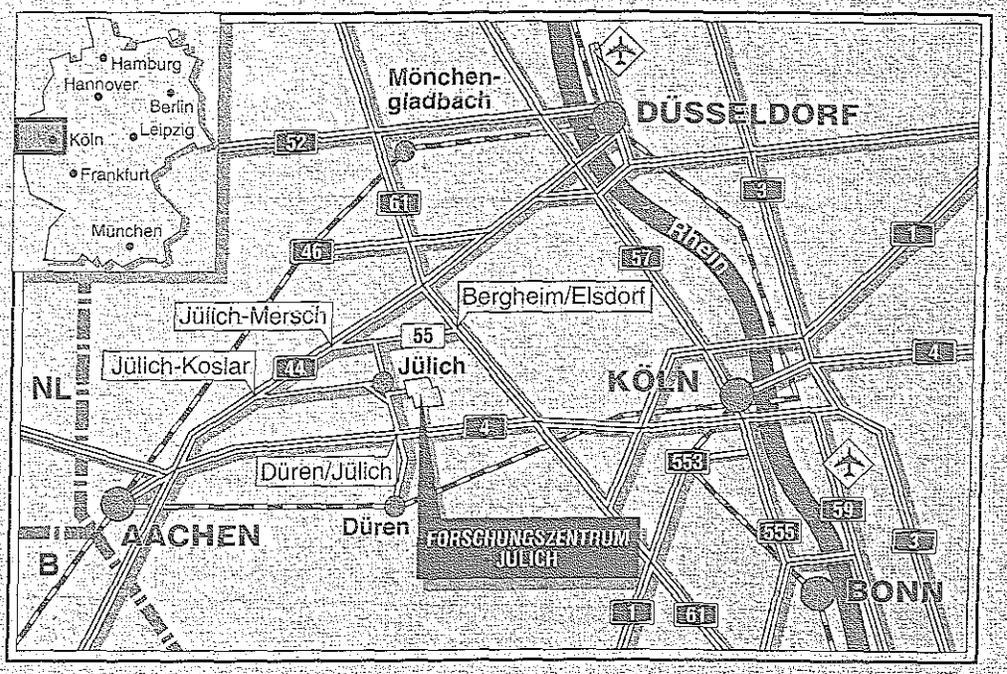
FORSCHUNGSZENTRUM JÜLICH GmbH

Zentralinstitut für Angewandte Mathematik

Eine Client-Server-Lösung für ein Inventory-Management-System

Jörg Sturm



Berichte des Forschungszentrums Jülich ; 3085
ISSN 0944-2952
Zentralinstitut für Angewandte Mathematik Jül-3085

Zu beziehen durch: Forschungszentrum Jülich GmbH · Zentralbibliothek
D-52425 Jülich · Bundesrepublik Deutschland
Telefon: 02461/61-6102 · Telefax: 02461/61-6103 · Telex: 833556-70 kfa d

Kurzfassung

In einem großen Rechenzentrum wie dem in der KFA ist die Dokumentation der Rechner- und Netzwerkinstallation so komplex, daß eine rechnergestützte Bearbeitung für das DV-Inventar unbedingt notwendig ist. Ziel dieser Arbeit ist es, ein Programmsystem zur Verfügung zu stellen, das wesentliche Kenndaten der DV-Komponenten speichert und sie so verknüpft, daß Rechner- und Netzwerkkonfigurationen abgebildet werden können. Dazu wurde eine Datenbank programmiert, die das relationale Datenbankmodell für diese Aufgabe erweitert. Da die Nutzung der Datenbank von vielen Workstations aus dem Netzwerk erfolgt, wurde ein Client-Server-Mechanismus implementiert. Für den Zugang zum Inventory-Management-System wurde eine graphische Oberfläche entwickelt, die dem Benutzer entsprechend des zu verarbeitenden Objektes vordefinierte Bildschirmmasken und Menüs bereitstellt.

Abstract

In a large computing centre, e. g. the Central Institute for Applied Mathematics of the Research Centre Jülich (KFA), the complexity of the installation of computers and networks makes it necessary to support the task of documenting inventory data by a programming system.

The objective of this work is to develop a computational system, which stores fundamental data of the DP-components and interlink them in a mode, that allows to reflect the configuration of computers and networks. For the solution of this problem, a database has been developped, that extends the relational database-model. Since many users have to access the database at the same time, a client server architecture is implemented. A graphical user interface supports the work with the inventory-management-system by supplying masks and menus corresponding to the object processed.

Inhaltsverzeichnis

Einleitung

Neue Hardware-und-Software-Technologien, verteilte Anwendungen, heterogene Online-Umgebungen und Client-Server Computing schaffen völlig neue Perspektiven für den Einsatz unternehmensweiter Kommunikationssysteme. In einem großen Rechenzentrum wie dem des Zentralinstitutes für angewandte Mathematik (ZAM) in der KFA ist die Dokumentation der Rechner- und Netzwerkinstallation so komplex, daß eine rechnergestützte Bearbeitung für das DV-Inventar unbedingt notwendig ist. Die Ursache dieser Komplexität ist bedingt durch die Vielfalt und Anzahl der zu verwaltenden Komponenten [?], z.B.:

- Workstations
- Personal Computer
- Drucker
- Bildschirme

Ziel dieser Arbeit ist es, ein Programmsystem zur Verfügung zu stellen, das die wesentlichen Kenndaten der DV-Komponenten speichert und sie so verknüpft, daß Rechner- und Netzwerkkonfigurationen abgebildet werden können.

Dazu wurde eine Datenbank programmiert, die das relationale Datenbankmodell für diese Aufgabe erweitert. Für den Datenbankentwurf ist es notwendig, die reale Welt der Netzwerkstruktur, die beliebig komplex sein kann, auf einen Ausschnitt abzubilden [?]. Dieser Ausschnitt ist im Inventory-Management-System wiedergegeben und soll durch die Datenbank XDB verwaltet werden.

Die Datenbankentwicklung befindet sich seit geraumer Zeit im Umbruch. Sogenannte 4GL-Systeme (Fourth Generation Language Systems) beginnen herkömmliche Softwaretechnologien vom Markt zu verdrängen. Leistungsstarke und graphikorientierte Workstations in schnellen und heterogenen Netzwerken mit Client-Server-Architektur geben diesen Programmierumgebungen einen zusätzlichen Akzeptanzschub [?].

Ein 4GL-System zeichnet sich durch eine menügestützte und graphische Benutzerschnittstelle aus. Eine solche Benutzerschnittstelle wurde auch im Inventory-Management-System implementiert. Dabei handelt es sich um eine Oberfläche, die dem Benutzer

entsprechend des zu verarbeitenden Objektes vordefinierte Bildschirmmasken und Menüs bereitstellt.

Da die Nutzung der Datenbank von vielen Workstations aus dem Netzwerk erfolgt, wurde ein Client-Server-Mechanismus entwickelt. Der Server stellt in diesem Ansatz einen Prozeß in einem beliebigen Rechner dar, der auf Dienstanforderungen eines Clients wartet [?].

Eine Einführung in die Anforderungen des Inventory-Management-Systems findet sich in Kapitel 1. Im zweiten Kapitel werden das hierarchische und das relationale Datenbankmodell beschrieben und in Bezug auf die Eignung für das Inventory-Management-System verglichen. Kapitel 3 beschreibt die Implementation der Datenbank XDB. Die Erarbeitung des Client-Server-Modells erfolgt in Kapitel 4. Als letzter Teil der Arbeit wird im fünften Kapitel eine Darstellung der graphischen Benutzeroberfläche gegeben. Das letzte Kapitel beschließt die Arbeit durch eine Zusammenfassung und einen kurzen Ausblick.

Kapitel 1

Die Anforderung an das Inventory-Management-System

Ein Inventory-Management-System muß sowohl das aktuell vorhandene DV-Inventar und die Zugehörigkeit der Komponenten zueinander, aber auch zu Benutzern und dem Betreuungspersonal speichern. Es müssen Funktionen vorhanden sein, die den gesamten Lebenszyklus eines Gerätes nachbilden lassen. Der Zugang zum Inventory-Management-System wurde erleichtert durch die Entwicklung einer graphischen Oberfläche.

1.1 Der Lebenszyklus eines Gerätes

Aus der Sicht des Inventory-Management-Systems umfaßt der Lebenszyklus eines Gerätes die folgenden Zustände:

- Die Bestellung
- Die Lieferung
- Die Erstinstallation
- Die Umkonfiguration
- Die Außerbetriebnahme
- Die Re-Installation
- Die Verschrottung

Dabei ist der Übergang von einem Zustand in einen anderen nicht beliebig. An folgender Abbildung soll dies deutlich gemacht werden.

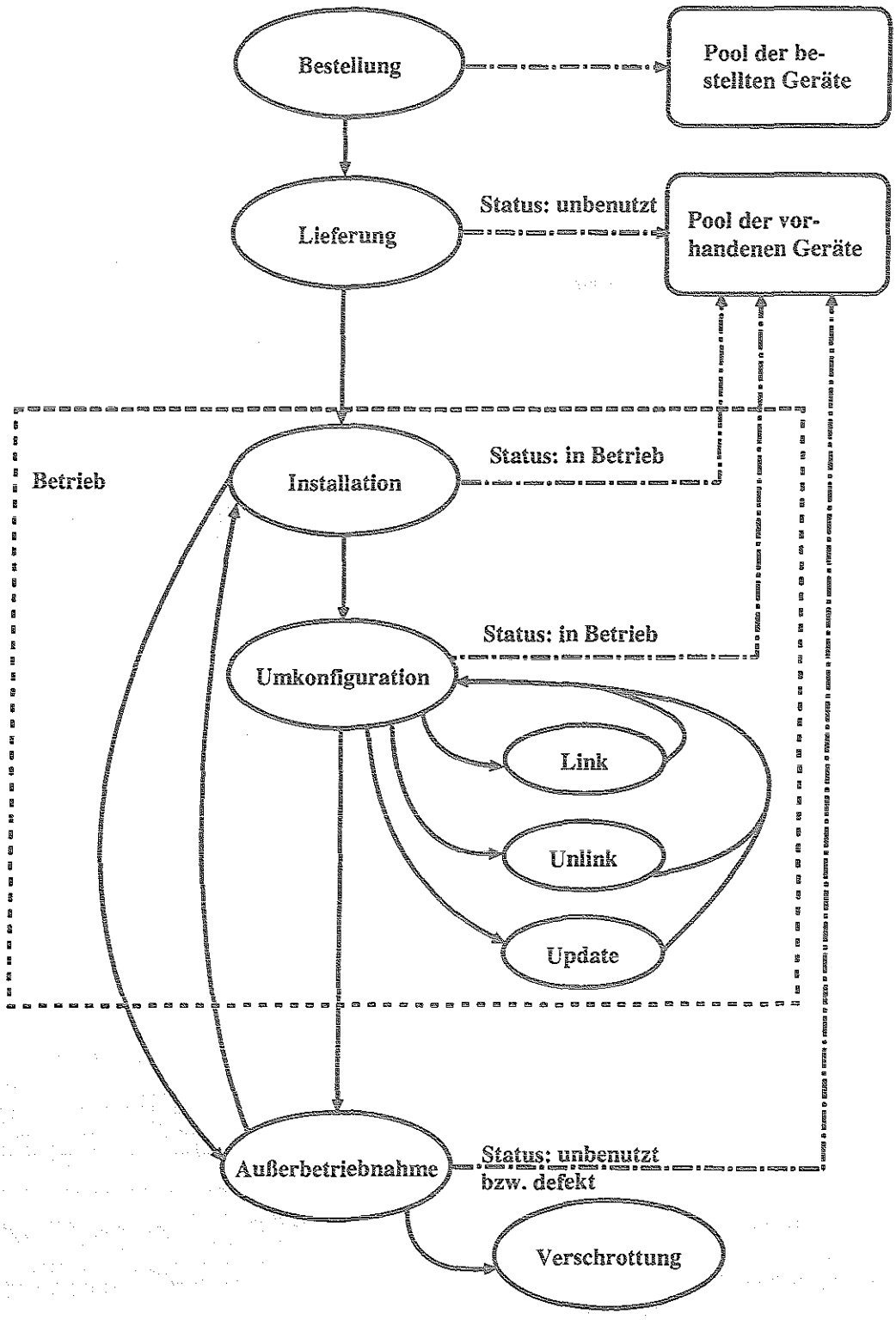


Abbildung 1.1: Lebenszyklus eines Gerätes

Abb. 1.1 zeigt die Lebensphase und die erlaubten Übergänge. In jedem dieser Zustände werden dem Datensatz weitere Informationen hinzugefügt bzw. werden Daten geändert. Die bestellten und die gelieferten Geräte befinden sich physikalisch im gleichen Pool, sie unterscheiden sich nur durch ihren Status. Die Zuordnung des Status zum Zustand des Lebenszyklus sieht wie folgt aus:

Bestellung	$\Rightarrow$	bestellt
Lieferung	$\Rightarrow$	unbenutzt
Installation	$\Rightarrow$	in Betrieb
Umkonfiguration	$\Rightarrow$	in Betrieb
Außerbetriebnahme	$\Rightarrow$	unbenutzt oder defekt

Bei der Verschrottung wird der Eintrag für das Gerät im Pool der vorhandenen Geräte gelöscht.

Ein Gerät besitzt Verbindungen zu anderen Geräten (z.B. Workstation $\leftrightarrow$ Bildschirm), die im gesamten Lebenszyklus nicht statisch sein müssen. So kann z.B. nach einer bestimmten Zeit ein neuer Bildschirm beschafft werden und an der vorhandenen Workstation betrieben werden. Diese Zustandsänderung wird durch die Funktionen *Link* und *Unlink* beschrieben.

Kapitel 2

Datenstrukturen

Im folgenden Kapitel werden die Datenmodelle *Hierarchisches Datenbankmodell* und *Relationales Datenbankmodell* erläutert. Dabei wird auf die Probleme eingegangen, die sich für das Speichern der Daten im Inventory-Management-System ergeben. Abschließend wird die Wahl des Datenmodells begründet.

2.1 Grundlagen

Wohlunterscheidbare Dinge, welche in der realen Welt existieren, werden in Anlehnung an den englischen Sprachgebrauch *Entitiy* genannt, z.B. Personen, Autos, Städte oder Firmen [?].

Einzelne Entities, welche vergleichbar oder zusammengehörig sind, werden *Entity-Set* genannt, z.B. alle Personen einer Stadt, alle Firmenwagen eines Unternehmens.

Die Eigenschaften eines Entity werden als *Werte* bezeichnet. Die Zusammenfassung aller Werte für eine Eigenschaft wird *Wertebereich* (engl. Domain) dieser Eigenschaft genannt. Auf der Ebene eines Entity-Sets werden die Eigenschaften als Attribute bezeichnet.

Zur eindeutigen Identifikation eines Entity ist meistens nicht die Angabe aller Attributwerte erforderlich, stattdessen reicht es aus, nur für bestimmte Attribute ihren Wert anzugeben, durch die das Entity dann eindeutig beschrieben ist. Diese Attribute werden Schlüsselattribute genannt, ihre Zusammenfassung ist ein Schlüssel für das Entity-Set. Falls ein Entity mehrere Schlüssel besitzt, wird ein Schlüssel als Primärschlüssel ausgezeichnet. Um Redundanzfreiheit eines Schlüssels zu garantieren, wird noch folgende Vereinbarung gemacht:

Falls K_1 und K_2 die Eigenschaften besitzen, ein Entity zu identifizieren, und weiterhin gilt: $K_1 \subseteq K_2$, so bezeichnet man K_1 als Schlüssel, K_2 hingegen als Schlüsselkandidaten.

Ein Schlüssel ist also eine minimale Attributenkombination. Jede echte Obermenge von K_1 ist ein Schlüsselkandidat.

Damit kann nun folgende Definition aus [?] gemacht werden:

Eine Entity-Deklaration hat die Form

$$E=(attr(E), K) \quad ;$$

sie besteht aus einem Namen (E), einem Format (attr(E)) und einem Primärschlüssel (K), welcher aus Elementen von attr(E) zusammengesetzt ist.

Beispiel: Die Entity-Deklaration für eine Workstation kann folgendes Aussehen haben:

Workstation = ({Hostname, Hersteller, Seriennr., Inventarnr.}, Inventarnummer)

oder bei einem zusammengesetzten Schlüssel:

Workstation = ({Hostname, Hersteller, Seriennr., Inventarnr.} Hersteller, Seriennr.) .

2.2 Einfluß der Struktur der Inventargegenstände

Im folgenden wird nun ein Ausschnitt aus einem Workstation-Verbund dargestellt. Dabei werden 2 Workstations betrachtet, die als Peripheriegeräte sowohl CD-ROM-Laufwerke als auch externe Platten besitzen können. Nimmt man den einfachsten Fall, in dem jede Workstation genau eine externe Platte und genau ein CD-ROM-Laufwerk besitzt, dann ist das Speichern der Daten einfach zu realisieren. In der Datenstruktur für die Workstation kann ein Feld CDROM definiert werden, das einen Zeiger auf die Stelle in der Datenbank besitzt, an der die Daten für das CD-ROM-Laufwerk liegen ; ebenso wird mit der externen Platte verfahren. Auch für den Fall, daß eine Workstation keine externe Platte oder kein CD-ROM-Laufwerk besitzt, stellt das Speichern kein Problem dar: das entsprechende Feld bleibt leer oder es wird mit einem Null-Zeiger besetzt.

Schwieriger wird es nun für Konfigurationen wie in Abb. 2.1 dargestellt.

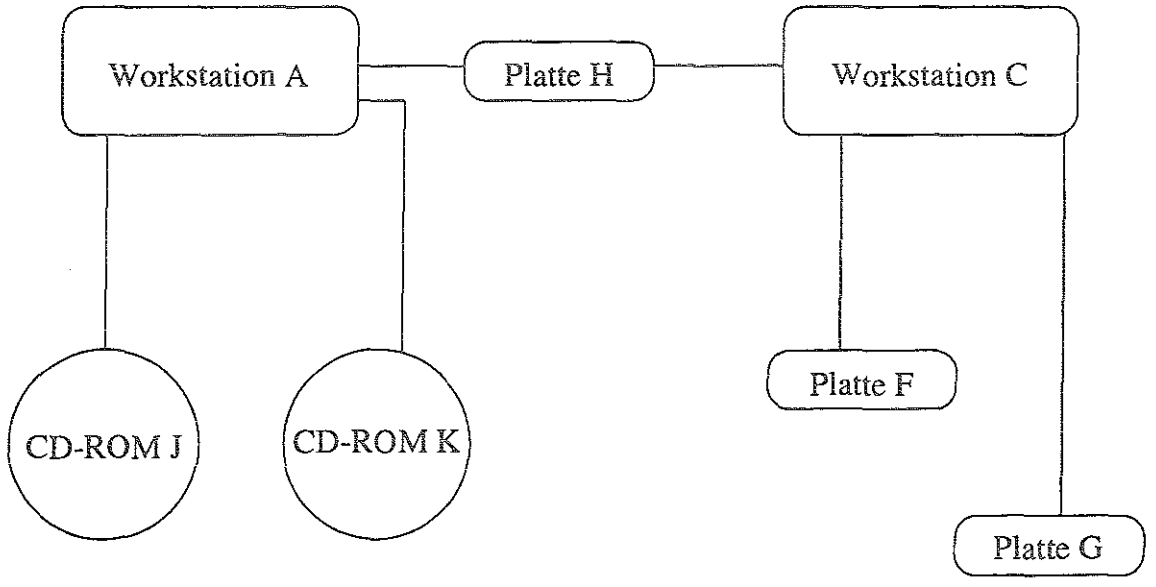


Abbildung 2.1: Workstationverbund

Hier besitzt eine Workstation eine variable Zahl an peripheren Geräten. Um diese Daten zu speichern, muß es eine variable Anzahl Felder für Zeiger auf andere Gerätetypen geben. Aus dem Wunsch, Datensätze variabler Länge und mit einer variablen Zahl von Feldern zu bearbeiten, wurde das hierarchische Datenmodell entwickelt, das im folgenden kurz beschrieben werden soll.

2.3 Das hierarchische Datenmodell

Das hierarchische Datenmodell geht davon aus, daß Daten, die für eine Anwendung verarbeitet werden, aus Grundanwendungsdaten und von diesen abhängigen Daten bestehen. Der auf der obersten Hierarchiestufe beschriebene Satztyp wird als Einstieg in die abhängigen Daten definiert, die Verbindung zwischen der oberen Stufe und der darunterliegenden wird durch Verkettung realisiert [?].

Durch diese Struktur erhält man einen Baum, in dem die Knoten die einzelnen Datensätze darstellen. Jeder Knoten, den man auch als Sohn bezeichnen kann, hat nur einen Vater.

Abbildung 2.2 zeigt ein einfaches Beispiel für ein hierarchisches Datenmodell.

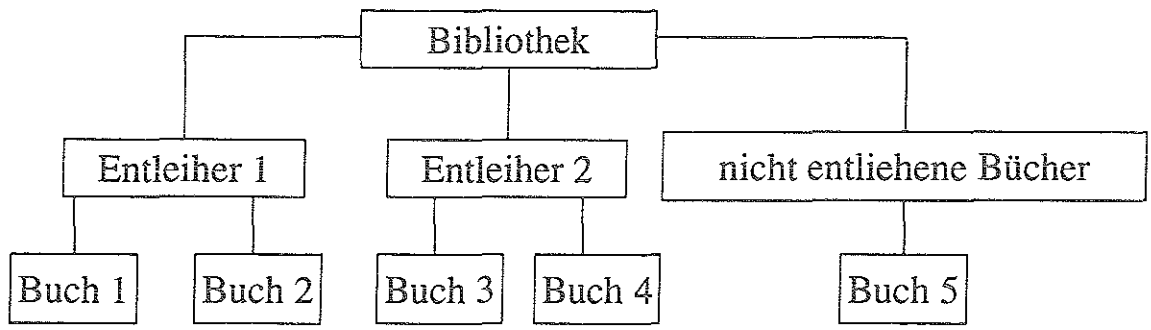


Abbildung 2.2: Hierarchisches Datenbankmodell

Das hier dargestellte Datenmodell basiert auf einer 1:n-Beziehung; für den Entleiher heißt das:

- Jeder Entleiher kann beliebig viele Bücher entleihen.
- Er kann aber nur in einer Bibliothek entleihen.

Genauso kann jede Bibliothek beliebig viele Entleiher bedienen, und jedes Buch kann zu einem Zeitpunkt nur von einer Person entliehen werden und natürlich auch nur einer Bibliothek gehören (Eine Bibliothek kann selbstverständlich mehrere Exemplare eines Buches besitzen). In der Datenbank würde jetzt der Datensatz Bibliothek eine Liste von Zeigern auf die Entleiher enthalten (und einen Zeiger auf den Bestand der nicht entliehenen Bücher). Jeder Datensatz Entleiher würde wiederum eine Liste mit Zeigern auf die von ihm entliehenen Bücher enthalten.

Bezogen auf die Abbildung 2.1 wäre mit dem hier vorgestellten Datenmodell das Problem der Speicherung der CD-ROM-Laufwerke gelöst. Bei dem Workstation-Verbund handelt es sich aber nicht um eine 1 : n-Beziehung, sondern um eine m:n-Beziehung. Betrachtet man in Abb. 2.1 die externen Platten, so sieht man, daß nicht nur der Fall auftritt, daß eine Workstation auf mehrere Platten zugreift, sondern daß die Platte H auch von mehreren Workstations genutzt werden kann. Übertragen auf das Beispiel der Bibliothek

hieße das, daß an einem Ort zwei oder mehr Bibliotheken wären, an denen ein Entleiher Bücher entleihen könnte. Die so entstandene Datenstruktur stellt aber keinen Baum mehr dar, sondern nur ganz allgemein einen Graphen, denn zu einem Sohn Entleiher gibt es mehrere Väter Bibliothek. Man muß nun eine Methode suchen, die diesen Graphen in eine hierarchische Baum-Struktur überführt. Als erstes bietet sich an, das Netzwerk durch Duplizieren derjenigen Recordsätze, die mehr als einem Vater besitzen, in eine Baumstruktur zu überführen. Daraus ergeben sich aber neue Probleme.

1. Durch die mehrfache Abspeicherung einiger Datensätze wird Redundanz erzeugt, die man in einer Datenbank aber grundsätzlich vermeiden will.
2. Wird in dem Baum ein Datensatz entfernt, so muß überprüft werden, ob dieser Datensatz mehrfach gespeichert wurde. Ist das der Fall, dann müssen auch die duplizierten Daten gelöscht werden.

Um wenigstens das Problem der redundanten Datenspeicherung zu lösen, wird inzwischen bei der Übertragung eines Graphen in einen Baum folgendermaßen vorgegangen: Die betroffenen Datensätze werden zwar dupliziert, allerdings nur als virtueller Datensatz. Physikalisch werden die Datensätze weiterhin nur einmal gespeichert, in der Hierarchie werden die virtuellen Duplikate durch Pointer realisiert, die auf die realen Records zeigen [?].

Bezogen auf den Workstationverbund in Abb. 2.1 enthält der Datensatz für Workstation A dann folgende Daten:

- allgemeine Daten, die nur Workstation A betreffen
- einen Zeiger auf CD-ROM-Laufwerk J
- einen Zeiger auf CD-ROM-Laufwerk K
- einen Zeiger auf die externe Platte H

Falls nun z.B. die Platte H aus dem Workstationverbund entfernt werden soll, muß in jedem Workstation-Datensatz gesucht werden, ob er einen Zeiger auf Platte H enthält. Ist das der Fall, so muß der Zeiger aus dem betroffenen Datensatz gelöscht werden. Ist der Bestand an Workstations groß, wächst auch der Aufwand für eine solche Aktion. Eine Möglichkeit, diesen Performance-Verlust zu verringern, bietet das relationale Datenmodell, das im folgenden Abschnitt erklärt wird.

2.4 Das relationale Datenmodell

2.4.1 Eigenschaften des relationalen Datenmodells

Das relationale Datenmodell bietet die einfachste strukturelle Sicht der hier angesprochenen Datenmodelle. Das Wort Relation steht für eine Tabelle, in der die Datensätze zeilenweise abgespeichert werden. Die Spalten der Tabelle werden über die Feldnamen angesprochen. Eine der Spalten muß die Eigenschaft besitzen, einen Schlüssel zu enthalten, über den die Zeilen eindeutig angesprochen werden können. Eine Datenbank besteht dann aus einer Menge von solchen Tabellen, d. h. Relationen [?].

Im Gegensatz zum hierarchischen Datenmodell kennt das relationale Datenmodell nur lineare Datenstrukturen, die Verarbeitung der Daten wird also wesentlich vereinfacht.

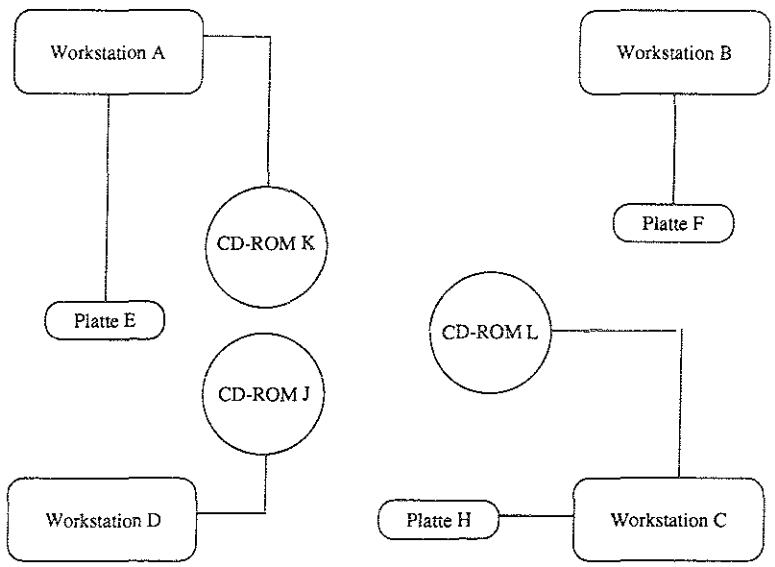


Abbildung 2.3: Workstationverbund

Als Beispiel für eine Relation wird nun die Tabelle für den Workstationverbund in Abb. 2.3 angegeben. Dabei wird erst einmal vereinfachend angenommen, daß eine Workstation nur mit einem CD-ROM-Laufwerk und einer externen Platte verbunden sein kann, außerdem hat jede Workstation einen Namen (A,B,C,D) und eine Inventarnummer, die auch den Primärschlüssel darstellt. Weiterhin besitzt auch jedes CD-ROM-Laufwerk und jede externe Platte einen Namen.

Für den Verbund ergibt sich dann eine Relation etwa in der Form:

Workstationverbund			
Workstation	Workstation-Inventarnummer	CD-ROM	Ext. Platte
A	145356	K	E
B	436654	–	F
C	335546	L	H
D	757677	J	–

Durch den Primärschlüssel, hier die Inventarnummer, ist eine Workstation eindeutig identifiziert. In der Relation fehlen in diesem Fall die Inventarnummern von CD-ROM-Laufwerk und externer Platte. Sie konnten wegfallen, weil durch den Namen des Peripheriegerätes der Zugriff auf die Daten eindeutig ist. Statt dem Namen der Peripheriegeräte hätte man auch die Inventarnummern in die Relation einbeziehen können. Durch die Herausnahme dieser Inventarnummern befindet sich die entstandene Relation in Normalform. Eine Normalform ist ein Hilfsmittel, mit der mögliche Redundanzen, die in einer Relation auftreten können, vermieden werden. Durch einen Normalisierungsprozeß wird ein nicht normalisiertes Relationenschema in ein solches überführt, das bestimmten Anforderungen genügt. Dabei wird das Relationenschema RS im allgemeinen in zwei (oder mehr) Schemata zerlegt, die die Anforderungen der Normalisierung erfüllen. Eine solche Zerlegung muß verlustfrei sein, d.h. der natürliche Verbund der Zerlegungen muß eindeutig die ursprüngliche Relation ergeben. Zur Erläuterung folgt ein Beispiel, bei dem letztgenannte Bedingung nicht erfüllt ist [?].

RS habe die Attributmenge $\{ABC\}$, und R sei die folgende Relation über RS:

R	A	B	C
	0	0	0
	1	0	1

Wird dann RS zerlegt in RS_1 mit $\{AB\}$ und RS_2 mit $\{BC\}$, so gilt für die durch Projektion aus R erhaltenen Relationen R_1 und R_2:

R_1	A	B
	0	0
	1	0

R_2	B	C
	0	0
	0	1

Der Verbund von R_1 und R_2 liefert

$R_1 \bowtie R_2$	A	B	C
	0	0	0
	0	0	1
	1	0	0
	1	0	1

und damit eine von R verschieden Relation. Das bedeutet, daß die Zerlegung nicht verlustfrei war und die in R enthaltenen Informationen verloren gingen. Folgende Zerlegung von R ist verlustfrei:

R_1	A	B
	0	0
	1	0

R_2	A	C
	0	0
	1	1

$R_1 \bowtie R_2$	A	B	C
	0	0	0
	1	0	1

2.4.2 Normalformen für Relationenschemata

In diesem Abschnitt werden die Normalformen und die Anforderungen, die sie an ein Relationenschema stellen, erläutert. Außerdem werden die Normalisierungsprozesse dargestellt, die eine Überführung der Relationenschemata in die gewünschte Normalform ermöglichen.

Die erste Normalform

Ein Relationenschema ist in erster Normalform, falls jeder Wertebereich eines Attributs dieses Schemas nur aus atomaren Werten besteht.

Dabei ist also insbesondere ausgeschlossen, daß ein Wertebereich zusammengesetzte Werte, d. h. Wertemengen als Werte (z. B. wieder Relationen) enthält [?].

Betrachtet werden nun die beiden Attributmengen Workstation und Zusatzausstattung. Der Wertebereich der Menge Workstation besteht aus dem eindeutigen Hostnamen. Die Menge Zusatzausstattung ist aber aus zwei Merkmalen zusammengesetzt: Objekt und Seriennummer. Eine Relation Workstationverbund würde dann wie folgt aussehen.

Workstation	Zusatzausstattung	
Hostname	Objekt	Seriennummer
zam144	CD-ROM	123456
	Drucker	234123
	Bildschirm	234789
zam999	Bildschirm	456324
	Drucker	234123

Unnormalisiert

Hostname	Objekt	Seriennummer
zam144	CD-ROM	123456
zam144	Drucker	234123
zam144	Bildschirm	234789
zam999	Bildschirm	456324
zam999	Drucker	234123

Normalisiert

Die Gruppe Zusatzausstattung ist eine Wiederholungsgruppe, die von der Gruppe Workstation abhängig ist. Einer Ausprägung in Workstation können mehrere Ausprägungen in Zusatzausstattung zugeordnet sein. Durch Zerlegen der Gruppe Zusatzausstattung in ihre Atome und das Einbringen des Schlüssels der übergeordneten Gruppe Workstation wird aus der unnormalisierten Relation eine Relation in 1. Normalform. Durch die Wiederholung des Schlüssels Hostname bleibt die hierarchische Struktur erhalten.

Betrachtet man nun die Mengen Workstation (Hostname) und Zusatzausstattung (Objekt, Beschreibung) mit Beschreibung (Hersteller,Seriennummer), so erhält man folgende Abhängigkeit:

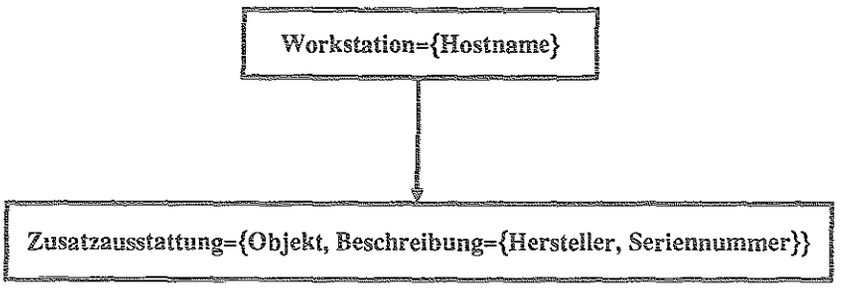


Abbildung 2.4: Relation Workstation-Verbund (unnormalisiert)

Hierbei ist die Relation Workstationverbund eine zusammengesetzte Gruppe, die auch wiederholend ist; einmal auf dem Niveau Workstation und einmal in Bezug auf Objekt. Für ein solches hierarchisches Schema geht der Normalisierungsprozeß wie folgt vor sich: [?]

1. Das Schema wird in einer Form angegeben, in der geschachtelte Klammern nicht mehr vorkommen.

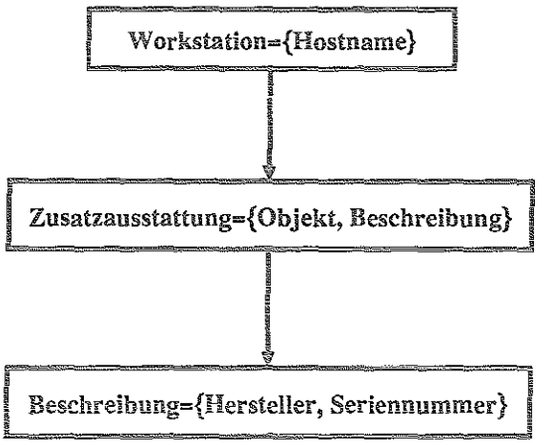


Abbildung 2.5: Relation Workstation-Verbund (normalisiert)

2. Man führt die Relationenschreibweise ein und bezeichnet die Primärschlüssel (hier durch Unterstreichung).

Workstation{Hostname}

⇓

Ausstattung{Objekt,Beschreibung}

⇓

Beschreibung{Seriennummer,Hersteller}

3. Man startet mit der Relation an der Baumwurzel. Der Primärschlüssel dieser Relation wird in die untergeordnete Relation eingefügt. Der Primärschlüssel der so erweiterten Relation besteht aus dem Primärschlüssel vor der Erweiterung und dem Primärschlüssel, der aus der übergeordneten Relation stammt. In der übergeordneten Relation können nun alle nicht einfachen Attribute gestrichen werden. Die Baumwurzel ist jetzt normalisiert und kann als solche entfernt werden.

Workstationausstattung{Hostname, Objekt, Beschreibung}

⇓

Beschreibung{Seriennummer, Hersteller}

Entfernt: Workstation{Hostname}

4. Der Schritt 3 wird für alle Unterbäume durchgeführt.

In den so entstandenen neuen Relationen sind keine Wiederholungsgruppen mehr enthalten.

Die zweite Normalform

Erweitert man die Attributmengen von Workstation und Zusatzausstattung, so könnte man etwa folgende Relation Workstationverbund erhalten:

Workstation	Zusatzausstattung			
Hostname	Peripheriegerät	Hersteller	Typbezeichnung	Seriennummer
zam999	CD-ROM	A	1232	345
zam999	Drucker	A	4755	432
zam999	Bildschirm	A	3006	987
zam144	CD-ROM	A	1232	432
zam144	CD-ROM	C	7411	438
zam144	Bildschirm	D	4410	666

Der Primärschlüssel ist durch {Hostname, Peripheriegerät, Seriennummer } gegeben. Dabei bezieht sich die Seriennummer auf das Peripheriegerät. Diese Relation befindet sich in der ersten Normalform. Sie enthält aber Daten, die für den Workstationverbund unwichtig sind (Typenbezeichnung des Herstellers). Der Sinn der zweiten Normalform besteht darin, in den Relationen die Attribute zusammenzuführen, die zusammengehören [?].

Dieser Workstationverbund hat im Hinblick auf Änderungsdienste der Datenbank folgende Konsequenzen:

1. Ändern eines Attributwertes (Änderungs-Anomalie)

   Ändert der Hersteller A die Typenbezeichnung seines CD-ROM-Laufwerkes, so muß, da zwei Workstations ein CD-ROM-Laufwerk des Herstellers A benutzen, die Relation Workstationverbund zwei mal geändert werden; dies kann z.B. in einem Multi-User-System zu Problemen führen.

2. Einfügen einer neuen Workstation (Einfüge-Anomalie)

   Eine neue Workstation kann nur in die Datenbank hinzugefügt werden, wenn sie über Zusatzausstattung verfügt, da sonst der Primärschlüssel {Hostname, Peripheriegerät, Seriennummer } unvollständig wäre.

3. Löschen einer Zusatzausstattung (Lösch-Anomalie)

   Die Zusatzausstattung einer Workstation soll gelöscht werden. Dann kann aber auch die Typenbezeichnung eines Gerätes verloren gehen, obwohl das vielleicht gar nicht gewünscht war.

Man stellt fest, daß die Relation Redundanzen aufweist, die Typenbezeichnung ist für das CD-ROM-Laufwerk des Herstellers A mehrfach gespeichert. Diese Redundanz ist offensichtlich auch der Grund für die oben genannten Anomalien. Man könnte zwar die Anomalien (2) und (3) durch die Benutzung von Nullwerten beheben, das würde aber dazu

führen, daß unter Umständen eine große Menge von Nullwerten verwaltet werden müßte, was bei Relationen mit vielen Tupeln zu Effizienzproblemen führen kann. Außerdem ließe sich damit die 1. Anomalie nicht beheben, und auch die Redundanzen würden bestehen bleiben.
Ein Weg, dieses Problem zu lösen, ist die Aufspaltung der Relation Workstationverbund in die neuen Relationen Workstationverbund (Hostname, Seriennummer, Peripheriegerät) und Gerät (Peripheriegerät, Hersteller, Typbezeichnung). Durch diese Aufspaltung geht keine Information verloren, denn die ursprüngliche Relation Workstationverbund läßt sich aus den neuen Relationen Workstationverbund und Gerät offensichtlich zurückgewinnen. Diese neuen Relationen sind in der zweiten Normalform.

Für eine präzise Definition der zweiten Normalform muß nun zuerst der Begriff *Funktionale Abhängigkeit* erklärt werden. Dazu die Definition der funktionalen Abhängigkeit nach [?].

Es seien A und B zwei Attribute in der Relation R. Das Attribut B ist dann funktional abhängig von A, wenn zu jedem Zeitpunkt jeder Wert von A nicht mehr als einen Wert in B hat.

Man schreibt $R.A \rightarrow R.B$ und faßt dies als eine Abbildung vom Typ "viele zu eins" auf.

- $R.A \rightarrow R.B$ bedeutet, daß Werte in A die Werte in B eindeutig identifizieren. Wenn ein Wert in A gegeben ist, so liegt der Wert in B fest.
- Wenn $R.A \rightarrow R.B$ gegeben ist, dann muß bei einer Änderung von A auch der Wert von B neu bestimmt werden. Dieser neue Wert von B kann natürlich mit dem alten Wert identisch sein.

Gegeben ist folgende Relation Workstationstandort mit der Attributmenge {Institut, Hostname}

Institut	Hostname
zam	zam144
zam	zam999
ikp	ikp001

Es gilt: Workstation.Institut → Workstation.Hostname, denn eine Änderung des Institutes zieht auch eine Neubestimmung des Hostnamens nach sich, während ein Ändern des Hostnamens kein neues Institut erfordert.

Als nächstes benötigt man noch den Begriff der *vollen funktionalen Abhängigkeit* . Auch dazu die Definition nach [?].

Gegeben seien zwei verschiedene Teilmengen von Attributen A und B in der Relation R, $A=\{A_1, A_2, ..., A_n\}$, $B=\{B_1, B_2, ..., B_n\}$. B ist voll funktional abhängig von A,

$$R.A \quad \Rightarrow \quad R.B,$$

wenn B funktional abhängig ist von A, aber nicht funktional abhängig ist von einer echten Teilmenge von A.

Die volle funktionale Abhängigkeit soll hier an einem einfachen Beispiel beschrieben werden.

Gegeben ist die Relation Workstationstandort {Institut, Gebäude, Hostname, Raum}, die den Standort der Workstation beschreibt. Dabei wird angenommen, daß sich ein Institut über mehrere Gebäude erstreckt. In dieser Relation gilt:

Workstationstandort.{Institut, Gebäude} $\Rightarrow$ Workstationstandort.{Raum}

Wegen

Workstationstandort.{Institut} $\rightarrow$ Workstationstandort.{Hostname}

gilt aber **nicht**:

Workstationstandort.{Institut, Gebäude} $\Rightarrow$ Workstationstandort.{Hostname}.

Mit der vollen funktionalen Abhängigkeit kann jetzt die zweite Normalform aus [?] definiert werden:

Eine Relation in der ersten Normalform ist auch in der zweiten Normalform, wenn jedes Attribut im Komplement eines Schlüsselkandidaten von diesem Schlüsselkandidaten voll funktional abhängig ist. Alle Attribute, die nicht zu einem Schlüsselkandidaten gehören, sind im Komplement des Schlüsselkandidaten.

Möchte man ein Relationenschema, welches sich in erster, aber nicht in zweiter Normalform befindet, in ein Relationenschema in zweiter Normalform überführen, so wird das ursprüngliche Schema derart zerlegt, daß die Nicht-Schlüsselattribute, welche von einer Teilmenge eines Schlüssels abhängen, mit dieser Teilmenge als Schlüssel ein neues Schema bilden und die restlichen Attribute mit dem Schlüssel von R ein weiteres. Diese Prozedur wird auf jedes so erhaltene Schema so lange angewandt, bis alle resultierenden Schemata in zweiter Normalform sind.

Die dritte Normalform

Zum Verständnis der dritten Normalform wird zunächst der Begriff der transitiven Abhängigkeit definiert [?].

Eine Menge von Attributen C in der Relation R ist transitiv abhängig von einer Menge von Attributen A in R, wenn:

1. **A und C disjunkt sind,**
2. **Eine Menge von Attributen B existiert, disjunkt in Bezug auf A und C, so daß gilt:**

   R.A → R.B und nicht R.B → R.A
   R.B → R.C

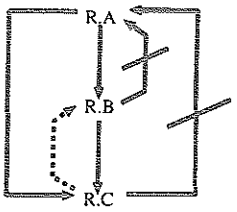


Abbildung 2.6: Transitivität

Das Diagramm dient zur Erläuterung von Transitivität. Strikte Transitivität liegt vor, falls zu den genannten Bedingungen auch noch hinzukommt, daß nicht R.C → R.B gilt. Zur Überführung von der zweiten in die dritte Normalform müssen die Schemata derart zerlegt werden, daß alle Transitivitäten ausgeschlossen werden. Zur Erläuterung der dritten Normalform soll folgendes Beispiel dienen:

Gegeben ist folgende Relation Workstation {Hostname, Institut, Gebäude}:

Workstation	Hostname	Institut	Gebäude
	zam144	ZAM	16
	zam145	ZAM	16
	ipp333	IPP	17
	ikp444	IKP	18
	ikp555	IKP	18

Diese Relation ist in der zweiten Normalform. Sie enthält aber immer noch Redundanzen, die Attributenmenge {Institut, Gebäude} ist wiederholt mit den Wertepaaren (ZAM, 16) bzw. (IKP, 18) besetzt.

Eine Änderung des Gebäudes eines Institutes (z.B. durch Umzug) hat mehrere Änderungen in der Tabelle zur Folge (Änderungs-Anomalie). In dieser Relation findet man die folgende transitive Abhängigkeit:

$$\text{Institut} \rightarrow \text{Gebäude}$$

Zerlegt man die Relation Workstation {Hostname, Institut, Gebäude} in die Relationen Workstation {Hostname, Institut} und Standort{Institut, Gebäude} und vergleicht die so entstandenen Relationen mit der Definition der dritten Normalform nach [?],

Eine Relation R in der zweiten Normalform ist auch in der dritten Normalform, wenn jedes Attribut im Komplement eines Schlüssselkandidaten nicht von diesem Schlüsselkandidaten transitiv abhängig ist.

so erkennt man, daß die neuen Relationenschemata in der dritten Normalform vorliegen und die Transitivitäten aufgelöst wurden. Auch diese Zerlegung ist verlustfrei, aus den neuen Relationenschemata können eindeutig die Relationenschemata in zweiter Normalform zurückgewonnen werden.

Die vierte Normalform

Zur Erklärung der vierten Normalform wird erneut die Relation Workstation benutzt. Als Zusatzausstattung stehen die CD-ROM-Laufwerke A und B, die Drucker C, D und E und die externen Platten F, G und H zur Verfügung. Die Relation hat folgendes Aussehen:

Hostname	CD-ROM	Drucker	Ext.-Platte
zam144	A	C	F
zam144	A	C	G
zam144	A	D	F
zam144	A	D	G
zam999	-	E	F
zam999	-	E	H
zam888	B	C	G
zam888	B	C	H
zam888	B	D	G
zam888	B	D	H

Diese Relation ist offensichtlich in dritter Normalform, enthält jedoch immer noch Redundanzen aufgrund von bestimmten Wertekombinationen. Falls die Workstation zam999 mit dem Drucker C verbunden wird, muß die Relation um zwei Einträge erweitert werden, da zam999 auch mit zwei externen Platten verbunden ist. Dies ist im Sinne der Anwendung jedoch nicht sinnvoll, da die Drucker einer Workstation nicht mit deren externen Platten im Zusammenhang stehen. Folgende Zerlegung vermeidet diese Problematik:

- Workstation-CD-ROM{Hostname, CD-ROM}
- Workstation-Drucker{Hostname, Drucker}
- Workstation-ext. Platte{Hostname, Ext.-Platte}

Die neuen Relationenschemata sehen folgendermaßen aus:

Hostname	CD-ROM
zam144	A
zam888	B

Hostname	Drucker
zam144	C
zam144	D
zam999	E
zam888	C
zam888	D

Hostname	Ext.-Platte
zam144	F
zam144	G
zam999	F
zam999	H
zam888	G
zam888	H

Dies ist die Darstellung in vierter Normalform. Im Gegensatz zu den bisher behandelten Normalformen garantiert die vierte Normalform keine verlustfreie Zerlegung. In Einzelfällen liefert der Verbund der Relationenschemata in der vierten Normalform nicht die ursprüngliche Relation zurück. Da solche Fälle selten sind, wird auf ein Beispiel verzichtet.

2.5 Vergleich der Datenmodelle

Eine Bewertung der beiden Modelle kann nur unter Berücksichtigung der Anwendung erfolgen. So kann es beim relationalen Datenmodell auch sinnvoll sein, auf eine Normalisierung völlig zu verzichten, falls die Datenbank hauptsächlich für Anfragen benutzt und nicht häufig geändert wird.

Folgende Kriterien werden bei der Wahl des Datenmodells berücksichtigt:

1. Abbildung der Inventarstruktur auf das Datenmodell
2. Zugriffsgeschwindigkeit auf die Daten (lesen)
3. Flexibilität bei Änderungen der Daten

Abbildung 2.6 zeigt noch einmal einen Workstationverbund vor und nach einer möglichen Änderung der Konfiguration.

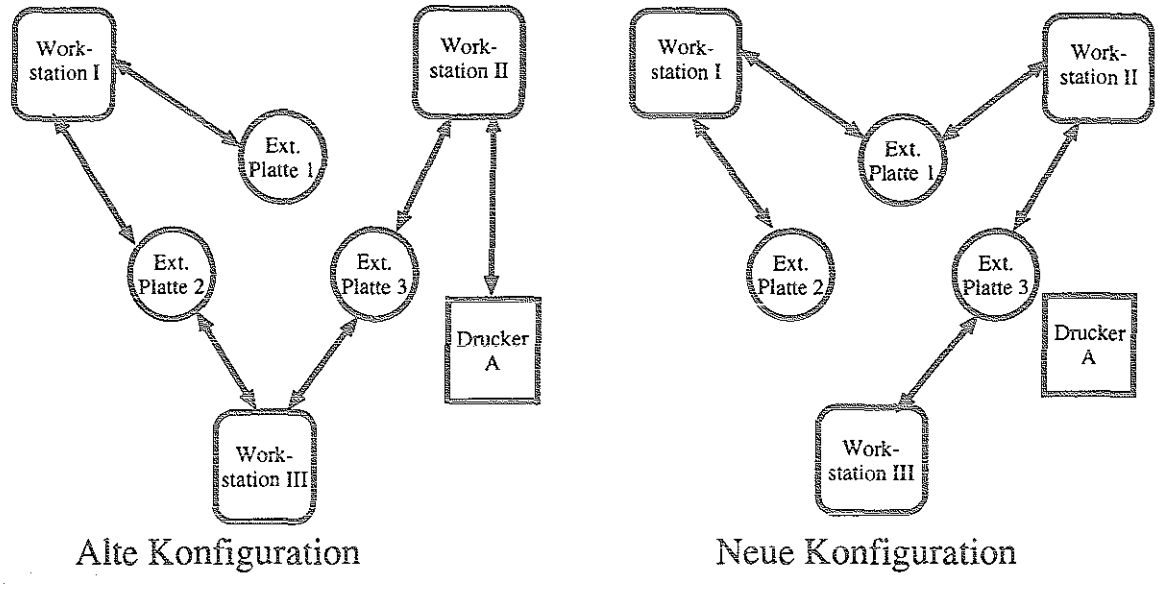


Abbildung 2.7: Workstationverbund

Durch das Zerlegen der Relationenschemata bei einem Normalisierungsprozeß müssen viele neue Relationen eingerichtet werden, die bei Anfragen eventuell verknüpft werden müssen. Enthalten die Tabellen auch noch eine hohe Anzahl an Zeilen, können die Antwortzeiten recht hoch werden [?].

Die größere Flexibilität der Datenstrukturen bietet das relationale Datenmodell. Hier können mehrere Anwendungen ihre individuellen Sichten auf den gleichen Datenbestand definieren. Ein Update ist durch das Ändern der entsprechenden Tabelle(n) einfach zu realisieren.

Falls die Anwendung festliegt und hierarchische Struktur besitzt, bietet das hierarchische Datenmodell einen unkomplizierten und schnellen Zugriff auf die Daten. Der Zugriff auf die Daten ist jedoch abhängig von der Wahl der Wurzel. Diese Wahl ist zu Beginn des Datenbankentwurfes festzulegen und kann zur Laufzeit der Anwendung nicht ohne weiteres geändert werden. Weiterhin wird die Abbildung der Daten auf das hierarchische Datenmodell durch die nicht-hierarchische Struktur des abzubildenden Inventars erschwert.

Für das Inventory-Management-System wurde das relationale Datenmodell gewählt, da die Abbildung der Konfiguration leichter durchzuführen ist als im hierarchischen Datenmodell. Eine Änderung der Daten wie in Abb. 2.6 kann durch die entsprechende Änderung in den Tabellen leichter realisiert werden.

Im hierarchischen Datenmodell hätte man den Vorteil gegenüber dem relationalen Datenmodell mit der kleineren Zahl an Tabellen und der damit verbundenen kleineren Zugriffszeit, nicht vollständig nutzen können, da auch hier durch die komplizierte Verzeigerung, speziell der externen Platten, Zwischentabellen hätten angelegt werden müssen.

Kapitel 3

Die Datenbank XDB

Da sich die Struktur der Daten im Inventory-Manangement-System nicht ohne umfangreiche Erweiterungen auf eine kommerzielle Datenbank abbilden läßt, wurde eine eigene Datenbank implementiert. Als Programmiersprache für die Implementation des Gesamtsystems wurde 'Tcl/Tk' verwendet [?] [?]. In diesem Kapitel werden der Aufbau der Datenbank und die zur Verfügung stehenden Dienste beschrieben.

3.1 Die Beschreibung der Daten

3.1.1 Data-Files

Jeder Datensatz im Sinne einer relationalen Datenbank wird in einer separaten Datei abgespeichert, im folgenden als Data–File bezeichnet. Im Data–File sind alle den Datensatz betreffenden Daten in der Form: *Feldname # Feldinhalt* enthalten. Dabei sind folgende Regeln einzuhalten:

- Die Feldnamen in einem Data–File müssen eindeutig sein, d.h. kein Feldname darf mehr als einmal in einem Data–File stehen.
- Es muß ein Feldname *object* existieren, der Inhalt des Feldes darf nicht leer sein.
- Es muß ein Feldname *key* existieren, der Inhalt des Feldes darf nicht leer sein
- Das Zeichen # darf weder im Feldnamen noch in dessen Inhalt vorkommen, es fungiert als Trennzeichen

Das Feld *object* bezeichnet eindeutig die Menge der Feldnamen (Attribute). Das Feld *key* enthält den Primärschlüssel, er bezeichnet eindeutig die Feldinhalte dieses Objektes. Er wird auch zur Namensgebung des Data-Files benutzt (die Daten werden in der Datei *key.data* gespeichert). Im Inventory-Management-System wird die KFAid als

Primärschlüssel benutzt, dabei handelt es sich um eine ganze Zahl, die zur Laufzeit vom Programm für jeden neuen Gegenstand vergeben wird. Die letzte vergebene KFAid wird in einer Datei festgehalten, wird eine neue KFAid benötigt, wird der Inhalt dieser Datei inkrementiert und die neue KFAid abgespeichert. Eine einmal vergebene KFAid wird auch bei Ausmusterung eines Gegenstandes nicht wieder freigegeben.

Abbildung 3.1 zeigt den Ausschnitt aus einem Data–File :

Abbildung 3.1: Ausschnitt eines Data-Files

Die Reihenfolge der Attribute und ihrer Daten in der Datei ist beliebig.

Eine weitere Einschränkung muß in der Zahl der Dateien gemacht werden, die vom System verarbeitet werden können. Die Zahl der in einem Unix-File-System handhabbaren Files hängt u.a. von den verfügbaren "inodes" ab [?]. Für das Inventory-Management-System wird angenommen, daß diese Zahl ausreichend hoch ist.

Die Datenbank kann neben dem Primärschlüssel *key* weitere Sekundärschlüssel enthalten. Diese Sekundärschlüsel werden in den Data-Files jedoch nicht weiter gekennzeichnet. Sie werden vielmehr durch das Table-File beschrieben, das im nächsten Abschnitt erläutert wird.

3.1.2 Table-Files

Das Table-File stellt eine Art Inhaltsverzeichnis der Datenbank dar. In dem Table-File werden alle Inventargegenstände gespeichert, jedoch nicht alle Daten zu diesen Gegenständen. Folgende Daten müssen mindestens im Table-File enthalten sein.

- Der Primärschlüssel
- Alle Sekundärschlüssel
- Der Inhalt des Feldes *object*

Zu dem Table-File existiert eine Table-Description, in der für jedes Feld die Eigenschaften der im Table-File gespeicherten Felder enthalten sind. Dadurch ist der Aufbau des Table-Files genau festgelegt. Durch die Table-Description werden auch die Sekundärschlüssel definiert. Die Table-Description ist unter dem Namen *tablefilename.dd* gespeichert.

Die Zahl der Table-Files in der Datenbank ist beliebig unter der Voraussetzung daß jeder Datensatz mindestens in einem Table-File vorkommt und in jedem Table-File die Datensätze eindeutig sind.

Abbildung 3.2 zeigt eine Table-Description:

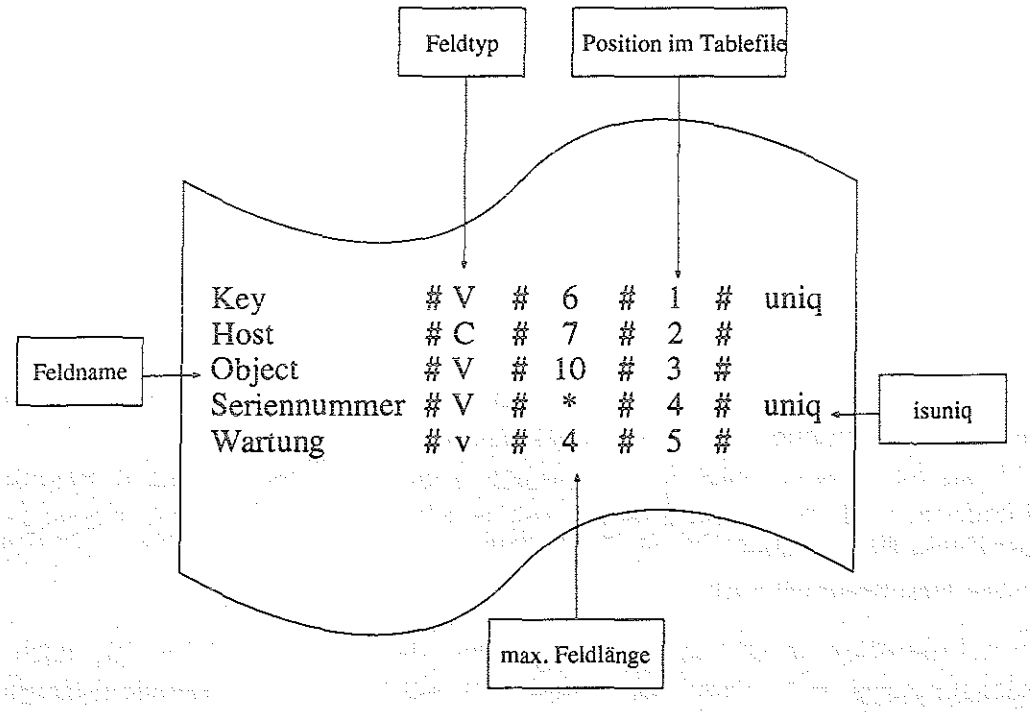


Abbildung 3.2: Ausschnitt einer Table-Description

Die Beschreibung eines Feldes setzt sich aus folgenden Angaben zusammen:

1. Der Feldname (dieser Name muß mit dem Feldnamen im Data–File identisch sein)
2. Der Feldtyp
3. Die Feldlänge
4. Die Position im Table-File
5. Eine Angabe, ob das Feld die Unique-Eigenschaft besitzt, also Sekundärschlüssel ist

Folgende Feldtypen können gewählt werden:

- **v** ...Character-Feld variabler Länge, Inhalt kann leer bleiben
- **V** ...Character-Feld variabler Länge, Feld muß beschrieben sein
- **c** ...Character-Feld fester Länge, Inhalt kann leer bleiben
- **C** ...Character-Feld fester Länge, Feld muß beschrieben sein
- **u** ...Integer-Feld ohne Vorzeichen, Inhalt kann leer bleiben
- **U** ...Integer-Feld ohne Vorzeichen, Feld muß beschrieben sein

Die Feldlänge eines Datenfeldes bezieht sich auf das Table-File . Der Wert im Data–File kann sich davon unterscheiden. Beispiel: Im Data–File steht der Eintrag

Hostname # DiesisteinlangerHostname .

In der Table-Description wurde das Feld vom Typ 'V' mit einer Länge von 12 definiert, dann steht im Table-File : *Diesisteinla*
Ein '*' als Feldlängenangabe bei den Typen V bzw. v bedeutet: keine Begrenzung der Länge im Table-File . Die Positionsangabe kennzeichnet die Nummer des mit dem Trennzeichen # unterteilten Abschnittes. Abbildung 3.3 zeigt ein Table-File , das sich auf die Table-Description aus Abbildung 3.2 bezieht.

Werden Datensätze in der Datenbank gesucht, dann wird im Table-File nach dem Primärschlüssel gesucht. Durch diesen "key" kann dann das entsprechende Data–File selektiert werden. Der Benutzer der Datenbank benötigt die Kenntnis des Primärschlüssels nicht.

Beispiel: Im Inventory-Management-System übernimmt die KFAid die Aufgabe des Primärschlüssels. Die KFAid wird vom Programm vergeben. Dadurch wird die Eindeutigkeit des Primärschlüssels garantiert. Da dieser Schlüssel jedoch keinen Bezug zu dem

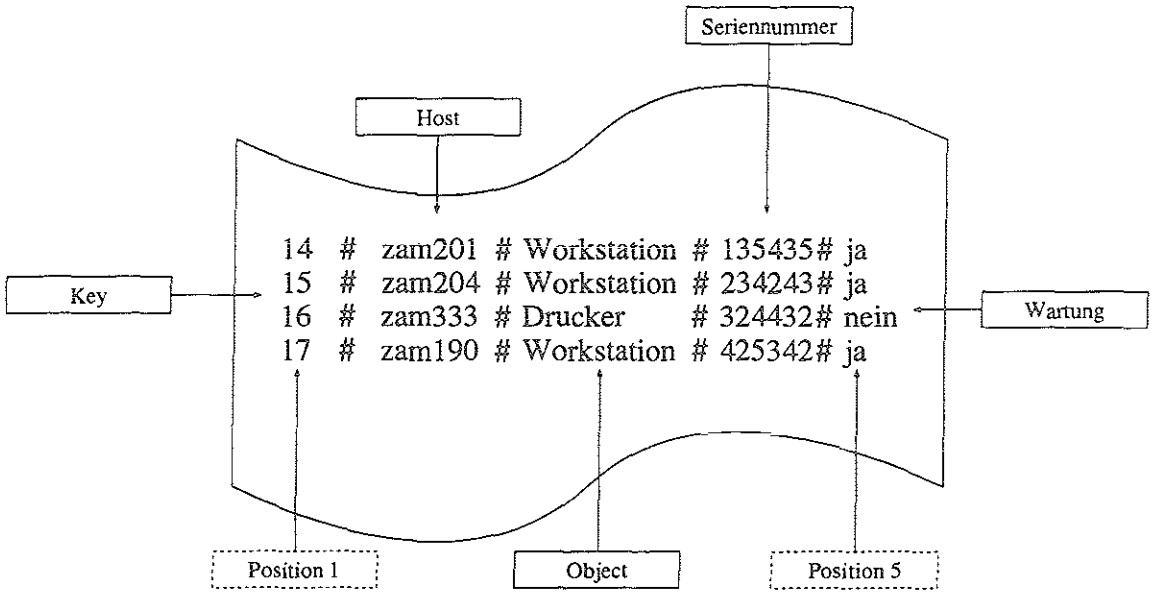


Abbildung 3.3: Ausschnitt eines Table-Files

Inhalt des Datensatzes hat, ist er für den Benutzer unhandlich. Der Benutzer selektiert die Daten deshalb nach einem anschaulichen Sekundärschlüssel, z.B. dem Hostnamen einer Workstation. Das Programm sucht nun im Table-File nach diesem Sekundärschlüssel. Nach dem Auffinden der entsprechenden Zeile kann es nun über den Primärschlüssel die gesamten Daten an den Benutzer weitergeben. Die Einführung eines "künstlichen" Primärschlüssels wurde gewählt, da sich alle Sekundärschlüssel während der Lebenszeit eines Gerätes ändern können.

3.1.3 Link-Files

Die Datenbank XDB bietet die Möglichkeit, Verbindungen zwischen einzelnen Datensätzen herzustellen. Diese Möglichkeit wurde eingeführt, um die Vernetzung der Inventargeräte auf die Datenstruktur abbilden zu können. In einer Datei, die den Namen *tablefile.pointer* trägt, werden diese Verbindungen festgehalten. Der Eintrag in ein solches Link-File hat folgendes Format:

$$key_1 \# key_2 \# object_1 \# object_2 \# Wertigkeit$$

Die folgende Abbildung zeigt einen Ausschnitt aus einem Link-File , wie es z.B. im Inventory-Management-System vorkommen könnte.

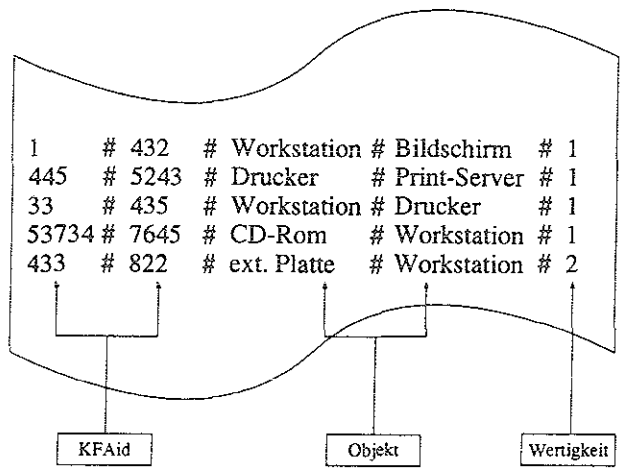


Abbildung 3.4: Ausschnitt eines Link-Files

Die Wertigkeit gibt dabei an, wie oft diese Verbindung vorliegt.

Beispiel: Es liegt eine externe Platte mit zwei Anschlußmöglichkeiten vor. Wird diese Platte nun mit einer Workstation verbunden, dann erfolgt ein entsprechender Eintrag in das Link-File mit der Wertigkeit 1. Wird nun auch der zweite Anschluß der Platte mit dieser Workstation verbunden, dann wird kein neuer Eintrag in das Link-File generiert, sondern die Wertigkeit des vorherigen Eintrages wird um eins erhöht.

Die Datenbank XDB macht keine Einschränkungen hinsichtlich der Verbindungsstruktur der Daten. Ein Gerät kann beliebig oft mit einem anderen logisch verbunden werden. Weiterhin ist es möglich, ein Gerät mit sich selbst zu verbinden. Die Einschränkungen werden außerhalb der Datenbank festgelegt und geprüft.

Die Reihenfolge des Eintrages in das Link-File (welcher Key zuerst gespeichert wird) wird durch einen numerischen Vergleich der beiden Keys festgelegt. Der numerisch kleinere Key wird zuerst gespeichert. Durch diese Ordnung kann bei einem Lesen des Link-File der gesuchte Eintrag sofort gefunden werden.

3.2 Die Dienste der Datenbank

3.2.1 Übersicht

Folgende Unterprogramme und Dienste der Datenbank stehen zur Verfügung:

- **xdb_init**

  Initialisiert die Datenbankbenutzung
  Aufruf: *xdb_init*

- **xdb_load_dd**

  Lädt die Datenbeschreibung für ein oder mehrere Table-Files
  Aufruf: *xdb_load_dd path tablenamelist*

- **xdb_is_uniq**

  Prüft für die aktuelle Datei, ob die Unique-Felder eindeutig sind, d.h. ob sie auch wirklich die Unique-Eigenschaft besitzen
  Aufruf: *xdb_is_uniq path tablenamelist infodata filepos*

- **xdb_add_row**

  Fügt einen neuen Gegenstand ein.
  Aufruf: *xdb_add_row path tablefile infodata*

- **xdb_update_row**

  Ändert einen bestehenden Datensatz.
  Aufruf: *xdb_update_row path tablefile infodata*

- **xdb_delete_row**

  Löscht einen Datensatz.
  Aufruf: *xdb_delete_row path tablefile infodata*

- **xdb_link**

  Stellt eine Verbindung zwischen zwei Geräten her
  Aufruf: *xdb_link path tablefile infodata1 infodata2*

- **xdb_unlink**

  Löst die Verbindung zwischen zwei Geräten
  Aufruf: *xdb_unlink path tablefile infodata1 infodata2*

- **xdb_checklink**

  Prüft, mit welchen anderen Geräten das Gerät verbunden ist.
  Aufruf: *xdb_checklink path tablefile key*

Die Parameter bei den Prozeduraufrufen haben folgende Bedeutung:
path: Pfadangabe, wo das File gespeichert ist
tablenamelist: Liste mit Table-Descriptions
infodata: Array mit den Datenfeldern und Inhalt entsprechend eines Data–File
filepos: Position im Table-File, an der eine Unique-Übereinstimmung gefunden wurde
key: Primärschlüssel eines Datensatzes

3.2.2 Beschreibung

Alle Prozeduren arbeiten mit der globalen Variablen *xdb*. Um diese Variable benutzen zu können, wird die Prozedur xdb_init aufgerufen. Diese Prozedur initialisiert die Variable *xdb* neu. In dieser Variable wird die Beschreibung eines Table-Files abgelegt.

Möchte man mit einem Table-File arbeiten, muß die Prozedur xdb_load_dd aufgerufen werden. Dieser Prozedur wird eine Liste von Table-File-Namen übergeben, mit denen man arbeiten möchte. Die Prozedur xdb_load_dd prüft nun, ob die Datenbeschreibungen zu diesen Table-Files existieren und ob sie nicht schon früher geladen wurden. Findet xdb_load_dd eine Datenbeschreibung, die noch nicht in der Variablen *xdb* festgehalten wurde, wird die Table-Description der Variablen *xdb* hinzugefügt. Bezogen auf folgende Table-Description:

name	type	length	pos	is_uniq
Key	V	6	1	uniq
Host	C	7	2	
Object	V	10	3	
Seriennr.	V	10	4	uniq
Wartung	v	4	5	

wird die Variable wie folgt besetzt:

xdb(dd,tablename,host,type)=V
xdb(dd,tablename,host,pos)=2
.
.
.
xdb(dd,tablename,wartung,unique)=no

Die Prozedur xdb_load_dd wird immer dann aufgerufen, wenn lesend auf ein Table-File zugegriffen wird.

Die Prozedur xdb_is_uniq wird aufgerufen, falls Daten geändert oder neue Daten hinzugefügt werden. Beim Aufruf wird sowohl das betroffene Table-File als auch der Datensatz übergeben. Die Prozedur sucht nun in der xdb-Variablen die Felder, die die Unique-Eigenschaft besitzen. Für diese Felder wird im Table-File nach Übereinstimmungen gesucht. Wird eine Übereinstimmung gefunden, dann wird die Zeile im Table-File in der Variablen *filepos* festgehalten. Die Suche wird mit dem nächsten Feld fortgesetzt, das die Unique-Eigenschaft besitzt. Wird eine zweite Übereinstimmung gefunden, dann endet die Suche. Zurückgegeben wird die Zahl der Übereinstimmungen, mit deren Hilfe das rufende Programm entscheiden kann, ob die Konsistenz der Datenbank erhalten bleibt.

Es gibt zwei Prozeduren, die xdb_is_uniq aufrufen:

1. xdb_add_row
2. xdb_update_row

Die Prozedur xdb_add_row fügt einen neuen Datensatz in die Datenbank ein. Der von xdb_is_uniq zurückgegebene Wert muß 0 sein. In diesem Fall wird ein Data–File im Zielverzeichnis angelegt. Dann wird der Eintrag in das Table-File gemäß der Beschreibung in der xdb-Variablen angelegt. Kann dieser Eintrag nicht erzeugt werden (z.B. weil ein Eintrag in einem Feld fehlt, dessen Inhalt nicht leer bleiben darf), wird das Data–File wieder gelöscht. Im anderen Fall wird der erzeugte Eintrag an das Table-File angehängt. Das Ergebnis der Aktion wird dem rufenden Programm mitgeteilt.

Die Prozedur xdb_update_row ändert einen Eintrag im Table-File und das entsprechende Data–File . Dazu muß das Ergebnis der Prozedur xdb_is_uniq genau 1 sein. Die zu ändernde Zeile im Table-File ist in der Variablen filepos festgehalten. Wie in der Prozedur xdb_add_row wird nun das Data–File erzeugt (wobei das alte Data–File überschrieben wird) und der Eintrag im Table-File vorbereitet. Im Gegensatz zu der Prozedur xdb_add_row muß aber jetzt ein neues Table-File erzeugt werden, da der Eintrag nicht an das Ende des Files angehängt werden kann, sondern an der Stelle *filepos* eingefügt werden muß. Dabei wird der alte Eintrag entfernt.

Mit der Prozedur xdb_delete_row können Einträge im Table-File sowie das entsprechende Data–File gelöscht werden. Datensätze, die gelöscht werden, dürfen nicht mit anderen Datensätzen verbunden sein. Dazu muß vor dem Löschen durch die Prozedur xdb_checklink geprüft werden, ob Verbindungen mit anderen Datensätzen bestehen. In diesem Fall muß der Auftrag zurückgewiesen werden.

Mit der Prozedur xdb_link können Verbindungen zwischen Datensätzen hergestellt werden. Dabei wird geprüft, ob es bereits Verbindungen zwischen den zwei übergebenen Datensätzen gibt. In diesem Fall wird die Wertigkeit der Verbindung um eins erhöht. Handelt es sich um eine neue Verbindung, wird ein neuer Eintrag mit der Wertigkeit eins an das Link-File angehängt.

Die Prozedur xdb_unlink wird benötigt, um bestehende Verbindungen aufzuheben. Dabei wird die Wertigkeit der Verbindung um eins erniedrigt oder, falls die Verbindung bereits die Wertigkeit eins hatte, der Eintrag aus dem Link-File gelöscht.

Die Prozedur xdb_checklink prüft für einen Datensatz, ob er Verbindungen zu anderen Datensätzen hat. Zurückgegeben wird eine Liste, die die Primärschlüssel und den Inhalt der Felder *object* von den Datensätzen enthält, zu denen eine Verbindung besteht.

Alle Prozeduren prüfen nach, ob die Files, mit denen sie arbeiten, überhaupt existent sind. Ist das nicht der Fall, dann wird das File entweder neu angelegt ,oder es geht eine Fehlermeldung an das rufende Programm. Alle Meldungen, die an das rufende Programm gehen, haben folgende Form:

i\n Kurzbeschreibung

Dabei bedeutet i=0, daß der Auftrag erfolgreich bearbeitet wurde, i>0 signalisiert einen Fehler.

Abb. 3.5 zeigt den Zugriff der Dienste auf die verschiedenen Files.

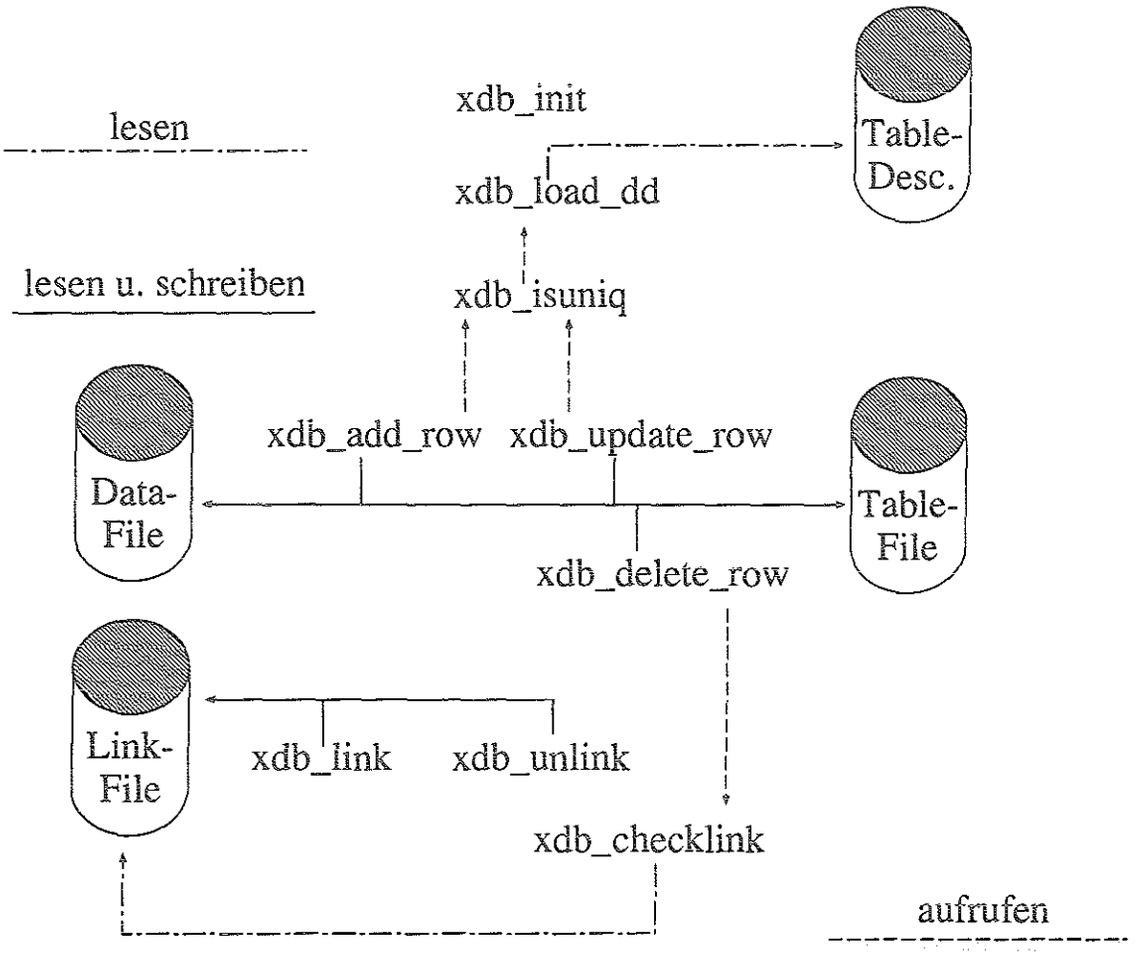


Abbildung 3.5: Die Datenbank XDB

Kapitel 4

Das Client-Server-Modell

Da im Inventory-Management-System mehrere Benutzer an verschiedenen Workstations mit den Daten arbeiten, muß sichergestellt werden, daß die Konsistenz der Datenbank erhalten bleibt. Dies geschieht durch die Client-Server-Lösung. Im folgenden Kapitel wird die Funktion des Client-Server-Konzeptes beschrieben. Dabei wird begründet, warum dieses Modell benutzt wurde. Weiterhin wird auf die sich ergebenden Sicherheitsaspekte eingegangen [?].

4.1 Zugriff auf die Datenbank

Beim Betrieb des Inventory-Management-Systems arbeiten gleichzeitig viele Benutzer mit der Datenbank. Dabei müssen auftretende Konsistenz-Probleme vermieden werden. Zwar können mehrere Benutzer die Daten gleichzeitig lesen, aber bei der Änderung von Daten muß sichergestellt werden, daß es zu keinen Konflikten kommt. Es darf z.B. nicht geschehen, daß ein Benutzer eine Workstation mit einem Drucker ausstattet und ein anderer Benutzer diese Workstation gerade löscht. Ist die Workstation gelöscht und aus dem Verzeichnis der verfügbaren Geräte entfernt worden, dann führt eine Änderung der Workstation-Daten zu einem fehlerhaften Inhalt der Datenbank, da nun versucht wird, in eine nicht mehr vorhandene Datei zu schreiben.

Ein Weg zur Vermeidung dieser Problematik liegt in der Vergabe von Zugriffsrechten für die einzelnen Dateien. Dabei wird z.B. in die Daten für eine Workstation die User-ID und die Group-ID des Benutzers eingefügt. Ein schreibender Zugriff kann dann nur noch auf die eigenen Daten gemacht werden.

Trotzdem muß es Benutzer mit privilegierten Rechten geben, die sowohl lesenden als auch schreibenden Zugriff zu allen Daten haben. Weiterhin gibt es Geräte, die nicht nur von einem Anwender genutzt werden, z.B. externe Platten. Daher muß ein anderer Weg gefunden werden, die Konsistenz der Datenbank zu gewährleisten. Eine Möglichkeit besteht darin, eine Instanz zu schaffen, die alle Schreibaufträge der Datenbankbenutzer sequentiell bearbeitet und dabei die Konsistenz der Datenbank sicherstellt. Diese Instanz

wird als Server bezeichnet. Die Zusammenarbeit zwischen dem Server und den Clients, d.h. den Datenbankbenutzern, ist in Abbildung 4.1 dargestellt und wird im folgenden Abschnitt beschrieben.

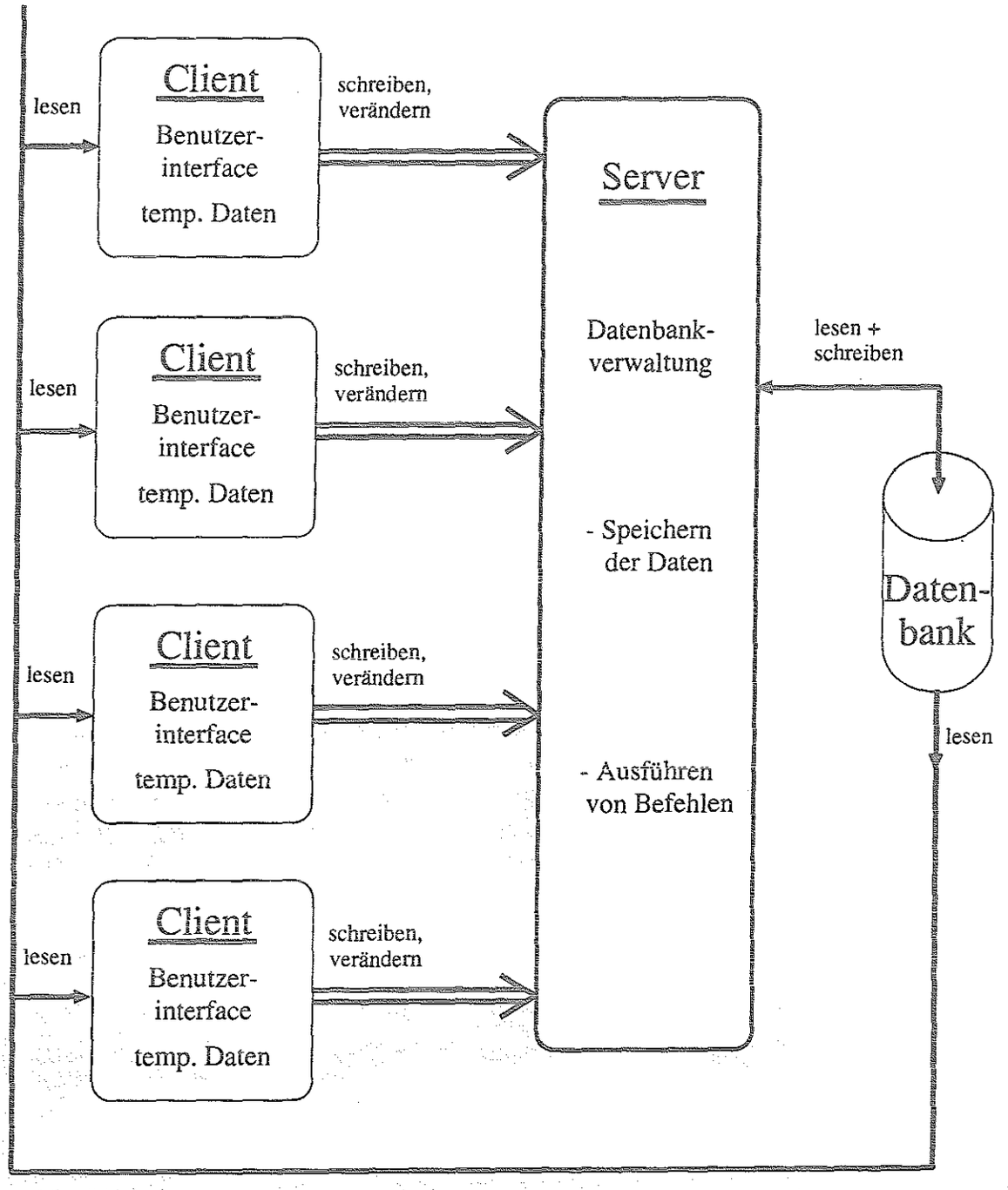


Abbildung 4.1: Das Client-Server-Modell

4.2 Beschreibung der Client-Server-Umgebung

4.2.1 Der Client

Der Benutzer der Datenbank übernimmt die Rolle eines Clients. Clients sind Prozesse auf unterschiedlichen Rechnern in einem Rechnerverbund. Der Client kann auf alle Daten lesend zugreifen. Dabei greift er mit Hilfe des Network File Systems (NFS) direkt auf den Datenbestand zu, ohne eine Anfrage an den Server zu richten. Über NFS stehen dem Client nicht nur die Daten, sondern auch die Datenbank-Programme (Datenbankdienste) zur Verfügung. Dabei handelt es sich ausschließlich um die Dienste, die keinen schreibenden Zugriff auf die Daten vornehmen.

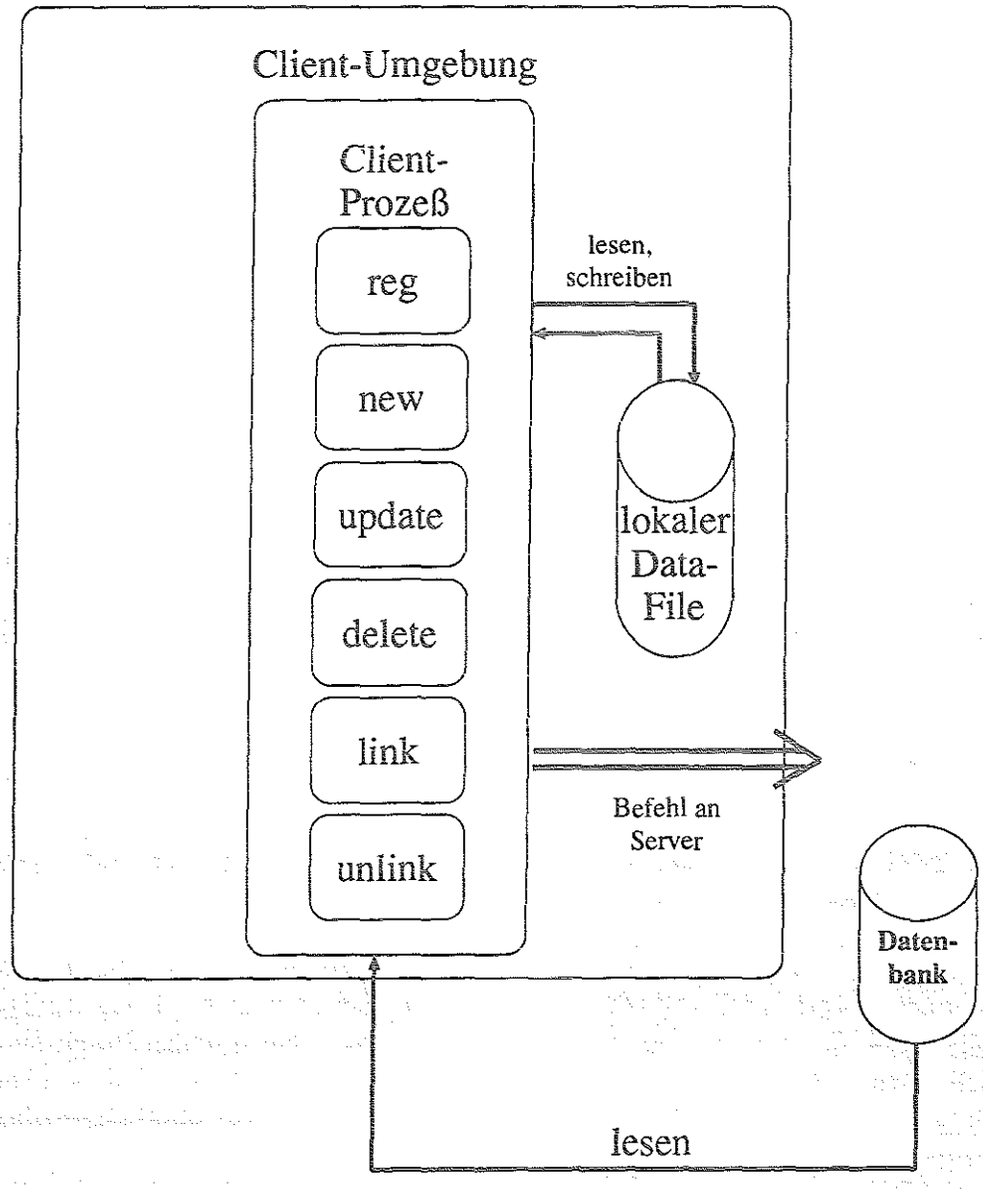


Abbildung 4.2: Die Client-Umgebung

Möchte der Client die Daten ändern, neue Daten hinzufügen oder Daten löschen, so muß er mit dem Server kommunizieren (s. Abb. 4.2). Dabei kann der Client folgende Dienste des Servers in Anspruch nehmen:

- Bei der ersten Anfrage an den Server kann er eine Registrierung beantragen (REG).
- Er kann den Server beauftragen, einen Datensatz zu löschen (DELETE).
- Er kann den Server beauftragen, einen neuen Datensatz in die Datenbank hinzuzufügen (NEW).
- Er kann den Server beauftragen, einen Datensatz zu ändern (UPDATE).
- Er kann den Server beauftragen, zwei Datensätze miteinander zu verbinden (LINK).
- Er kann den Server beauftragen, die Verbindung zwischen zwei Datensätzen aufzuheben (UNLINLK).

Jeder dieser Dienste wird vom Client mit dem UNIX-Befehl "rsh" an den Server geschickt ohne zu prüfen, ob eine Berechtigung zu diesen Diensten vorliegt. Dafür muß der Client sicherstellen, daß die Daten der durch den Server vorgegebenen Syntax entsprechen. Die Daten werden vom Client in ein temporäres File geschrieben; der Dateiname erhält zur eindeutigen Identifikation durch den Server den Suffix

hostname.process_id

In dem File sind neben den Daten auch die User-ID und, falls es sich um bereits existierende Daten handelt, die KFAid gespeichert. Die KFAid besitzt den Status des Primärschlüssels und wird vom Server vergeben. Dieses Prinzip, das den Client bzw. den Benutzer davon entbindet, ein Attribut der Datenbestände als Primärschlüssel auszuwählen, garantiert die Invarianz des Schlüssels. Ein solcher Schlüssel wird als *Surrogat* bezeichnet [?].

Ein Surrogat ist ein vom System vergebener, invarianter Merkmalwert, welcher jedes Tupel der Datenbank eindeutig identifiziert.

Das die Daten enthaltende File wird bei den Diensten NEW und UPDATE über den UNIX-Befehl "rcp" in ein vorgegebenes Verzeichnis des Servers kopiert.

Benutzt der Client zum ersten Mal die Datenbank XDB, so muß er sich zunächst zur Einhaltung der Zugriffssicherheit durch den Server registrieren lassen. Dazu sendet er den Auftrag REG an den Server. Von ihm erhält er ein Paßwort, das in seiner Home-Directory gespeichert wird. Bei allen folgenden Sitzungen wird auf dieses Paßwort zugegriffen. Der Wert dieses Paßwortes bleibt für den Client ohne Bedeutung, da er sämtliche Aufträge an den Server nur mit seiner User-ID abgibt.

Der Client erhält vom Server eine Meldung über den Erfolg bzw. Mißerfolg seines Auftrages.

4.2.2 Der Server

Der Server ist ein auf einem eigenen Rechner ständig laufendes Programm und steht allen Clients zur Verfügung. Abbildung 4.2 beschreibt die Server-Umgebung.

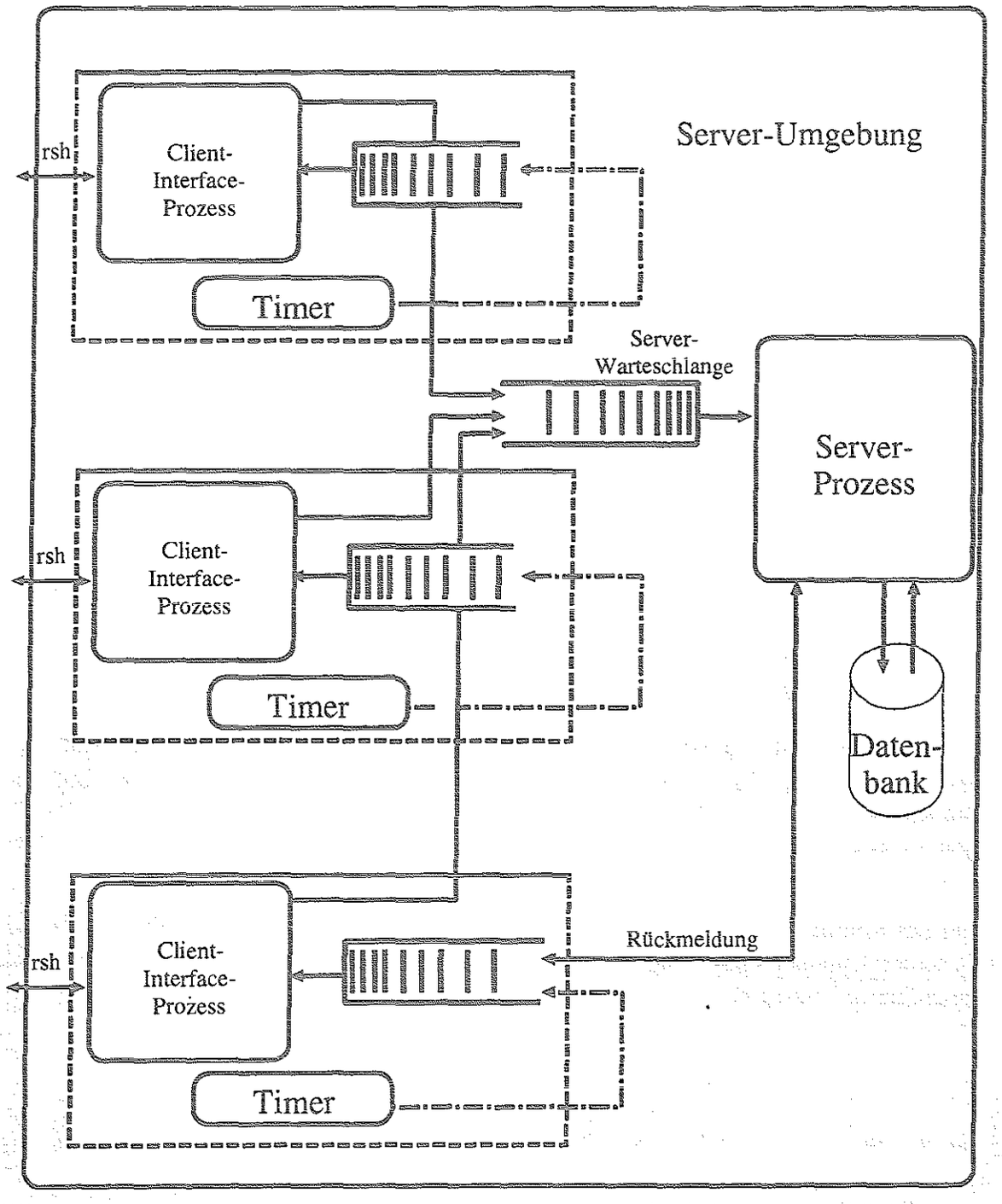


Abbildung 4.3: Die Server-Umgebung

Der Server erzeugt zunächst eine FIFO-Warteschlange und wartet dann auf Eingänge in diese Warteschlange. Sobald von einem Client ein Auftrag an den Server gestellt wird, liegt diese Anfrage in der Warteschlange vor. Der Server liest nun eine Zeile aus der Warteschlange, die abhängig vom übergebenen Kommando noch weitere Informationen enthält. Folgende Befehle können dem Server übergeben werden:

- REG # cpid # uid_name # ident
- NEW # cpid # ident # tempfile
- UPDATE # cpid # ident # tempfile
- DELETE # cpid # ident # tempfile
- LINK # cpid # ident # tempfile_1 tempfile_2
- UNLINK # cpid # ident # tempfile_1 tempfile_2
- FINISH

Die Doppelkreuze haben dabei die Aufgabe eines Trennzeichens. Im Feld 'cpid' ist die Prozeß-ID des rufenden Client-Interface-Prozesses enthalten, im Feld 'ident' die User-ID und die Group-ID des Benutzers.

Mit der Prozeß-ID des rufenden Prozesses kann der Server eine eindeutige Return-FIFO-Warteschlange adressieren, um dem rufenden Prozeß eine Rückmeldung über den Erfolg bzw. Mißerfolg seiner Arbeit zu geben. Diese Return-Pipe wurde von dem rufenden Prozeß bereits erzeugt.

Da der Server Aufträge von vielen Clients entgegenzunehmen hat, muß er die Aufträge serialisieren. Empfängt der Server einen Auftrag über seine Pipe, dann öffnet er zunächst die Return-Pipe und sendet dem rufenden Prozeß eine Empfangsbestätigung "Working". Dann ruft er, abhängig vom Auftrag, eine Unterprozedur auf und beginnt mit der Auftragsabwicklung.

Für das Kommando REG erzeugt der Server zunächst ein neues Paßwort. Dann öffnet er in seinem eigenen Verzeichnis die Datei *server.access* und hängt an das Ende der Datei einen neuen Eintrag in der Form:

uid_name # ident # passwd

an. Anschließend übergibt er dem Client das neue Paßwort und begibt sich wieder in den Status *Blocking Read.* Der Client legt das erzeugte Paßwort in der Datei ".rzman_pass" der Client-Home-Directory ab.

Erhält der Server das Kommando NEW, gibt er zunächst wieder eine Empfangsbestätigung an den rufenden Prozeß. Dann ruft er die Unterprozedur call_new auf. Da es sich um neue Daten handelt, wird zunächst ein Primärschlüssel, die KFAid, vergeben. Dieser Schlüssel

wird den Daten hinzugefügt. Die neuen Daten sind in einem temporären File enthalten, das zuvor vom Client mit dem 'rcp'-Befehl in eine Directory des Servers transferiert wurde. Der Name dieser Datei wird dem Server mit dem Parameter tempfile übergeben. Die Daten werden nun dem Datenbankdienst xdb_add_row übergeben. Dort wird eine Prüfung der Unique-Felder vorgenommen. Verläuft diese erfolgreich , werden die Daten im Zielverzeichnis unter Angabe des Primärschlüssels abgelegt und in dem Katalog all_dev aufgenommen. Sie sind jetzt für alle Benutzer verfügbar. Der Server gibt nun über die Return-Pipe eine Meldung über das Resultat des Auftrags zurück.

In gleicher Weise werden die Aufträge UPDATE und DELETE bearbeitet. Beim Auftrag DELETE wird jedoch zuerst der Eintrag im Katalog all_dev gelöscht und dann erst das Data–File . Dadurch wird vermieden, daß ein Benutzer Daten liest, die im Katalog zwar noch existieren, die aber als Data–File vom Server soeben gelöscht wurden. Aus dem gleichen Grund wird bei den Aufträgen NEW und UPDATE auch zuerst das Data–File angelegt und dann erst der Eintrag in das Table-File erzeugt. Ein Benutzer kann nun nicht versuchen, Daten zu lesen, die zwar im Katalog bereits vorhanden sind, aber noch nicht als Data–File vorliegen. Diese Methode setzt voraus, daß der Benutzer Daten nur aus dem Katalog auswählt und nicht direkt auf die Data-Files zugreift.

Beim Kommando LINK werden zwei File-Namen, tempfile_1 und tempfile_2, übergeben. Der Server ermittelt nun anhand dieser Files die Primärschlüssel der zu verbindenden Datensätze. Mit dem ersten Schlüssel wird die Prozedur xdb_checklink aufgerufen. Diese Prozedur liefert eine Liste, die die bestehenden Verbindungen zum ersten Schlüssel beinhaltet. Die Länge dieser Liste gibt die Anzahl der Verbindungen an. Durch die Funktion *abs_link* kann die maximale Zahl an Verbindungen für den ersten Datensatz ermittelt werden. Diese Zahl ist für jedes Gerät, das verbunden werden kann, im File *link.abs* festgehalten. Für das zweite Gerät wird auf die selbe Weise festgestellt, ob es noch mit weiteren Geräten verbunden werden kann.

Als nächstes wird geprüft, ob eine Verbindung zwischen diesen Objekten zulässig ist und wieviele Verbindungen es zwischen diesen Geräten geben darf. Diese Zahl ist im File *link.rel* festgehalten. Dabei ist zu beachten, daß diese Zahl für zwei Geräte unterschiedlich sein kann: Ein CD-ROM-Laufwerk kann mit genau einer Workstation verbunden werden, eine Workstation aber mit mehreren CD-ROM-Laufwerken.

Ein Stern hinter dem Eintrag "workstation # cdrom" bedeutet, daß es beliebig viele Verbindungen zwischen diesen Geräten geben darf. Die Prüfung der Zahl zulässiger Verbindungen zwischen zwei Geräten muß daher zweimal durchgeführt werden.

Ist eine Verbindung zwischen den zwei Geräten möglich, wird die Prozedur xdb_link aufgerufen, die diese Verbindung in das Link-File einträgt.

Der Auftrag UNLINK bewirkt das Gegenteil, d.h. er löscht eine existierende Verbindung zwischen zwei Geräten.

Es existiert neben den bereits genannten Diensten ein weiteres Kommando, der Befehl FINISH, der den Server-Prozeß beendet. Dieses Kommando kann jedoch nur von privilegierten Benutzern gegeben werden.

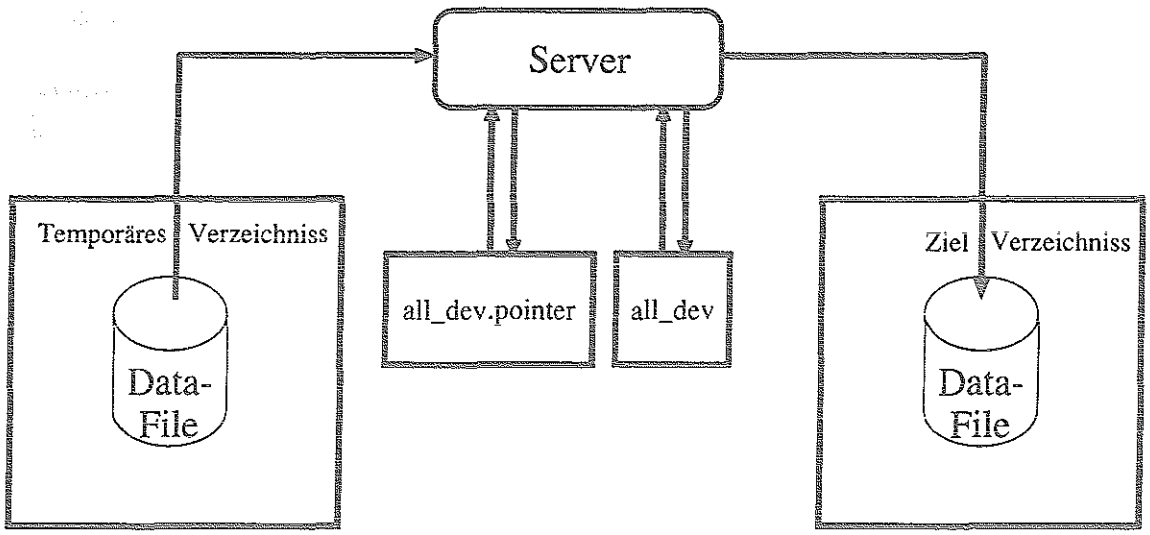


Abbildung 4.4: Durch die Arbeit des Servers betroffenen Datenbestände

Abb. 4.5 zeigt alle Server-Routinen im Überblick. Die in der Zeichnung links angeordneten Routinen können vom Client aufgerufen werden.

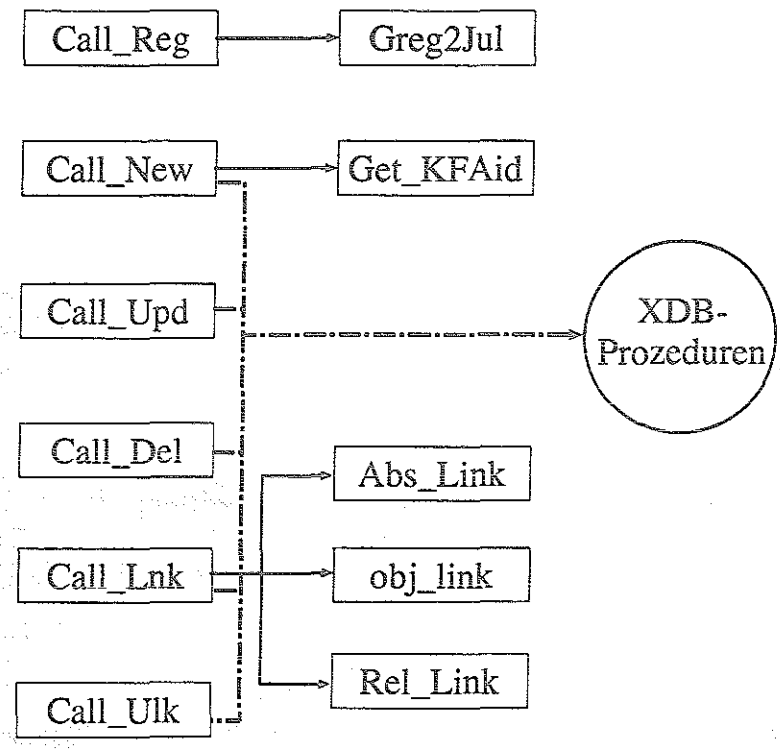


Abbildung 4.5: Die Prozeduren des Server-Prozesses

4.2.3 Die Schnittstelle zwischen Client und Server

Der Client kommuniziert nicht direkt mit dem Server, sondern er benutzt eine Schnittstelle, die ebenfalls in der Umgebung des Servers liegt. Die Schnittstelle besteht aus zwei Routinen:

- do_reg
- do_inventory

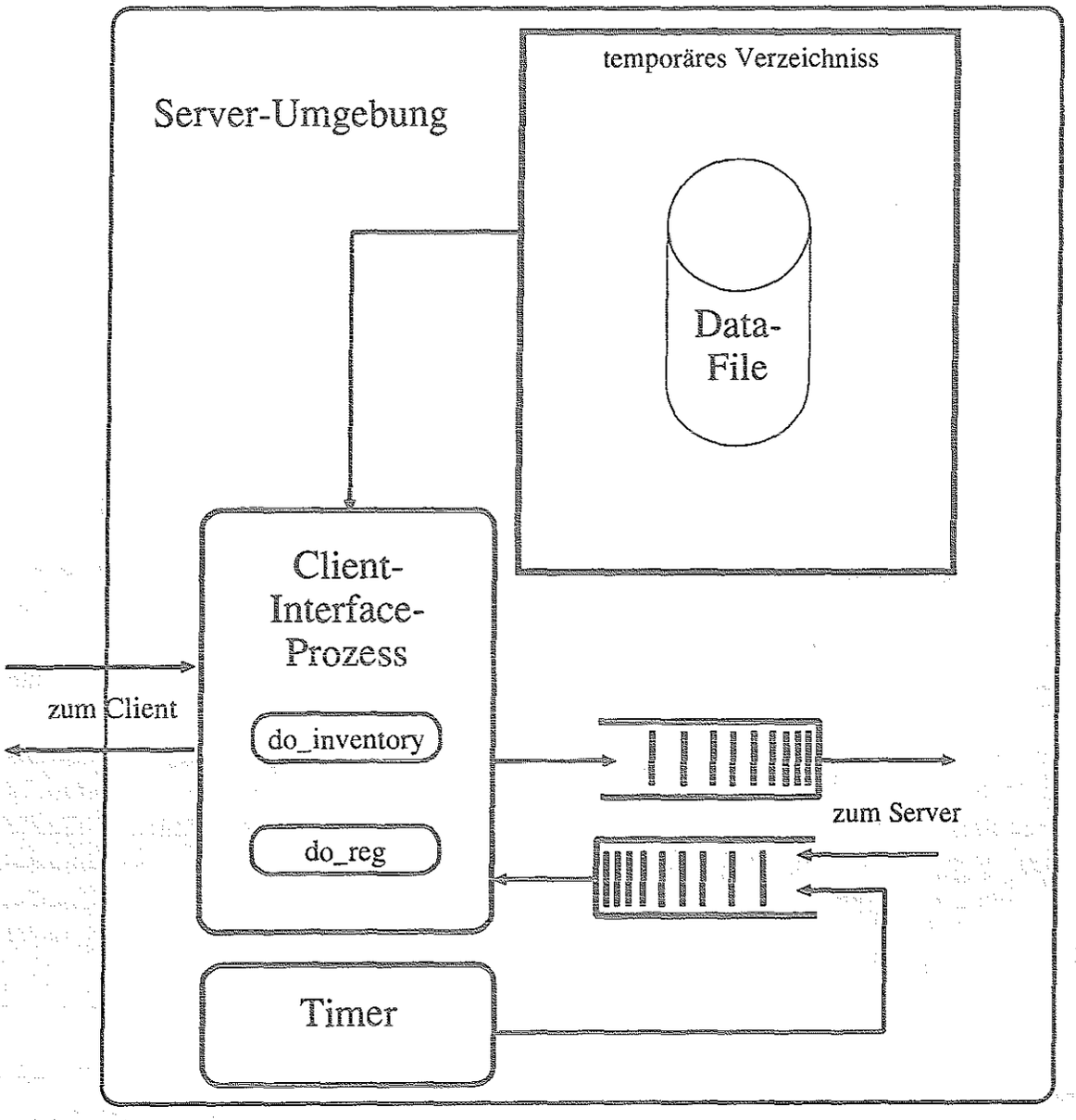


Abbildung 4.6: Die Schnittstelle

Die Schnittstelle ist notwendig, um auch den Benutzern Zugang zur Datenbank zu gewähren, die sich nicht auf demselben Host wie der Server befinden. Außerdem entlastet sie den Server von Aufgaben, die zur Bearbeitung zwar Daten aus der Server-Umgebung erfordern, aber nicht notwendig durch einen Server serialisiert werden müssen.

Die do-Routinen werden vom Client über den UNIX-Befehl "rsh" aufgerufen. Der Befehl "rsh" startet einen Prozeß auf einem entfernten Host. Dieser Host wird durch einen Parameter angegeben (Dabei kann es sich natürlich auch um den lokalen Host handeln). Der Aufruf der do-Routinen sieht wie folgt aus:

```
rsh server_host -l server-userid -n do_reg register
```

bzw.

```
rsh server_host -l user -n do_inventory cmd register passwd filename
```

Dabei ist *server_host* der Host, auf dem der Server läuft, *server-userid* der User-Name des Server-Programms und *register* die Angabe über User-ID und Group-ID des Clients. Die für den Aufruf der do_inventory-Routine zusätzlich benötigten Parameter bedeuten:

cmd : Das Kommando (NEW, UPDATE, DELETE, LINK, UNLINK)

passwd : Das Paßwort des Users

filename : Der Name des zu bearbeitenden Files

Wird die do_reg-Routine aufgerufen, erzeugt sie zunächst ihre eigene Return-Pipe, über die sie später die Antworten des Servers erhält. Dann öffnet sie die Pipe des Servers und übergibt dem Server einen REG-Auftrag. Anschließend wird im Hintergrund ein Timeout-Prozeß gestartet. Dieser Timeout-Prozeß wartet eine bestimmte Zeitspanne (bei dem Kommando REG 20 sec), bevor er in die Return-Pipe das Timeout-Signal schreibt. Der do_reg-Prozeß befindet sich inzwischen im Status *Blocking Read* und wartet auf eine Nachricht in seiner Return-Pipe. Ist dieses Signal ein Timeout, dann wird der Client über eine Nachricht dazu aufgefordert, die Anfrage zu einem späteren Zeitpunkt erneut zu stellen, da der Server nicht in der vorgegebenen Zeit geantwortet hat. Bekommt er vom Server die Empfangsbestätigung *Working*, dann initialisiert er den Timeout-Prozeß neu. Dann wartet er weiter auf den nächsten Eingang in seiner Pipe. Erhält er nun ein Timeout, dann gibt er wieder eine entsprechende Nachricht an den Client und beendet sich. Kommt eine Antwort des Servers, so wird diese an den Client weitergeleitet, bevor der do_reg-Prozeß sich beendet.

Wird die do_inventory-Routine aufgerufen, dann richtet sich die Arbeitsweise nach dem übergebenen Kommando. Für das Kommando NEW wird zunächst geprüft, ob der Client im File server.access registriert ist. Dann schaut der Prozeß nach, ob das Data-File in das temporäre Verzeichnis kopiert wurde. Fallen diese Prüfungen negativ aus, dann gibt der Prozeß eine entsprechende Meldung an den Client. Im anderen Fall wird das Data-File auf seine Syntax geprüft. Ist auch dieser Test erfolgreich, ruft der Prozeß über die Server-Pipe den Server und beantragt die Aufnahme des neuen Data-Files in die Datenbank. Dann ruft er wie in der do_reg-Routine den Timeout-Prozeß auf und begibt sich in den Status *Blocking Read.* Für die Kommandos Link, Unlink, Delete und Update ist die Vorgehensweise der Routine ähnlich.

Insgesamt sieht das Client-Server-Modell also wie folgt aus:

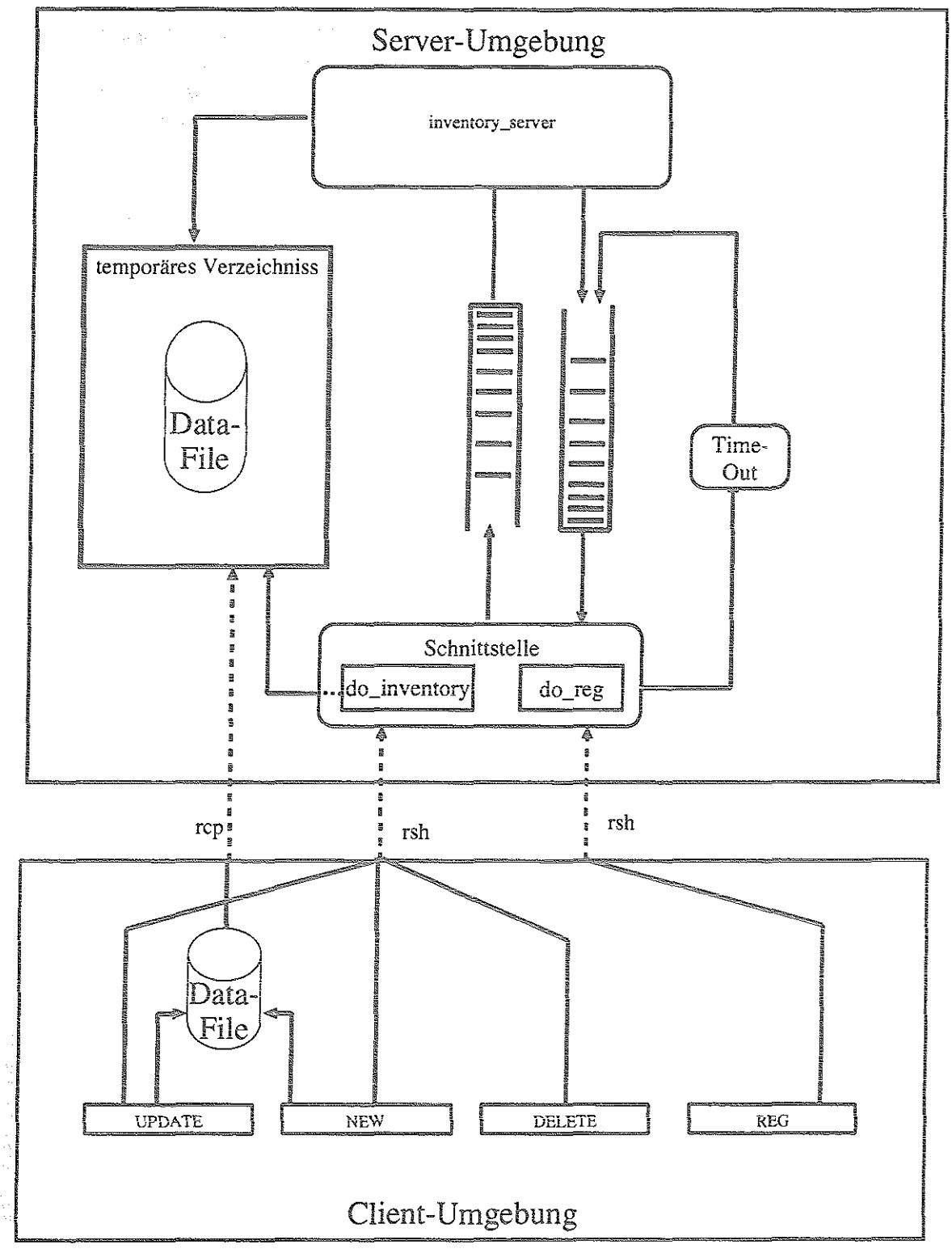


Abbildung 4.7: Das Client-Server-Modell

4.3 Sicherheitsaspekte

Datensicherheit wird verstanden als der gewährleistete Schutz der Daten in einer Datenbank vor einem unberechtigten Zugriff [?]. In diesem Abschnitt werden die systemtechnischen und organisatorischen Mittel beschrieben, um diesen Schutz zu garantieren. Die Ziele eines vorsätzlichen Eindringens in ein DB-System können sein:

1. Gewinnung von Information
2. Herausfinden, welches Informationsinteresse ein Benutzer hat
3. Ändern und Zerstören von Information
4. Kostenlose Nutzung von Ressourcen des Systems oder Nutzung von Systemressourcen

Zunächst muß der Zugang zu den Daten für die verschiedenen Benutzerschichten geregelt werden [?]. Dabei ist zwischen schreibenden und lesenden Zugriff zu unterscheiden. Für die Daten des Inventory-Management-Systems kann dabei so vorgegangen werden, daß jeder berechtigte Benutzer lesenden Zugriff auf alle Daten hat, schreibenden Zugriff aber nur auf seine eigenen Geräte. Daneben gibt es Benutzer, die schreibenden Zugriff auch auf andere bzw. alle Daten haben. Diese Rechte werden in der Datei server.access registriert, in der jeder Benutzer mit seinem Paßwort verzeichnet ist. Ein Eintrag in der Datei server.access hat folgende Form:

User-Name # User-ID.Group-ID # Paßwort # Zugriffsrechte

Dabei können die Zugriffsrechte wie folgt kodiert sein:

own : Schreibberechtigung nur bei "eigenem" Inventar

Liste von Kategorien : Schreibberechtigung auf Inventar zu bestimmten Kategorien (z.B. PC's oder Workstations)

all : Schreibberechtigung auf alle Daten

Neben den Zugriffsrechten ist die Vergabe eines Paßwortes eine Möglichkeit, Unbefugten den Zugriff auf die Daten zu verweigern. Dazu kann man das Paßwort benutzen, das auch zum Zugang ins System benutzt wird. Dann darf das Paßwort aber nicht mehr in dem Verzeichnis des Benutzers abgespeichert werden, da das Datenbank-Management-System nicht verhindern kann, daß diese Datei nicht durch den Eigentümer unwissentlich auch anderen Benutzern zugänglich gemacht wird.

Das Ziel dieser Sicherheitsmaßnahmen ist es, das Eindringen in die Datenbank durch Unbefugte zu vermeiden.

Für das Inventory-Management-System ist nur der dritte Punkt von besonderer Bedeutung und wurde durch die Vergabe eines Paßwortes und der Zuteilung von bestimmten Zugriffsrechten weitgehend ausgeschlossen. Ein Weg, um die Gewinnung von Information zu verhindern, besteht darin, die gespeicherten Daten zu kodieren. Für das Inventory-Management-System sind die gespeicherten Daten für Außenstehende ohne Bedeutung, daher wird auf eine Kodierung im Rahmen dieser Arbeit verzichtet.

Kapitel 5

Die graphische Oberfläche

Für den Zugang zum Inventory-Management-System wurde eine graphische Oberfläche entwickelt, die dem Benutzer entsprechend des zu verarbeitenden Objektes vordefinierte Bildschirmmasken und Menüs bereitstellt. Als Werkzeug für die Generierung der Menüs und der Dialoganzeigen stand das System Xhibition zur Verfügung [?] [?]. In diesem Kapitel werden die Menüs und ihre Funktionen beschrieben.

5.1 Das Hauptmenü

Das Hauptmenü des Inventory-Management-Systems ist eingebettet in ein allgemein zugängliches Menüsystem 'siat' [?]. Von hier gelangt man durch den impliziten Aufruf des Tcl-Programms *inventory_client* in den im folgenden beschriebenen Menübaum (Abb. 5.1).

Das Hauptmenü soll den Lebenszyklus aus Abb. 1.1 wiedergeben. Nicht im Hauptmenü anwählbar sind die Zustandsänderungen Außerbetriebnahme und Verschrottung. Diese Punkte werden vielmehr in Untermenüs angewählt.

Im Hauptmenü sind demnach folgende Funktionen aufrufbar:

- Gerät bestellen

  Es folgt ein Menü, mit dem der zu bestellende Gerätetyp ausgewählt werden kann.

- Gerät erhalten

  Es folgt ein Menü, mit dem der gelieferte Gerätetyp ausgewählt werden kann, daran anschließend erscheint eine Liste mit den bestellten Geräten dieses Gerätetyps.

Abbildung 5.1: Das Hauptmenü

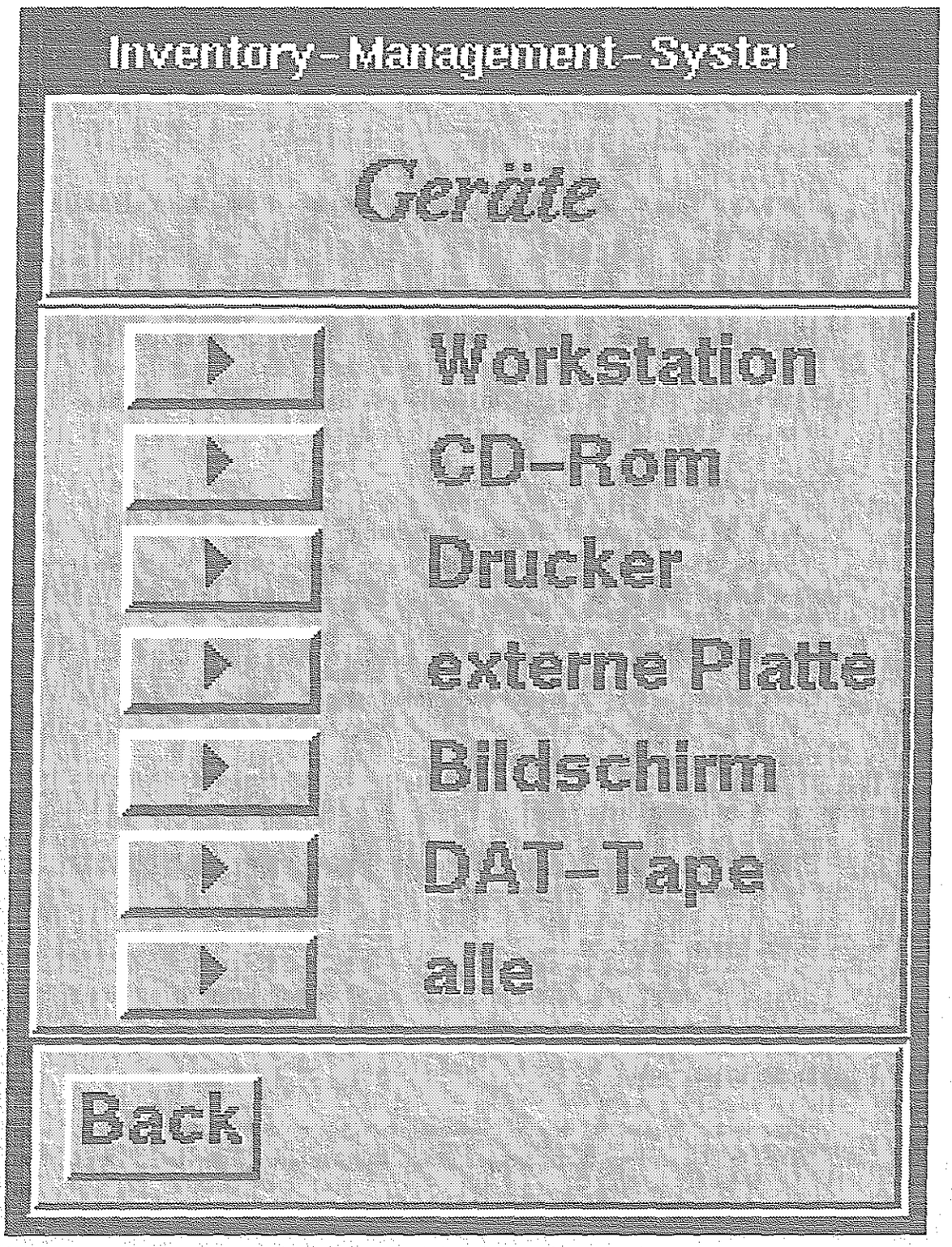


Abbildung 5.2: Das Gerätemenü

- Gerät installieren

  Nach der Wahl des Gerätetyps wird eine Liste der gelieferten Geräte angezeigt.

- Gerät umkonfigurieren

  Nach der Wahl des Gerätetyps wird eine Liste der installierten Geräte angezeigt.

- Gerät verbinden

  Zunächst wird wieder ein Gerätetyp ausgewählt, dann kann aus einer Liste der installierten Geräte dieses Typs ein spezielles selektiert werden. Nach dem gleichen Schema wird das zweite für diese Aktion erforderliche Gerät gewählt.

- Verbindung aufheben

  Das erste Gerät wird wie bei den vorherigen Menüpunkten ausgewählt, zu diesem wird dann eine Liste geliefert, anhand der das zweite Gerät gewählt werden kann.

- bestellte Geräte anzeigen

  Es wird eine Liste der bestellten Geräte angezeigt, wobei entweder ein bestimmter Gerätetyp spezifiziert wird oder alle bestellten Geräte unabhängig vom Typ gezeigt werden.

- gelieferte Geräte anzeigen

  Eine dem Menüpunkt *bestellte Geräte anzeigen* entsprechende Liste der gelieferten Geräte wird angezeigt.

- installierte Geräte anzeigen

  Eine dem Menüpunkt *bestellte Geräte anzeigen* entsprechende Liste der installierten Geräte wird angezeigt.

- defekte Geräte anzeigen

  Eine dem Menüpunkt *bestellte Geräte anzeigen* entsprechende Liste der defekten Geräte wird angezeigt. Dabei werden alle Geräte unabhängig vom Gerätetyp angezeigt.

Sollte eine Liste nicht verfügbar sein, weil sie beispielsweise leer ist, so wird der jeweilige Menüpunkt abgebrochen und in das Hauptmenü zurückgekehrt.

Die Zustandsänderung Außerbetriebnahme wird im Menüpunkt *Gerät umkonfigurieren* durchgeführt.

5.2 Gerät bestellen

Für das Inventory-Management-System beginnt ein Gerät mit der Bestellung zu existieren. In diesem Menüpunkt muß zunächst ein Gerätetyp (Abb. 5.2) gewählt werden. Abhängig vom Typ wird dann ein Bestell-Panel aufgerufen. Abb. 5.3 zeigt ein solches Panel für die Bestellung einer Workstation.

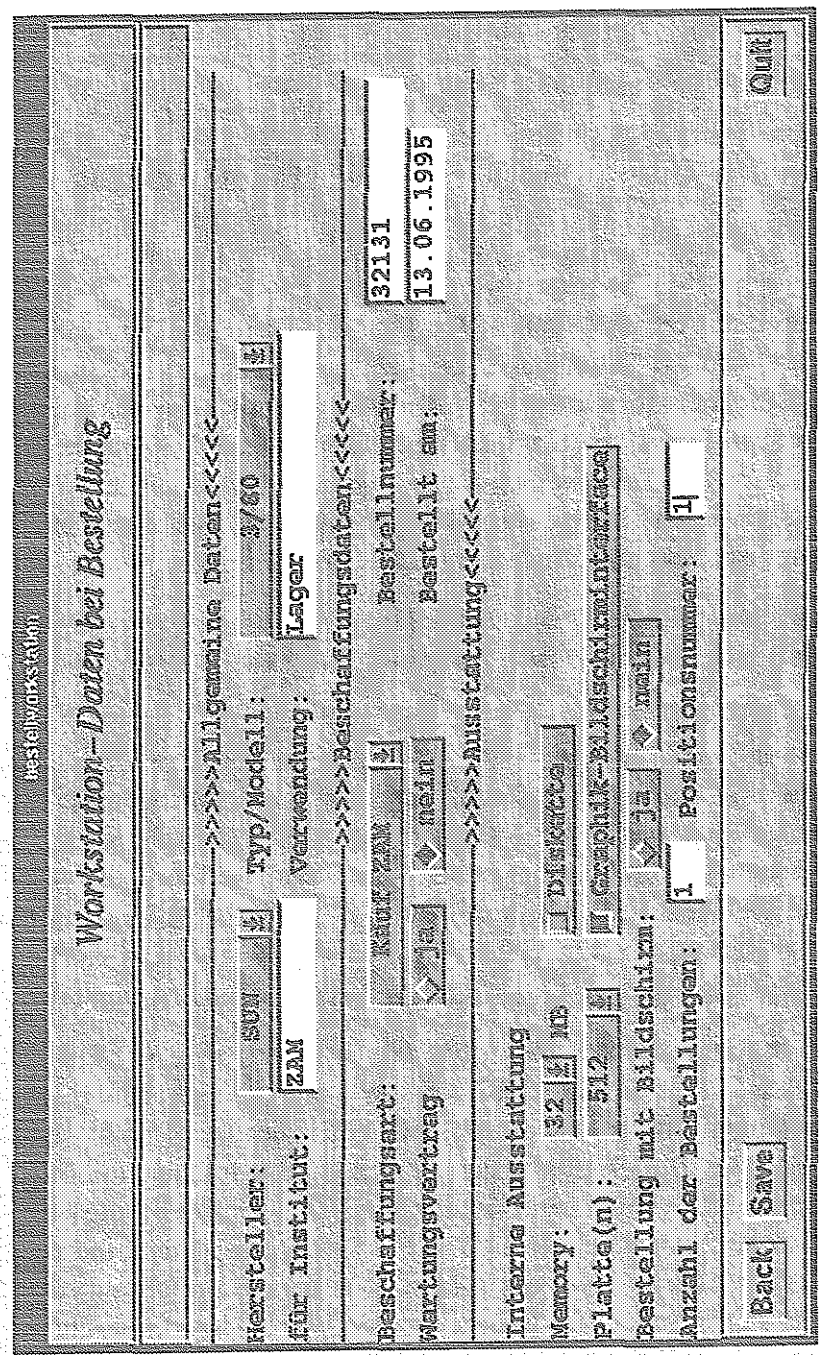


Abbildung 5.3: Bestellung einer Workstation

In diesem Panel sind alle die Bestellung betreffenden Daten enthalten. Bevor in der unteren Reihe der Button Save gedrückt wird, müssen verschiedene Felder ausgefüllt werden. Die Felder *Hersteller, Typ/Modell* und *Beschaffungsart* können nicht beschrieben werden, hier erhält man durch Anklicken des jeweiligen Buttons eine Auswahl, wie Abb. 5.4 für das Feld *Beschaffungsart* zeigt. Die Felder *Memory* und *Platte(n)* werden auf die gleiche Weise

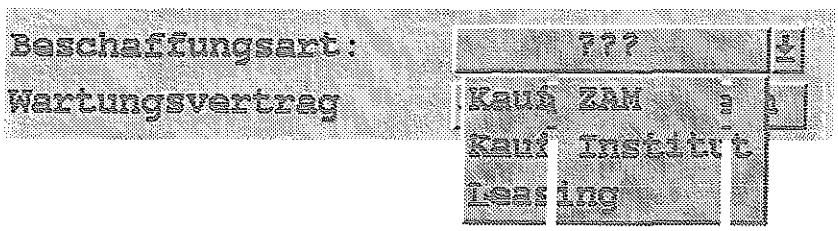


Abbildung 5.4: Auswahl bei Beschaffungsart

gewählt, sind aber mit einem Default-Wert besetzt. Das Feld *Typ/Modell* ist abhängig vom Hersteller. Wird der Hersteller ausgewählt, erscheint bei Anklicken des Buttons *Typ/Modell* eine Liste der für diesen Hersteller verfügbaren Modelle. Bei beiden Buttons (*Hersteller* und *Typ/Modell*) besteht die Möglichkeit, eigene Angaben zu definieren. Die Eingabefelder für *Institut, Verwendung, Bestellnummer, Bestellt am* und *Positionsnummer* müssen ausgefüllt werden, für das Bestelldatum wird eine Syntaxprüfung durchgeführt. Das Bestelldatum ist dabei mit dem aktuellen Datum vorbesetzt.

Möchte man mehrere Geräte mit der angegebenen Konfiguration bestellen, so kann man dies im Feld *Anzahl der Bestellungen* angeben. Dieses Feld ist mit der Voreinstellung eins besetzt.

Wird in der unteren Leiste der Save-Button gedrückt, wird zunächst geprüft, ob alle Felder ausgefüllt wurden. Dann wird, falls kein Bildschirm mitbestellt wurde, der Auftrag zum Sichern an den Server geschickt. Da es sich um ein neues Gerät handelt, wird hierzu der Dienst new benutzt. Die Rückmeldung über Erfolg bzw. Mißerfolg der Aktion erscheint in der oberen Leiste (Abb. 5.5). Wurde das Gerät in die Datenbank aufgenommen, erfährt der Benutzer die KFAid, die dem Gerät vergeben wurde. Falls der Auftrag vom Server zurückgewiesen wurde, wird der Grund in der oberen Leiste angezeigt (Abb. 5.5).

Ist das Feld Bestellung eines Bildschirmes mit ja belegt, wird vor der Weitergabe der Bestellung an den Server das Panel Bildschirmbestellung aufgerufen. Dabei wird die Zahl der Bestellungen übernommen und kann in diesem Panel nicht geändert werden. Weiterhin können das Bestelldatum, der Hersteller, das Institut, die Verwendung, die Bestellnummer und die Positionsnummer aus dem Panel zur Bestellung einer Workstation übernommen werden. Der Benutzer muß nun lediglich das Modell, die Bildschirmgröße und die Option *color/monochrome* auswählen. Wie bei der Bestellung einer Workstation bekommt man auch hier unter dem Button Typ/Modell eine herstellerabhängige Liste von Modellen angeboten. Wird nun der Save-Button gedrückt, wird die angegebene Zahl an Bildschirmen bestellt. Anschließen wird in das Panel zur Bestellung einer Workstation zurückgekehrt. Wird das Bildschirm-Panel verlassen, ohne zu speichern, dann wird das Feld im Panel Workstation-Daten bei Bestellung auf nein gesetzt. Das Programm

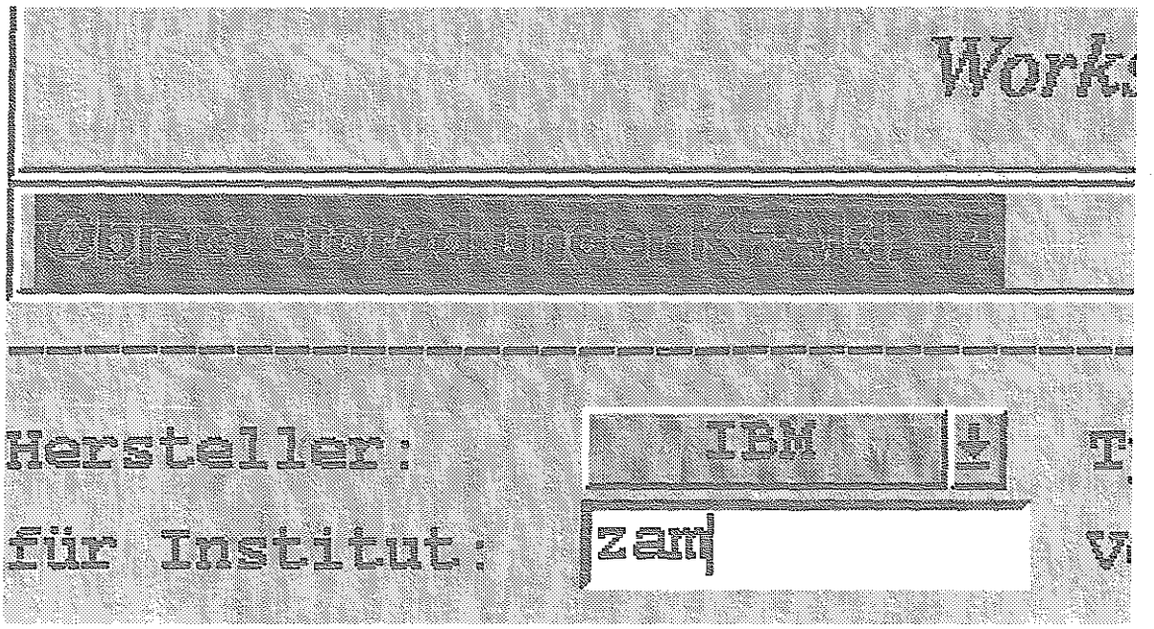


Abbildung 5.5: Meldung im Panel

beginnt nun damit, die Bestellungen der Workstations durchzuführen. Ist dieser Vorgang abgeschloßen, wird das Feld *Anzahl der Bestellungen* gelöscht. Bevor der Benutzer nun neue Bestellungen durchführt, muß er zunächst dieses Feld mit einem sinnvollen Wert besetzen. Dadurch wird vermieden, daß der Benutzer durch ein versehentliches erneutes Drücken des Save-Buttons eine weitere Bestellung in Auftrag gibt.

Wird in irgendeiner Phase der Back-Button betätigt, dann wird geprüft, ob die aktuellen Daten bereits gespeichert wurden. Ist dies nicht der Fall, muß das Back-Kommando noch einmal bestätigt werden (Abb. 5.6).

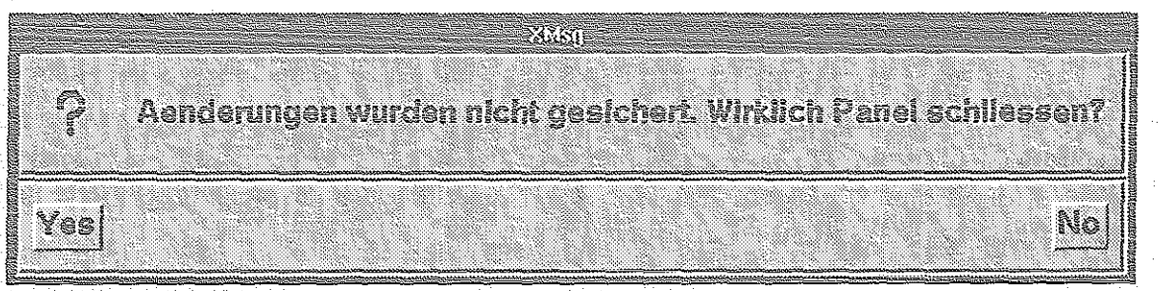


Abbildung 5.6: Bestätigung des Back-Kommandos

5.3 Gerät erhalten

In diesem Menüpunkt wird wie beim Menüpunkt "Gerät bestellen" zunächst der Gerätetyp gewählt (Abb. 5.2). Abhängig vom gewählten Gerätetyp wird nun eine Liste angezeigt (Abb. 5.7), in der die bestellten Geräte des gewählten Typs stehen.

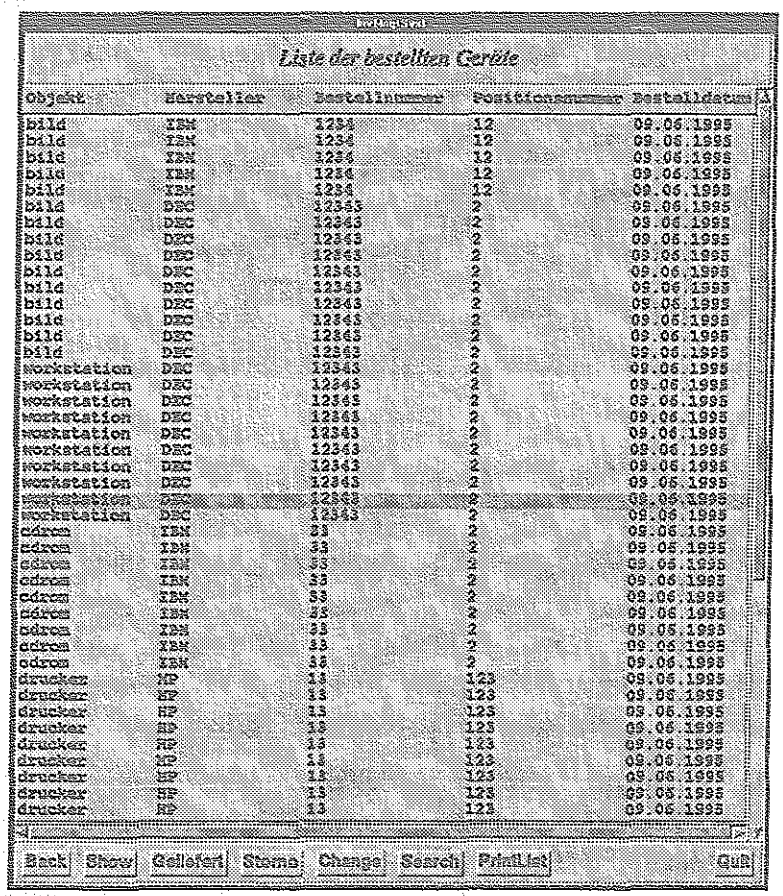


Abbildung 5.7: Liste der bestellten Geräte

Über den Search-Button kann nach einem bestimmten Eintrag in der Liste gesucht werden. Angezeigt wird hierbei der erste Eintrag in der Liste, der mit dem einzugebenden Suchmuster übereinstimmt; mit dem 'Tab'-Key der Tastatur können die weiteren gefundenen Einträge aufgesucht werden.

Man kann eine gewünschte Zeile durch 'Klick' anwählen und anschließend den Button 'Geliefert' betätigen. In der Folge erhält man das Dialog-Panel von Abb. 5.8.

Die Felder, die bereits bei der Bestellung eingegeben wurden, sind in diesem Panel nicht mehr beschreibbar, ihr Wert wurde aus dem bisherigen Datensatz übernommen. Lediglich das Feld *Verwendung* kann geändert werden. Abhängig davon, ob bei der Bestellung die Beschaffungsart *Kauf* oder *Leasing* gewählt wurde, erscheint im Panel das Feld *Inventarnr.* oder *Leasingnr.* Diese Nummer muß zu diesem Zeitpunkt noch nicht eingegeben werden, im Gegensatz zu den Feldern *Seriennummer* und *Geliefert am.* Wird

der Save-Button gedrückt, erfolgt eine Prüfung der neu eingegebenen Daten, dabei wird auch geprüft, ob die Lieferung später als die Bestellung erfolgt.

Das nun gelieferte Gerät wird aus dem Pool der bestellten Geräte entfernt und in den Pool der vorhandenen Geräte mit dem Zustand *unbenutzt* eingetragen. Dann wird in die Liste der bestellten Geräte zurückverzweigt.

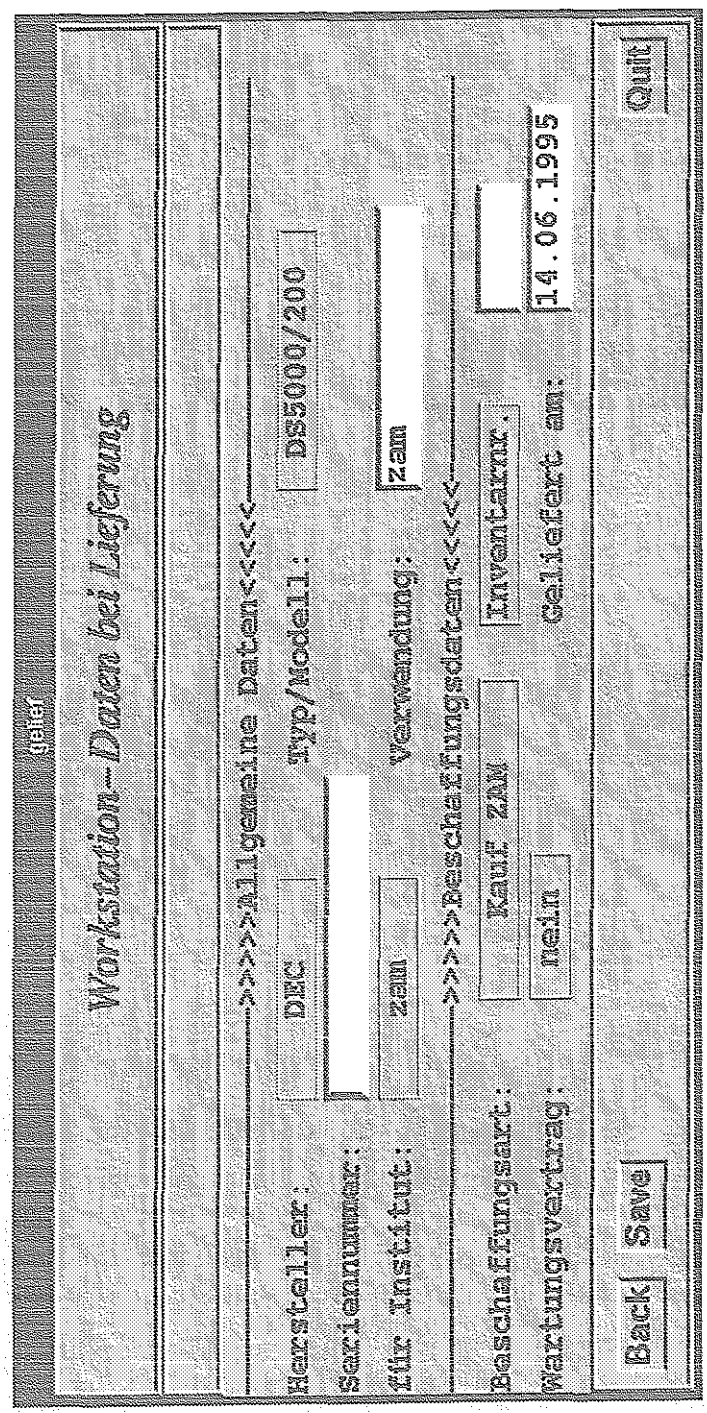


Abbildung 5.8: Lieferung einer Workstation

In der Liste der bestellten Geräte sind weitere Buttons angeordnet. Mit dem Button Change können die Daten eines bestellten Gerätes geändert werden, hierzu wird das Bestell-Panel aufgerufen. Möchte man eine Bestellung rückgängig machen, kann über den Button Storno das Gerät aus der Datenbank gelöscht werden. Über den Show-Button kann man sich die bisher bekannten Daten eines Gerätes anzeigen lassen.

5.4 Gerät installieren

Bei Wahl des Menüpunktes "Gerät installieren" erscheint, nachdem der Gerätetyp spezifiziert wurde, eine Liste der gelieferten Geräte. Wiederum gibt es einen Search-Button, mit dem ein Gerät in der Liste gesucht werden kann. Nach der Auswahl erscheint das Dialog-Panel für die Installation (Abb. 5.9).

Dem Datensatz des Gerätes können nun weitere Daten hinzugefügt werden, die zum Zeitpunkt der Lieferung noch nicht bekannt waren:

1. Das Gerät erhält einen Internet-Namen.
2. Die Verwendung wird nun genauer spezifiziert durch die Angabe des Standortes, des Benutzers und des Institutes.
3. Inventar- bzw. Leasing-Nummer müssen jetzt angegeben werden. Wurde bereits bei Lieferung eine solche Nummer vergeben, wird sie in das Installations-Panel übernommen, kann jedoch geändert werden.
4. Das Installationsdatum muß angegeben werden.
5. Die Betriebsart muß ausgewählt werden.
6. Das Betriebssystem muß ausgewählt werden.

Das Betriebssystem ist abhängig vom Hersteller. Ist z.B. als Hersteller DEC ausgewählt so stehen als Betriebssystem VMS, ULTRIX und DEC-OSF/1 zur Verfügung. Je nach Wahl der Betriebsart sind weitere Felder auszufüllen:

bei Betriebsart Server: Aufgabe des Servers.
bei Betriebsart Client: Hostname des Servers der Workstation-Gruppe und Name der Gruppe.

Felder, die abhängig von der Betriebsart keine Eingabe erlauben, werden gesperrt und mit dem Nullwert der Datenbank, xxxxxxxxxx, belegt.

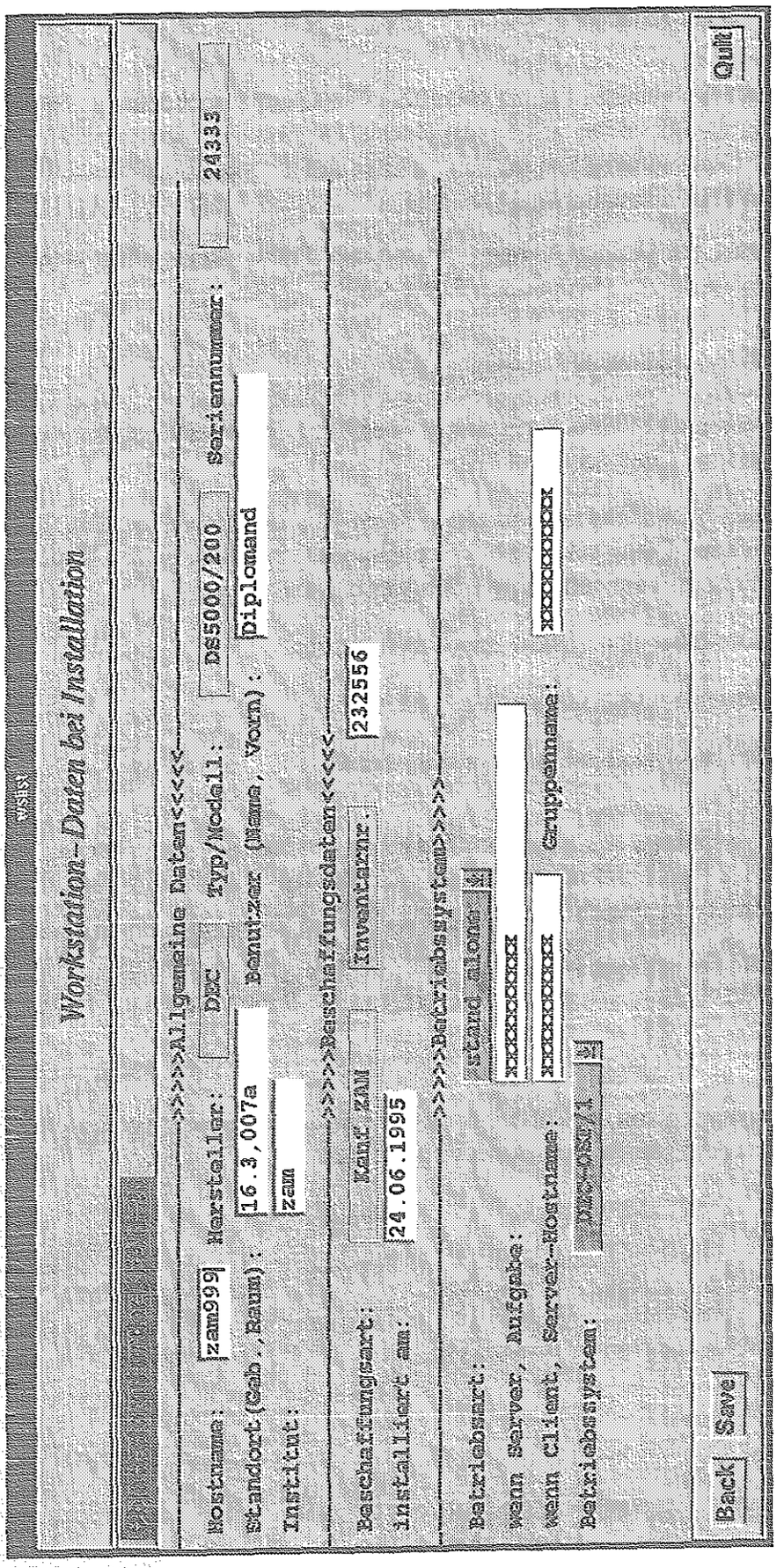


Abbildung 5.9: Installation einer Workstation

5.5 Gerät umkonfigurieren

Sollen bestimmte Installationsdaten eines Gerätes geändert werden, geschieht das durch die Menü-Funktion "Gerät umkonfigurieren". Der Benutzer wählt wie in den bisher beschriebenen Funktionen ein Gerät aus. Zu diesem Gerät erscheint nun ein Dialog-Panel (Abb. 5.10), in dem die Umkonfiguration vorgenommen werden kann.

Folgende Felder können geändert werden:

1. die allgemeinen Daten Hostname, Standort, Benutzer und Institut
2. die Beschaffungsart und die Inventar- bzw. Leasingnummer
3. die Betriebsart sowie die dazugehörigen Felder
4. das Betriebssystem
5. die interne Ausstattung: Hauptspeicher, interne Platte, Disketten-Laufwerk und Graphik-Interface

Wie in Abb. 1.1 ersichtlich, versteht man unter einer Umkonfiguration im Sinne des Inventory-Management-Systems nicht nur das Ändern der Inventardaten, sondern auch die Registrierung von Verbindungen zu anderen Geräten. Dementsprechend sind zusätzlich zum Save-Button in der unteren Leiste hier auch ein Link- und Unlink-Button hinzugefügt worden. Die Funktion dieser Buttons entspricht den Hauptmenüpunkten Gerät verbinden bzw. Verbindung aufheben. Diese Punkte sind im nächsten Kapitel beschrieben.

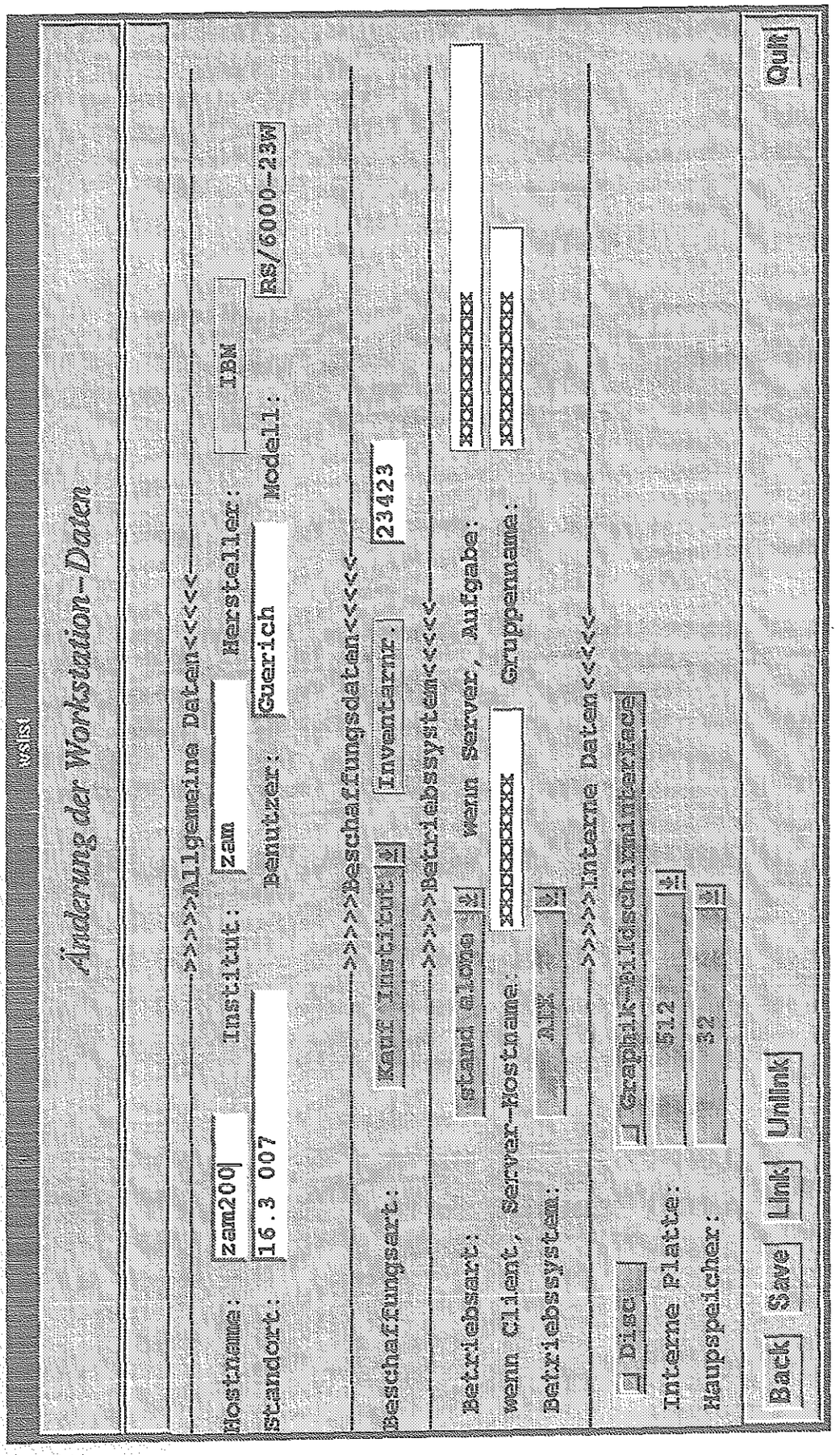


Abbildung 5.10: Umkonfiguration einer Workstation

5.6 Gerät verbinden und Verbindung aufheben

Zunächst muß ein bestimmtes Gerät ausgewählt werden. Dies geschieht wieder durch Gerätetypauswahl und Selektieren des gewünschten Gerätes aus der Liste. Diese Liste enthält alle im Betrieb befindlichen Geräte des gewählten Typs. Nun muß das zweite Gerät gewählt werden. Dies geschieht auf die gleiche Art. Im Gegensatz hierzu ist im Panel Umkonfiguration (Abb. 5.11) bereits das erste Gerät festgelegt, es muß also nur noch das zweite Gerät gewählt werden.

Im Untermenüpunkt "Verbindung aufheben" wird wieder ein Gerät ausgewählt. Für dieses wird nun eine Liste erzeugt, in dem die Links enthalten sind. Hier kann nun ein weiteres Gerät ausgewählt werden, für das die Verbindung aufgehoben wird. Auch hier liegt im Panel Umkonfiguration das erste Gerät bereits fest. Wird der Unlink-Button gedrückt, erscheint sofort die Liste mit den Verbindungen. Abb. 5.12 zeigt eine solche Liste für eine Workstation, die mit zwei Druckern und einem Bildschirm verbunden ist.

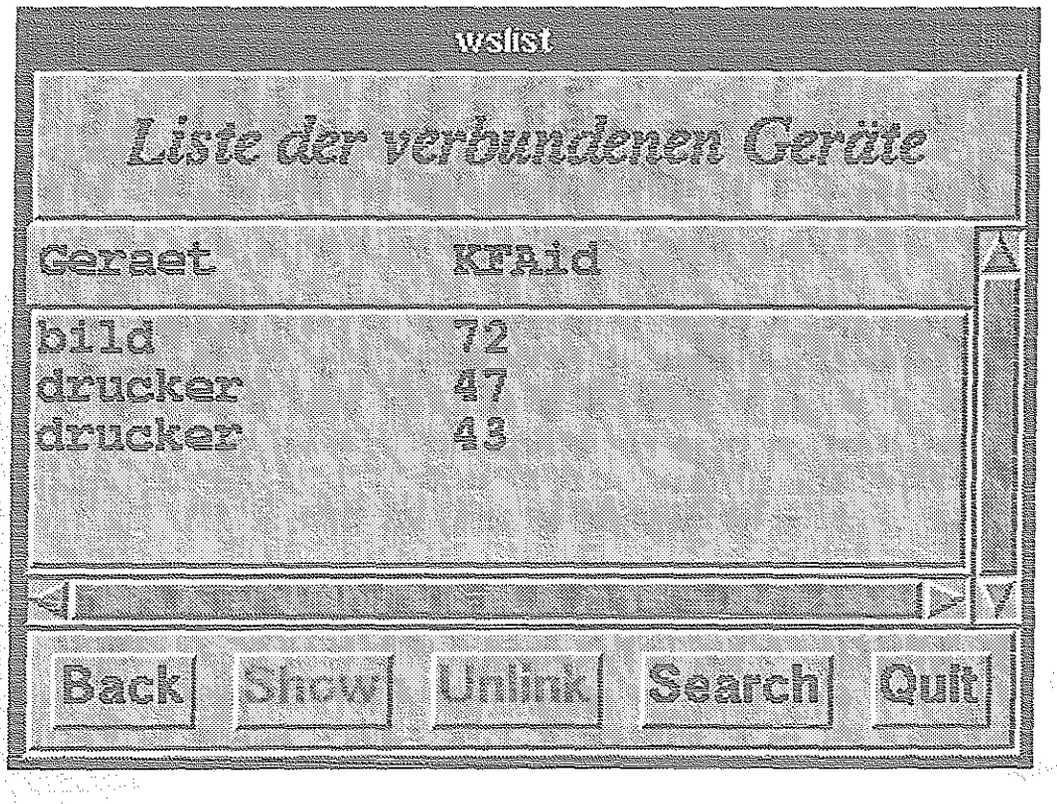


Abbildung 5.11: Links einer Workstation

Wird eine Verbindung zwischen zwei Geräten hergestellt, dann wird untersucht, ob die Daten zu den Standorten der Geräte identisch sind. Ist dies nicht der Fall, erfolgt eine Fehlermeldung. Ist eines der betroffenen Geräte eine Workstation, dann gleicht das Programm den Standort des zu verbindenden Gerätes mit dem der Workstation ab.

5.7 Gerät anzeigen

In diesen Menüpunkten ist es möglich, sich Geräte eines bestimmten Zustandes anzeigen zu lassen. Dabei kann ein bestimmter Typ spezifiziert werden, weiterhin besteht die Möglichkeit, sich alle Geräte anzeigen zu lassen.

In jeder Liste gibt es einen oder mehrere Buttons, die das Gerät in mögliche neue Zustände überführen. So kann z.B. in der Liste der bestellten Geräte ein Objekt ausgewählt werden und durch Betätigung des Geliefert-Buttons das entsprechende Panel aufgerufen werden. Weiterhin kann jedes Gerät aus dem Inventory-Management-System gelöscht werden. Dies geschieht durch Betätigen des Storno-Buttons bzw. des Ausmustern-Buttons.

Abb. 5.13 zeigt eine Liste der installierten Geräte für alle Gerätetypen. Dabei gibt

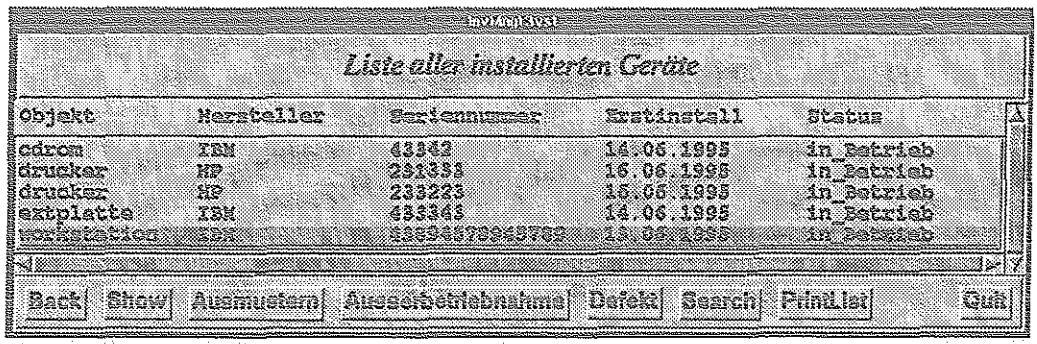
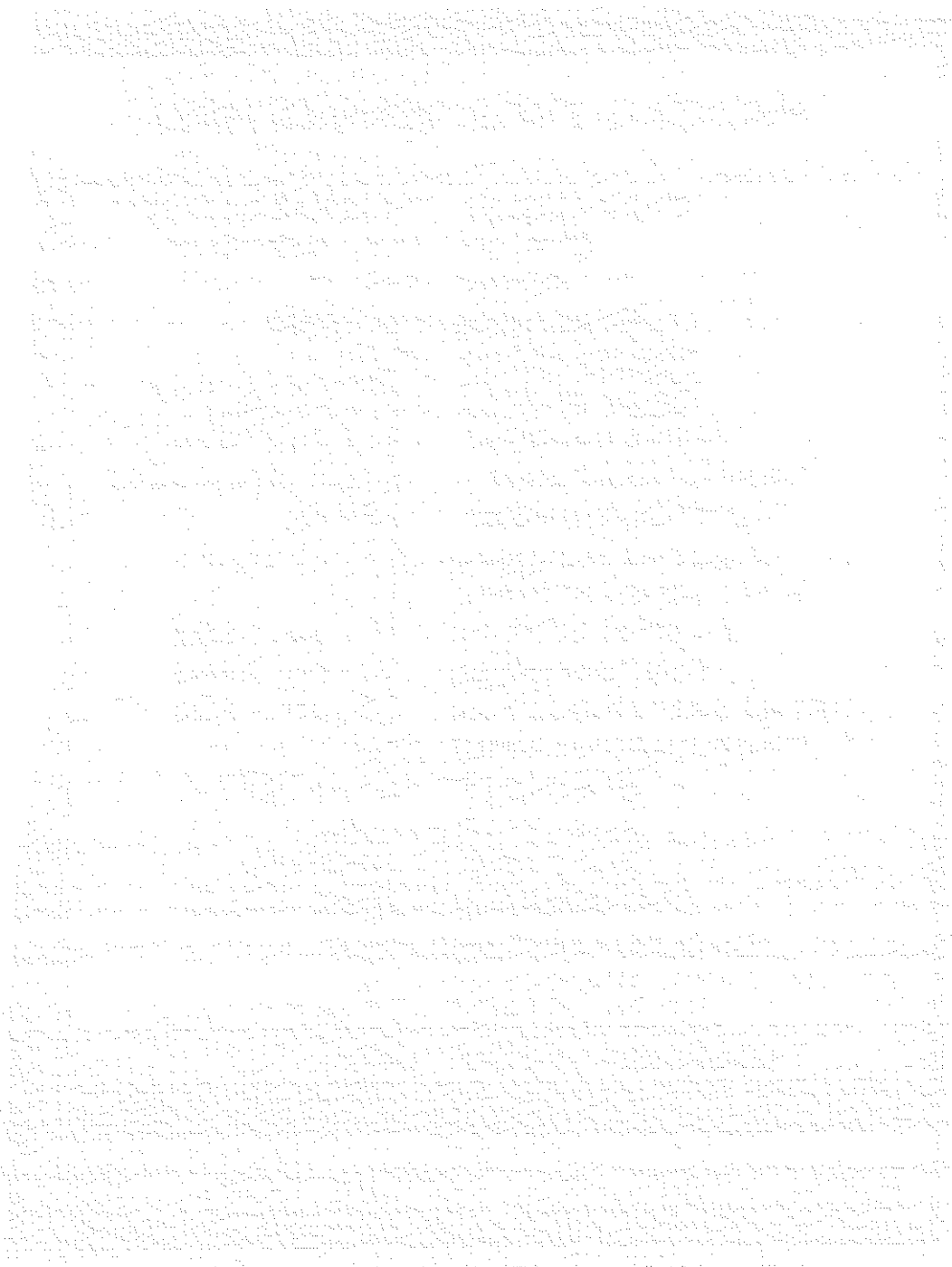


Abbildung 5.12: Liste der installierten Geräte

es in dieser Liste noch den Defekt-Button, mit dem der Status des Gerätes auf defekt gesetzt werden kann. Weiterhin kann es vorkommen, daß man ein Gerät außer Betrieb nehmen möchte. Dies geschieht durch Wählen des Gerätes und Betätigen des gewünschten Buttons. Ein solches Gerät wird dann wieder in den Zustand *geliefert* überführt. In der Liste der defekten Geräte gibt es einen entsprechenden Re-Install-Button, mit dem defekte Geräte wieder in den Zustand *installiert* überführt werden können.

In allen Listen ist ein Show-Button verfügbar. Über diesen Button kann man sich in einem Textfenster die aktuellen Daten des Gerätes ansehen (Abb. 5.13). In diesem Fenster gibt es den Linked_Dev-Button, durch den man eine Liste der verbundenen Geräte erhält. In dieser Liste wiederum existiert ein Show-Button. Dadurch ist es möglich, alle Daten einer Workstation und der mit ihr verbundenen Geräte zu erhalten. Über den Print-Button können diese Listen auf einem angegebenen Postscript-Drucker ausgegeben werden.

Abbildung 5.13: Textfenster zu den Daten einer Workstation



Kapitel 6

Zusammenfassung und Ausblick

In dieser Arbeit wurde ein Inventory-Management-System entwickelt, das die systematische Verwaltung des DV-Inventars unterstützt und zu jeder Zeit Auskunft über den aktuellen Zustand der Geräte geben soll. Wesentliche Merkmale des Inventory-Management-Systems sind die Implementation einer Datenbank, die die Abbildung der Netzwerkstruktur der betroffenen Daten ermöglicht, der Client-Server-Mechanismus, der die von mehreren Benutzern parallel eingehenden Aufträge an das System serialisiert und so die Konsistenz der Datenbank gewährleistet und die graphische Oberfläche, die den Lebenszyklus der einzelnen Geräte aus der Sicht eines Inventory-Management-Systems nachbildet und die Bearbeitung der Daten für den Benutzer vereinfacht.

Ausgehend vom relationalen Datenbankmodell wurde die Datenbank XDB entworfen, die das relationale Datenbankmodell um zwei weitere Funktionen (LINK und UNLINK) erweitert. Durch diese Funktionen ist es möglich, die Verbindungsstruktur der einzelnen Geräte untereinander in der Datenbank festzuhalten. Dabei werden die Datensätze der einzelnen Geräte in separaten Dateien gespeichert. Weiterhin existiert ein Katalog-File , in der die für die Inventarisierung relevanten Daten der Geräte, insbesondere der Primär- und alle Sekundärschlüssel, enthalten sind. Wird nach bestimmten Datensätzen gesucht, so wird dieses Katalog-File zur Hilfe genommen. Um die Verbindungsstruktur der Geräte festzuhalten, wurde ein Link-File eingeführt.

Da das Inventory-Management-System durch mehrere Anwender genutzt wird, ist es erforderlich, die von den Benutzern eingehenden Aufträge zu serialisieren. Dies geschieht durch die Implementation einer Client-Server-Architektur. Bei einem solchen Mechanismus stellt ein Server Dienstleistungen für ein Rechnernetz bereit. Ein Client fordert Leistungen des Servers an und benutzt sie [?]. Durch die Schaffung einer zentralen Instanz, die die Aufträge über eine Warteschlange einzeln abwickelt, wird die Konsistenz der Datenbank gewährleistet. Die Daten werden von den Benutzern lokal auf ihren Workstations bearbeitet, der Server agiert nur dann, wenn er eine Anfrage eines Clients in seiner Warteschlange empfängt. [?]

Ein Gerät kann aus der Sicht seiner Inventarisierung verschiedene "Lebensphasen" durchlaufen. Ziel bei der Entwicklung der graphischen Oberfläche war es, diesen Lebenszyklus

möglichst gut auf das Inventory-Management-System abzubilden. Dazu wurde dieser Lebenszyklus in einzelne Zustände zerlegt. Jede Aktion im Inventory-Management-System löst nun eine Zustandsänderung des Gerätes aus (falls Daten geändert wurden). Die Oberfläche erlaubt nur solche Transaktionen, die ein Gerät von einem Zustand in einen anderen überführen. Dabei legen spezielle Regeln fest, welche Zustandsänderungen möglich sind.

Die drei Module Datenbank, Client-Server-Architektur und Oberfläche wurden voneinander abgegrenzt. Dadurch ist es möglich, einzelne Module zu erweitern bzw. zu ersetzen. So wurde bei der Implementierung der Datenbank auf einen Recovery-Manager verzichtet. Ein solcher Recovery-Manager dient dazu, bei Systemausfall die Datenbank in einen Zustand zurückzuversetzen, in dem sie sich vor dem Start der Transaktion befand. [?]

Für eine über das Inventory-Management-System hinausgehende Nutzung der Datenbank XDB wäre auch eine Datenbeschreibungssprache einzuführen, mit der Datensätze manipuliert werden könnten. Für das Inventory-Management-System wurden nur direkte Anfragen an die Datenbank integriert.

Parallel zu dieser Arbeit wurde ein bestehendes Problem-Management-System ebenfalls auf eine Client-Server-Architektur umgestellt [?] [?]. Es ist in Zukunft geplant, beide Teilsysteme miteinander zu verbinden, so daß bei einer Problemmeldung zu einem Hardware-Gerät sofort über das Inventory-Management-System der Standort und die noch mitbetroffenen verbundenen Geräte bestimmt werden können.

Abbildungsverzeichnis

Danksagung

Die vorliegende Arbeit wurde als Diplomarbeit in Elektrotechnik am Lehrstuhl für Technische Informatik und Computerwissenschaften der Rheinisch-Westfälischen Technischen Hochschule (RWTH) Aachen erstellt.

Dem Lehrstuhlinhaber Herrn. Prof. Dr. F. Hoßfeld danke ich für die Möglichkeit, die Arbeit am Zentralinstitut für Angewandte Mathematik (ZAM) des Forschungszentrums Jülich anzufertigen und die Betriebsmittel zu nutzen.

Mein besonderer Dank gilt Herrn W. Gürich für die Betreuung dieser Arbeit und zahlreiche konstruktive Diskussionen.

An dieser Stelle möchte ich auch meinen Eltern danken, die mich auf dem Weg durch mein Studium begleitet und unterstützt haben, dessen Abschluß diese Arbeit darstellt.

Abschließend möchte ich noch die gute Atmosphäre unter den Studenten und Doktoranden des ZAM hervorheben, die zum Gelingen der Arbeit beigetragen hat.

FORSCHUNGSZENTRUM JÜLICH GmbH

KFA

Jül-3085
Juni 1995
ISSN 0944-2952