley, CA; 1988; S.191-202

Tre88 G. W. Treese : “Berkeley UNIX on 1000 Workstations: Athena Changes to 4.3BSD”; Proc. of the USENIX Winter 1988 Techn. Conf.,Dallas, TX,9-12.2.1988; USENIX,Berkeley, CA; 1988; S.175-182

Weg90 Peter Wegner, “Concepts and Paradigms of Object-Oriented Programming”, *ACM OOPS Messenger*, No. 1, pp. 8-84, Jul. 1990

WLH81 William A. Wulf, Roy Levin and Samuel P. Harbinson. *HYDRA/C.mmp – An Experimental Computer System.* McGraw-Hill, New York, 1981.

Zwi91 Elisabeth Zwicky, “Torture testing Backup and Archive Programs: Things You Ought to Know But Probably Would Rather Not”; (Proc. of the Fifth Large Installation Systems Administration Conference, San Diego, CA, USA, 30. ,9. 1991 - 3. 10. 1991); USENIX,Berkeley, CA; 1991; pp. 181-190