FernUniversität
Gesamthochschule in Hagen

Diplomarbeit

Entwurf und Implementierung eines Rechner-Administrierungssystems

Nora Mamblona Fischer

Aufgabensteller:	Univ.-Prof. Dr. Claus Unger
	Univ.-Prof. Dr. Heinz-Gerd Hegering
Betreuer:	Dipl.-Math. Robert Hofer
	Dipl.-Ing. (FH) Annette Kostelezky

FernUniversität
Gesamthochschule in Hagen

Diplomarbeit

Entwurf und Implementierung eines Rechner-Administrierungssystems

Nora Mamblona Fischer

Aufgabensteller:	Univ.-Prof. Dr. Claus Unger
	Univ.-Prof. Dr. Heinz-Gerd Hegering
Betreuer:	Dipl.-Math. Robert Hofer
	Dipl.-Ing. (FH) Annette Kostelezky
Abgabedatum:	7. 5.1999

Hiermit versichere ich, daß ich die vorliegende Diplomarbeit selbständig verfaßt und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 7. Mai 1999

....

Nora Mamblona Fischer

Inhaltsverzeichnis

1 Einleitung 1

1.1 Motivation und Aufgabenstellung 1

1.2 Gliederung 3

2 Grundlagen 4

2.1 Domänen 4

2.2 Entity-Relationship-Modell 6

2.3 Kommunikationsprotokolle 8

2.3.1 Directory Access Protocol (DAP) 8
2.3.2 Lightweight Directory Access Protocol (LDAP) 8
2.3.3 Simple Network Management Protocol (SNMP) 8
2.3.4 Zusammenfassung 9

2.4 Kryptographie 10

2.5 Netzwerk-Informationsdienste 12

2.5.1 Definition 12
2.5.2 Standard X.500 12
2.5.3 Domain Naming Service (DNS) 14
2.5.4 Network Information Service (NIS) 15
2.5.5 Network Infomation Service Plus (NIS+) 16
2.5.6 Network Directory Service (NDS) 17
2.5.7 Microsoft Active Directory (MAD) 21
2.5.8 Zusammenfassung 21

2.6 Objektorientierte Programmierung in Java 22

3 Entwurf des Rechner-Administrierungssystems 24

3.1 Domänenkonzept 25

3.1.1 Domänen 25
3.1.2 Objekte 25
3.1.3 Eigenschaften 25
3.1.3.1 Domäneneigenschaften 25
3.1.3.2 Objekteigenschaften 26
3.1.3.3 Rechte 26
3.1.3.4 Eigenschaft „bezüglich“ 27
3.1.4 Vererbung 28
3.1.4.1 Vererbungstyp AT 28
3.1.4.2 Vererbungstyp AW 28
3.1.5 Domänenbaum 28
3.1.5.1 Wurzeldomäne 29
3.1.5.2 Oberdomänen 29
3.1.5.3 Unterdomänen 29
3.1.5.4 Objekte 29
3.1.5.5 Mehrfachzuordnung 30
3.1.5.6 Eigenschaften 30
3.1.5.7 Operationen auf Domänen 30
3.1.5.8 Operationen auf Objekten 31
3.1.5.9 Vererbung 32
3.1.5.10 Auswirkung der Operationen auf den Domänenbaum 33

3.2 Datenbankkonzept 35

3.2.1 Umsetzung des Domänenkonzepts anhand des Entity-Relationship-Modells (ER-Modell) 35
3.2.1.1 Entitäten 35
3.2.1.2 Beziehungen 37
3.2.1.3 ER-Diagramm zum Domänenkonzept 39
3.2.2 Relationales Datenmodell 40
3.2.3 Datenbankschema 40
3.2.3.1 Tabelle Domänen 41
3.2.3.2 Tabelle Objekte 41
3.2.3.3 Tabelle Domänenobjekte 41
3.2.3.4 Tabelle Vererbung 42
3.2.3.5 Tabelle Domäneneigenschaften pro Domäne 42
3.2.3.6 Tabelle Objekteigenschaften pro Objekt 43
3.2.3.7 Tabelle Eigenschaftswerte pro Eigenschaft 44
3.2.3.8 Operationen auf Domänen und Objekten 45
3.2.4 Zugriffsrechte 48
3.2.4.1 Benutzerebene 48
3.2.4.2 Gruppenebene 48
3.2.4.3 Abstrahierung von Benutzer- und Gruppenebenen 48
3.2.4.4 Realisierung 48
3.2.4.5 Operationen auf Domänen und Objekte 51
3.2.4.6 Rechteverwaltung und -überprüfung 52
3.2.4.7 Applikationsserver 52
3.2.5 Datenbankzugriff über Benutzerkennung und Paßwort 52

3.3 Client-Server-Konzept 54

3.3.1 Netzwerke 54
3.3.2 Client-Server-Architektur 54
3.3.3 Datenbankserver 54
3.3.4 Applikationsserver 54
3.3.5 Kommunikationsprotokoll 55

3.4 Sicherheitskonzept 57

3.4.1 Sicherheit im Domänenkonzept 58
3.4.2 Sicherheit im Datenbankkonzept 58
3.4.2.1 Referentielle Integrität der Daten 59
3.4.2.2 Authentifizierung 60
3.4.2.3 Sicherung der Datenbank 60
3.4.2.4 Wiederherstellung der Datenbank im Fehlerfall 60
3.4.3 Sicherheit im Client-Server-Konzept 60
3.4.3.1 Sicherheitsmaßnahmen der Clients und Server 60
3.4.3.2 Verschlüsselung 61
3.4.3.3 Ausfallsicherheit 61

4 Implementierung des Rechner-Administrierungssystems 62

4.1 Aufgabenstellung 62

4.1.1 Domänenbaum 62
4.1.2 Konfigurationsdateien 62

4.2 Entwicklungsumgebung des Prototyps 63

4.2.1 Wahl der Programmiersprache 63
4.2.2 Wahl der Datenbank 63
4.2.3 Wahl des Datenbanktreibers 64
4.2.4 Wahl des Betriebssystems (Plattform) 64

4.3 Umsetzung des Konzepts 64

4.3.1 Domänenkonzept 64
4.3.2 Datenbankkonzept 64
4.3.2.1 Modul db 65
4.3.2.2 Modul dom 65
4.3.2.3 Modul obj 65

4.3.2.4 Modul attr 65
4.3.2.5 Modul domtree 66
4.3.3 Client-Server-Konzept 66
4.3.3.1 Kommunikationsprotokoll des Servers 66
4.3.3.2 Kommunikationsprotokoll der Clients 66
4.3.3.3 Aufteilung der generischen Module auf Clients und Server 66
4.3.4 Sicherheitskonzept 66
4.4 Kommandozeilen-Schnittstelle 67
4.5 Testumgebung 67
4.6 Dokumentation 67
5 Zusammenfassung und Ausblick 68
5.1 Zusammenfassung 68
5.2 Ausblick 69
5.3 Mögliche Erweiterungen 69
6 Abkürzungsverzeichnis 73
7 Literaturverzeichnis 75
Anhang A Benutzerhandbuch 77
Software-Voraussetzungen 77
Quellcode entpacken 77
Umgebungsvariable CLASSPATH 78
Konfigurationsdatei des Datenbanktreibers twz1 78
Konstanten im Quellcode 78
Starten des Prototyps 78
Kommandozeilen-Schnittstelle 78
Menü 78
Anhang B Dokumentation des Prototyps 79
Klasse domtree 79
Klasse dom 80
Klasse obj 95
Klasse attr 105
Klasse db 109

1 Einleitung

1.1 Motivation und Aufgabenstellung

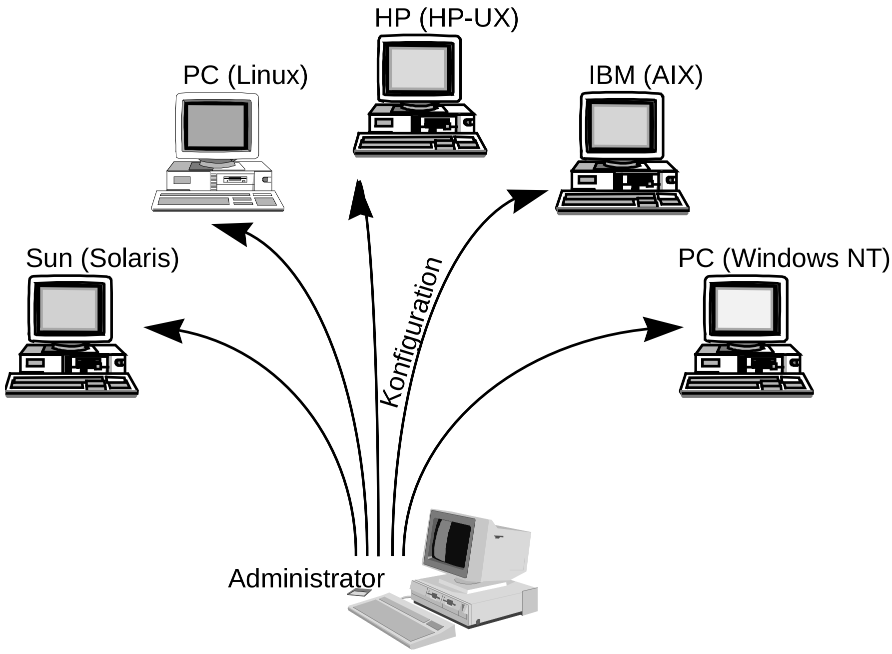
Das Institut für Informatik der Ludwig-Maximilians-Universität München benötigt ein Programm für die Systemverwaltung heterogener Rechnerarchitekturen. Ziel dieser Arbeit ist es, dafür ein Konzept zu entwerfen und einen Prototyp zu implementieren. Die Aufgabenstellung enthält den Entwurf eines Domänen-, Datenbank-, Client-Server- und Sicherheitskonzepts. Der Prototyp für ein Rechner-Administrierungssystem soll für ein bis zwei Konfigurationsdateien mit Kommandozeilen-Schnittstelle und Testumgebung, auf die in folgenden Arbeiten aufgesetzt werden kann, entwickelt werden. Zum Prototyp ist eine Dokumentation anzufertigen. Im Hinblick auf zukünftige Erweiterungen wird besonderer Wert auf einen modularen Entwurf und ein flexibles Datenbankkonzept gelegt. Eine weitere Anforderung besteht darin, daß der Prototyp aus Komponenten bestehen soll, die am Institut für Informatik bereits vorhanden oder kostenlos erhältlich sind.

Anforderungen an das zu erstellende Rechner-Administrationssystem sind:

- Client-Server-Struktur
- Datenbank zur Speicherung der Informationen
- hierarchisches Domänenkonzept
- Verschlüsselung der Daten
- Erweiterbarkeit der angebotenen Dienste
- Plattformunabhängigkeit

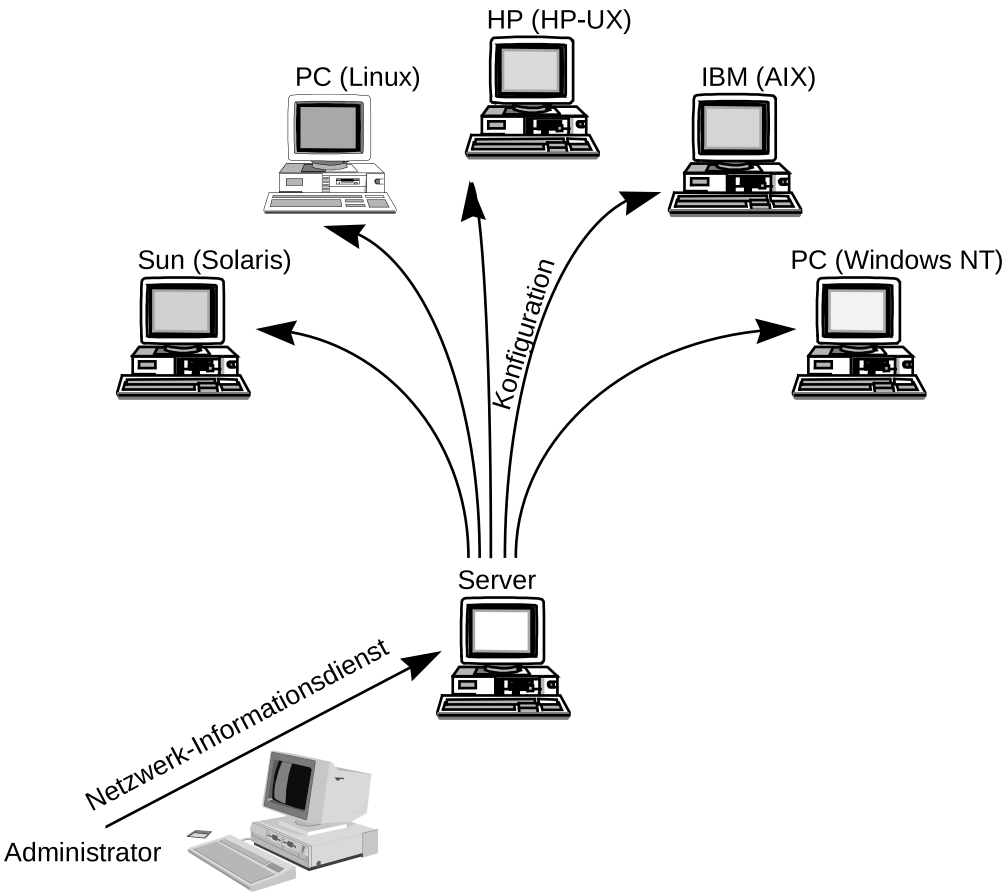
Die Aufgabenstellung ergibt sich aus der Situation am Institut für Informatik und gilt allgemein für viele Rechenzentren: im Laufe der Jahre wurden an den verschiedenen Lehrstühlen Rechner mit unterschiedlichen Betriebssystemen in verschiedenen Versionen aufgestellt. Diese Rechner sind über ein oder mehrere Netzwerke miteinander verbunden. Es gilt, die verschiedenen Verwaltungstätigkeiten der Systemadministration auf den unterschiedlichen Rechnern möglichst effizient zu organisieren. Ein Programm, welches diese Aufgabe übernimmt, wird als Netzwerk-Informationsdienst bezeichnet (network directory). Netzwerk-Informationsdienste speichern Informationen, die Benutzer, Workstations und Anwendungen für die Kommunikation über das Netzwerk benötigen. Bei den am Institut für Informatik vorkommenden Betriebssystemen handelt es sich vor allem um verschiedene UNIX- und Windows-Varianten. Damit richtet sich die vorliegende Arbeit hauptsächlich an System- und Netzwerkverwalter von UNIX- und Windows-Betriebssystemen.

Die folgende Abbildung zeigt, die für System- und Netzwerkverwalter typische Arbeitssituation:



Der Administrator muß jeden Rechner einzeln verwalten.

Ein Netzwerk-Informationsdienst soll folgendes erreichen:



Der Administrator gibt ein Kommando im Netzwerk-Informationsdienst ein, der die entsprechende Operation auf allen Rechnern des Netzwerkes ausführt.

1.2 Gliederung

Die vorliegende Arbeit behandelt den Entwurf und die Implementierung eines Rechner-Administrierungssystems. Das erste Kapitel enthält eine Einleitung. Das zweite Kapitel „Grundlagen“ beginnt mit Defi

nitionen zum Begriff Domäne. Es folgt eine Beschreibung von Kommunikationsprotokollen mit den Vertretern DAP, LDAP und SNMP. Nach einem kurzen Abschnitt über das Gebiet der Kryptographie folgt eine Beschreibung und Definition von Netzwerk-Informationsdiensten. Dabei werden wichtige Vertreter vorgestellt. Anschließend werden objektorientierte Begriffe, wie sie in der Programmiersprache Java verwendet werden, erläutert. Im dritten Kapitel wird das Konzept für das Rechner-Administrierungssystem behandelt. Es gliedert sich in vier Teile: dem Domänen-, Datenbank-, Client-Server- und Sicherheitskonzept. Zum Datenbankkonzept wird ein Entity Relationship-Diagramm erstellt. Das vierte Kapitel beschreibt die Implementierung des Rechner-Administrierungssystems. Es enthält eine Beschreibung der Entwicklungs- und Testumgebung, der erstellten Module, der Kommandzeilenschnittstelle und der Dokumentation. Im fünften Kapitel wird das bisherige zusammengefaßt und mögliche Erweiterungen für eine Folgeversion des Prototyps werden behandelt. Im sechsten Kapitel befindet sich ein Verzeichnis der verwendeten Abkürzungen. Das siebte Kapitel besteht aus einem Literatur- und Quellenverzeichnis gibt. Das Benutzerhandbuch zum Prototyp des Rechner-Administrierungssystems befindet sich im Anhang A. Die Dokumentation zum Quellcode steht im Anhang B. Der Quellcode ist auf einer Diskette abgespeichert, die sich auf der letzten Seite der vorliegenden Arbeit in einer Diskettentasche befindet.

2 Grundlagen

In diesem Kapitel werden die Grundlagen für das Konzept eines Rechner-Administrierungssytem und dessen Implementierung beschrieben. Es folgt eine Erläuterung des Begriffs Domäne. Danach wird eine E Einführung in Entity-Relationship-Modelle gegeben. In Client-Server-Architekturen werden Kommunikations-protokolle verwendet, die in diesem Kapitel eingeführt und vorgestellt werden. Anschließend folgen Begriffe und Algorithmen der Kryptographie, welche im Sicherheitskonzept für die Verschlüsselung von Daten diskutiert werden. Der darauf folgende Abschnitt über Netzwerk-Informationsdienste ist von zentraler Bedeutung für die vorliegende Arbeit. Es werden dort Vertreter von Netzwerk-Informationsdiensten vorgestellt und miteinander verglichen. Das Kapitel Grundlagen endet mit einem Abschnitt zu Begriffen aus der Programmiersprache Java.

2.1 Domänen

Für den Begriff Domäne (domain) gibt es unterschiedliche Definitionen. Aufgrund der nicht einheitlichen Verwendung dieses Begriffs folgt im Kapitel 3.1.1 „Domänenkonzept“ eine in der vorliegenden Arbeit benutzte Definition. Die folgenden Definitionen zeigen die vielfältigen und oft zu Unklarheiten führenden Verwendungen des Begriffs:

2.1.1 Definition in einem Computer-Englisch-Fachwörterbuch

Die Übersetzung in [Schu 97] gibt Bereich, Domäne eines relationalen Datenbanksystems, Landesnetz, Fachbereich und Geltungsbereich an.

2.1.2 Definition in relationalen Datenmodellen

In relationalen Datenmodellen wird der Begriff Domäne (Domain) als die Menge aller möglichen Werte eines bestimmten Typs verwendet.

2.1.3 Definition im Open System Interconnect (OSI)-Standard

Im Organisationsmodell des Open System Interconnect (OSI)-Standard wird der Begriff Domäne (Domain) folgendermaßen definiert: eine Domäne kann sich über eine organisatorische Einheit in einem Unternehmen, einem Unternehmensstandort oder über die Geräte eines Rechnerherstellers innerhalb eines Unternehmens erstrecken. Domänen können über eine Vielzahl von Subdomänen verfügen (siehe [HeGr 94] auf S.27 und 28).

2.1.4 Definition in Windows NT

In Windows NT wird eine Domäne als Zusammenstellung von Computern definiert, die vom Administrator eines Windows NT Server-Netzwerkes festgelegt wird.

2.1.5 Definition im Domain Naming Service (DNS)

Domänen sind laut Request for Comment (RfC) 1032 Verwaltungseinheiten, welche eine dezentralisierte Verwaltung bei der Vergebung von Namen und Adressen für Rechner erlauben.

2.1.6 Definition im Network Information System Plus (NIS+)

NIS+ bezeichnet mit Domänen (Domains) eine Sichtweise auf eine Gruppe von Objekten.

2.1.7 Domänenbaum

Im Kapitel 3.1.5 wird der Begriff **Domänenbaum** eingeführt. Er entspricht den Begriffen **Zuordnungsraum** und **Verzeichnisbaum**. Der Begriff Zuordnungsraum wird von den Netzwerk-Informationsdiensten Domain Naming Service (DNS) (siehe Kapitel 2.5.3), Network Information Service (NIS) (siehe Kapitel 2.5.5.1) und Network Information Service Plus (NIS+) (siehe Kapitel 2.5.5.1) verwendet. Der Begriff Verzeichnisbaum wird im Standard X.500 (siehe Kapitel 2.5.2) und vom Netzwerk-Informationsdienst NDS (siehe Kapitel 2.6.2) benützt.

2.2 Entity-Relationship-Modell

Das Entity-Relationship-Modell (ER-Modell) besteht aus Entitäten (entities) und Beziehungen (relationships) zwischen den Entitäten. Es stellt einen formalen Rahmen zur Beschreibung von Datenstrukturen und Operationen auf Daten dar.

2.2.1 Entitäts-Typ

Entitäts-Typen sind nach [KeEi 96] physisch oder gedanklich existierende Konzepte der in der Datenbank zu modellierenden Welt. In [KaKl 93] findet sich eine formale Definition, die einen Entitäts-Typ als benannte Menge von Merkmalen (Attributen) definiert, denen eindeutig Wertebereiche zugeordnet sind. Die Ausprägung eines Entitäts-Typ wird **Entität** genannt. Man erhält eine Ausprägung eines Entitätstyps, indem jedem Attribut (siehe Abschnitt 2.2.3) des Entitätstyp Werte zugeordnet werden. Erweiterungen von Entitätstypen sind Subentitätstypen:

Subentitätstypen übernehmen Attribute von den Entitätstypen, zu denen sie gehören. Sie besitzen zusätzlich weitere Attribute.

2.2.2 Beziehungs-Typ

Ein **Beziehungs-Typ** ist nach [KaKl 93] eine benannte Menge von mindestens zwei Bezeichnern, denen eindeutig Entitäts-Typen zugeordnet sind. Er verknüpft Entitätstypen miteinander. Die Ausprägung eines Beziehungs-Typs wird **Beziehung** genannt.

2.2.3 Attribute

Attribute dienen nach [KeEi 96] dazu, Entitäten bzw. Beziehungen zu charakterisieren.

2.2.4 Schlüssel

Schlüssel. Ein Schlüssel stellt nach [KeEi 96] eine minimale Menge von Attributen dar, deren Werte die zugeordnete Entität eindeutig innerhalb aller Entitäten eines Typs identifiziert.

2.2.5 Ordnung von Beziehungstypen

Im ER-Modell wird die Ordnung von Beziehungstypen durch Zahlen annotiert. Diese Zahlen geben an, wieviel Entitäten der beteiligten Entitätstypen an einer Beziehung teilhaben können. Häufig vorkommende Ordnungen von Beziehungstypen sind:

- 1:1-Beziehung
- 1:n-Beziehung
- n:1-Beziehung
- n:m-Beziehung

Beispiel für eine n:m-Beziehung:

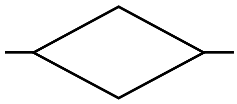
Die Zahlen haben die Bedeutung „jede Entität des Entitätstyps Objekt besitzt höchsten m Entitäten des Entitätstyps Attribut“ und „jede Entität des Entitätstyps Attribut gehört zu höchstens n Entitäten des Entitätstyps Objekt“.

2.2.6 Entity-Relationship-Diagramm

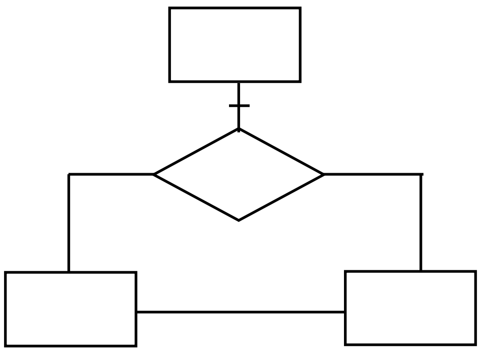
Graphisch wird das ER-Modell im Entity-Relationship-Diagramm (**ER-Diagramm**) dargestellt. Die Entitäten werden als **Rechtecke**, die Beziehungstypen als **Rauten** zwischen den Entitäten dargestellt. An den Beziehungstypen befindet sich die Art der Beziehung (**1:1**, **1:n**, **n:1** oder **n:m)**. Subentitätstypen sind dadurch gekennzeichnet, daß sie in der Raute keine Bezeichnung tragen und sich über der Raute ein **Querstrich** befindet.

Symbole eines ER-Diagramms:

Entitätstyp



Beziehungstyp



Subentitätstypen

2.3 Kommunikationsprotokolle

Kommunikationsprotokolle legen die Kommunikation zwischen Clients und Server fest. Es folgt ein Überblick über die verschiedenen Kommunikationsprotokolle, die sich für Netzwerk-Informationsdienste anbieten.

2.3.1 Directory Access Protocol (DAP)

Das Directory Access Protocol (DAP) wurde entwickelt, um in einer standardisierten Form auf Informationen eines X.500-Verzeichnis-Dienstes (siehe Kapitel 2.5.2) zuzugreifen. Im DAP gibt es folgende fünf Anfragefunktionen und vier Änderungsfunktionen zu einem X.500-Verzeichnis:

Anfragefunktionen:

Lesen	Die Werte gewählter oder aller Eigenschaften eines Eintrags werden angezeigt
Vergleichen	Der Wert einer Eigenschaft wird mit dem angegebenen Wert verglichen
Auflisten	Gibt eine Liste der unmittelbaren untergeordneten Einträgen (subordinates) aus
Suchen	Einträge, die der Filterbedingung genügen ,werden ausgegeben
Abbrechen	Eine laufende Anfrage kann mit diesem Befehl abgebrochen werden

Änderungsfunktionen:

Hinzufügen	Fügt einen neuen Eintrag in den Verzeichnisbaum ein
Entfernen	Ein Eintrag wird aus dem Verzeichnisbaum entfernt
Ändern	Zu einem Eintrag werden die Werte von Eigenschaften hinzugefügt, geändert oder ersetzt.
Namen Ändern	Der Name eines Eintrags und Änderungen bei den untergeordneten Einträgen werden geändert.

Im DAP werden auch Anfragen für X.500-Verzeichnis-Dienste, die auf verteilten Datenbanken basieren, berücksichtigt.

2.3.2 Lightweight Directory Access Protocol (LDAP)

Das Lightweight Directory Access Protocol (LDAP) stellt nach [KeIv 98] eine im Vergleich zum DAP einfachere und weniger Betriebsmittel verbrauchende Client-Schnittstelle für Verzeichnisdienste dar, die dem Standard X.500 entsprechen. LDAP ist inzwischen ein offizieller Internet-Standard für ein Directory Access Protocol. Die Version 2 von LDAP (LDAPv2) ist in RFC 1777, RFC 1778 und RFC 1779 spezifiziert. Im Gegensatz zum DAP verfügt LDAP über nur drei Anfragefunktionen:

Vergleichen	Entspricht der Funktion Vergleichen des DAP
Suchen	Entspricht den Funktionen Suchen, Lesen und Auflisten des DAP
Abbrechen	Entspricht der Funktion Abbrechen des DAP

Die vier Änderungsfunktionen des LDAP entsprechen den Änderungsfunktionen des DAP.

2.3.3 Simple Network Management Protocol (SNMP)

Das Simple Network Management Protocol (SNMP) dient nach [HeGr 94] der Bereitstellung und dem Transport von Management-Informationen zwischen Netzwerkkomponenten. Netzwerkadministratoren können Netzwerkparameter interaktiv abfragen und bestimmte Netzwerkzustände abfragen. Das SNMP wurde 1988 veröffentlicht. Ursprünglich wurde es nur für Ethernet-basierte Produkte entwickelt. Inzwischen unterstützt es vielzählige Local Area Network (LAN)-Übertragungsmechanismen und Wide Area Network (WAN)-Technologien. Die SNMP-Architektur basiert auf einem Modell, das aus einer oder mehreren Netzwerkmanagement-Stationen und mehreren Netzwerkelementen besteht.
Es ist die Aufgabe dieser Netzwerkmanagement-Stationen, die Netzwerkelemente zu überwachen und zu kontrollieren ([HeGr 94] S. 117/118). Netzwerkelemente werden alle Geräte im Netz bezeichnet, die einen oder mehrere Network Management Agents implementiert haben. Das SNMP-Protokoll wird zur

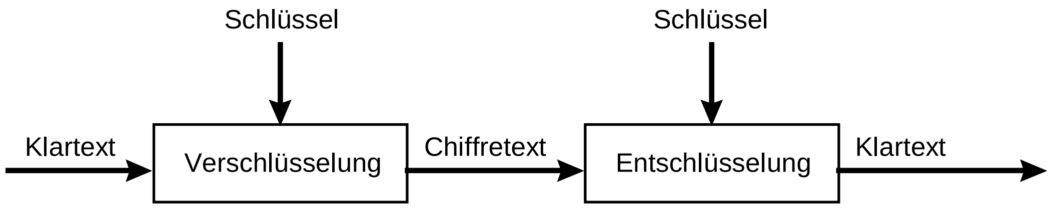
Kommunikation zwischen den Network Management Agents und der Netzwerkmanagementstation eingesetzt. Es kann unabhängig von den Hardware- und den Software-Bedingungen des jeweiligen Geräts implementiert werden. Daher ist das SNMP-Protokoll als allgemeine Basis für das Netzwerkmanagement in heterogenen Netzwelten realisierbar. Es hat sich nach [HeGr 94] als Standardmanagement-Protokoll in heterogenen Systemumgebungen durchgesetzt. Im April 1993 wurde die Version 2 des SNMP (SNMPv2) von der Network Working Group veröffentlicht und in den Request for Comments (RFCs) 1441 bis 1452 festgehalten.

2.3.4 Zusammenfassung

Die hier vorgestellten Kommunikationsprotokolle besitzen einen großen Funktionsumfang. Dadurch sind sie vor allem für Netzwerk-Informationsdienste, die in großen Organisationen wie z.B. international tätigen Wirtschaftsunternehmen eingesetzt werden sollen, relevant. Für die Aufgabenstellung der vorliegenden Arbeit kommt aus Zeitgründen eine Anbindung an die Kommunikationsprotokolle DAP, LDAP oder SNMP nicht in Frage. Im Kapitel 3.3 wird ein für den Prototyp des Rechner-Administrationssystems entwickeltes Kommunikationsprotokoll vorgestellt.

2.4 Kryptographie

Kryptographie ist die Wissenschaft, die sich mit der Absicherung von Nachrichten beschäftigt. Eine Nachricht besteht aus Klartext. **Klartext** kann eine Bitfolge, eine Textdatei und vieles mehr darstellen. In der Computerkryptographie besteht Klartext aus binären Daten. Eine **verschlüsselte** Nachricht besteht aus dem **Chiffretext**. Es stellen ebenfalls binäre Daten den Chiffretext dar. Die Umwandlung von Chiffretext in Klartext wird **Entschlüsselung** genannt. Ein kryptographischer Algorithmus, auch Chiffrierung genannt, ist die mathematische Funktion, die zur Ver- und Entschlüsselung verwendet wird. In der folgenden Abbildung wird die Ver- und Entschlüsselung mit einem Schlüssel veranschaulicht.



Die vorstehende Darstellung wurde [Schn 96] auf S. 3 (Abb. 1.2: Ver- und Entschlüsselung mit einem Schlüssel) entnommen.

Sinn der Kryptographie ist es, den Klartext (und Schlüssel) vor Personen zu verbergen, die nicht autorisiert sind, den Inhalt des Klartextes zu lesen.

2.4.1 Algorithmen

Es gibt zwei Arten von Algorithmen, die auf **Schlüsseln** basieren: symmetrische Algorithmen und Algorithmen mit öffentlichen Schlüsseln (public key).

Symmetrische Algorithmen. Der Chiffrierschlüssel läßt sich aus dem Dechiffrierschlüssel berechnen und umgekehrt. Meist sind Chiffrier- und Dechiffrierschlüssel identisch. Die Sicherheit eines symmetrischen Algorithmus liegt im Schlüssel. Wird der Schlüssel preisgegeben, können beliebige Personen Nachrichten ver- und entschlüsseln. Soll die Kommunikation geheim bleiben, ist auch der Schlüssel geheimzuhalten.

Algorithmen mit öffentlichen Schlüsseln. Public-Key-Algorithmen werden auch asymmetrische Algorithmen genannt. Public-Key-Algorithmen besitzen unterschiedliche Chiffrier- und Dechiffrierschlüssel. Der Dechiffirierschlüssel kann nicht (in angemessener Zeit) aus dem Chiffrierschlüssel berechnet werden. Diese Verfahren heißen Algorithmen mit öffentlichen Schlüsseln, da der Chiffrierschlüssel öffentlich bekannt gemacht werden kann. Nur die Person mit dem entsprechenden Dechiffrierschlüssel kann die Nachricht (in angemessener Zeit) entschlüsseln. Der Chiffrierschlüssel wird auch öffentlicher Schlüssel und der Dechiffrierschlüssel privater oder geheimer Schlüssel genannt.

Die folgenden drei Algorithmen stellen die am weitesten verbreiteten Computer-Algorithmen dar: DES, RSA und DSA.

Data Encryption Standard (DES). Dies ist der nach [Schn 96] am häufigsten verwendete Computer-Algorithmus zur Verschlüsselung. Er gehört zu den symmetrischen Algorithmen. Zur Ver- und Entschlüsselung wird derselbe Algorithmus verwendet.

RSA. Der Algorithmus wurde nach seinen Entwicklern Rivest, Shamir und Adleman benannt. Er ist nach [Schn 96] der am häufigsten eingesetzte Algorithmus mit öffentlichen Schlüsseln. RSA kann zur Verschlüsselung und auch für digitale Signaturen verwendet werden. RSA ist nach [Schn 96] ein De-Facto-Standard in weiten Teilen der Welt.

Digital Signature Algorithm (DSA). Ist ein Algorithmus mit öffentlichem Schlüssel. Er kann zur Verschlüsselung und für digitale Signaturen verwendet werden.

Eine Beschreibung der Arbeitsweise der Algorithmen befindet sich in [Schn 96].

2.4.2 Sicherheit von Algorithmen

Die verschiedenen Algorithmen bieten ein unterschiedliches Maß an Sicherheit an. Ein Algorithmus ist wahrscheinlich sicher, wenn

- der zum Aufbrechen eines Algorithmus erforderliche Geldaufwand den Wert der verschlüsselten Daten übersteigt;
- die zum Aufbrechen eines Algorithmus notwendige Zeit größer ist als die Zeitspanne, welche die verschlüsselten Daten geheim bleiben müssen;
- das mit einem bestimmten Schlüssel chiffrierte Datenvolumen kleiner ist als die Datenmenge, die zum Aufbrechen des Algorithmus erforderlich ist.

Ein Algorithmus ist **uneingeschränkt** sicher, wenn der Klartext auch dann nicht ermittelt werden kann, wenn Chiffretext in beliebigem Umfang vorhanden ist.

Die folgende Tabelle gibt Aufschluß, wovon die Sicherheit der Computer-Algorithmen DES, RSA und DAS abhängt:

Algorithmus	**Sicherheit des Algorithmus**
DES	hängt ab von Schlüssellänge, Anzahl der Iterationen, Gestalt der S-Boxen [Schn 96]
RSA	basiert auf Problem der Faktorisierung großer Zahlen [Schn 96]
DSA	hängt ab von Schlüssellänge

2.4.3 Vertraulichkein, Datenintegrität und Authentifizierung

Unter **Vertraulichkeit** versteht man, daß nur die an der Kommunikation beteiligten Parteien die ausgetauschten Nachrichten lesen können und selbst beim Abhören der Datenleitungen, ein Zugriff auf den Inhalt der Nachrichten verhindert werden kann. Dies wird durch Verschlüsselung aller übertragenen Daten erreicht.

Datenintegrität der übertragenen Daten beschreibt, daß die gesendete Nachricht mit der empfangenen Nachricht übereinstimmt, d.h. nicht manipuliert wurde. Clients und Server müssen die Möglichkeit haben, festzustellen, daß der jeweilige Partner auch der ist, für den er sich ausgibt. Daher müssen sich alle Rechner **authentizifieren**.

Authentifizierung. Woher weiß der Empfänger einer Nachricht, daß der Sender derjenige ist, für den er sich ausgibt? Nach [Schn 96] wird dieses Problem mit Paßwörtern gelöst. Es reicht aus, daß der Empfänger der Nachricht ein gültiges von einem ungültigen Paßwort unterscheiden kann. Er braucht nicht das Paßwort des Senders zu kennen. Anstelle eines Paßwortes kann der Empfänger den Sender einer Nachricht auch mit Public-Key-Algorithmen identifizieren (siehe [Schn 96]).

2.4.4 Digitale Signaturen

Eine digitale Signatur wird zur Feststellung der Echtheit von elektronisch übermittelten Nachrichten verwendet. Sie ist das elektronische Äquivalent zur handschriftlichen Untertschrift. Durch Überprüfung der digitalen Signatur läßt sich feststellen, ob die zugehörige Nachricht verändert wurde. Zur Überprüfung können assymetrische Verschlüsselungsverfahren eingesetzt werden.

2.5 Netzwerk-Informationsdienste

Nach einer Definition des Begriffs Netzwerk-Informationsdienst werden in diesem Abschnitt der Standard X.500 und die Netzwerk-Informationsdienste DNS, NIS, NIS+ und NDS vorgestellt. Bei deren Beschreibung wird auf die Anforderungen in der Aufgabenstellung für die Diplomarbeit eingegangen: Client-Server-Architektur, Datenbank zur Speicherung der Informationen, hierarchisches Domänenkonzept, Verschlüsselung der Daten, Erweiterbarkeit der angebotenen Dienste und Plattformunabhängigkeit.

2.5.1 Definition

Netzwerk-Informationsdienste (network directories) speichern und organisieren Informationen über Netzwerk-Betriebsmittel. Unter Netzwerk-Betriebsmittel versteht man Server, die Netzwerkdienste bereitstellen, Drucker und Benutzer. Netzwerk-Informationsdienste bieten Administratoren und Benutzern Dienste an, die Informationen über Netzwerk- und System-Betriebsmittel enthalten. Die Dienste regeln auch den Zugriff auf diese Betriebsmittel. Beispiele für Netzwerkinformationen sind:

- Netzwerkadressen
- Sicherheits-Informationen
- Mail-Informationen
- Informationen über Ethernet-Schnittstellen
- Informationen über Netzwerk-Dienste
- Informationen über Benutzergruppen mit Zugriffsberechtigung auf das Netzwerk
- Informationen über die im Netzwerk zu Verfügung stehenden Dienste

Diese Informationen werden in Konfigurationsdateien gespeichert. Beispiele für Konfigurationsdateien des Betriebssystem UNIX sind die /etc-Dateien wie /etc/hosts, /etc/protocols, /etc/passwd und weitere. Wenn sich diese Informationen ändern, müssen Systemverwalter sie auf jeder Workstation im Netzwerk aktualisieren. Das ist in einem großen Netzwerk sehr zeitaufwendig und auch unüberschaubar. Ein Netzwerk-Informationsdienst übernimmt diese Aufgabe. Er speichert die Netzwerk-Informationen auf einem Server und stellt sie jeder sie anfragenden Workstation zur Verfügung. Der Systemverwalter aktualisiert nur die Informationen auf dem Server. Dies reduziert die Fehlerzahl, Inkonsistenzen zwischen den Daten und den Umfang der Aufgabe.

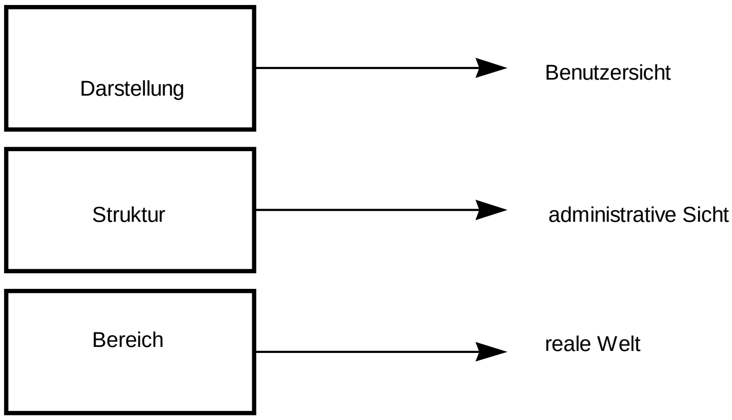
2.5.2 Standard X.500

X.500 ist der aktuelle internationale Standard für Verzeichnisdienste (directory services). Verzeichnisdienste umfassen Netzwerk-Informationsdienste und andere Arten von Verzeichnisdiensten wie Systemverzeichnisdienste und Verzeichnisdienste für Applikationen. Ein Verzeichnisdienst ist nach [KeIv 98] eine Art Datenbank. Er erlaubt Benutzern und Anwendungsprogrammen, Objekte ausfindig zu machen, indem Informationen benützt werden, die mit diesen Objekten verbunden sind. In Netzwerk-Informationsdiensten erhalten Benutzer und Anwendungsprogramme Zugang zu Informationen über Netzwerk-Betriebsmittel, während zugleich für die Erhaltung, Verteilung und Sicherheit der Information gesorgt wird.

X.500 wurde in Zusammenarbeit des Consultative Committee for International Telegraphy and Telephony (CCITT), der International Standards Organization (ISO) und der International Electrotechnical Commission (IEC) entwickelt. Die erste Version wurde 1988, die zweite, aktuelle Version 1992 und 1993 veröffentlicht. Die zweite Version stellt eine Überarbeitung der ersten dar, ohne sie jedoch zu ersetzen. Beide wurden offiziell mit Version 1 bezeichnet. Im folgenden wird nur auf die aktuelle Version „1“ von 1992 und 1993 eingegangen.

Schichtenmodell für Verzeichnisdienste

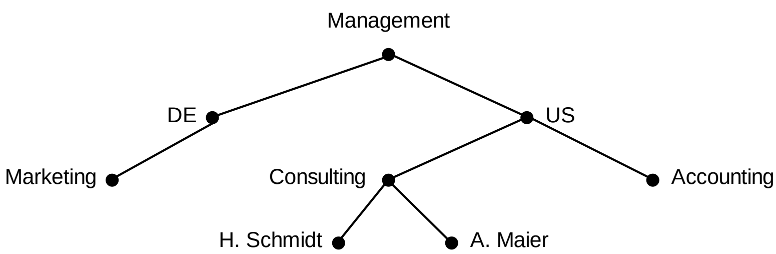
In X.500 besteht ein Verzeichnisdienst aus den Schichten Bereich (scope), Struktur (structure) und Darstellung (presentation):



Die unterste Schicht des Verzeichnisdienstes ist der **Bereich** eines Verzeichnisdienstes. Er legt die Grenzen des Ausschnitts der realen Welt fest, welche die Elemente des Verzeichnisdienstes und die Informationen, die mit den Elementen verknüpft sind, darstellen. Der Bereich stellt somit die reale Welt bzw. physikalische Sicht des Verzeichnisdienstes dar. Im Bereich werden Objektklassen und Attribute definiert.

Die **Struktur** eines Verzeichnisdienstes bestimmt, wie die Informationen im Verzeichnisdienst gespeichert und verwaltet werden. Somit stellt die Struktur die administrative Sicht eines Verzeichnisdienstes dar. Sie dient der Objektidentifizierung (object identification) und der Objektorganisation (object organization). Die Objektidentifizierung sorgt dafür, daß jedes Objekt eindeutig identifizierbar ist. Die Objektorganisation dient der Erhaltung von logischen Unterklassen von Objekten. Dadurch soll der Zugang zu Informationen von Objekten vereinfacht und die Übertragbarkeit von Teilen des Verzeichnisdienstes möglich gemacht werden. Mit DIB (Directory Information Base) wird die Gesamtheit der Information, welche die Datenbank des Verzeichnisdienstes in Form von Objektklassen und Attributen enthält, bezeichnet. Die Einträge im DIB sind hierarchisch aufgebaut. Der Verzeichnis-Informationsbaum (directory information tree, DIT) stellt die logische Organisierung des Inhalt eines Verzeichnisdienstes dar.

Die folgende Abbildung zeigt als Beispiel den Verzeichnis-Informationsbaum für ein internationales Wirtschaftsunternehmen:



Das **Schema** eines Verzeichnisdienstes ist ein Teil der Directory Information Base (DIB). Es enthält die Definition aller Objektklassen, Attribute und Beziehungen zwischen den Objektklassen und Attributen. Des weiteren enthält es hierarchische Beziehungen zwischen den Objektklassen. Im Schema wird sichergestellt, daß ein Objekt alle obligatorischen Attribute besitzt, keine unerlaubten Werte für die Attributstypen benutzt und nur die optionalen Attribute erhält, die für die entsprechende Objektklasse erlaubt sind.

Replikation

X.500 sieht die Replikation von Informationen vor. Darunter versteht man das Speichern von mehrfachen Kopien derselben Information an verschiedenen Stellen im Verzeichnisdienst. Durch Replikation können Laufzeit und Verfügbarkeit verbessert werden.

Zugriffskontrolle

Um unerlaubten Zugriff auf Elemente des Verzeichnisdienstes zu verhindern sind in X.500 zwei Arten von Authentifizierungen vorgesehen: einfache und starke Authentifizierung. In der einfachen Authentifizierung wird ein Paßwort verlangt, um den Benutzer zu identifizieren. Die starke Authentifizierung benützt kryptographische Verfahren, die auf öffentliche Schlüssel basieren. Des weiteren gibt es in X.500 Richtlinien für die Verwendung von digitalen Signaturen und Zertifikaten.

Die **Darstellung** des Verzeichnisdienstes stellt die Benutzersicht und die oberste Schicht des Verzeichnisdienstes dar. Bei der Darstellung des Verzeichnisdienstes gibt es verschiedene Ziele:

- Vollständigkeit der Informationen
- Zugriffsmöglichkeiten auf alle Objekte
- Exaktheit der Informationen
- Effizienz bezüglich Laufzeit und Kosten der Prozesse

2.5.3 Domain Naming Service (DNS)

Der Domain Naming Service (DNS) ist ein im Internet für Netzwerke unter TCP/IP weit verbreiteter Netzwerk-Informationsdienst. DNS stellt als grundlegenden Dienst die **Namenskonvertierung** zu Verfügung. Darunter versteht man die Übersetzung von Namen der Workstations in deren IP-Adressen und umgekehrt.

Das Modell des Domain Naming Service ist in den Requests For Comments (RFCs) 1034 und 1035 beschrieben. Es verfügt über eine Client-Server-Architektur und einem hierarchischen Zuordnungsraum.

2.5.3.1 Hierarchischer Zuordnungsraum

Als DNS-**Zuordnungsraum** wird die Menge aller untereinander vernetzten Workstations bezeichnet. Der DNS-Zuordnungsraum kann in eine Hierarchie von DNS-Domains unterteilt werden. Eine DNS-**Domain** oder Zone stellt eine Gruppe von Elementen des zu verwaltenden Netzwerkes dar. Beispiele für DNS-Domain sind Drucker oder Workstations. Organisationen mit Netzwerken, die an das Internet angeschlossen werden können zwischen geographischen und organisatorischen Hierarchien wählen.

2.5.3.2 Client-Server-Struktur

DNS-**Server** werden auch als Domain-Server, Zuordnungsserver oder name server bezeichnet. Sie unterteilen sich in Principal- und Neben-Server. Jede DNS-Domain wird von einem Principal- und einem oder mehreren Neben-Server verwaltet. Die Server haben die Aufgabe, die Namen und IP-Adressen der Workstations zu speichern. Principal-Server speichern die Originalinformationen. Nebenserver speichern Kopien der Originalinformationen.

DNS-**Clients** sind Rechner, die zu einer bestimmten DNS-Domain gehören. Sie senden ihre Anfragen nur an die Server, die ihre DNS-Domain verwalten. Verfügt der Server nicht über die Information, leitet er die Anfrage an den Vaterserver weiter. Dies ist der Server der übergeordneten Domain in der Hierarchie. Wenn die Anfrage den Wurzelserver erreicht, der oberste Server in der Hierarchie, prüft er die Gültigkeit der Domain. Falls die Domain nicht gültig ist, wird die Antwort „nicht gefunden“ bis zum Client weitergereicht. Im Fall, daß die Domain gültig ist, leitet der Wurzelserver die Anfrage an den Server weiter, der die entsprechende Domain verwaltet.

2.5.4 Network Information Service (NIS)

Der Network Information Service (NIS) wurde von der Firma Sun Microsystems für das Betriebssystem Solaris entwickelt. Ursprünglich lautete der Name für diesen Netzwerk-Informationsdienst Yellow Pages (YP). Nachdem dieser Name jedoch geschützt war, wurde er in NIS abgeändert. NIS erweitert bzw. ersetzt viele der /etc-Dateien, der Konfigurationsdateien in UNIX-Systemen. Die Netzwerk-Informationen können auf einem einzigen System gespeichert und von vielen Rechnern miteinander geteilt werden. Somit bietet NIS zentrale Kontrolle über Netzwerk-Informationen. Beispiele für Netzwerk-Informationen sind Informationen über:

- Workstation-Namen
- Workstation-Adressen
- Anwender
- das Netzwerk selbst
- Netzwerkdienste

2.5.4.1 Kein hierarchischer Zuordnungsraum

Die Sammlung von Netzwerk-Informationen wird als **NIS-Zuordnungsraum** bezeichnet. Innerhalb des Zuordungsraumes verwendet NIS **Domains**, um Workstations, Anwender und Netzwerke zu organisieren. In NIS existiert jedoch **keine** Domain-Hierarchie. Ein NIS-Zuordnungsraum hat nur eine Ebene.

2.5.4.2 Anbindung an das Internet

Eine NIS-Domain kann nicht direkt an das Internet angeschlossen werden. Eine Anbindung an das Internet wird durch eine Kombination von DNS (Domain Naming Service) und NIS erreicht.

2.5.4.3 Client-Server-Struktur

Es gibt zwei Arten von **NIS-Servern**: Master-Server und Slave-Server. Letzere werden auch als Sicherheits-Server bezeichnet. Beide speichern die NIS-Karten. **NIS-Karten** oder auch NIS-Maps wurden entworfen, um die /etc-Dateien von UNIX und andere Konfigurationsdateien zu ersetzen. Sie enthalten die Netzwerk-Informationen. NIS-Karten sind überwiegend zweispaltige Listen. Eine Spalte enthält den Schlüsselwert, die andere Informationen über den Schlüsselwert.

Redundanz der Informationen

Informationen für Clients findet NIS nur über den Schlüsselwert. Daher müssen manche Informationen mehrfach gespeichert werden, wobei dann jede Karte einen anderen Schlüssel verwendet. Zur Verdeutlichung ein Beispiel: benötigt ein Server den Namen einer Workstation und kennt die Adresse, sucht er in der NIS-Karte host.byaddr. Für den Fall, daß der Server den Namen der Workstation kennt und ihre Adresse benötigt, muß er in der NIS-Karte hosts.byname suchen.

NIS-Clients stellen Anfragen zu Netzwerk-Informationen an einen oder mehrere Server. Als erstes wird die Adresse eines aktiven NIS-Server ermittelt. An diesen wird dann die Anfrage gesendet, und der Client erhält die gewünschte Information.

2.5.4.4 Plattformen

NIS wurde für das Betriebssystem SunOS (Solaris) entwickelt. Es ist auch für weitere UNIX-Systeme verwendbar.

2.5.4.5 Sicherheit

Aufgrund der fehlenden Authentifizierungs-Möglichkeiten bei NIS-Servern wurde NIS+ entwickelt.

2.5.5 Network Infomation Service Plus (NIS+)

Der Network Information Service Plus (NIS+) ist der Nachfolger von NIS.

2.5.5.1 Hierarchischer Zuordnungsraum

Der **NIS+-Zuordungsraum** wird als Anordnung der von NIS+ gespeicherten Netzwerk-Informationen definiert. Er besitzt Verzeichnisse, Listen und Gruppen als Bestandteile. In NIS+ existiert im Gegensatz zu NIS ein **hierarchischer Zuordnungsraum**. Dadurch wird eine flexiblere und dezentralisierte Verwaltung ermöglicht. Somit kann NIS+ im Gegensatz zu NIS auch in sehr großen Netzwerken eingesetzt werden.

2.6

Die Bestandteile des NIS+-Zuordnungsraumes werden **NIS+-Objekte** genannt. Die NIS+-Objekte können in einer Hierarchie, in sogenannten hierarchischen Domains, angeordnet werden.

Eine **NIS+-Domain** (Domänen) speichert neben Namen- und Adreßinformationen der Clients eine Sammlung von Informationen über Workstations, Anwender und Netzwerk-Dienste innerhalb eines Teils einer Organisation. Dies erlaubt dem Administrator eine dezentralisierte Verwaltung. Eine NIS+-Domain besteht aus einem Verzeichnisobjekt, welches NIS+-Gruppen enthält, einem Verzeichnisobjekt, welches NIS+-Listen enthält, und mehreren NIS+-Listen. NIS+-Domains sind logische, aber nicht physikalische Bestandteile des Zuordnungsraumes. Somit ist eine Domäne kein Objekt, sondern nur eine Sichtweise auf eine Gruppe von Objekten. In NIS+ kann eine Domäne aus einer anderen Domäne heraus verwaltet werden.

2.6.1.1 Client-Server-Struktur

Wie NIS besitzt NIS+ eine Client-Server-Struktur. Jede NIS+-Domain wird von einer Gruppe von **NIS+-Servern** versorgt. Diese Server speichern die Verzeichnisse, Gruppen und Listen der Domänen und beantworten die Zugriffsanfragen von Anwendern, Verwaltern und Anwendungen.

In NIS+ gibt es zwei Arten von Server: einen Master-Server und seine Sicherheitsserver, die Replika-Server genannt werden. Beide verwalten NIS+-Listen. Jede Workstation des Betriebssystems Solaris in der Version 2.3 kann als Server arbeiten.

Master-Server

Master-Server speichern die Originallisten. Alle Änderungen von Informationen müssen auf dem Master-Server vorgenommen werden. Die Änderungen werden auf Festplatte und in einer Transaktionsliste gespeichert. Nach Abschluß werden die Änderungen automatisch an die Replika-Server weitergereicht und stehen dann dem gesamten Zuordnungsraum zur Verfügung. Hat der Master-Server alle seine Replika-Server aktualisiert, löscht er die Transaktionsliste.

Replika-Server

Replika-Server speichern die Kopien der Originallisten.

2.6.1.2 NIS+-Listen

NIS+ speichert seine Informationen in NIS+-Listen (vgl. NIS-Karten bei NIS). Eine NIS+-Liste enthält Informationen über ihre lokale, nicht jedoch über andere Domänen. Neben vordefinierten Systemlisten kann man auch eigene Listen erzeugen, sogenannte Nicht-Systemlisten. NIS+-Listen sind aus Spalten und Einträgen aufgebaut. Auf NIS+-Listen kann über jede Spalte zugegriffen werden.
Dadurch wird die bei NIS „unnötige“ Redundanz von Daten vermieden. Die Informationen in NIS+-Listen können auf Listen-, auf Eintrags- und auf Spaltenebene verändert werden. Da das Aktualisieren der Listen

langwierig ist, verwenden alle NIS+-Server Logdateien für ihre Listen. In diesen werden die Veränderungen vorübergehend gespeichert. NIS+-Listen können auf drei Arten gefüllt werden:

- aus NIS-Karten
- aus ASCII-Dateien wie den /etc-Dateien
- per Hand

2.6.1.3 Sicherheit

Sie teilt sich in Berechtigung und Freigabe:

Unter **Berechtigung** versteht NIS+ die Festlegung aller Zugriffsrechte und Zugriffsarten. Bei der Berechtigung identifiziert ein Server die Zugriffsrechte, die diesem Principal bewilligt wurden.

Jede Anfrage nach einem Zugriff auf den Zuordnungsraum kann erlaubt, dies bezeichnet die **Freigabe**, oder zurückgewiesen werden. Bei der Freigabe identifiziert ein NIS+-Server einen NIS+-Principal, der eine bestimmte Anfrage gesendet hat.

NIS+-Principals sind Benutzer, Anwendungen oder Workstations (Clients). Sie werden über ihre Kennungen identifiziert. Bei den NIS+-Principals gibt es die Principalklassen: Eigentümer, Gruppe, Welt und Niemand.

NIS+-Objekte unterscheiden folgende Arten von **Zugriffsrechten**, die Berechtigungen für NIS+-Principals festlegen:

1. Lesen
2. Verändern
3. Erzeugen
4. Löschen

Die Zugriffsrechte werden nicht in einer NIS+-Liste gespeichert, sondern sind Bestandteil der Objektdefinition.

Workstations, die mit Solaris 2.x arbeiten, können ihre Informationen von mehr als einem Netzwerk-Informationsdienst erhalten:

- aus lokalen Dateien (/etc-Dateien)
- aus NIS-Karten
- aus DNS-Zonen-Dateien
- aus NIS+-Listen

Dies stellt eine wesentliche Verbesserung zu NIS da. Denn bei NIS gab es das Problem, bei Nichtverfügbarkeit eines Servers keine Eintragungen mehr vornehmen zu können.

2.6.2 Network Directory Service (NDS)

NDS wurde von der Firma Novell entwickelt. Es stellt den am weitesten verbreiteten und am häufigsten eingesetzten Netzwerk-Informationsdienst dar. Novell selbst bezeichnet NDS als verteilte, replizierte Datenbank. NDS wurde auf Grundlage des Standards X.500 entwickelt.

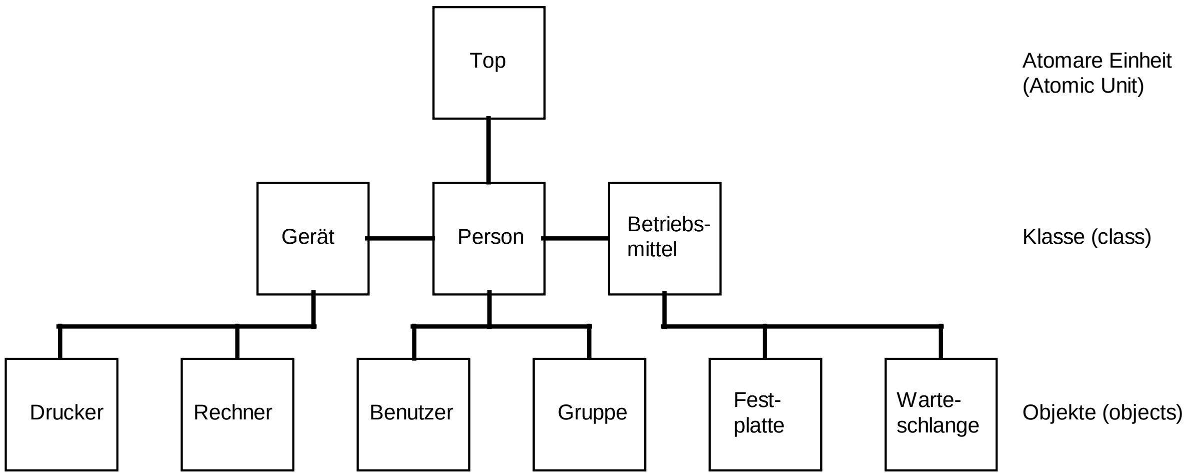
2.6.2.1 Client-Server-Struktur

NDS besitzt eine Client-Server-Architektur. **Server** sind Anwendungen (software applications), die auf Anforderungen der Clients antworten. **Clients** können Anwendungen, Benutzer oder Rechner sein.
Die Server dienen der Steuerung von gemeinsam benutzten Betriebsmitteln. Beispiele für Server sind Dateiserver, Druckerserver und Datenbankserver. Die Software für die Clients wird in [Kelv 98] im Kapitel 14 detailliert beschrieben.

2.6.2.2 Schema

Das **NDS-Schema** definiert die Struktur der NDS-internen Datenbank und legt zugleich fest, wie der **Verzeichnisbaum** konstruiert ist. Die physikalische Struktur wird in Form von Tabellen, Spalten, Feldern, Feldtypen und Schlüsseln im Schema festgelegt. Die „metaphysische“ Struktur legt die Objektklassen, Attribute, Beziehungen zwischen Objektklassen und Attributen und hierarchische Beziehungen zwischen Objektinstanzen fest. NDS stellt ein sogenanntes Basisschema bereit. Es enthält vordefinierte Objektklassen wie Drucker (Printer), Rechner (Computer), Benutzer (User) Gruppe (Group) und andere. Dieses Basis-Schema kann erweitert werden, so daß eigene Objektklassen hinzugefügt werden.

Beispiel für die Konstruktion von Objekten nach einem Schema



Dieses Beispiel ist [Kelv 98] auf S. 93 entnommen. Die **Atomare Einheit** hat die Eigenschaften Name, Ort und Erweiterbarkeit. In der Abbildung wird die Atomare Einheit mit Top bezeichnet. Aus dieser wurden in diesem Beispiel die drei Objektklassen Gerät, Person und Betriebsmittel erstellt. Die drei Objektklassen erben die Eigenschaften Name, Ort und Erweiterbarkeit. Sie besitzen zusätzliche Eigenschaften. Jeder der Klassen besitzt Objekte. Die Klasse Gerät besitzt z.B. die Objekte Drucker und Rechner. Das Objekt Drucker besitzt neben der Eigenschaften seiner Klasse Gerät weitere spezifische Eigenschaften. Durch die spezifischen Eigenschaften unterscheidet es sich vom Objekt Rechner, welches zur gleichen Klasse gehört.

2.6.2.3 Objekte

Jedes NDS-Objekt basiert auf einer Objektklasse im NDS-Schema. Im Schema wird definiert, an welcher Stelle im Verzeichnisbaum sich das Objekt befindet, welche Attribute es besitzen muß, welche es besitzen darf und welche es nicht besitzen darf. NDS Objekte werden in Behälterobjekte (container objects) und Blätterobjekte (leaf objects) unterteilt:

Behälterobjekte sind Objekte, die weitere Objekte enthalten können. NDS definiert vier Behälterobjekte:

- Land
- Örtlichkeit
- Organisation
- organisatorische Einheit

Jeder Verzeichnisbaum muß wenigstens aus einem Organisationsobjekt bestehen.

Blätterobjekte sind Objekte, die keine weiteren Objekte enthalten. Beispiele für Blätterobjekte sind Drucker, Rechner, Benutzer, Gruppe und weitere.

Zugriffskontrollisten

Es wird für jede Person, die sich im Netzwerk anmeldet, ein Benutzerobjekt erstellt. Als Eigenschaften erhält das Benutzerobjekt z.B. ein Paßwort und ein Heimverzeichnis (home directory). Administratoren definieren Gruppen hauptsächlich für die Übertragung von Zugriffsrechten auf Dateisysteme. Alle Mitglieder einer Gruppe erben die Zugriffsrechte der Gruppe. Jedes Objekt im Verzeichnisbaum besitzt als Eigenschaft eine **Zugriffskontrolliste** (ACL). Die Zugriffskontrolliste enthält Informationen, wer oder was auf das Objekt zugreifen und es ändern darf.

NDS unterscheidet zwischen zwei Arten von Rechten: Objektrechte und Eigenschaftsrechte.

Objektrechte regeln, was die Verwalter (trustees) eines Objekts mit dem Objekt tun dürfen. Es gibt fünf verschiedene Objektrechte:

supervisor	erlaubt Zugriff auf alle Rechte des Objekts und seiner Eigenschaften
browse	das Objekt darf im Verzeichnisbaum angezeigt werden
create	ein neues Objekt darf in einem Behälterobjekt im Verzeichnisbaum erzeugt werden
delete	ein Objekt darf aus dem Verzeichnisbaum gelöscht werden
rename	der Name des Objekts darf geändert werden

Eigenschaftsrechte regeln, auf welche Eigenschaften eines Objekts die Verwalter zugreifen und mit ihnen tun dürfen. Wobei die Rechte für jede Eigenschaft einzeln geregelt werden. Es gibt fünf verschiedene Eigenschaftsrechte:

add or delete self	erlaubt, sein eigenes Objekt im Verzeichnisbaum hinzuzufügen oder zu löschen; das Eigenschaftsrecht write enthält dieses Recht
compare	der Wert der Eigenschaft darf mit dem einer anderen Eigenschaft verglichen werden
read	die Eigenschaftswerte dürfen gelesen werden; dieses Recht enthält das Recht compare
supervisor	alle Rechte der Eigenschaft werden gewährt
write	sämtliche Werte der Eigenschaft dürfen hinzugefügt, geändert oder entfernt werden; dieses Recht enthält das Recht add or delete self

2.6.2.4 Replikation und Partitionierung

Besondere Bedeutung im Konzept von NDS und auch X.500 haben Verteiltheit (distribution) und Replikation. Unter **Verteiltheit** versteht man die Möglichkeit, verschiedene Teile der Daten innerhalb der Datenbank in Untermengen zu trennen. Unter **Replikation** versteht man die Möglichkeit, identische Kopien der Daten auf getrennten Plattformen zu erzeugen. Je mehr Kopien man hat, desto länger braucht das Synchronisieren (siehe unten). Je mehr Partitionen es gibt, deso schneller kann das Synchronisieren erfolgen. Nach Möglichkeit sollten Benutzer, die sich im Netzwerk anmelden, dies auf Partitionen vornehmen, die auf Servern gespeichert sind, die sich physikalisch nahe zu ihnen befinden. Die Vorteile der Replikation sind Fehlertoleranz und Verfügbarkeit. Fehlertoleranz wird erlangt, indem identische Information auf verschiedenen Servern verteilt wird. Falls ein Server ausfällt, ist die Information immer noch von anderen Servern aus erreichbar. Diese Server müssen dann synchronisiert werden. Dazu verwendet NDS das Datum und Zeitstempel.

Bei NDS gibt es drei Arten von Kopien (replicas):

- Master
- Read/WRITE
- Read-Only

Pro Partition darf es nur ein Master-Replikat geben. Es wird von NDS empfohlen, von jeder Partition mindestens ein Master- und zwei Read/Write-Replikate zu halten, um die Fehlertoleranz zu steigern. Es sollten jedoch nicht mehr als 10 bis 15 Replikate gehalten werden, um die Bandbreite während der Synchronisation zu minimieren. Auf einem Server kann jeweils nur eine Kopie einer bestimmten Partition gespeichert werden. NDS legt aus folgenden Gründe wert auf seine Verteiltheit und Repliziertheit, indem es die Nachteile von zentralisierten Verzeichnissystemen aufführt:

- Ein Server, Host oder Domain-Controller muß erreichbar sein, um sich anmelden zu können.
- Hat man eine Wide Area Network (WAN)-Verbindung zu diesem, kann die Authentifizierung sehr langsam sein.
- Fällt er aus, können unter Umständen keine Änderungen vorgenommen und möglicherweise das gesamte Verzeichnis nicht wiederherstellbar sein.

2.6.2.5 Zeitsynchronisation

Bei jeder Änderung verwendet NDS Zeitstempel, um sicherzustellen, daß alle Replikate der Datenbank in der exakt gleichen Reihenfolge geändert werden. Dieses System funktioniert jedoch nur, wenn alle Server, die von der Replikation betroffen sind, in ihrer Zeit miteinander übereinstimmen. Um dies zu erreichen, besitzt NDS Zeit-Server.

2.6.2.6 Pflege der Datenbank

Die hohen und vielfältigen Sicherheitsvorkehrungen, die in NDS getroffen werden, unterscheiden es von anderen Netzwerk-Informationsdiensten. Dies erfordert jedoch eine hohe Netzlast. Daher sieht NDS für die meisten seiner Sicherheitsprogramme die Zeit nach der Arbeitszeit als Laufzeit vor. Für international übergreifende Organisationen kann dies aufgrund eines 24-Stunden-Betriebes zu Problemen führen. Je mehr Kopien es gibt, desto länger brauchen die Konsistenzprüfungen. Werden bei den Konsistenzprüfungen Fehler festgestellt, verfügt NDS über Programme, die diese Fehler beheben können. Falls eine Fehlerbehebung nicht möglich ist, dienen die Sicherheitskopien der Wiederherstellung der Datenbank.

2.6.2.7 Sicherheit

NDS stellt verschiedene Sicherheitsmechanismen zur Verfügung, um unerlaubten Zugriff auf und Änderungen von Netzwerk-Informationen zu verhindern. Dabei werden folgende Sicherheitsmechanismen von NDS verwendet:

Zugriffskontrollisten

Jedes Element bzw. Objekt im Baum verfügt über eine Zugriffskontrolliste (access control list, ACL). In der Zugriffskontrolliste stehen Informationen wer oder was auf ein Objekt zugreifen darf und/oder was ein anderes Objekt mit dem erst genannten Objekt und seinen Eigenschaften anstellen darf.

Identifizierung und Authentifizierung

Darunter versteht man den Vorgang, bei welchem die Identität eines Benutzers oder eines Rechners, der auf Netzwerk-Informationen zugreifen möchte, festgestellt wird. In NDS (und X.500) ist eine einfache (simple) sowie eine starke (strong) Authentifizierung möglich. Für die einfache Authentifizierung wird lediglich ein Paßwort zur Verifizierung der Identität des Benutzers benötigt. Für die starke Authentifizierung werden öffentliche Schlüssel (RSA) benützt. Des weiteren können digitale Signaturen und Zertifikate verwendet werden.

Auditing

Unter Auditing versteht man den Vorgang, bei dem authorisierte Prüfer die Aufzeichnungen eines Unternehmens begutachten. Dabei wird unter aderem die Sicherheit von vertraulichen Informationen überprüft. NDS stellt für das Auditing Dateien zur Verfügung, die eine Beurteilung dieser Sicherheit erlauben.

2.6.2.8 Plattformen

NDS ist bisher nur mit dem Betriebssystem NetWare in der Version 4.2 oder höher einsetzbar. Es ist vorerst nur auf folgenden Betriebssystemen lieferbar (Stand Dezember 1998):

- NetWare 5, NetWare 4 und intraNetWare
- Windows NT 3.51 und Windows NT 4
- Solaris ab dem 1. Quartal 1999
- weitere UNIX-Betriebssysteme in Bearbeitung

2.6.3 Microsoft Active Directory (MAD)

Microsoft Active Directory (MAD) ist ein Netzwerk-Informationsdienst der Firma Microsoft. Er befindet sich noch im **Entwicklungsstadium**. Die erste Version ist nach [Nove 99] für das Jahr 2000 vorgesehen. Nach [Nove 99] ist MAD nur für Betriebssysteme der eigenen Firma entwickelt und somit **nicht plattformunabhängig**.

2.6.4 Zusammenfassung

Es folgt eine Einordnung der vorgestellten Netzwerk-Informationsdienste bezüglich der für die vorliegende Arbeit gestellten Anforderungen (siehe Kapitel 1). Die Zeilen in der folgenden Matrix stellen die einzelnen Anforderungen, die Spalten die verschiedenen Netzwerk-Informationsdienste dar :

	DNS	**NIS**	**NIS+**	**NDS**	**MAD**
Client-Server-Struktur	ja	ja	ja	ja	ja
Datenbank	ja	nein	nein	ja	ja
hier. Domänen-Konzept	ja	nein	ja	ja	ja
Datenverschlüsselung	nein	nein	ja	ja	ja
Erweiterbarkeit d. Dienste	nein	nein	ja	ja	ja
Plattformunabhängigkeit	ja	nein	nein	nein	nein

Fazit:

Aus der Matrix geht hervor, daß keiner der vorgestellten Netzwerk-Informationsdienste die in der Aufgabenstellung genannten Anforderungen erfüllt.

In Eingrenzung der Aufgabenstellung wurde darauf verzichtet, daß der Entwurf für das Rechner-Administrierungssystem (siehe Kapitel 3) auf den Standard X.500 aufbaut oder mit den anderen vorgestellten Netzwerk-Informationsdiensten verglichen wird. Die Einarbeitung in die in diesem Kapitel vorgestellten Konzepte haben jedoch wesentlich zur Erstellung des im Kapitel 3 beschriebenen eigenen Konzepts für das Rechner-Administrierungssystem beigetragen.

2.7 Objektorientierte Programmierung in Java

Die Implementierung des Prototyps für das Rechner-Administrierungssystem erfolgt in Java. Java ist eine objektorientierte Programmiersprache. Die hier vorgenommenen Definitionen sind strikt auf Java bezogen und nicht für alle objektorientierten Programmiersprachen gültig.

2.7.1 Objekte und Klassen

Ein **Objekt** ist ein Datentyp, der eine Struktur und einen Zustand besitzt. Jedes Objekt definiert Operationen, die den Zustand des Objekts ändern oder auf dieses zugreifen können. Es können neue Objekte erzeugt und ggf. initialisiert, kopiert und auf Gleichheit verglichen und Ein- und Ausgabeoperationen auf Objekten ausgeführt werden. Ein Objekt in Java ist eine Instanz einer Klasse. Durch die Definition einer Klasse wird der Zustand und die Funktionalität eines Objekts bestimmt. Ein Objekt wird durch die Instantiierung einer Klasse erzeugt. Das Erzeugen einer Instanz einer Klasse bezeichnet man als Instantiierung einer Klasse. Eine **Klasse** besteht aus Methoden und Feldern.

Die im Domänenkonzept verwendeten Begriffe „Objekt" und „Klasse" beziehen sich nicht auf Java und die objektorientierte Programmierung. Im Domänenkonzept stellt eine „Klasse" nur einen „Behälter" dar, der Domänen und Objekte enthält (siehe Kapitel 3.1).

2.7.2 Dokumentation

Das Programm **Javadoc** des Java Development Kits (JDK) erstellt aus den Kommentaren einer Klassendatei automatisch eine Dokumentation im HTML-Format. Die Kommentare müssen die folgende Form besitzen:

*/** Text, der als Kommentar erscheinen soll */*

Der Befehl javadoc *Klassenname*.java erstellt die Dokumentation in Form einer Datei mit dem Namen *Klassenname*.html. Für *Klassenname* wird der entsprechende Name der Klasse eingetragen. Die Dokumentation für den Prototyp des Rechner-Administrierungssystems im Anhang B wurde mit Javadoc erstellt.

2.7.3 Datenbankanbindung

Die Schnittstelle in Java (API), die eine Datenbank-unabhängige Datenbankanbindung herstellt, heißt **Java Database Connectivity** (**JDBC**).

Es können Datenbanken mit relationalen oder objekt-orientierten Datenmodellen benützt werden. Nach [Rees 97] werden hauptsächlich Datenbanken mit relationalen Datenmodellen verwendet.

Das JDBC setzt voraus, daß die entsprechende Datenbank die relationale Datenbanksprache **Structured Query Language** (**SQL**) in der Version 2 der Standardisierung des **American National Standards Institute** (**ANSI**), die mit **ANSI SQL-2** abgekürzt wird, unterstützt. Mit dem JDBC können SQL-Befehle als Argumente für Methoden der JDBC-Schnittstellen verwendet werden.

Die Klassen der JDBC-Schnittstelle, die ein bestimmtes Datenbank-Verwaltungssystem (DBMS) realisieren, werden (JDBC-)**Datenbanktreiber** genannt. Im Internet befinden sich eine Reihe von Datenbanktreibern für die verschiedenen DBMS. Im Dezember 1998 gab es diese nur für die Version 1.1.7 des JDKs und noch nicht für die neueste Version 1.2.

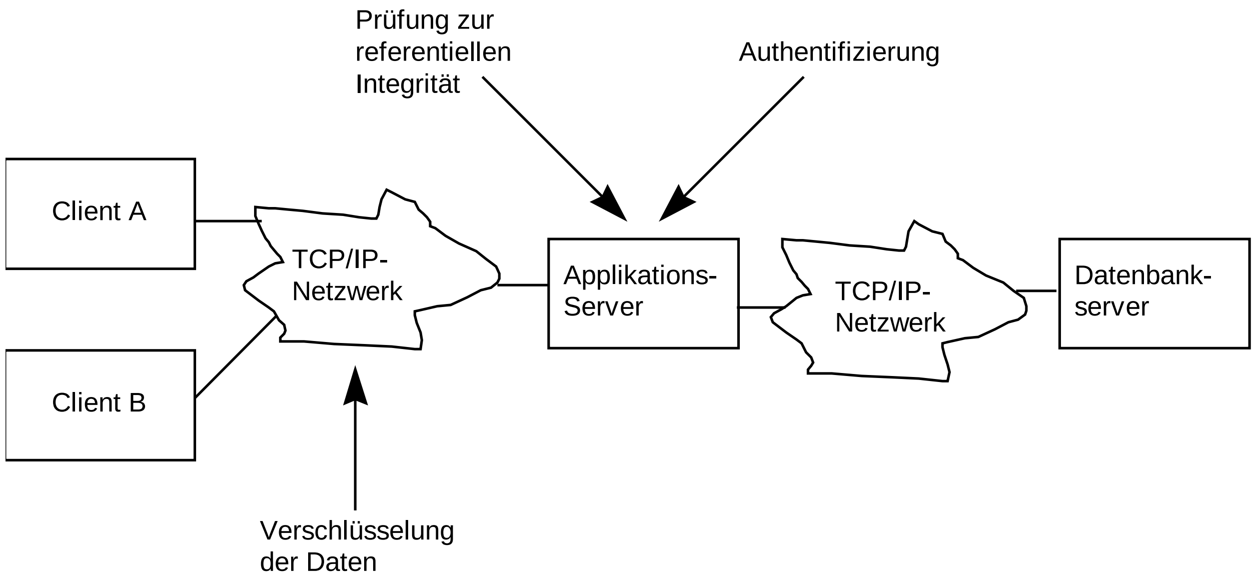
2.7.4 Verschlüsselung in Java

Die im Kapitel 2.4 vorgestellten Verschlüsselungs-Algorithmen stehen nicht in der Version 1.1.7 des JDK (vgl. letzter Abschnitt) zur Verfügung. Es gibt jedoch die Möglichkeit, über das Internet Verschlüsselungs-Algorithmen zu beziehen (shareware). Zum Zeitpunkt der Recherche im Dezember 1998 gab es von denen hier vorgestellten Verfahren nur RSA.

3 Entwurf des Rechner-Administrierungssystems

Der Entwurf für das Rechner-Administrierungssystem läßt sich in ein Domänen-, Datenbank-, Client-Server- und Sicherheitskonzept unterteilen. Es folgt eine Erläuterung des Domänenkonzepts, welches anschließend in das Datenbankkonzept umgesetzt wird. Eingebettet sind beide Konzepte in ein Client-Server-Modell, das sich für Netzwerke anbietet. Das Sicherheitskonzept bezieht sich auf alle drei Konzepte. Es umfaßt referentielle Integrität der Datenbank, Authentifizierung, Datenverschlüsselung, Sicherung der Daten in der Datenbank (Backup) und Wiederherstellung der Datenbank im Fehlerfall (Recovery).

Die folgende Abbildung zeigt das Konzept zum Rechneradministrierungssystems:



Mehrere Clients sind über ein Netzwerk mit einem Applikationsserver verbunden. Alle zwischen Clients und Servern ausgetauschten Daten werden **verschlüsselt**. Die Clienst melden sich an das Rechner-Administrierungssystem mit ihrer Benutzerkennung und ihrem Paßwort an. Diese Eingaben werden in verschlüsselter Form an den Applikationsserver gesendet. Dort findet eine **Authentifizierung** statt. Die Anfragen, welche die Clients über Tastatur oder Kommandozeilen-Schnittstelle in das Rechner-Administrierungssystem eingeben, werden ebenfalls in verschlüsselter Form an den Applikationsserver gesendet. Der Applikationsserver leitet die Anfrage an den Datenbankserver, mit dem er über ein Netzwerk verbunden ist, weiter. Im Datenbankserver wird die entsprechende Information der Datenbank entnommen und an den Applikationsserver gesendet. Der Applikationsserver sendet die Antwort an den Client, der die entsprechende Anfrage gestellt hat. Im Applikationsserver werden **Prüfungen zur referentiellen Integrität** der Datenbank durchgeführt. Das Domänenkonzept bildet die Grundlage für das Schema des Rechner-Administrierungssystems (vgl. Kapitel 2.5.6.2):

3.1 Domänenkonzept

3.1.1 Domänen

Der Begriff „Domäne“ wird an dieser Stelle für eine Verwaltungseinheit eingeführt, die mehrere gleichartige Elemente des Rechner-Administrierungssystems kennzeichnet. So werden z.B. alle Drucker eines Rechner-Administrierungssystems zur Domäne *Drucker* zusammengefaßt, oder auch alle Benutzer zur Domäne *Benutzer*.

Im folgenden wird als Symbol für Domänen ein Rechteck verwendet. Die Domäne wird durch den im Rechteck befindlichen Domänennamen gekennzeichnet:

3.1.2 Objekte

Eine Domäne besteht aus endlich vielen Elementen, die Objekte genannt werden. Sie kann auch leer sein, d.h. keine Objekte enthalten. In der Domäne *Benutzer* kann es z.B. die Elemente Otto, Hans und Agnes geben. Dann sind Otto, Hans und Agnes Objekte.

Im folgenden wird als Symbol für Objekte ein Achteck verwendet. Im Achteck befindet sich der Name des Objekts:

Ein bestimmtes Objekt kann sich in mehreren Domänen befinden (siehe Kapitel 3.1.5.5 Mehrfachzuordnung).

3.1.3 Eigenschaften

Domänen und Objekte besitzen endlich viele Eigenschaften. Dabei wird zwischen den Eigenschaften von Domänen und den Eigenschaften von Objekten unterschieden.

3.1.3.1 Domäneneigenschaften

Domäneneigenschaften sind Eigenschaften, die für eine Domäne und alle Objekte, die dieser Domäne angehören, **zwingend** gelten.

Domäneneigenschaften werden immer im Zusammenhang mit Domänen abgebildet. Der Querstrich an dem Domänensymbol stellt die Domäneneigenschaft dar:

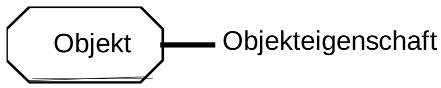
Ein Beispiel für eine Domäneneigenschaft ist: die Domäne Student hat die Eigenschaft, das Rechenzentrum benützen zu dürfen. Die Eigenschaft *benützt Rechenzentrum* gilt dann für alle Studenten.

Beispiel für Domäneneigenschaften

3.1.3.2 Objekteigenschaften

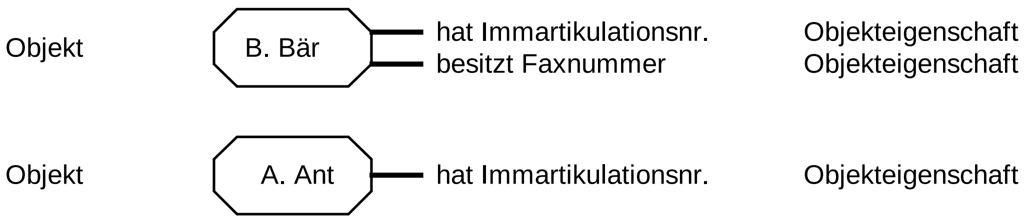
Jedes Objekt besitzt standardmäßig alle Eigenschaften der Domäne, zu der es gehört. Zwei Objekte derselben Domäne besitzen somit die gleichen Domäneneigenschaften, die auch im Wert übereinstimmen. Sie können jedoch über unterschiedliche Objekteigenschaften verfügen.

Das Symbol für Objekteigenschaften wird analog zum Symbol für Domäneneigenschaften festgelegt:



Ein Beispiel für eine Objekteigenschaft ist: Student x besitzt die Faxnummer a. Dabei haben nicht alle Studenten Faxanschluß und daher nicht alle Studenten diese Eigenschaft zugewiesen bekommen. Ein Beispiel für eine Objekteigenschaft, die alle Objekte der Domäne Student besitzen, ist die Immatrikulationsnummer. Diese lautet für jeden Studenten anders.

Beispiel für Objekteigenschaften



Eine Objekteigenschaft unterscheidet sich von einer Domäneneigenschaft dadurch, daß der Wert einer Domäneneigenschaft für alle Objekte dieser Domäne der gleiche ist. Die Werte der Objekteigenschaften können sich im Gegensatz zu den Werten der Domäneneigenschaften unterscheiden.

3.1.3.3 Rechte

Rechte regeln den Zugriff auf Domänen und Objekte. Sie können in Domänen- und Objektrechte unterteilt werden:

3.1.3.3.1 Domänenrechte

Es gibt Rechte, die für alle Objekte einer Domäne gelten. Diese heißen **Domänenrechte**. Sie gelten für eine bestimmte Domäne oder Objekt des Rechner-Administrierungssystems und bestimmen, wie auf eine andere Domäne zugegriffen werden darf.

Es gibt folgende Arten von Domänenrechten:

- es darf auf eine Domäne lesend (**get**) zugegriffen werden

- es darf eine neue Domäne erzeugt (**create**) werden
- es darf eine bestehende Domäne geändert (**set**) werden
- es darf eine bestehende Domäne gelöscht (**delete**) werden

Ein **Beispiel** zum Domänenrecht **delete** lautet: Das Objekt *R. Hofer* der Domäne *Administrator* besitzt das Domänenrecht **delete** bezüglich aller Domänen. Dann darf *R. Hofer* sämtliche Domänen löschen. Da ein Domänenrecht für alle Objekte einer Domäne gilt, bedeutet dies, daß *R. Hofer* auch sämtliche Objekte löschen darf.

3.1.3.3.2 Objektrechte

Objektrechte regeln den Zugriff auf einzelne Objekte. Sie bestimmen, ob und auf welche Weise eine Domäne oder ein Objekt auf ein anderes Objekt zugreifen darf.

Es können folgende Arten von Objektrechten unterschieden werden:

- es darf auf ein Objekt lesend (**get**) zugegriffen werden
- es darf ein neues Objekt erzeugt (**create**) werden
- es darf ein bestehendes Objekt geändert (**set**) werden
- es darf ein bestehendes Objekt gelöscht (**delete**) werden

Es folgt ein **Beispiel** zum Objektrecht **get**: Das Objekt *B. Bär*, welches der Domäne *Benutzer* angehört, besitzt das Objektrecht **get** bezüglich des Objekts *Biologengruppe* der Domäne *Gruppe*; d.h. B. Bär darf auf die *Biologengruppe* lesend zugreifen.

3.1.3.3.3 Einschränkung der Rechte auf einzelne Eigenschaften

Falls nicht auf alle Eigenschaften einer Domäne oder eines Objekts zugegriffen werden darf, können durch Angabe des Namens der Eigenschaften die Rechte auch auf eine Auswahl von Eigenschaften eingeschränkt werden.

Das folgende **Beispiel** verdeutlicht, wie ein Zugriff auf Eigenschaften eingeschränkt werden kann: es soll nur die Eigenschaft *Name* der Domäne *Drucker* für *Benutzer* angezeigt werden können. Dies kann durch folgende Rechtevergabe erreicht werden: Die Domäne *Benutzer* besitzt das Recht **get** gegenüber der Eigenschaft *Name* der Domäne *Drucker*. Dann können alle Objekte der Domäne *Benutzer* nur auf die Eigenschaft *Name* des Domäne *Drucker* lesend zugreifen.

Fehlt die Angabe von Eigenschaftsnamen bedeutet dies, daß der Zugriff auf alle Eigenschaften der Domäne bzw. des Objekts erlaubt ist.

3.1.3.4 Eigenschaft „bezüglich“

Eigenschaften sollen **bezüglich** Domänen oder Objekte definiert werden können. Ein Beispiel für eine solche Eigenschaft lautet: Domäne *Student* besitzt ein *Druckerquotum von 500 Seiten* **bezüglich** der Domäne

Drucker. Durch die Information „**bezüglich**" wird ein Querverweis zu einer anderen Domäne oder einem anderen Objekt hergestellt. Die Eigenschaft bezüglich kann in vier Formen vorkommen:

1. Domäne D1 besitzt die Domäneneigenschaft d bezüglich der Domäne D2.
2. Domäne D1 besitzt die Domäneneigenschaft d bezüglich des Objekts O1.
3. Objekt O1 besitzt die Objekteigenschaft o bezüglich der Domäne D1.
4. Objekt O1 besitzt die Objekteigenschaft o bezüglich des Objekts O2.

Im 1. und 2. Fall kann die Eigenschaft bezüglich **vererbt** werden (siehe nachfolgender Abschnitt 3.1.4). Es sind beide Vererbungstypen AT und AW erlaubt. Im Fall 3. und 4. ist keine Vererbung vorgesehen, da nur Domäneneigenschaften vererbbar sind (siehe Kapitel 5.3.1.2 „Vererbung von Objekteigenschaften").

Im Kapitel 5.3.1.3 „Mehrfache Eigenschaften der Art bezüglich" wird auf den im Konzept nicht behandelten Fall eingegangen, daß eine Domäne oder ein Objekt mehrere „bezüglich"-Verweise besitzen kann.

3.1.4 Vererbung

Zwischen Domänen können Eigenschaften vererbt werden. Vererbung bezieht sich nur auf **Eigenschaften** einer Domäne. Objekteigenschaften sind nicht vererbbar (siehe Kapitel 5.3.1.2 Vererbung von Objekteigenschaften). Es gibt zwei Arten von Vererbungen: **Attributstyp** (**AT**) und **Attributswert** (**AW**):

3.1.4.1 Vererbungstyp AT

AT die Eigenschaft wird vererbt, ohne daß der Wert der Eigenschaft vererbt und somit übernommen werden muß;

3.1.4.2 Vererbungstyp AW

AW die Eigenschaft wird zusammen mit dem Wert der Eigenschaft vererbt; d.h. der Wert der Eigenschaft muß übernommen und darf nicht geändert werden;

Die beiden Vererbungstypen unterscheiden sich darin, ob eine Domäne den Wert einer geerbten Eigenschaft ändern darf (AT) oder nicht (AW).

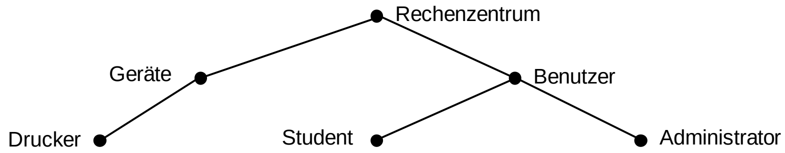
Im nächsten Abschnitt (3.1.5 Domäne) werden die Voraussetzungen erläutert, die eine Domäne erfüllen muß, um von einer anderen Domäne Eigenschaften zu erben. Dort befinden sich auch Beispiele für die zwei Vererbungstypen AT und AW.

3.1.5 Domänenbaum

Die Domänen werden in einem hierarchischen Modell angeordnet. Das Domänenkonzept ist dann als Domänenbaum darstellbar. Ein Domänenbaum wird auch als Verzeichnisbaum bezeichnet (siehe Kapitel 2.1.7 Domänenbaum).

Ein **Domänenbaum** ist ein hierarchischer Baum, der aus endlich vielen Knoten besteht. Die Knoten des Baumes sind die Domänen. Die gerichteten Kanten stellen die Eigenschaften dar, die vererbt werden. Der Graph enthält nur gerichtete Kanten, da Oberdomänen nur Eigenschaften an ihre Unterdomänen vererben können. Unterdomänen dürfen also keine Eigenschaften an ihre Oberdomänen vererben. Jede Domäne darf nur einmal im Domänenbaum vorkommen.

Beispiel für einen Domänenbaum



In diesem Beispiel sind *Student* und *Administrator* Unterdomänen der Domäne *Benutzer*. *Drucker* ist eine Unterdomäne der Domäne *Geräte*. *Benutzer* und *Geräte* sind Unterdomänen der Wurzeldomäne *Rechenzentrum*.

3.1.5.1 Wurzeldomäne

Jeder Domänenbaum muß aus mindestens einem Wurzelelement bestehen und darf nur ein Wurzelelement enthalten. Dies wird im folgenden als **Wurzeldomäne** bezeichnet. Eine Wurzeldomäne kann beliebig viele Unterdomänen besitzen.

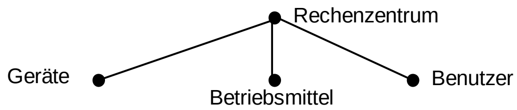
3.1.5.2 Oberdomänen

Eine Oberdomäne ist eine Domäne, die Unterdomänen besitzt.

3.1.5.3 Unterdomänen

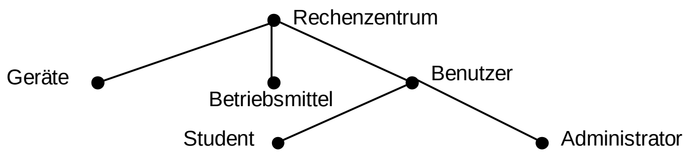
Eine Unterdomäne ist eine Domäne, welche genau eine Oberdomäne besitzt. Diese Unterdomänen können wiederum ihrerseits weitere Unterdomänen besitzen.

Beispiel für eine Wurzeldomäne mit drei Unterdomänen



Die Unterdomäne Benutzer in der nachfolgenden Graphik besitzt beispielsweise die Domänen *Student* und *Administrator* als Unterdomänen.

Beispiel für eine Unterdomäne, die selbst zwei Unterdomänen besitzt



Es gilt die Einschränkung, daß eine Domäne nicht zugleich die Oberdomäne einer anderen Domäne und deren Unterdomäne sein kann. Im Beispiel ist die Domäne *Student* eine Unterdomäne der Domäne *Benutzer*. Dann ist es nicht erlaubt, daß die Domäne *Student* zugleich die Oberdomäne für die Domäne *Benutzer* darstellt, da es sonst bei der Vererbung von Domäneneigenschaften zu Inkonsistenzen kommt (siehe Kapitel 3.1.5.9).

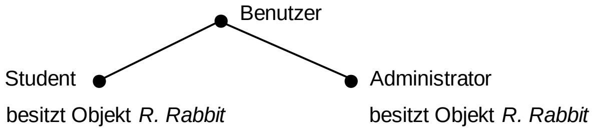
3.1.5.4 Objekte

Eine Domäne kann kein, ein oder mehrere Objekte besitzen. Die Stellung eines Objekts im Domänenbaum ist durch die Stellung der zum Objekt zugehörigen Domäne bestimmt. Da ein Objekt zu mehreren Domänen gehören kann, ist es möglich, daß ein Objekt mehrfach im Domänenbaum vorkommt.

3.1.5.5 Mehrfachzuordnung

Es ist möglich, ein bestimmtes Objekt mehreren Domänen zuzuordnen. Dabei dürfen die Domänen, zu denen das Objekte gehört, nicht Ober- bzw. Unterdomänen voneinander sein und auch keine gemeinsamen Ober- oder Unterdomänen besitzen. Dadurch werden Inkonsistenzen bei der Vererbung von Eigenschaften vermieden.

Beispiel für Mehrfachzuordnung



Das Objekt mit Namen *R. Rabbit* kommt im Beispiel zweifach vor. Es gehört der Domäne Student und der Domäne Administrator an.

3.1.5.6 Eigenschaften

Im Domänenbaum kommen Domänen und Objekte vor, die Domänen- bzw. Objekteigenschaften besitzen können.

3.1.5.7 Operationen auf Domänen

Im Domänenbaum können Benutzer je nach ihren Zugriffsrechten folgende Operationen auf Domänen ausgeführen:

Operation	Erläuterung zur Operation
new	neue Domäne definieren
ins	Domäne im Domänenbaum einfügen
mov	Domäne im Domänenbaum verschieben
get	Informationen zu einer Domäne anzeigen
set	Namen einer Domäne ändern
del	Domäne aus dem Domänenbaum löschen
new_attr	Domäneneigenschaften definieren
ins_attr	Domäneneigenschaften einfügen
get_attr	Domäneneigenschaften anzeigen
set_attr	Domäneneigenschaften ändern
del_attr	Domäneneigenschaften löschen
ins_attr_value	Wert einer Domäneneigenschaft angeben
get_attr_value	Wert einer Domäneneigenschaft anzeigen
set_attr_value	Wert einer Domäneneigenschaft ändern
ins_refer	Domäne oder Objekt zu einer Domäneneigenschaft bezüglich angeben
get_refer	Domäne oder Objekt der Domäneneigenschaft bezüglich anzeigen
set_refer	andere Domäne oder anderes Objekt zur Domäneneigenschaft bezüglich bestimmen
del_refer	Domäne oder Objekt der Domäneneigenschaft bezüglich löschen
get_inh_attr	geerbte Domäneneigenschaften anzeigen
get_at_aw	Vererbungstyp einer Domäne anzeigen
set_at_aw	Vererbungstyp einer Domäne ändern
get_ober	Oberdomäne einer Domäne anzeigen
get_sub	Unterdomänen einer Domäne anzeigen
get_obj	Objekte einer Domäne anzeigen

3.1.5.8 Operationen auf Objekten

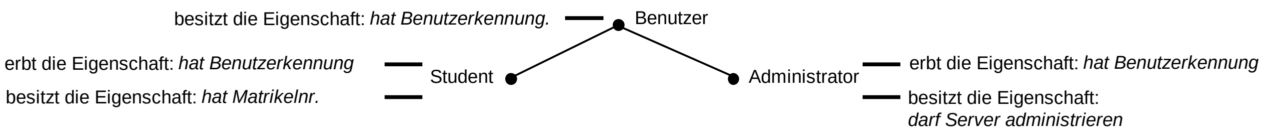
Im Domänenbaum gibt es folgende Operationen, die auf Objekten ausgeführt werden können:

Operation	Erläuterung zur Operation
new_obj	Objekt definieren
ins_obj	Objekt in eine Domäne einfügen
mov_obj	Objekt von einer Domäne zu einer anderen verschieben
get_obj	Informationen zu einem Objekt anzeigen
set_obj	Name eines Objekts ändern
del_obj	Objekt löschen
new_obj_attr	Objekteigenschaften definieren
ins_obj_attr	Objekteigenschaften einfügen
get_obj_attr	Objekteigenschaften anzeigen
set_obj_attr	Objekteigenschaften ändern
ins_obj_attr_value	Wert einer Objekteigenschaft angeben
get_obj_attr_value	Wert einer Objekteigenschaft anzeigen
set_obj_attr_value	Wert einer Objekteigenschaft ändern
del_obj_attr_value	Wert einer Objekteigenschaft löschen
ins_obj_refer	Domäne oder Objekt zu einer Objekteigenschaft bezüglich angeben
get_obj_refer	Domäne oder Objekt der Objekteigenschaft bezüglich anzeigen
set_obj_refer	andere Domäne oder anderes Objekt zur Objekteigenschaft bezüglich bestimmen
del_obj_refer	Domäne oder Objekt der Objekteigenschaft bezüglich löschen
ins_mult	Objekt zu einer weiteren Domäne hinzufügen
del_mult	Objekt, welches zu mehreren Domänen gehört, aus einer Domäne entfernen
get_dom	Domänen, zu denen ein Objekt gehört, anzeigen

3.1.5.9 Vererbung

Bei der Vererbung im Domänenbaum erbt eine Unterdomäne alle **Domäneneigenschaften** ihrer Oberdomäne. Sie kann zusätzlich weitere Eigenschaften besitzen.

Beispiel für die Vererbung von Eigenschaften



Der Querstrich in der Abbildung zeigt eine Eigenschaft einer Domäne an.

In diesem Beispiel wird die Domäneneigenschaft *hat Benutzerkennung* der Domäne *Benutzer* an die Unterdomänen *Student* und *Administrator* vererbt. Die Domäne *Student* besitzt die nicht geerbte Domäneneigenschaft *hat Matrikelnr.* und die Domäne *Administrator* besitzt die nicht geerbte Domäneneigenschaft *darf Server administrieren*.

Unterdomänen können nur von ihren Oberdomänen Eigenschaften erben. Sie dürfen nicht Eigenschaften von Geschwisterdomänen oder anderen Domänen, mit denen sie in keiner Verbindung stehen, erben. Somit ist die Voraussetzung für Domäne B, um Eigenschaften von einer Domäne A zu erben:

Domäne B muß eine Unterdomäne von Domäne A sein.

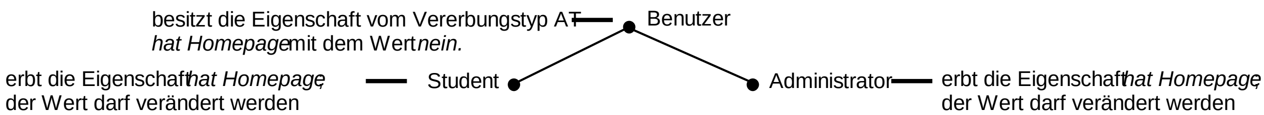
Dies ist identisch mit folgender Formulierung:

Domäne A ist die Oberdomäne von Domäne B.

3.1.5.9.1 Vererbungstyp AT

Die Domäne *Benutzer* vererbt im folgenden Beispiel die Eigenschaft *hat Homepage* an die Subdomänen *Student* und *Administrator*. Diese übernehmen erst einmal automatisch den Wert der Eigenschaft von ihrer Oberdomäne. Sie können den Wert jedoch jederzeit mit der Operation **set_at_aw** ändern.

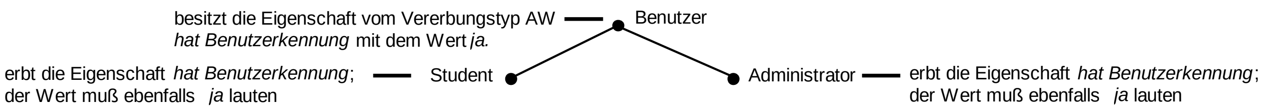
Beispiel zum Vererbungstyp AT



3.1.5.9.2 Vererbungstyp AW

Im nachfolgenden Beispiel zum Vererbungstyp AW wird die Eigenschaft *hat Benutzerkennung* von der Domäne *Benutzer* an ihre Subdomänen *Student* und *Administrator* vererbt. Da der Vererbungstyp AW lautet, ist es den Subdomänen nicht erlaubt, den Wert der Eigenschaft zu ändern. Er muß unverändert übernommen werden.

Beispiel zum Vererbungstyp AW



3.1.5.10 Auswirkung der Operationen auf den Domänenbaum

Auf die Vererbung wirken sich nur Operationen auf Domänen und nicht auf Objekte aus, da die Vererbung nicht für Objekteigenschaften definiert ist.

Es folgt eine Beschreibung der einzelnen Domänenoperationen im Bezug auf Auswirkungen auf den Domänenbaum:

Operation	Auswirkung der Operation auf die Vererbung im Domänenbaum
new	keine
ins	Unterdomänen erben alle Eigenschaften mit Wert und Vererbungstyp AW der Oberdomäne
mov	geerbte Eigenschaften der alten Oberdomäne werden bei der Domäne und all ihren Unterdomänen bis hinab zu den Blättern gelöscht; es werden alle Eigenschaften der neuen Oberdomäne mit Wert und Vererbungstyp AW geerbt; dies wirkt sich auf alle Unterdomänen hinab bis zu den Blättern aus
get	keine
set	keine
del	keine, da eine Domäne nur gelöscht werden darf, wenn sie weder Unterdomänen noch Objekte besitzt
new_attr	keine
ins_attr	Unterdomänen und deren Unterdomänen hinab bis zu den Blättern erben die Eigenschaft mit Wert und Vererbungstyp AW
get_attr	keine
set_attr	Name der Eigenschaft ändert sich; dies wirkt sich auf den gesamten Domänenbaum aus
del_attr	geerbte Eigenschaften dürfen nicht gelöscht werden; nicht geerbte Eigenschaften werden auch bei den Unterdomänen und deren Unterdomänen hinab bis zu den Blättern gelöscht
ins_attr_value	der Wert kann nur bei nicht geerbten Eigenschaften eingefügt werden; Unterdomänen und deren Unterdomänen hinab bis zu den Blättern übernehmen den Wert der Eigenschaft
get_attr_value	keine
set_attr_value	der Wert kann nur bei nicht geerbten Eigenschaften und Eigenschaften, deren Oberdomäne den Vererbungstyp AT besitzt, geändert werden; auch bei den Unterdomänen und deren Unterdomänen hinab bis zu den Blättern wird der neue Wert der Eigenschaft übernommen, falls sie den Vererbungstyp AW besitzen
ins_refer	Eigenschaft bezüglich kann nur bei nicht geerbten Eigenschaften eingefügt werden; auch bei den Unterdomänen und deren Unterdomänen hinab bis zu den Blättern wird die Eigenschaft bezüglich eingefügt
get_refer	keine
set_refer	der Wert kann nur bei nicht geerbten Eigenschaften und Eigenschaften, deren Oberdomäne den Vererbungstyp AT besitzt, geändert werden; auch bei den Unterdomänen und deren Unterdomänen hinab bis zu den Blättern wird die Eigenschaft bezüglich geändert
del_refer	der Wert kann nur bei nicht geerbten Eigenschaften und Eigenschaften, deren Oberdomäne den Vererbungstyp AT besitzt, gelöscht werden; auch bei den Unterdomänen und deren Unterdomänen hinab bis zu den Blättern wird die Eigenschaft bezüglich gelöscht
get_inh_attr	keine
get_at_aw	keine
set_at_aw	der Wert kann nur bei nicht geerbten Eigenschaften und Eigenschaften, deren Oberdomäne den Vererbungstyp AT besitzt, geändert werden; auch bei den Unterdomänen und deren Unterdomänen hinab bis zu den Blättern wird die Eigenschaft gelöscht
get_ober	keine
get_sub	keine
get_obj	keine

3.2 Datenbankkonzept

Im Datenbankkonzept wird das im letzten Unterkapitel beschriebene Domänenkonzept umgesetzt. Dieser Abschnitt beginnt mit einer Erstellung eines ER-Modells zum Domänenkonzept. Anhand des ER-Modells erfolgt dann die Umsetzung in ein Datenbankschema für ein relationales Datenmodell.

3.2.1 Umsetzung des Domänenkonzepts anhand des Entity-Relationship-Modells (ER-Modell)

Das Entity-Relationship-Modell (**ER-Modell**) legt Entitäten fest und beschreibt Beziehungen zwischen diesen Entitäten. Graphisch wird das ER-Modell im Entity-Relationship-Diagramm (**ER-Diagramm**) dargestellt. Die Entitäten werden als **Rechtecke**, die Beziehungstypen als **Rauten** zwischen den Entitäten dargestellt. An den Beziehungstypen befindet sich die Art der Beziehung, ob **1:1**, **1:n** oder **n:m**. Subentitätstypen sind dadurch gekennzeichnet, daß sie in der Raute keine Bezeichnung tragen und sich über der Raute ein **Querstrich** befindet (siehe auch Kapitel 2.2.6).

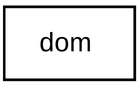
3.2.1.1 Entitäten

Aus dem Domänenkonzept werden die Entitäten **Domänenbaum**, **Domäne**, **Objekt** und **Eigenschaft** abgeleitet. Aus den im Kapitel 3.2.1.2 beschriebenen Beziehungen lassen sich zusätzlich die Entitäten **Domäneneigenschaft** und **Objekteigenschaft** bestimmen. Diese sind Subentitäten der Entität Eigenschaft. Um die Beziehung **bezüglich** (siehe Kapitel 3.2.1.2) darzustellen, wurde eine weitere Entität **Klasse** aufgenommen, die als Subentitäten die Entität Domäne und die Entität Objekt besitzt.

3.2.1.1.1 Domänenbaum

Die Entität **Domänenbaum** wird im ER-Diagramm mit *d tree* (domain tree) bezeichnet. Das für das Domänenkonzept erstellte ER-Diagramm wird nur mit einem Domänenbaum dargestellt. Das Konzept berücksichtigt nicht, daß im Domänenkonzept mehrere Domänenbäume vorkommen können (siehe Kapitel 5.3).

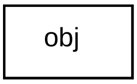
3.2.1.1.2 Domäne



Die Entität **Domäne** besitzt die Attribute *Domänenid*, im folgenden mit **did** abgekürzt, und *Domänenname*, im folgenden mit **dname** abgekürzt. Eine Domäne kann sowohl eine Oberdomäne von Subdomänen sein, wie selbst Subdomäne einer Oberdomäne. Insgesamt darf eine Domäne nur einmal im Domänenbaum vorkommen.

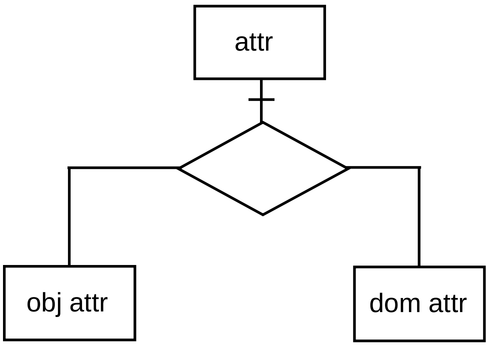
Der hier verwendete Begriff Attribut bezieht sich auf Merkmale einer Entität oder einer Beziehung eines ER-Modells (siehe Kapitel 2.2). Davon zu unterscheiden sind die Begriffe Eigenschaft, Domäneneigenschaft und Objekteigenschaft, die im Domänenkonzept (siehe Kapitel 3.1) definiert wurden. In diesem Fall sind did und dname Attribute der Entität Domäne im ER-Modell und Domäneneigenschaften jeder Domäne im Domänenbaum.

3.2.1.1.3 Objekt



Die Entität **Objekt** besitzt analog zur Entität Domäne und deren Attribute did und dname die Attribute *Objektid* (**oid**) und *Objektnamen* (**oname**). Im ER-Diagramm trägt sie die Bezeichnung *obj*.

3.2.1.1.4 Eigenschaft



Die Entität **Eigenschaft** wird im ER-Diagramm mit *attr* bezeichnet. Sie besitzt folgende Attribute: Id der Eigenschaft (**id**), Name der Eigenschaft (**attr**), Wert der Eigenschaft (**value**) und der Name der Domäne oder des Objekts bezüglich der die Eigenschaft gilt (**bzgl**). Sie hat zwei Subentitäten: **Domäneneigenschaft** und **Objekteigenschaft**.

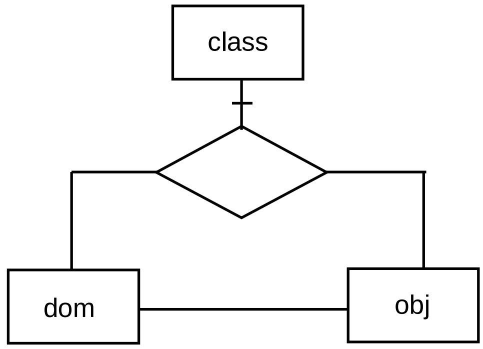
3.2.1.1.5 Domäneneigenschaft

Die Entität **Domäneneigenschaft** besitzt zusätzlich die Attribute Identifikator der Oberdomäne (**odid**) und Vererbungstyp (**at_aw**), der die Werte AT oder AW enthalten kann. Im ER-Diagramm trägt die Entität Domäneneigenschaft die Bezeichnung *dom attr*. Bei der Subentität Domäneneigenschaft werden im Attribut id, welches von der Entität Eigenschaft geerbt wird, nur Domänenids eingetragen.

3.2.1.1.6 Objekteigenschaft

Die Entität **Objekteigenschaft** wird mit *obj attr* bezeichnet. Sie besitzt zu den Attributen der Entität Eigenschaft das Attribut Identifikator der Domäne (**did**). Bei der Objekteigenschaft werden im Attribut id, welches von der Entität Eigenschaft geerbt wird, nur Objektids eingetragen.

3.2.1.1.7 Klasse

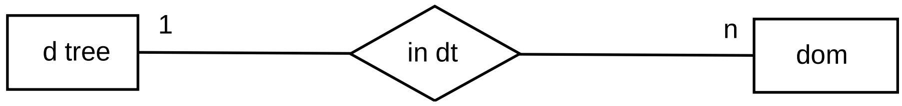


Die Entität **Klasse** wird im ER-Diagramm mit *class* bezeichnet. Sie wurde eingeführt, um die Eigenschaft **bezüglich** darzustellen. Die Entität Klasse besitzt die Subentitäten **Domäne** und **Objekt**.

3.2.1.2 Beziehungen

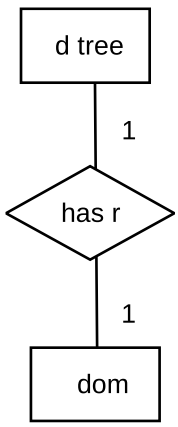
Im Domänenbaum gibt es folgende Beziehungen zwischen den verschiedenen Entitäten:

3.2.1.2.1 Beziehungen zwischen Domänenbaum und Domäne

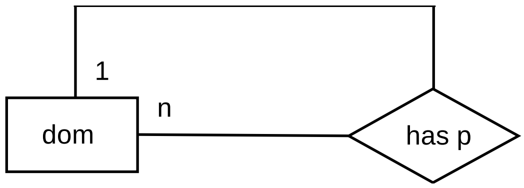


Es gibt zwischen der Entität **Domänenbaum** und der Entität **Domäne** die **1:n-Beziehung** *DomäneGehörtZuDomänenbaum*, die im ER-Diagramm mit **in dt** (in domain tree) bezeichnet. Im vorliegenden Konzept wird nur ein Domänenbaum behandelt (siehe Kapitel 5.3.1.4 Mehrere Domänenbäume). Dadurch ist eindeutig, zu welchem Domänenbaum eine Domäne gehört.

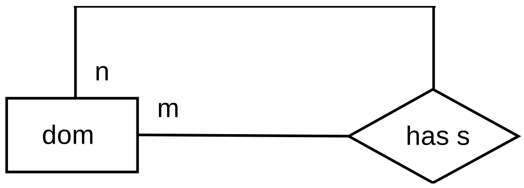
Jeder **Domänenbaum** besitzt genau eine **Wurzeldomäne**. Das wird im ER-Diagramm durch die **1:1-Beziehung** *DomänenbaumBesitztWurzeldomäne* (**has r** für has root) ausgedrückt.



3.2.1.2.2 Beziehungen zwischen Domänen

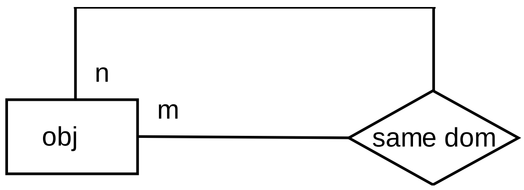


Die Entität **Domäne** besitzt die selbstbezügliche Beziehung *istOberdomäneVon* (**has p** für has parent) und *istUnterdomäneVon* (**has s** für has subdomain). has p ist eine **1:n-Beziehung**. Sie besagt, daß jede Subdomäne nur eine Oberdomäne besitzen darf.

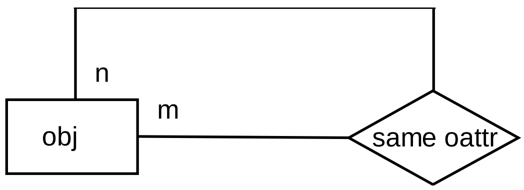


has s ist eine **n:m-Beziehung**. Jede Oberdomäne kann mehrere Subdomänen besitzen.

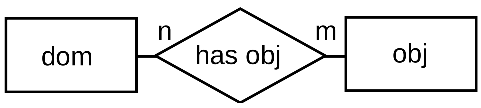
3.2.1.2.3 Beziehungen zwischen Objekten



Zwischen **Objekten** gibt es die **n:m-Beziehungen** *gehörtZurGleichenDomäne* (**same dom**) und *besitztDieGleichenEigenschaften* (**same oattr** für same object attribute).



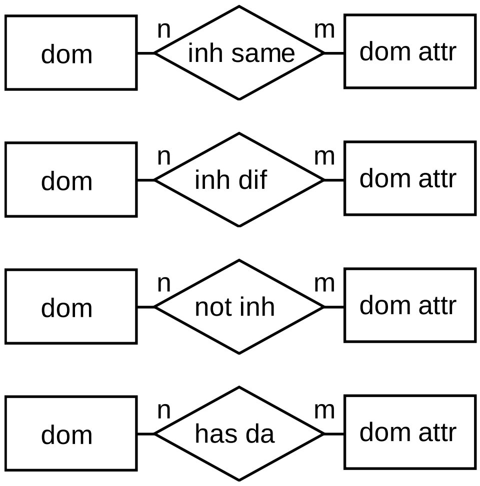
3.2.1.2.4 Beziehungen zwischen Domäne und Objekt



Zwischen **Domänen** und **Objekten** gibt es die **n:m-Beziehungen** *ObjektGehörtZuDomäne* (**has dom**) bzw. *DomäneBesitztObjekte* (**has obj**).

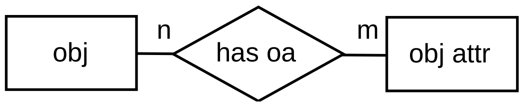
Eine Domäne kann mehrere Objekte besitzen und zugleich kann ein Objekt zu mehreren Domänen gehören.

3.2.1.2.5 Beziehungen zwischen Domäne und Domäneneigenschaft



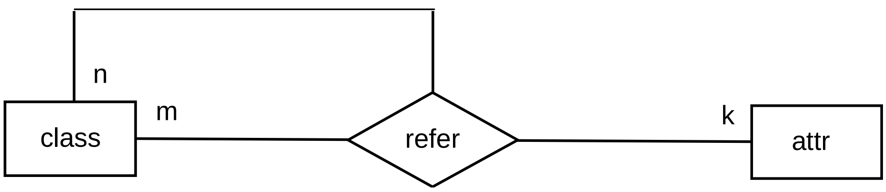
Zwischen **Domäneneigenschaften** und **Domänen** gibt es die **n:m-Beziehungen** *istGeerbtMitGleichemWert* (**inh same**), *istGeerbtMitAnderemWert* (**inh dif**) , *istNichtGeerbt* (**not inh**) und *besitztDomäneneigenschaft* (**has da**). Die Beziehung has da besagt, daß eine Domäne mehrere Domäneneigenschaften besitzen und zugleich eine Domäneneigenschaft zu mehreren Domänen gehören kann.

3.2.1.2.6 Beziehungen zwischen Objekt und Objekteigenschaft



Zwischen der Entität **Objekt** und der Entität **Objekteigenschaft** gibt es die Beziehung *besitztObjekteigenschaft* (**has oa**). Diese **n:m-Beziehung** bedeutet, daß ein Objekt beliebig viele Objekteigenschaften besitzen kann. Zugleich kann eine Objekteigenschaft zu beliebig vielen Objekten gehören.

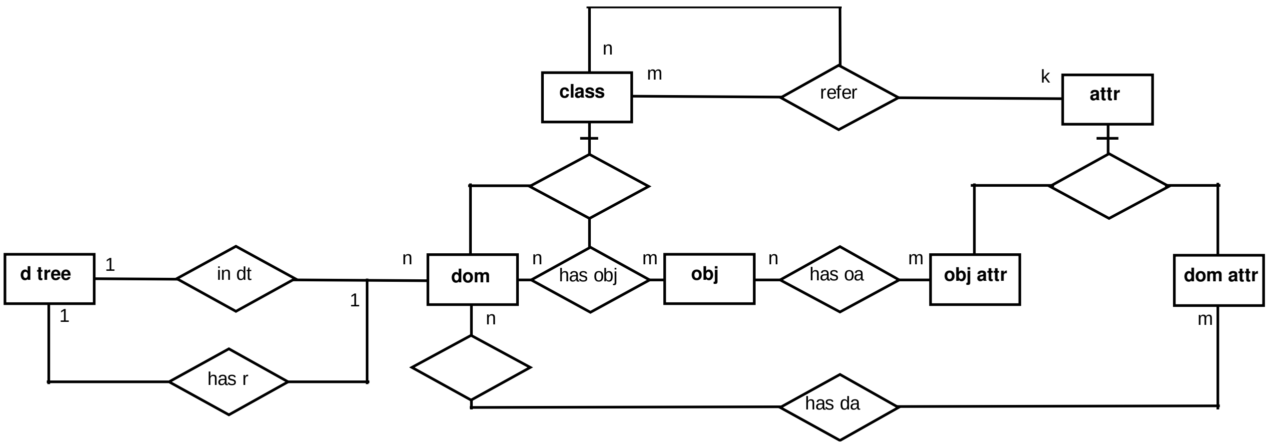
3.2.1.2.7 Beziehungen zwischen Klasse und Eigenschaft



Die Entität **Klasse** und die Entität **Eigenschaft** verfügen über die **k:n:m-Beziehung** *bezüglich* (**refer**). Ein Beispiel für die Beziehung ist: die Klasse *a* besitzt die Eigenschaft *E* bezüglich der Klasse *b*. *a* und *b* können entweder Domänen oder Objekte sein. Bei *k* kann es sich dann um eine Domäneneigenschaft oder Objekteigenschaft handeln (siehe Kapitel 3.1.3.4).

3.2.1.3 ER-Diagramm zum Domänenkonzept

Das ER-Diagramm zum Domänenkonzept für das Rechner-Administrierungssystem setzt die im Kapitel 3.1 beschriebenen Elemente um: Domänenbaum, Domänen, Objekte, Mehrfachzuordnung von Objekten, Eigenschaften, Domäneneigenschaften, Objekteigenschaften, Eigenschaft bezüglich und Vererbung von Domäneneigenschaften mit den Vererbungstypen AT und AW.



Aus Gründen der Übersichtlichkeit wurden folgende Beziehungen nicht in das ER-Diagramm eingezeichnet:

- istOberdomäneVon (siehe Abschnitt 3.2.1.2.2)
- istUnterdomäneVon (siehe Abschnitt 3.2.1.2.2)
- gehörtZurGleichenDomäne (siehe Abschnitt 3.2.1.2.3)
- besitztDieGleichenEigenschaften (siehe Abschnitt 3.2.1.2.3)
- ObjektGehörtZuDomäne (siehe Abschnitt 3.2.1.2.4)
- istGeerbtMitGleichemWert (siehe Abschnitt 3.2.1.2.5)
- istGeerbtMitAnderemWert (siehe Abschnitt 3.2.1.2.5)
- istNichtGeerbt (siehe Abschnitt 3.2.1.2.5)

3.2.2 Relationales Datenmodell

Der Abschnitt 3.1.5 über den Domänenbaum macht deutlich, daß Hierarchien eine wesentliche Rolle in Netzwerk-Informationsdiensten spielen. Hierarchien lassen sich durch hierarchische Datenbankmodelle darstellen. Jedoch sind hierarchische Datenbanken nach [Voss 94] fast ausgestorben. Heute gibt es vor allem Datenbanksysteme mit relationalen und objektorientierten Datenmodellen. Als Programmiersprache für den Prototyp des Rechner-Administrierungssystems wird die objektorientierte und plattformunabhängige Programmiersprache Java gewählt (siehe Kapitel 4.2.1). Dies legt auch ein objektorientiertes Datenmodell für die Datenbank nahe. Die Hierarchie im Domänenkonzept läßt sich gut mittels Vererbung ausdrücken. Eine Anforderung an die vorliegende Arbeit bestand jedoch darin, daß die zu verwendende Datenbank kostenlos erhältlich sein sollte. Die Recherche nach möglichen Datenbankmodellen mit JDBC-Schnittstelle ergab nur Datenbanksysteme mit relationalen Datenmodellen wie z.B. MySQL und msql. Deshalb folgt eine Umsetzung des ER-Modells in ein **relationales** statt eines objektorientierten Datenmodells.

Die Festlegung der strukturellen Elemente des relationalen Datenmodells beziehen sich auf die relationale Datenbanksprache Standard Query Language (**SQL**). Danach besteht eine relationale Datenbank aus Tabellen, die über Zeilen und Spalten verfügen. Das **Datenbankschema** definiert die Struktur der Datenbank. Die Tabellen, Zeilen, Spalten und Schlüssel werden im Datenbankschema definiert und stellen die **physikalische Struktur** dar.

3.2.3 Datenbankschema

Das Datenbankschema besteht aus Regeln, die definieren, wie der Domänenbaum aufgebaut ist. Das im vorherigen Abschnitt erstellte ER-Modell zum Domänenkonzept wird nun in das Datenbankschema für ein relationales Datenmodell umgesetzt: jedes Objekt, jede Domäne und jede Eigenschaft haben einen eindeutig identifizierbaren Namen und besitzen eine bestimmte Position im Domänenbaum.

3.2.3.1 Tabelle Domänen

Es wird eine Tabelle **Domänen** definiert, in der die Zeilen die einzelnen Domänen darstellen. In der Tabelle sind alle Domänen enthalten, die zum Rechner-Administrierungssystem gehören. Die Tabelle Domänen besitzt die Spalten **did** (*Domänenid*) und **dname** (*Domänenname*). In der Spalte did wird ein numerischer Wert als **Primärschlüssel**, der vom Datenbankverwaltungssystem (DBMS) vergeben wird, eingetragen. In der Spalte dname wird der Name der Domäne, der eindeutig sein muß, eingetragen. Die Einführung der Spalte did hat den Vorteil, daß der Name einer Domäne ohne Auswirkungen auf andere Tabellen geändert werden kann. Denn alle anderen Tabellen benutzen als Verweis auf eine Domäne nur die Spalte did. Die did einer Domäne dagegen bleibt, solange es die Domäne gibt, unverändert. Auf sie darf nur lesend zugegriffen werden.

Tabelle **Domänen**

did	**dname**
0	root
1	user
2	printer

Beispiel. Die Tabelle Domänen enthält drei Domänen mit den Domänennamen root, user und printer und den zugehörigen Domänenids 0, 1 und 2.

3.2.3.2 Tabelle Objekte

Es gibt eine Tabelle **Objekte**, in der die Zeilen die einzelnen Objekte darstellen. Die Tabelle Objekte besteht aus den Spalten **oid** (Objektid) und **oname** (Objektname). Die Ausführungen über die Spalten did und dname der Tabelle Domänen gelten analog für die Tabelle Objekte und deren Spalten oid und oname.

Tabelle **Objekte**

oid	**oname**
33	Ant
101	Bär

Beispiel. In der Tabelle Objekte wurden die zwei Objekte mit den Objektnamen Ant und Bär und den zugehörigen Objektsids 33 und 101 eingetragen.

3.2.3.3 Tabelle Domänenobjekte

Die Tabelle **Domänenobjekte** setzt die n:m-Beziehungen *DomäneBesitztObjekte* (**has obj**) und *ObjektGehörtZuDomäne* (**has dom**) zwischen Domänen und Objekten um. Sie besitzt die Spalten **did** und

oid. Dabei steht did für Domänenid und oid für Objektid. Das Feld **id** der ersten Spalte wird als **Primärschlüssel** verwendet. Es wird automatisch vom DBMS erzeugt und verwaltet.

Tabelle **Domänenobjekte**

id	did	oid
0	1	33
1	1	101

Beispiel. In der Domänentabelle wurden zwei Eintragungen vorgenommen, die sich auf die vorherigen Beispiele beziehen: Das Objekt mit der Objektid 33 gehört zur Domäne mit der Domänenid 1. Ebenso besitzt die Domäne mit der Domänenid 1 ein Objekt mit der Objektid 101.

Betrachtet man die vorherigen Tabellen Objekte und Domänen, so findet man dort die zu den numerischen Identifikatoren gehörigen Namen der Objekte und Domänen. Die Domänenid 1 entspricht der Domäne mit Namen user, die Objektid 33 dem Objekt mit Namen Ant und die Objektid 101 dem Objekt mit Namen Bär. D.h. die Objekte Ant und Bär gehören zur Domäne user und sind damit Benutzer im Rechner-Administrierungssystem.

3.2.3.4 Tabelle Vererbung

Die Position einer Domäne im Domänenbaum wird in der Tabelle **Vererbung** ausgedrückt. Dort werden die Beziehungen *istOberdomäneVon* (**has p**) und *istUnterdomäneVon* (**has s**) des ER-Modells umgesetzt. Die Tabelle Vererbung besteht aus den Spalten **did1**, **richtung** und **did2**. did1 und did2 enthalten die numerische Identifikatoren der Domänen, zwischen denen eine Vererbung besteht. Die Spalte richtung muß einen der Werte „<" oder „>" enthalten. „<" bedeutet, daß die Domäne did1 Unterdomäne der Domäne did2 ist. Im Fall „>" ist did1 Oberdomäne der Domäne did2. Das Feld **id** der ersten Spalte wird als **Primärschlüssel** verwendet. Es wird automatisch vom DBMS erzeugt und verwaltet.

Tabelle **Vererbung**

id	did1	richtung	did2
0	0	>	1

Anmerkung. Die Spalte **richtung** kann wegfallen, indem man folgendes festlegt: in der Spalte **did1** werden nur Oberdomänen und in der Spalte **did2** nur deren Unterdomänen eingetragen. Im Konzept wurde der erste Ansatz gewählt, da der Administrator des Netzwerk-Informationsdienstes dabei mehr Wahlmöglichkeiten hat (siehe Abschnitt 5.3.2).

Beispiel. In der Tabelle Vererbung ist die Domäne mit der Domänenid 0 Oberdomäne der Domäne mit der Domänenid1.

Aus der Tabelle Domänen kann man die zu den Domänenids gehörigen Namen der Domänen entnehmen. Dann erhält man folgende Beziehung zwischen den Domänen: root ist Oberdomäne von user.

3.2.3.5 Tabelle Domäneneigenschaften pro Domäne

Um festzustellen, welche Eigenschaften eine Domäne hat, wird <u>eine</u> Tabelle für <u>jede</u> Domäne erstellt, die den Namen der Domäne besitzt. In dieser Tabelle werden die **n:m-Beziehungen** zwischen Domänen und Domäneneigenschaften *istGeerbtMitGleichemWert* , *istGeerbtMitAnderemWert* , *istNichtGeerbt* und *besitztDomäneneigenschaft* (**has da**) und die **k:n:m-Beziehung** *bezüglich* (**refer**) des ER-Modells umgesetzt.

Jede Tabelle enthält in der Spalte **attr** eine Eigenschaft der Domäne für jede Zeile der Tabelle. Zusätzlich besitzt diese Tabelle Spalten, die anzeigen, ob diese Eigenschaft geerbt wurde oder nicht:

- **odid** ist eine Abkürzung für *Oberdomänenid*. Besitzt diese Spalte den Spaltenwert -1, bedeutet dies, daß die Eigenschaft nicht vererbt wurde. Im anderen Fall enthält sie die oid der Oberdomäne.
- Die Spalte **at_aw** zeigt an, ob der Wert einer geerbten Eigenschaft verändert werden darf. Dann enthält sie den Wert at (Attributstyp), sonst aw (Attributswert).
- Die Spalte **bzgl** bezieht sich auf die Eigenschaft bezüglich. Falls sie ein o (kleines „o“) enthält, steht in der Spalte **bzglid** die oid des Objekts bezüglich dessen die Eigenschaft gilt. Analog kann die Spalte bzgl ein d (kleines „d“) für Domäne und die Spalte bzglid die entsprechende Domänenid enthalten. Wenn in der Spalte bzgl ein Leerzeichen steht, muß in der Spalte bzglid eine -1 stehen. Dies bedeutet dann, daß die Eigenschaft nicht bezüglich eines anderen Objekts oder einer Domäne gilt.
- Das Feld **id** der ersten Spalte wird als **Primärschlüssel** für die Tabelle verwendet. Es wird automatisch vom DBMS erzeugt und verwaltet.

Anmerkung: Der Wert einer Domäneneigenschaft befindet sich in einer eigenen Tabelle, die im Abschnitt „Tabelle Eigenschaftswerte pro Eigenschaft“ beschrieben wird. Die Spalte odid wird, um eine Verbesserung der Ausführungszeit des Rechner-Administrierungssystems zu erreichen, in der Datenbank redundant gehalten (siehe Tabelle Domänen 3.2.3.1).

Tabelle pro Domäne, welche alle Domäneneigenschaften einer Domäne enthält:

id	attr	odid	at_aw	bzgl	bzglid

Beispiel. In der folgenden Tabelle befinden sich alle Eigenschaften der Domäne user. Daher trägt die Tabelle den Namen **user**. Die erste Spalte namens id dient als Primärschlüssel für die Tabelle. Anhand der zweiten Spalte attr lassen sich die Domäneneigenschaften der Domäne user feststellen. Sie besitzt in diesem Beispiel die drei Domäneneigenschaften hatFaxnr, Titel und Druckerquota. Anhand der Spalte odid läßt sich feststellen, welche der Eigenschaften geerbt wurden. Da nur die erste und die dritte Zeile eine Zahl ungleich -1 enthält, wurden nur die Eigenschaften hatFaxnr und Druckerquota geerbt. 0 steht hierbei für die Domänenid der Oberdomäne von der Domäne user. Die Spalte at_aw zeigt den Vererbungstyp der entsprechenden Eigenschaft an. Da der Vererbungstyp für die Eigenschaft hatFaxnr aw lautet, darf der Wert der Eigenschaft nicht geändert werden. In diesem Beispiel können nur die Werte der Eigenschaften Titel und Druckerquota verändert werden, da es sich bei Titel um eine nicht geerbte Eigenschaft und bei Druckerquota um eine geerbte Eigenschaft des Vererbungstyps at handelt. Die letzten zwei Spalten beziehen sich auf die Eigenschaft bezüglich. Im Beispiel gilt die Eigenschaft Druckerquota bezüglich der Domäne mit der Domänenid 2.

Tabelle **user**

id	attr	odid	at_aw	bzgl	bzglid
0	hatFaxnr	0	aw		-1
1	Titel	-1	aw		-1
2	Druckerquota	0	at	d	2

3.2.3.6 Tabelle Objekteigenschaften pro Objekt

Pro Objekt gibt es ebenfalls eine <u>eigene</u> Tabelle, die den Namen des Objekts trägt. Es gibt so viele Zeilen in der Tabelle, wie ein Objekt Objekteigenschaften besitzt. Die Spalte **did** enthält die Domänenid der Domäne, zu der das Objekt gehört. Wenn das Objekt mehrfach vorkommt, also zu mehreren Domänen gehört, wird die did gewählt, der die Eigenschaft zugeschrieben werden soll. Die Spalten **at_aw**, **bzgl** und **bzglid** entsprechen den Spalten der eben beschriebenen Domänentabelle. Das Feld **id** der ersten Spalte wird als **Primärschlüssel** verwendet. Es wird automatisch vom DBMS erzeugt und verwaltet.

Anmerkung: Der Wert der entsprechenden Objekteigenschaft befindet sich in einer eigenen Tabelle, die im folgenden Abschnitt „Tabelle Eigenschaftswerte pro Eigenschaft“ beschrieben wird. Die Spalte did wird, um eine Verbesserung der Ausführungszeit des Rechner-Administrierungssystems zu erreichen, in der Datenbank redundant gehalten (siehe Tabelle Objekte 3.2.3.2).

Tabelle pro Objekt, welche alle Objekteigenschaften eines Objekts enthält:

id	attr	did	bzgl	bzglid

Beispiel. In der folgenden Tabelle Bär befinden sich alle Objekteigenschaften des Objekts mit Namen Bär. Dabei handelt es sich um die Eigenschaften Vorname und Nachname. In der Spalte did befindet sich die Domänenid der Domäne, zu der die Eigenschaft gehört. Der Tabelle Domänenobjekte dieses Abschnitts kann man entnehmen, daß das Objekt Bär nur zur Domäne mit der Domänenid 1 gehört. Daher ist der einzige zulässige Wert für die Spalte did die Domänenid der Domäne user, nämlich 1. Im Beispiel handelt es sich bei den Eigenschaften Vorname und Nachname um Eigenschaften, die nicht bezüglich einer Domäne oder eines anderen Objekts definiert wurden, denn in der Spalte bzgl sind Leerzeichen und in der Spalte bzglid der Spaltenwert -1 eingetragen.

Tabelle **Bär**

id	attr	did	bzgl	bzglid
0	Vorname	1		-1
1	Nachname	1		-1

3.2.3.7 Tabelle Eigenschaftswerte pro Eigenschaft

Für <u>jede</u> Eigenschaft wird eine <u>eigene</u> Tabelle definiert, die als Tabellennamen den Namen der Eigenschaft hat. Sie enthält dann zwei Spalten, eine mit der **id** und die andere mit dem Wert für das entsprechende Objekt bzw. für die Domäne, welche den Namen **value** trägt. Die Spalte id stellt den Primärschlüssel für die Tabelle dar.

Ob es sich bei der entsprechenden Eigenschaftstabelle um eine Domänen- oder Objekteigenschaft handelt, ist nicht direkt ersichtlich.

Tabelle pro Eigenschaft, welche die Werte dieser Eigenschaft für jede Domäne bzw. für jedes Objekt enthält:

id	value

Beispiel. Die folgende Tabelle Vorname enthält zu der Eigenschaft mit Namen Vorname, die Identifikatoren und die zugehörigen Werte der Eigenschaft. Der Tabelle Bär des letzten Abschnitts kann man entnehmen, daß es sich bei der Tabelle um Objekteigenschaften und somit bei den Identifikatoren um Objektidentifikatoren handelt. Das Objekt mit der Objektid 33 besitzt den Vornamen Annika und das Objekt mit der Objektid 101 Berthold.

Tabelle **Vorname**

id	value
33	Annika
101	Berthold

Anmerkung. In Rechenzentren tritt häufig folgende Situation auf: Alle Studenten besitzen ein Druckerquotum von 500 Seiten, doch Student Maier benötigt für ein spezielles Studienprojekt ein Druckerquotum von 2000 Seiten. Die Situation läßt sich allgemein so formulieren: Alle Objekte einer Domäne besitzen für eine bestimmte Eigenschaft den gleichen Wert mit der Ausnahme eines Objekts, das einen anderen Wert für diese Eigenschaft hat. Normalerweise werden Eigenschaften und ihre Werte in einer Tabelle gehalten. Dabei entsteht für den Administrator das Problem, daß er dann eine Domäneneigenschaft in eine Objekteigenschaft ändern und für alle Objekte der Domänen deren Eigenschaftswerte eingeben muß, obwohl nur ein Objekt einen anderen Wert für diese Eigenschaft besitzt.

Indem die Werte der Eigenschaften getrennt von den Eigenschaften gehalten werden, vermindert man den Zeitaufwand. Mit der Voraussetzung, daß Domänenidentifikatoren und Objektidentifikatoren voneinander zu unterscheiden sind, kann man nun das Problem folgendermaßen lösen:

Tabelle **Druckerquotum**

id	value
d3	500
o11	2000

Alle Objekte der Domäne mit der Domänenid 3 besitzen zur Eigenschaft Druckerquotum den Wert 500. Eine Ausnahme bildet das Objekt mit der Objektid 11, welches den Wert 2000 besitzt.

Es war zwischen der Häufigkeit, mit der das Problem „alle bis auf wenige“ auftritt, und dem nun auf jeweils zwei Tabellen benötigten Zugriff abzuwägen, um den Wert einer Eigenschaft eines Objekts oder einer Domäne zu erhalten. Mit der Begründung, den Aufwand für Administratoren so gering wie möglich zu halten, selbst auf Kosten der Laufzeit, wurde in der vorliegenden Arbeit die eben vorgestellte Lösung gewählt.

In dem Beispiel zur Eigenschaft Druckerquotum wird davon ausgegangen, daß sich in der Tabelle Domänen die Domäne Student mit der Domänenid 3, in der Tabelle Objekte das Objekt Maier mit der Objektid 11 und in der Tabelle Domänenobjekte eine Zeile, welche die Beziehung „Objekt Maier gehört zur Domäne Student“ ausdrückt, befindet.

3.2.3.8 Operationen auf Domänen und Objekten

Die folgenden Tabellen beschreiben die Auswirkung der im Abschnitt Domänenbaum definierten Operationen auf Domänen und Objekte auf die Tabellen der Datenbank. Dabei wird zwischen lesendem und schreibendem Zugriff auf die Tabellen differenziert:

Operationen auf Domänen:

Operation	lesender Zugriff auf Tabellen	schreibender Zugriff auf Tabellen
new	Domänen	Domänen Domäneneigenschaftstabelle
ins	Domänen Domäneneigenschaftstabelle Eigenschaftstabellen Vererbung	Domäneneigenschaftstabelle Eigenschaftstabellen Vererbung
mov	Domänen	Domäneneigenschaftstabelle

	Domäneneigenschaftstabelle Eigenschaftstabellen Vererbung	Eigenschaftstabellen Vererbung
get	Domänen Domäneneigenschaftstabelle Domänenobjekte Eigenschaftstabellen Objekte Vererbung	
set	Domänen	Domänen Domäneneigenschaftstabelle (Name)
del	Domänen Domänenobjekte Vererbung	Domänen Domäneneigenschaftstabelle Eigenschaftstabellen Vererbung
new_attr		Eigenschaftstabelle
ins_attr	Domänen Domäneneigenschaftstabelle Eigenschaftstabelle	Domäneneigenschaftstabelle
get_attr	Domänen Domäneneigenschaftstabelle	
set_attr	Domänen Domäneneigenschaftstabelle Eigenschaftstabelle	Domäneneigenschaftstabelle Eigenschaftstabelle (Name)
del_attr	Domänen Domäneneigenschaftstabelle	Domäneneigenschaftstabelle Eigenschaftstabelle
ins_attr_value	Domänen Domäneneigenschaftstabelle	Eigenschaftstabelle
get_attr_value	Domänen Domäneneigenschaftstabelle Eigenschaftstabelle	
set_attr_value	Domänen Domäneneigenschaftstabelle	Eigenschaftstabelle
ins_refer	Domänen Domänenobjekte Objekte	Domäneneigenschaftstabelle
get_refer	Domänen Domäneneigenschaftstabelle Domänenobjekte Objekte	
set_refer	Domänen Domäneneigenschaftstabelle Domänenobjekte Objekte	Domäneneigenschaftstabelle
del_refer	Domänen Domäneneigenschaftstabelle	Domäneneigenschaftstabelle
get_inh_attr	Domänen Domäneneigenschaftstabelle	
get_at_aw	Domänen Domäneneigenschaftstabelle	
set_at_aw	Domänen Domäneneigenschaftstabelle	Domäneneigenschaftstabelle
get_ober	Domänen Vererbung	
get_sub	Domänen Vererbung	
get_obj	Domänen Domänenobjekte Objekte	

Operationen auf Objekten:

Operation	lesender Zugriff auf Tabellen	schreibender Zugriff auf Tabellen
new_obj	Objekte	Objekte Objekteigenschaftstabelle
ins_obj	Domänenobjekte Objekte	Domänenobjekte
mov_obj	Domänenobjekte Objekte	Domänenobjekte
get_obj	Domänen Domänenobjekte Eigenschaftstabellen Objekte Objekteigenschaftstabelle	
set_obj	Objekte	Objekte Objekteigenschaftstabelle (Name)
del_obj	Objekte Objekteigenschaftstabelle	Domänenobjekte Eigenschaftstabellen Objekte Objekteigenschaftstabelle
new_obj_attr		Eigenschaftstabelle
ins_obj_attr	Objekte Objekteigenschaftstabelle	Objekteigenschaftstabelle
get_obj_attr	Objekte Objekteigenschaftstabelle	
set_obj_attr	Objekte Objekteigenschaftstabelle	Eigenschaftstabelle (Name) Objekteigenschaftstabelle
del_obj_attr	Objekte Objekteigenschaftstabelle	Eigenschaftstabelle Objekteigenschaftstabelle
ins_obj_attr_value	Objekte Objekteigenschaftstabelle	Eigenschaftstabelle
get_obj_attr_value	Eigenschaftstabelle Objekte Objekteigenschaftstabelle	
set_obj_attr_value	Objekte Objekteigenschaftstabelle	Eigenschaftstabelle
ins_obj_refer	Domänen Domänenobjekte Objekte Objekteigenschaftstabelle	Objekteigenschaftstabelle
get_obj_refer	Domänen Domänenobjekte Objekte Objekteigenschaftstabelle	
set_obj_refer	Domänen Domänenobjekte Objekte Objekteigenschaftstabelle	Objekteigenschaftstabelle
del_obj_refer	Objekte Objekteigenschaftstabelle	Objekteigenschaftstabelle
ins_mult	Domänen Domänenobjekte Objekte	Domänenobjekte
del_mult	Domänenobjekte Objekte	Domänenobjekte
get_dom	Domänen Domänenobjekte Objekte	

3.2.4 Zugriffsrechte

Die Zugriffsrechte bestimmen, wer oder was auf welche Tabelle und innerhalb dieser Tabelle auf welche Zeilen in welcher Weise zugreifen darf. Dabei gibt es die folgenden Zugriffsarten:

- löschend (**del**)
- lesend (**get**)
- erzeugend (**new**)
- ändernd (**set**)

Der Zugriff kann in Anlehnung an das Betriebssystem UNIX auf zwei Ebenen erfolgen: auf Benutzer- und auf Gruppenebene.

3.2.4.1 Benutzerebene

Im Domänenbaum gibt es die Domäne *Benutzer*, die als Objekte alle Benutzer des Netzwerk-Informationsdienst enthält. Jeder Benutzer besitzt spezifische Zugriffsrechte. Der Begriff Benutzer umfaßt **Personen**, **Workstations** und **Anwendungen**. Personen müssen eine Benutzerkennung besitzen, die ihnen Zugriff auf das Netzwerk erlaubt. Workstations und Anwendungen müssen ebenfalls im Netzwerk zugelassen sein.

Die Zugriffsrechte, die den Benutzern direkt zugeordnet werden sollen, sind dann spezielle Eigenschaften.

3.2.4.2 Gruppenebene

Die Benutzer können zu Gruppen zusammengefaßt werden, die bestimmte Zugriffsrechte besitzen. Dabei kann ein Benutzer zu keiner, einer oder mehreren Gruppen gehören. Im Domänenbaum kann man die verschiedenen Gruppen ablesen. Diese Gruppen stellen ebenso wie die Benutzer eine Domäne dar. Die Arten von Zugriffsrechten von Gruppen entsprechen denen der Benutzer: es gibt lesende, ändernde, erzeugende und löschende Zugriffe.

3.2.4.3 Abstrahierung von Benutzer- und Gruppenebenen

Statt nur für Benutzer und Gruppen können Zugriffsrechte für alle Objekte und Domänen des Rechner-Administrierungssystems vergeben werden. Der Administrator vergibt Zugriffsrechte für Objekte und Domänen.

3.2.4.4 Realisierung

Es folgen zwei Entwurfsmöglichkeiten, wie Zugriffsrechte in das Datenbankmodell umgesetzt werden können:

3.2.4.4.1 als Eigenschaften

Im Domänenkonzept werden Zugriffsrechte den Eigenschaften zugeordnet. Eine direkte Umsetzung des Domänenkonzepts führt zu folgender Realisierungsmöglichkeit im Datenbankkonzept:

Der Administrator fügt eine Eigenschaft zu jeder Domäne bzw. Objekt ein, für die ein Zugriffsrecht definiert werden soll. Als Eigenschaftswert wird die Domäne oder das Objekt, auf die bzw. das der Zugriff definiert werden soll, in die entsprechende Eigenschaftstabelle eingegeben. Darf eine Domäne oder ein Objekt auf alle Domänen zugreifen, wird der Wert „**all**“ eingetragen.

Beispiel. Benutzer Z. Kohl darf auf seine Daten lesend zugreifen, d.h. er besitzt das Zugriffsrecht *get*. Er befindet sich als Objekt mit der Objektid 3 in der Domäne Benutzer. Dann wird folgende Zeile in seine Objekteigenschaftstabelle und in die Eigenschaftstabelle *get* eingefügt:

Tabelle **Kohl**

id	attr	did	bzgl	bzglid
0	get	1		-1

Tabelle **get**

id	value
3	o3

o3 steht für Objekt mit der id 3.

Beispiel. Benutzer B. Klein darf auf die Eigenschaft *Name* der Domäne Benutzer lesend zugreifen, d.h. er besitzt auf die Eigenschaft *Name* eingeschränkt das Zugriffsrecht *get*. Dann wird folgende Zeile in seine Objekteigenschaftstabelle und in die Eigenschaftstabelle *getName* eingefügt:

Tabelle **Kohl**

id	attr	did	bzgl	bzglid
0	getName	1		-1

Tabelle **getName**

id	value
3	d1

d1 steht für Domäne mit der id 1.

Bewertung. Diese Entwurfsmöglichkeit erlaubt es, Zugriffsrechte, wie Eigenschaften zu behandeln. Mit denen für Eigenschaften definierten Operationen kann der Administrator der Rechner-Administrierungssystem Zugriffsrechte für Domänen und Objekte definieren, einfügen, ändern, löschen und sich anzeigen lassen.

3.2.4.4.2 Tabelle Zugriffsrechte

Die in diesem Abschnitt beschriebene zweite Entwurfsmöglichkeit, definiert eine eigene Tabelle für Zugriffsrechte: DieTabelle **Zugriffsrechte** enthält in der ersten Spalte den Primärschlüssel *id*. Bei der *id* handelt es sich um den Identifikator eines Objekts oder einer Domäne. Die Objekte können Benutzer, welche auch Workstations und Anwendungen umfassen, Gruppen oder andere Objekte oder Domänen sein. Die Spalten *del*, *get*, *new* und *set* beziehen sich auf die Zugriffsarten. Wenn ein Zugriffsrecht für alle Domänen und Objekte des Rechner-Administrierungssystems gelten soll, enthält es den Wert „**all**“. Für den Fall, daß das Zugriffsrecht nur für ein bestimmtes Objekt oder eine bestimmte Domäne gelten soll, wird die Objektid bzw. die Domänenid eingetragen. Objektids können durch Voranstellung des Buchstaben **o** von Domänenids, die durch den Buchstaben **d** gekennzeichnet sind, unterschieden werden. Besitzt ein Objekt oder eine Domäne ein Zugriffsrecht, welches für eine Domäne gilt, kann es dieses Zugriffsrecht auf alle

Objekte der bestimmten Domäne ausüben. Durch die Spalte *attr* wird eine Einschränkung auf bestimmte Eigenschaften erreicht. Befindet sich kein Eintrag in der Spalte attr, gilt das Zugriffsrecht für alle Eigenschaften des Objekts bzw. der Domäne. Soll ein Zugriffsrecht nur für eine Eigenschaft eines Objekts bzw. einer Domäne gelten, wird in der Spalte attr der Name dieser Eigenschaft eingetragen.

Tabelle **Zugriffsrechte**

id	**del**	**get**	**new**	**set**	**attr**

Beispiel. Benutzer A. Hagen darf auf seine Daten lesend zugreifen, d.h. er besitzt das Zugriffsrecht *get*. Er befindet sich als Objekt mit der Objektid 3 in der Domäne Benutzer. Dann wird folgende Zeile in die Tabelle Zugriffsrechte eingefügt:

id	**del**	**get**	**new**	**set**	**attr**
o3		o3			

o3 steht für Objekt mit der id 3.

Beispiel. Das Objekt mit der Objektid 2001 besitzt die Zugriffsrechte get und set bezüglich der Domänen, welche die id 4 und 95 tragen. Jedoch gelten die Zugriffsrechte für die Domäne mit der Domänenid 95 nur für die Eigenschaften Vorname und Nachname. Dann werden folgende Zeilen in die Tabelle Zugriffsrechte hinzugefügt:

id	**del**	**get**	**new**	**set**	**attr**
o2001		d4		d4	
o2001		d95		d95	Vorname
o2001		d95		d95	Nachname

d4 und d95 beziehen sich auf die Domänen mit der id 4 und 95. o2001 bezieht sich auf das Objekt mit der Objektid 20001.

Beispiel. Das Objekt mit der Objektid 0 darf auf alle Objekte und Domänen des Rechner-Administrierungssystems lesend zugreifen. Dann existiert folgender Eintrag in der Tabelle Zugriffsrechte:

id	**del**	**get**	**new**	**set**	**attr**
o0		all			

Datenbankverwalter

Es gibt einen (oder mehrere) Datenbankverwalter die sämtliche Zugriffsrechte für alle Spalten, Zeilen und Tabellen erhalten, einen sogenannter Superuser. In der Tabelle Zugriffsrechte erkennt man ihn daran, daß er bei allen Zugriffsarten den Wert "**all**" besitzt. D.h. er darf auf alle Einträge aller Objekte und Domänen in allen möglichen Zugriffsarten zugreifen.

In der folgenden Abbildung besitzt der Superuser die Objektid 0:

id	**del**	**get**	**new**	**set**	**attr**
o0	all	all	all	all	all

Bewertung. Diese in diesem Abschnitt beschriebene Entwurfsmöglichkeit zu Zugriffsrechten benötigt eigene Operationen zur Verarbeitung von Zugriffsrechten. Deshalb wurde für das Rechner-Administrationssystem die im letzten Abschnitt beschriebene Realisierung gewählt.

3.2.4.5 Operationen auf Domänen und Objekte

Die Benutzer des Rechner-Administrierungssystems können die im Domänenbaum definierten Operationen auf Domänen und Objekte nur ausüben, wenn sie über die entsprechenden Zugriffsrechte verfügen. Die folgenden Tabellen zeigen, für welche Operation welches Zugriffsrecht benötigt wird.

Operationen auf Domänen:

Operation	Zugriffsrechte del, get, new und set
new	new
ins	new
mov	new und del
get	get
set	set
del	del
new_attr	new
ins_attr	new
get_attr	get
set_attr	set
del_attr	del
ins_attr_value	new
get_attr_value	get
set_attr_value	set
ins_refer	new
get_refer	get
set_refer	set
del_refer	del
get_inh_attr	get
get_at_aw	get
set_at_aw	set
get_ober	get
get_sub	get
get_obj	get

Operationen auf Objekte:

Operation	Zugriffsrechte del, get, new und set
new_obj	new
ins_obj	new
mov_obj	new und del
get_obj	get
set_obj	set
del_obj	del
new_obj_attr	new
ins_obj_attr	new
get_obj_attr	get
set_obj_attr	set

del_obj_attr	del
ins_obj_attr_value	new
get_obj_attr_value	get
set_obj_attr_value	set
ins_obj_refer	new
get_obj_refer	get
set_obj_refer	set
del_obj_refer	del
ins_mult	new
del_mult	del
get_dom	get

3.2.4.6 Rechteverwaltung und -überprüfung

Um festzustellen, ob Benutzer B auf die Zeile Z der Tabelle T z.B. lesend zugreifen darf, werden seine Zugriffsrechte und die Zugriffsrechte der Gruppen, zu denen er gehört, überprüft. Die Rechteverwaltung und -überprüfung kann in einer künftigen Version des Rechner-Administrierungssystems vom Datenbank Management System (DBMS) übernommen werden (Stichwort „stored procedures"). Eine der Anforderungen an die Arbeit besteht darin, daß die zu verwendeten Datenbanksysteme kostenlos erhältlich sein sollen. Diese Anforderung erfüllen z.B. MySQL und msql. Beide verfügen nicht über eine ausreichende Rechteverwaltung und -überprüfung. Deshalb wird die Prüfung und Verwaltung der Zugriffsrechte außerhalb der Datenbank in einem Modul programmiert, welches Applikationsserver genannt wird.

3.2.4.7 Applikationsserver

Auf Datenbankebene gibt es eine Tabelle für die Zugriffsrechte. Im Applikationsserver erfolgt die Überprüfung der Zugriffsrechte, indem auf diese Tabelle zugegriffen wird. Ein Benutzer darf nur dann auf ein bestimmtes Objekt oder eine Domäne in einer bestimmten Art zugreifen, wenn er über das zugehörige Recht verfügt. Dieses Recht muß sich in der Zugriffstabelle befinden. Im Applikationsserver werden der Reihe nach alle Rechte des Benutzers und der Gruppen, zu denen er gehört, kontrolliert.

Beispiel. Benutzer H. Schmidt, der die Objektid 22 besitzt, möchte sich alle Eigenschaften des Objekts mit der id 43 ansehen. Daraufhin wird im Applikationsserver geprüft, ob sich folgende Zeile in der Zugriffsrechtetabelle befindet:

Tabelle **Zugriffsrechte**

id	**del**	**get**	**new**	**set**	**attr**
o22		o43			

3.2.5 Datenbankzugriff über Benutzerkennung und Paßwort

Zugriff auf die Datenbank erhält ein Benutzer über seine Benutzerkennung und sein Paßwort. Die Benutzerkennung wird als Objektname (oname) für den Benutzer in der Tabelle Objekte verwendet (siehe Kapitel 3.2.3.2 „Tabelle Objekte"). Das Paßwort wird als Objekteigenschaft zum entsprechenden Benutzer abgespeichert (siehe Kapitel 3.2.3.6 „Tabelle Objekteigenschaften pro Objekt" und Kapitel 3.2.3.7 „Tabelle Eigenschaftswerte pro Eigenschaft").

Im Applikationsserver findet auch die Überprüfung statt, ob die beim Aufrufen des Prototyps eingegebene Benutzerkennung und das Paßwort mit den in der Datenbank gespeicherten Werten übereinstimmen. Nur wenn die Daten übereinstimmen, ist der Benutzer **berechtigt** (**autorisiert**), mit der Verarbeitung des Rechner-Administrierungssystems fortzufahren. Der Benutzer erhält dann die Möglichkeit, seine ihn betreffenden Daten zu lesen und deren Eigenschaften zu ändern. Von seinen Zugriffsrechten und

Gruppenzugehörigkeiten hängt es ab, ob er auch weitere Operationen ausführen kann bzw. ob er auf weitere Zeilen und Tabellen der Datenbank zugreifen darf.

3.3 Client-Server-Konzept

3.3.1 Netzwerke

Netzwerke verbinden einzelne Rechner oder ganze Netzwerke (LANs) untereinander. Im Rechenzentrum des Instituts für Informatik der Ludwig-Maxmilian-Universität in München sind die Rechner über ein **TCP/IP-Netzwerk** miteinander verbunden.

3.3.2 Client-Server-Architektur

Die Rechner, die im Netzwerk miteinander verbunden sind, sind Clients, Server oder beides zugleich.

Clients stellen Anfragen an Server. Server beantworten diese Anfragen. Die Clients sind meist Workstations, auf denen sich die Benutzer anmelden.

Client-Server-Architekturen besitzen meist zwei oder mehr Prozesse auf zwei oder mehreren Rechnern.

3.3.2.1 2-Ebenen-Architektur

Die einfachste Form einer Client-Server-Architektur wird mit 2-Ebenen-Architektur (two tier) bezeichnet. Sie besteht aus mehreren Clients und einem Server.

3.3.2.2 3-Ebenen-Architektur

Die 3-Ebenen-Architektur (three tier) besteht aus den Clients und zwei Servern, z.B. einem Datenbankserver und einem Applikationsserver.

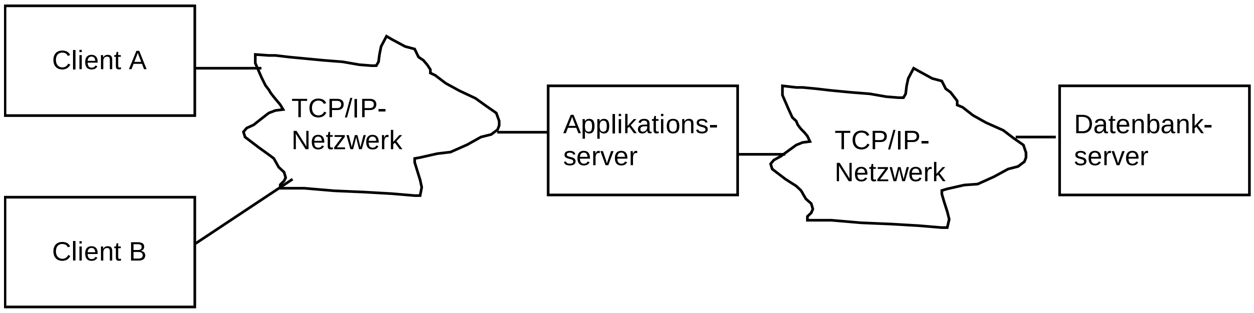
3.3.3 Datenbankserver

Im vorliegenden Rechner-Administrierungssystem gibt es nur eine Datenbank und somit auch nur einen Datenbankserver.

3.3.4 Applikationsserver

Das Rechner-Administrierungssystem besteht aus einer 3-Ebenen-Architektur mit einem Datenbankserver und einem Applikationsserver. Im Applikationsserver befinden sich die Prüfungen, welche die Konsistenz des Domänenkonzepts und der Datenbank gewährleisten.

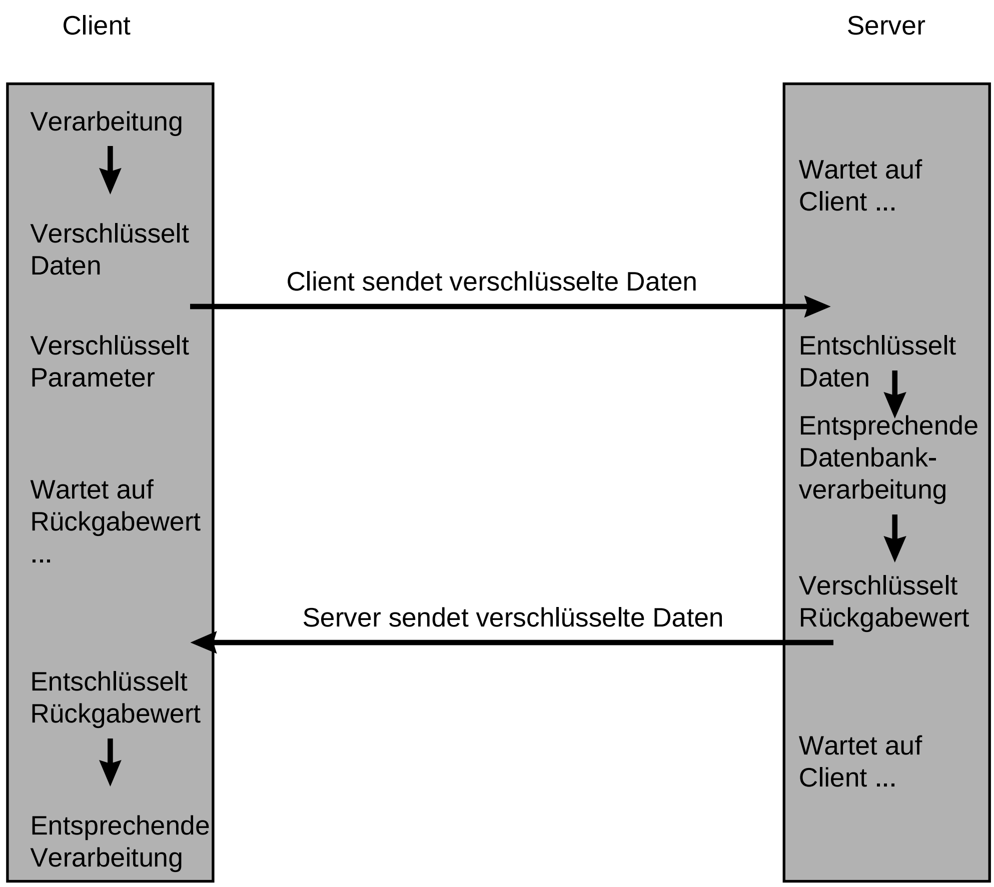
Die folgende Abbildung zeigt eine 3-Ebenen-Architektur mit einem Applikationsserver und einem Datenbankserver:



3.3.5 Kommunikationsprotokoll

Das Kommunikationsprotokoll legt die Kommunikation zwischen Clients und Server fest. Im Kapitel 2.3 erfolgte ein Überblick über die verschiedenen Kommunikationsprotokolle, die sich für Netzwerk-Informationsdienste anbieten: DAP, LDAP und SNMP. Die Anbindung dieser sehr umfangreichen und dadurch aufwendigen Protokolle würde den zeitlichen Rahmen dieser Arbeit sprengen. Daher wird für den Prototyp des Rechner-Administrierungssystems ein **eigenes Kommunikationsprotokoll** für den Austausch der Daten zwischen Clients und Server verwendet. Es besteht aus den sendenden und empfangenden Daten zwischen Clients und Server. Das Kommunikationsprotokoll ist auch für die Ver- und Entschlüsselung der gesendeten und empfangenen Daten zuständig (**siehe Kapitel 3.4.3.2 Verschlüsselung**).

Die folgende Abbildung beschreibt die Kommunikation zwischen Clients und Server des Rechner-Administrierungssystems:



Kommunikationsprotokoll des Clients

Nachdem der Benutzer ein Kommando aus der Kommandozeilen-Schnittstelle oder dem Menü des Rechner-Administrierungssystems ausgewählt hat, wird er aufgefordert, die für diese Operation benötigten Parameter über eine Datei oder die Tastatur einzugeben. Das Kommunikationsprotokoll des Clients hat die Aufgabe, jeden Parameter zu verschlüsseln und an den Server zu senden. Danach wartet es auf den Rückgabewert, empfängt diesen, entschlüsselt ihn und gibt ihn an den Client weiter.

Das Kommunikationsprotokoll des Clients wird vom Client mit den zu sendenden Daten als Parameter aufgerufen. Der Client übergibt dem Server als ersten Parameter den Namen der Operation, die der Server ausführen soll. Danach folgen die der Operation entsprechenden Parameter.

Der Ablauf des Kommunikationsprotokolls des Client ist folgendermaßen:

1. alle Parameter nacheinander verschlüsseln
2. alle verschlüsselten Parameter nacheinander an Applikationsserver senden
3. auf Rückgabewert warten
4. verschlüsselten Rückgabewert empfangen
5. Rückgabewert entschlüsseln
6. Rückgabewert an Client übergeben

Kommunikationsprotokoll des Servers

Der Server empfängt als ersten Parameter den Namen der Operation, die er ausführen soll. Danach folgen die je nach Kommando bzw. Operation benötigten Parameter, wie z.B. die Daten aus den Konfigurationsdateien, in verschlüsselter Form. Entsprechend dem empfangenen Kommando werden unterschiedlich viele Parameter vom Client gesendet und im Server empfangen. Der Server ruft nun eine Methode im Kommunikationsprotokoll (des Servers) mit allen verschlüsselten Parametern auf. Das Kommunikationsprotokoll des Servers hat die Aufgabe die Parameter nacheinander zu entschlüsseln und eine entsprechende Methode im Datenbankserver aufzurufen, die das Kommando ausführt. Es erhält von dieser Methode einen Rückgabewert, den es verschlüsseln und an den entsprechenden Client senden muß.

Der Server wartet in einer Endlosschleife auf die von den verschiedenen Clients gesendeten Daten. Sobald er diese empfängt, ruft er das Kommunikationsprotokoll des Servers auf und übergibt ihm die verschlüsselten Daten als Parameter. Dann ist der Ablauf des Kommunikationsprotokolls des Servers folgendermaßen:

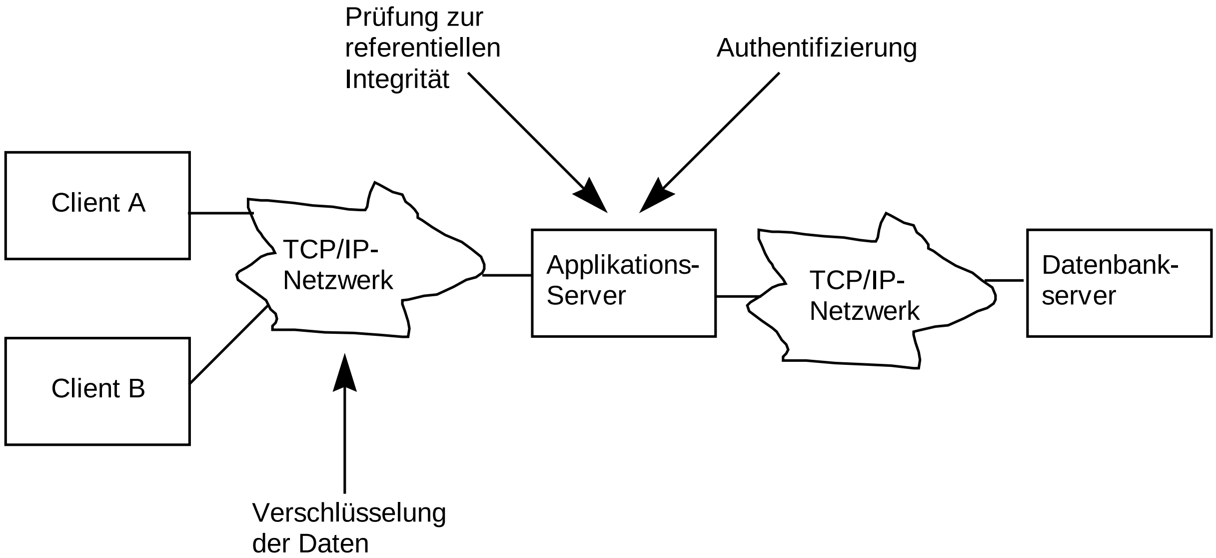
1. alle Parameter nacheinander verschlüsseln
2. alle verschlüsselten Parameter nacheinander an Datenbankserver senden
3. auf Rückgabewert warten
4. verschlüsselten Rückgabewert empfangen
5. Rückgabewert an Client senden

Beispiel. Der Benutzer meldet sich beim Rechner-Administrierungssystem mit seiner Kennung und seinem Paßwort an. Daraufhin sendet der Client die eingegebene Benutzerkennung und das Paßwort über das Kommunikationsprotokoll des Clients an das Kommunikationsprotokoll des Servers. Im Kommunikationsprotokoll des Servers wird dann eine entsprechende Prozedur aufgerufen, die überprüft, ob in der Datenbank einen Eintrag für die Benutzerkennung mit diesem Paßwort besteht. Je nach Datenbankinhalt wird true oder false an das Kommunikationsprotokoll des Servers zurückgeliefert, der nun den Rückgabewert an den wartenden Client sendet. Der Client ist nur dann berechtigt, mit der Vererarbeitung des Rechner-Administrierungsssytem fortzufahren, wenn der Rückgabewert true lautet. Stimmen seine Kennung und Paßwort nicht mit den in der Datenbank gespeicherten Werten überein, erhält er eine Fehlermeldung. Der Benutzer hat nur noch die Möglichkeit, die richtige Kennung mit dem dazu gehörigen Paßwort nochmals einzugeben oder das Rechner-Administrierungssystem zu beenden.

3.4 Sicherheitskonzept

Das Sicherheitskonzept besteht aus den Abschnitten Sicherheit im Domänenkonzept, Sicherheit im Datenbankkonzept und Sicherheit im Client-Server-Konzept.

Die folgende Abbildung zeigt, wie sich die Sicherheitsaspekte (Prüfung zur referentiellen Integrität, Authentifizierung und Verschlüsselung der im Netzwerk gesendeten Daten) im Konzept des Rechner-Administrierungssystems einordnen:



3.4.1 Sicherheit im Domänenkonzept

Um die Konsistenz des Domänenbaums nicht zu zerstören, müssen folgende Regeln im Domänenbaum immer gültig sein:

- der Name einer Domäne, eines Objekts oder einer Eigenschaft muß eindeutig sein; d.h. jeder Name kommt nur einmal im Rechner-Administrierungssystem vor; keine andere Domäne, Objekt oder Eigenschaft darf diesen Namen besitzen
- der Name einer Domäne, eines Objekts oder einer Eigenschaft darf nur geändert werden, wenn dieser neue Namen noch nicht im Domänenbaum vorkommt, d.h. eindeutig ist

- jede Domänen besitzt eine Domänenid und einen Namen
- jede Domäne gehört zu einem bestimmten Domänenbaum
- es gibt nur eine Wurzeldomäne pro Domänenbaum
- alle Domänen bis auf die Wurzeldomäne besitzen eine Oberdomäne
- eine Oberdomäne einer Domäne darf nicht zugleich deren Unterdomäne sein
- Geschwisterdomänen besitzen dieselbe Oberdomäne

- jede Domäne im Domänenbaum erbt alle Domäneneigenschaften der Wurzeldomäne
- jede Domäne erbt alle Domäneneigenschaften ihrer Oberdomäne
- jede Unterdomäne besitzt alle Domäneneigenschaften ihrer Oberdomäne
- lautet der Vererbungstyp einer Domäneneigenschaft AW, müssen alle Unterdomänen und deren Unterdomänen hinab bis zu den Blättern auch den gleichen Wert bei dieser Domäneneigenschaft besitzen
- der Wert einer Domäneneigenschaft kann nur verändert werden, wenn diese Domäneneigenschaft nicht geerbt wurde oder den Vererbungstyp AT besitzt

- jedes Objekt besitzt eine Objektid und einen Namen
- jedes Objekt muß zu mindestens einer Domäne im Domänenbaum gehören
- ein Objekt kann nur zu Domänen gehören, die im Domänenbaum existieren

- jede Eigenschaft besitzt einen Namen
- jede Eigenschaft gehört zu einer bestimmten Domäne oder einem bestimmten Objekt, die im Domänenbaum existieren
- jede Eigenschaft einer Domäne oder eines Objekts besitzt einen Eigenschaftswert

- jede Domäne oder jedes Objekt, welches in der Eigenschaft bezüglich angegeben wird, muß im Domänenbaum existieren

Diese Regeln müssen nun passend für das Datenbankkonzept, welches den Domänenbaum enthält, umformuliert werden:

3.4.2 Sicherheit im Datenbankkonzept

Die Sicherheit im Datenbankkonzept umfaßt folgende Bereiche:

1. referentielle Integrität der Daten
2. nur befugter Zugriff auf die Datenbank (Authentifizierung)
3. Sicherung der Datenbank (Backup)
4. Wiederherstellung der Datenbank im Fehlerfall (Recovery)

3.4.2.1 Referentielle Integrität der Daten

Der vorherige Abschnitt über die Regeln zur Erhaltung der Konsistenz des Domänenkonzepts, wird nun auf das Datenbankschema übertragen. Um die logische Konsistenz des Datenbank nicht zu zerstören, müssen folgende Regeln immer Gültigkeit besitzen, dabei wird angenommen, daß es nur einen Domänenbaum im Rechner-Administrierungssystem gibt:

- der Name einer Tabelle darf nur geändert werden, wenn es keine Tabelle in der Datenbank, keinen Eintrag in der Tabelle Domänen und keinen Eintrag in der Tabelle Objekte mit dem neuen Namen gibt

- jede Domäne besitzt eine Eintrag in der Tabelle Domänen mit Domänenid und Namen
- jede Domäne besitzt genau eine Domäneneigenschaftstabelle, die den Namen der Domäne trägt
- gibt es mehr als einen Eintrag in der Tabelle Domänen, muß auch in der Tabelle Vererbung ein Eintrag vorhanden sein
- gibt es n Einträge in der Tabelle Domänen, müssen mindestens n-1 Einträge in der Tabelle Vererbung vorhanden sein
- jede Domänenid der Tabelle Domänen muß mindestens einmal in der Tabelle Vererbung in der Spalte did1 oder did2 erscheinen
- es darf nur ein Eintrag in der Tabelle Vererbung zu zwei bestimmten Domänen existieren

- in der Domäneneigenschaftstabelle einer Domäne müssen mindestens so viele Einträge wie in der Domäneneigenschaftstabelle ihrer Oberdomäne vorhanden sein
- zu jedem Eintrag in der Domäneneigenschaftstabelle muß es genau eine Eigenschaftstabelle geben, die den Namen der Eigenschaft trägt
- jede Eigenschaft, die in der Domäneneigenschaftstabelle den Vererbungstyp AW besitzt, muß in der entsprechenden Eigenschaftstabelle Einträge mit dem gleichen Wert und der Domänenid aller Unterdomänen und deren Unterdomänen hinab bis zu den Blättern besitzen
- gibt es in einer Eigenschaftstabelle Einträge mit verschiedenen Werten, so muß beim entsprechenden Eintrag in der Domäneneigenschaftstabelle der Vererbungstyp AT stehen

- zu jedem Objekt muß es genau einen Eintrag in der Tabelle Objekte mit Objektid und Namen geben
- zu jedem Objekt muß es genau eine Objekteigenschaftstabelle geben, die den Namen des Objekts hat
- zu jedem Objekt muß es mindestens einen Eintrag in der Tabelle Domänenobjekte geben
- jede Objektid der Tabelle Objekte muß mindestens einmal in der Tabelle Domänenobjekte stehen

- zu jeder Eigenschaft gibt es genau eine Eigenschaftstabelle, die den Namen der Eigenschaft hat
- in der Tabelle Domänen und Objekte dürfen keine Einträge vorkommen, die den Namen einer Eigenschaftstabelle besitzen
- jeder Name einer Eigenschaftstabelle muß mindestens einen Eintrag in einer Objekt- oder Domäneneigenschaftstabelle besitzen
- zu jedem Eintrag in einer Domänen- bzw. Objekteigenschaftstabelle muß es mindestens einen Eintrag mit der Domänen- bzw. Objektid und dem Wert in der entsprechenden Eigenschaftstabelle geben
- in jeder Eigenschaftstabelle muß es mindestens einen Eintrag geben
- jede in einer Eigenschaftstabelle vorkommende Domänen- bzw. Objektid, muß einen entsprechenden Eintrag in der Tabelle Domänen bzw. Objekte besitzen

- die Spalten bzgl und bzglid von Domänen- und Objekteigenschaftstabellen müssen auf eine in der Tabelle Domänen oder Objekte vorkommende gültige Domänen- bzw. Objektid verweisen

Konsistenzprüfungen im Applikationsserver

Jede Update-Operation, die ein Benutzer im Rechner-Administrierungssystem ausführt, kann die Konsistenz der Datenbank zerstören. Deshalb müssen im Applikationsserver Prüfungen stattfinden, welche die logische Konsistenz der Daten gewährleisten sollen.

3.4.2.2 Authentifizierung

Die **Authentifizierung** im Rechner-Administrierungssystem umfaßt den Vorgang der Zugriffsberechtigungs-prüfung im Rechner-Administrierungssystem. Jeder Benutzer besitzt bereits eine Benutzerkennung und ein Paßwort, um sich im bestehenden Netzwerk des Instituts für Informatik anzumelden. Diese können im Rechner-Administrierungssytem übernommen werden. Die **Autorisierung** des Benutzers erfolgt durch seine Benutzerkennung und sein Benutzerpaßwort, die der Benutzer beim Starten des Rechner-Administrierungssystems eingeben muß. Im Applikationsserver erfolgt eine Überprüfung der eingegebenen Daten mit den in der Datenbank gespeicherten Werten. Nur wenn die Daten übereinstimmen, erhält der Benutzer die Berechtigung, mit der Vererarbeitung im Rechner-Administrierungssystem zu beginnen. Im Rahmen des Prototyps wird von digitalen Zertifikaten und Signaturen abgesehen. In der Zugriffstabelle der Datenbank ist festgelegt, welcher Benutzer welchen Zugriff auf welche Operationen des Domänenbaums ausführen darf. Diese Prüfung erfolgt ebenfalls im Applikationsserver.

3.4.2.3 Sicherung der Datenbank

Zur Sicherung der Datenbank werden Sicherungskopien erstellt. Jeder Benutzer besitzt die Möglichkeit selbst eine Sicherung vorzunehmen. Zusätzlich erfolgt eine automatische Sicherung der Datenbank im Server zu bestimmten Zeitabständen. Im Kapitel 5.3.4.1 wird auf die Möglichkeit eingegangen, nur partielle Sicherungen vorzunehmen.

3.4.2.4 Wiederherstellung der Datenbank im Fehlerfall

Ein weiterer Sicherheitsaspekt betrifft die Wiederherstellung der Datenbank im Fall von Inkonsistenzen, Festplattenfehlern und Abstürzen der Server und Clients. Ein Rechner kann nur für einen Moment nicht erreichbar sein oder auch für längere Zeit. Beide Fälle müssen effizient, die Konsistenz bewahrend bzw. wiederherstellend behandelt werden. Im Fehlerfall muß das Protokoll genau betrachtet werden (manuell). Die Wiederherstellung der Datenbank erfolgt manuell anhand von Kopien und der Log-Dateien. Dieselbe Vorgehensweise findet auch bei Festplattenfehlern Anwendung. Im Kapitel 5.3.4.3 wird auf die Möglichkeit eingegangen, ein Programm zur Konsistenzüberprüfung der Datenbank zu verwenden.

3.4.3 Sicherheit im Client-Server-Konzept

3.4.3.1 Sicherheitsmaßnahmen der Clients und Server

Sicherheitsmaßnahmen auf Seiten der Server

Nur Administratoren dürfen direkt auf die Server zugreifen. Damit ist außer von Seiten der Administratoren kein direkter Zugriff auf die Datenbank möglich. Die Administratoren können auch über Clients auf die Server und die jeweilige Datenbank zugreifen.

Sicherheitsmaßnahmen auf Seiten der Clients

Sicherheitsmaßnahmen auf Seiten der Clients werden zum einen über Benutzerkennungen und Paßwörter und zum anderen über Verschlüsselungsmechanismen geregelt.

Benutzerkennungen und Paßwörter

> Benutzerkennungen und Paßwörter regeln den Zugriff auf das Rechner-Administrierungssystem. Jeder Benutzer besitzt seine eigene Benutzerkennung und ein Paßwort.

3.4.3.2 Verschlüsselung

Die Verschlüsselung betrifft die über das Netzwerk gesendeten Daten zwischen Clients und Server. Ein Client stellt eine Anfrage an einen Server und ein Server antwortet dem Client. In beiden Fällen müssen die Daten verschlüsselt werden. Für die Verschlüsselung der Daten des Rechner-Adminsitrierungssystem wird der RSA-Algorithmus vorgesehen (siehe Abschnitt 2.4). Zu Beginn der vorliegenden Arbeit wurde festgelegt, das umfassendere SSL-Protokoll anstatt des RSA-Algorithmus zu verwenden. Doch gab es zum Zeitpunkt der Recherche im Januar 1999 kein kostenlos erhältliches Java-Paket für das SSL-Protokoll. Die Verschlüsselung der Daten findet in den Kommunikationsprotokollen des Clients und des Servers statt. Dort werden sämtliche Daten verschlüsselt, bevor sie über das Netz gesendet werden, und entsprechend entschlüsselt, ehe sie weiterverarbeitet werden.

3.4.3.3 Ausfallsicherheit

Im Netzwerk können Clients bzw. Server ausfallen:

3.4.3.3.1 Serverausfall

Der Ausfall eines Servers macht sich folgendermaßen beim Client bemerkbar: Der Client sendet eine Anfrage an den Server und erhält keine Antwort. Der Server kann ausgefallen sein, bevor er die Anfrage erhalten hat. Eine weitere Möglichkeit ist, daß der Server ausgefallen ist, nachdem er die Anfrage erhalten hat und bevor er eine Antwort an den Client senden konnte. Handelt es sich bei der Anfrage des Clients um einen schreibenden Zugriff auf die Datenbank, kann dies zu kritischen Situationen führen. Manche Änderungen dürfen nur einmal ausgeführt werden, wie z.B. eine bestimmte Kontoüberweisung. Im Prototyp für das Rechner-Administrierungssystem wird davon ausgegangen, daß eine Änderung auch mehrmals ausgeführt werden darf. Dies hat den Vorteil, daß nicht zwischen Ausfall des Clients vor Erhalt der Anfrage und nach Erhalt der Anfrage unterschieden werden muß. Der Client erhält eine Fehlermeldung, wenn der Server nicht erreichbar ist. Es bleibt dem Administrator überlassen, manuell festzustellen, was die Ursache dafür ist.

3.4.3.3.2 Clientausfall

Fällt der Client aus, bevor der Server die Anfrage erhalten hat, wird die Wiederholung der Anfrage dem Benutzer überlassen. Im Falle, daß der Client ausgefallen ist, nachdem er eine Anfrage an den Server gesendet hatte und bevor er die Antwort erhalten hat, liegt folgende Situation beim Server vor: er hat eine Anfrage bekommen und ausgeführt. Die Antwort an den Client, der die Anfrage gesendet hat, ist nicht möglich. Es wird dem Anwender, der die Anfrage gesendet hatte, überlassen, nachdem seine Workstation wieder verfügbar ist, die letzte Anfrage zu wiederholen bzw. zu überprüfen, ob sie ordnungsgemäß auf dem Server ausgeführt wurde.

4 Implementierung des Rechner-Administrierungssystems

Nach der Beschreibung der Aufgabenstellung für den Prototyp des Rechner-Administrierungssystems folgt eine Erläuterung dessen Entwicklungsumgebung. Anschließend wird die Umsetzung des Konzepts erläutert. Es folgen eine Beschreibung der Kommandozeilen-Schnittstelle und der Testumgebung.

4.1 Aufgabenstellung

Die Aufgabenstellung für den Prototyp des Rechner-Administrierungssystems lautet, einen Domänenbaum anhand ein bis zwei Konfigurationsdateien zu erstellen, der das im vorigen Kapitel beschriebene Gesamtkonzept umsetzt. Folgende weitere Anforderungen werden an den Prototyp gestellt:

- modularer Entwurf
- Prototyp mit Kommandozeilen-Schnittstelle
- Testumgebung auch für Folgeversionen verwendbar
- Erstellung einer Dokumentation
- Java als Programmiersprache für den Server
- plattformunabhängige Programmiersprache für den Client
- kostenlos erhältliche Datenbank mit Schnittstelle zu Java (JDBC-Treiber)

4.1.1 Domänenbaum

Hauptgewicht bei der Implementierung des Prototyps für das Rechner-Administrierungssystem liegt auf der Realisierung des Domänenkonzepts, insbesondere der Vererbung (siehe Kapitel 3.1.4 Vererbung).

4.1.2 Konfigurationsdateien

Konfigurationsdateien sind Dateien, die für die Verwaltung eines Rechners bzw. Betriebssystems notwendige Informationen enthalten. Im Betriebssystem UNIX werden diese Dateien auch nach dem Verzeichnis, in dem sie sich befinden, /etc-Dateien genannt. Für den Prototyp des Rechner-Administrierungssystems sollten gemäß Anforderung der Betreuer die Konfigurationsdateien /etc/passwd und /etc/group implementiert werden.

4.1.2.1 Inhalt und Aufbau der Konfigurationsdatei /etc/passwd

Die Datei /etc/passwd enthält Informationen über die Benutzer eines Rechners: den (alphanumerischen) Namen der Benutzerkennung, das Paßwort, Vor- und Nachname des Benutzers, eine numerische Benutzerkennung UID, eine numerische Gruppenkennung GID, das Heimverzeichnis des Benutzers und den Kommandointerpreter. Die Datei besteht aus einer Zeile pro Benutzer, in der die einzelnen Felder durch Doppelpunkte voneinander getrennt sind. Ein Beispiel einer Zeile für die Konfigurationsdatei /etc/passwd ist:

maier:x:Hans Maier:3200:45:home/stud/maier:/bin/csh

4.1.2.2 Inhalt und Aufbau der Konfigurationsdatei /etc/group

Die Konfigurationsdatei /etc/group enthält Informationen über die Gruppen eines Rechners. Mehrere Benutzer können zu einer Gruppe zusammengefaßt werden. Alle Mitglieder einer Gruppe besitzen in der Gruppe dieselben Rechte. Die Datei besteht aus den folgenden Feldern: den (alphanumerischen) Namen der Gruppe, das Paßwort der Gruppe, einer numerische Gruppenkennung GID und einer Liste der zur Gruppe zugehörigen Benutzer. Wie bei der Konfigurationsdatei /etc/passwd sind die einzelnen Felder durch Doppelpunkte voneinander getrennt. Es folgt ein Beispiel für eine Zeile in der Datei /etc/group:

praktikum3:x:4015:maier,mueller,bauer

4.1.2.3 Konfigurationsdateien /etc/shadow und /etc/gshadow

Manche Betriebssysteme erlauben allen Benutzern lesenden Zugriff auf die Konfigurationsdateien /etc/passwd und /etc/group. Damit die Benutzer die Paßwörter der anderen Benutzer nicht lesen und unerlaubt verwenden können, stehen in diesen Dateien statt der Paßwörter Platzhalter in Form eines x. Die “echten” Paßwörter befinden sich dann in den Konfigurationsdateien /etc/shadow und /etc/gshadow. Diese beiden Dateien sind geschützt, so daß ein lesender Zugriff von nicht autorisierten Benutzern verhindert wird. Die Datei /etc/shadow wird der Datei /etc/passwd und die Datei /etc/gshadow der Datei /etc/group zugeordnet. Auf Anweisung der Betreuer sollten die Dateien /etc/shadow und /etc/gshadow ebenfalls im Prototyp realisiert werden.

4.2 Entwicklungsumgebung des Prototyps

4.2.1 Wahl der Programmiersprache

Eine Anforderung an den Prototyp des Rechner-Administrierungssystems lautet, Java als Programmiersprache für die Programmierung des Servers zu verwenden. Auf Seiten des Clients gab es die Wahl zwischen den plattformunabhängigen Programmiersprachen Java und Perl. Da es sonst keine zwingende Gründe gab, die für die Verwendung von Perl gesprochen hätten, fiel die Wahl auch für den Client auf Java, um nicht zwischen zwei verschiedenen Programmiersprachen wechseln zu müssen.

4.2.2 Wahl der Datenbank

Bei der Wahl des Datenbankverwaltungssystems (DBMS) sollte folgendes beachtet werden: das DBMS muß eine Java-Schnittstelle besitzen, kostenlos und in möglichst vielen UNIX- wie Windows-Betriebssystemen in unterschiedlichen Versionen erhältlich sein. Dabei kann es sich um ein relationales oder objektorientiertes Datenbankmodell handeln. Eine Recherche im Internet ergab, daß es zur Zeit der Suche im Dezember 1998 zwei weit verbreitete Datenbanksysteme gab, die die oben genannten Anforderungen erfüllten: msql und MySQL. Beide sind kostenlos erhältlich, besitzen unter anderem auch Schnittstellen zu den Programmiersprachen Java und Perl, sind auf unterschiedlichsten Plattformen einsetzbar und besitzen ein relationales Datenbankmodell. Da msql weniger SQL-Funktionalitäten im Vergleich zu MySQL besitzt und MySQL bereits im Rechenzentrum des Instituts für Informatik der Ludwig-Maximilian-Universität in München, für welches das Rechner-Administrierungssystem realisiert werden sollte, installiert war, wurde MySQL gewählt.

4.2.3 Wahl des Datenbanktreibers

In der Dokumentation von MySQL wurden drei Datenbanktreiber erwähnt, die eine Verbindung zwischen dem DBMS und der Programmiersprache Java herstellen. Ihnen gemeinsam war, daß sie die Version 1.1 von Java (JDK) unterstützen und frei erhältlich sind. Zum Zeitpunkt der Suche im Januar 1999 gab es keine Treiber für die Version 1.2 von Java (JDK), die eben freigegeben wurde. Da einer der Treiber nach eigenen Angaben des Autors veraltet war, wurden nur die zwei neueren Datenbanktreiber getestet. Die Wahl fiel auf twz1, da es beim anderen Treiber bereits beim Setup Probleme gab. Nachdem in der Umgebungsvariable CLASSPATH, auf welche das JDK zugreift, der richtige Pfad, auf dem sich der Datenbanktreiber befindet, eingefügt wurde und in die zum Treiber gehörige Konfigurationsdatei der Rechner, auf dem das DBMS läuft und der Name der Datenbank, welches die Daten des Rechner-Administrierungssystems enthalten sollte, mit Kennung und Paßwort richtig gesetzt wurden, liefen die Testprogramme des Treibers twz1 fehlerfrei. Die Konfigurationsdatei des Datenbanktreibers muß im gleichen Unterverzeichnis stehen, in dem sich die Programme befinden, sonst wird die Konfigurationsdatei nicht gefunden und die Verbindung mit der Datenbank schlägt fehl.

4.2.4 Wahl des Betriebssystems (Plattform)

Der Prototyp sollte nach der Fertigstellung auf unterschiedlichsten UNIX- und Windows-Systemen ablauffähig sein. Dabei sind im Rechenzentrum des Instituts für Informatik Solaris und Linux und WindowsNT im Einsatz. Da die Rechner, auf denen Linux installiert war, sowohl das Datenbanksystem MySQL als auch die Programmiersprache Java in den für den Datenbanktreiber benötigten Versionen (MySQL 3.21 oder höher und JDK 1.1.7) bereits im Einsatz hatten, wurde Linux als Entwicklungsumgebung für den Prototyp des Rechner-Administrierungs-systems gewählt.

4.3 Umsetzung des Konzepts

Im Prototyp des Rechner-Administrierungssystems sollen laut Aufgabenstellung alle wesentliche Bestandteile des Konzepts realisiert werden:

4.3.1 Domänenkonzept

Die wesentlichen Bestandteile des Domänenkonzepts sind der Domänenbaum mit Domänen, Objekten und Eigenschaften und die Vererbung von Domäneneigenschaften. Die Kommandos der Kommandozeilen-Schnittstelle des Rechner-Administrierungssystems bestehen aus den Operationen auf die Domänen und Objekte, wie sie in Abschnitt 3.1.5.7 und 3.1.5.8 definiert wurden.

4.3.2 Datenbankkonzept

Das Datenbankkonzept setzt das Domänenkonzept um. Die Hauptaufgabe bei der Implementierung besteht nun darin, die grundlegenden Tabellen in der Datenbank zu erzeugen und die Operationen mit den benötigten Konsistenzprüfungen im Applikationsserver zu implementieren.

4.3.2.1 Modul db

Im Modul **db** werden Methoden für die Datenbankverarbeitung implementiert, die speziell auf das Rechner-Administrierungssystem konfiguriert werden:

1. Verbindung zur Datenbank, in der die Daten des Rechner-Administrierungssystems gespeichert werden sollen, öffnen
2. Datenbankkatalog setzen
3. Befehl, der für alle weiteren Datenbankzugriffe benötigt wird, initialisieren
4. Initialisieren der Datenbank
5. Verbindung zur Datenbank schließen

Der Datenbanktreiber twz1 liefert grundlegende Methoden für diese fünf Schritte. Im Modul db mußten die Parameter wie z.B. der Name der Datenbank oder der Name des Datenbankkatalogs für das Rechner-Administrierungssystem gesetzt werden.

Initialisierung der Datenbank bezeichnet das Anlegen folgender Tabellen in der Datenbank:

1. Tabelle **Domänen** mit den Spalten did (Domänenid) und dname (Domänenname)
2. Tabelle **Objekte** mit den Spalten oid (Objektid) und oname (Objektname)
3. Tabelle **Domänenobjekte** mit den Spalten id (intern vergebener Primärschlüssel für die Tabelle), did und oid
4. Tabelle **Vererbung** mit den Spalten id (intern vergebender Primärschlüssel für die Tabelle) did1, richtung („<“ odr „>“) und did2
5. Tabelle **Zugriffsrechte** mit den Spalten id, del, get, new , set und attr.

Mit dem SQL-Befehl **create table** konnten diese Tabellen in der Datenbank angelegt werden.

4.3.2.2 Modul dom

dom stellt eine Abkürzung für Domäne bzw. domain dar. In der Klasse dom befinden sich Methoden zum Erstellen, Ausgeben, Ändern und Löschen von Domänen und ihren Attributen. Dabei wird die Vererbung entsprechend der Stellung bzw. Lage im Domänenbaum beachtet. Das Modul enthält alle für Domänen benötigten Konsistenzprüfungen. Im Anhang B folgt in der Beschreibung der Dokumentation für den Prototyp des Rechner-Administrierungssystems eine Beschreibung der Methoden.

4.3.2.3 Modul obj

Die Klasse obj enthält Methoden, mit denen Objekte und ihre Eigenschaften erstellt, gelöscht, geändert, angezeigt um mehrfach zugeordnet werden können. Die Konsistenzprüfungen für Operationen auf Objekten befinden sich in diesem Modul. Im Anhang B folgt in der Beschreibung der Dokumentation für den Prototyp des Rechner-Administrierungssystems eine Beschreibung der Methoden.

4.3.2.4 Modul attr

Die Klasse attr enthält Methoden zum Erstellen, Ausgeben, Ändern und Löschen von Eigenschaften. Dabei werden in der Klasse dom Eigenschaften behandelt, die den Domänen zugerechnet werden, und in der Klasse obj Eigenschaften, die Objekten zugeordnet werden. Das Modul enthält Konsistenzprüfungen, die sich auf Eigenschaften beziehen. Im Anhang B folgt in der Beschreibung der Dokumentation für den Prototyp des Rechner-Administrierungssystems eine Beschreibung der Methoden.

4.3.2.5 Modul domtree

Das Modul domtree enthält die Kommandozeilen-Schnittstelle mit den Operationen der Moduln dom, obj und attr.

Arbeitsweise des Moduls domtree:

1. Das Kommando der Kommandozeilen-Schnittstelle bzw. des Menüs einlesen
2. die Parameter für die entsprechende Operation aus einer Datei oder über die Tastatur einlesen
3. den entsprechenden Zugriff auf die Datenbank vornehmen
4. Rückgabewert und Meldungen auf Bildschirm ausgeben

4.3.3 Client-Server-Konzept

Im Client-Server-Konzept muß das Kommunikationsprotokoll umgesetzt werden. Dabei kann man zwischen dem Kommunikationsprotokoll für den Server und dem Kommunikationsprotokoll für die Clients differenzieren. Sämtliche Daten, die zwischen Clients und Server gesendet und empfangen werden, sollen nur in verschlüsselter Form im Netzwerk gesendet und empfangen werden. Aus Sicherheitsgründen empfiehlt es sich, eine strikte Trennung von Zugriffen auf die Konfigurationsdateien und die Datenbank einzuhalten.

4.3.3.1 Kommunikationsprotokoll des Servers

Nur der Server soll (lesenden und schreibenden) Zugriff auf die Datenbank besitzen.

4.3.3.2 Kommunikationsprotokoll der Clients

Nur die Clients sollen (lesenden und schreibenden) Zugriff auf die Konfigurationsdateien besitzen. Ein Client darf nur auf „seine“, d.h. sich auf seinem Rechner befindenden Konfigurationsdateien zugreifen.

4.3.3.3 Aufteilung der generischen Module auf Clients und Server

Die folgende Tabelle beschreibt eine Aufteilung der Module auf Clients und Server:

Modul	Clients	Server
Konfigurationsdateien lesen und schreiben	x	
Format der Konfigurationsdateien berücksichtigen	x	
Kommunikationsprotokoll der Clients	x	
Ver- und Entschlüsseln der Daten	x	x
Kommunikationsprotokoll des Servers		x
Datenbank lesen und schreiben		x
Konsistenzprüfungen berücksichtigen (Applikationsserver)		x

x hat die Bedeutung, daß das entsprechende Modul in der entsprechenden Spalte (Client bzw. Server) realisiert werden soll.

4.3.4 Sicherheitskonzept

Das Sicherheitskonzept umfaßt folgende Bereiche: Konsistenzprüfungen des Domänen- und Datenbankkonzepts, Backup und Recovery der Datenbank und Verschlüsselung der Daten, die zwischen Clients und Server ausgetauscht werden und anschließende Entschlüsselung der Daten.

Die Konsistenzprüfungen für das Domänen- und das Datenbankkonzept wurden in den Moduln dom, obj und attr realisiert.

4.4 Kommandozeilen-Schnittstelle

Der Prototyp des Rechner-Administrierungssystems soll laut Anforderung über eine Kommandozeilen-Schnittstelle verfügen. Eine Kommandozeile ist eine Befehlszeile, die über die Tastatur eingegeben oder in einem Programm aufgerufen werden kann. Es können alle Operationen auf Domänen und Objekte über die Kommandozeilen-Schnittstelle eingegeben werden (siehe Abschnitt 4.3.2.5). Die Daten in den Konfigurationsdateien (siehe Abschnitt 4.1.2) können ebenfalls über die Kommandos zu Domänen und Objekten verarbeitet werden:

- **new_dom** erstellt eine Domäne *user* für die Konfigurationsdatei /**etc**/**passwd** erstellt und fügt dieses im Domänenbaum ein
- **new_obj** fügt einen bestimmten Benutzer unter Angabe der Benutzerkennung als Name für das Objekt in die Domäne *user* ein
- **new_attr_obj** erstellt die Objekteigenschaft Paßwort (analog weitere Eigenschaften wie Vor- und Nachname des Benutzers etc. erstellen)
- **ins_attr** fügt die Objekteigenschaft zu einem Objekt ein
- **ins_attr_value** fügt einen Wert für die Eigenschaft zu einem Objekt ein

Analog lauten die Kommandos für die Konfigurationsdateien /**etc**/**group**, /**etc**/**shadow** und /**etc**/**gshadow**.

4.5 Testumgebung

Eine der Anforderungen an den Prototyp des Rechner-Administrierungssystems lautet, eine Testumgebung zu entwickeln, auf die in Folgearbeiten aufgesetzt werden kann.

Das im Abschnitt 4.3.2.5 beschriebene Modul **domtree** wurde so realisiert, daß es eine Kommandozeilen-Schnittstelle, ein Menü und eine Testumgebung für den Prototyp liefert: mit dem Modul können die Methoden der Moduln dom, obj und attr getestet werden.

4.6 Dokumentation

Für die Dokumentation des Prototyps wurde das frei erhältliche Programm **Javadoc** verwendet, welches ein Bestandteil des JDK bei der Installierung von Java ist. Dieses kann automatisch aus den Kommentaren in Klassendateien Dokumentationsdateien im HTML-Format erstellen. Diese Dateien oder auch Klassendokumentationen wandeln Kommentare in eine einheitliche Form um. Konstruktoren und Methoden der Klasse können mit Parametern beschrieben werden. Im Anhang B befindet sich die Dokumentation für denPrototyp des Rechner-Administrierungssystems.

5 Zusammenfassung und Ausblick

5.1 Zusammenfassung

Zunächst erfolgte eine Literaturrecherche, welche im Kapitel 2 „Grundlagen" zusammengefaßt wurde. Ein wesentlicher Aspekt der Literaturrecherche bestand in der Untersuchung von bestehenden Netzwerk-Informationsdiensten. Es zeigte sich, daß zum Zeitpunkt der Literaturrecherche, die bis Ende Dezember 1998 andauerte, kein Netzwerk-Informationsdienst gefunden werden konnte, der alle in der Aufgabenstellung für diese Arbeit genannten Anforderungen erfüllen konnte.

Anschließend folgte die Erstellung eines eigenen Konzepts für ein Rechner-Administrierungssystem (siehe Kapitel 3), welches unabhängig von den im Kapitel 2.5 beschriebenen Konzepten entworfen wurde. Es wurden folgende Teilkonzepte entwickelt: ein Domänen-, ein Datenbank-, ein Client-Server- und ein Sicherheitskonzept. Dabei lag der Schwerpunkt dieser Arbeit in der Entwicklung des Domänenkonzepts. Domänen werden hierarchisch in einem Domänenbaum angeordnet. Sie können ihre Eigenschaften unter gewissen Voraussetzungen in zwei Varianten an andere Domänen des Domänenbaums vererben. Das Datenbankkonzept baut dann auf das Domänenkonzept auf. Es wurde ein ER-Modell zum Domänenkonzept entwickelt, welches in ein relationales Datenmodell umgesetzt wurde. Die Rechner des Rechenzentrums, für welches das Rechner-Administrierungssystem entworfen werden sollte, sind über ein TCP/IP-Netzwerk miteinander verbunden. Das Client-Server-Konzept geht von einer 3-Ebenen-Architektur für dieses Netzwerk aus. Die drei Ebenen sind die Clients, der Applikationsserver und der Datenbankserver. Es wurde ein eigenes Kommunikationsprotokoll auf Seiten der Clients und des Applikationsservers entworfen, über welches der Austausch der Informationen des Rechner-Administrierungssystems erfolgt. Das Sicherheitskonzept umfaßt die Bereiche referentielle Integrität der Datenbank, Verschlüsselung der Daten im Netzwerk, Sicherung (Backup) und Wiederherstellung (Recovery) der Datenbank. Im Applikationsserver finden Püfungen statt, welche die Konsistenz des Domänenkonzepts und die referentielle Integrität der Datenbank gewährleisten. Zwischen den Clients und Servern des Rechner-Administrierungssystems ist eine Verschlüsselung aller im Netzwerk ausgetauschten Daten mittels des RSA-Algorithmus vorgesehen.

Es wurde ein Prototyp für das Rechner-Administrationssystem in der plattformunabhängigen und objektorientierten Programmiersprache Java implementiert. Die Speicherung der Daten erfolgt in der relationalen Datenbank MySQL. Als Datenbankschnittstelle für Java wurde der Datenbanktreiber twz1 gewählt. Das Betriebssystem Linux stellte die Entwicklungsumgebung für den Prototyp. Der Benutzer kann den Prototyp für das Rechner-Administrierungssystem über eine Kommandozeilen-Schnittstelle oder ein Menü bedienen.

Anhand des Prototyps können die wesentlichen Bestandteile des Konzepts getestet und überprüft werden: Es kann ein hierarchischer Domänenbaum mit Domänen, Objekten, Domänen- und Objekteigenschaften verwaltet werden. Dabei wird die Vererbung von Domäneneigenschaften, die Eigenschaft bezüglich und die Mehrfachzuordnung von Objekten berücksichtigt. Die referentielle Integrität der Daten wird im Applikationsserver gewährleistet.

5.2 Ausblick

Viele Rechenzentren besitzen Netzwerke mit heterogenen Rechner-Architekturen in den verschiedensten Versionen und suchen nach einem Programm, welches den hohen Verwaltungsaufwand für diese Rechner reduziert. Dabei hat die Firma Novell mit ihrem Netzwerk-Informationsdienst NDS ein Programm erstellt, welches viele Anforderungen an einen Netzwerk-Informationsdienst erfüllt. Am Netzwerk-Informationsdienst NDS waren und sind in einer Entwicklungszeit von über zehn Jahren viele Mitarbeiter beteiligt. Es liegt auf der Hand, daß der Prototyp des Rechner-Administrierungssystems, der in dieser Arbeit realisiert wurde, im Vergleich zu NDS nur einen geringen Umfang an Funktionen besitzt. Ein wichtiger Vorteil des hier entwickelten Rechner-Administrierungssystems liegt jedoch in der geringen Anzahl von Funktionen, die genau auf die Zielanforderungen eines Rechenzentrums zugeschnitten sind. Dadurch wird zudem die Komplexität niedrig gehalten. Für das kommerzielle Produkt NDS fallen hohe Kosten für die Anschaffung des Produkts und für die Schulung der Mitarbeiter in Form von Kursen und langer Einarbeitungszeiten an. Rechenzentren kleiner Unternehmen und öffentlicher Einrichtungen wie Universitäten, die nur wenige Funktionen von NDS benötigen, können sich mit diesem Rechner-Administrierungssystem hohe Kosten und Arbeitsaufwand sparen.

5.3 Mögliche Erweiterungen

Es folgen nach den Teilkonzepten gegliedert Erweiterungen, die in einer Folgeversion berücksichtigt werden können:

5.3.1 Domänenkonzept

5.3.1.1 Obligatorische und optionale Eigenschaften

Im vorliegenden Prototyp gibt es nur obligatorische Domäneneigenschaften und ausschließlich optionale Objekteigenschaften. In einer Folgeversion sollten auch optionale Domäneneigenschaften und obligatorische Objekteigenschaften definiert werden können.

5.3.1.2 Vererbung von Objekteigenschaften

Bisher können nur Domäneneigenschaften vererbt werden. In einer künftigen Version ist es wünschenswert, daß auch Vererbung bei Objekteigenschaften möglich ist.

Dies bezieht sich auch auf die Eigenschaft bezüglich. Bisher wird die Vererbung nur im Fall „Domäne besitzt Eigenschaft bezüglich einer anderen Domäne oder eines Objekts“ beachtet. Für den Fall „Objekt besitzt Eigenschaft bezüglich einer Domäne oder eines anderen Objekts“ ist bisher keine Vererbung möglich (siehe Kapitel 3.1.3.4 Eigenschaft „bezüglich).

5.3.1.3 Mehrfache Eigenschaften der Art bezüglich

Der Prototyp behandelt nur die zwei Fälle, daß eine bestimmte Domäne bezüglich <u>einer</u> anderen Domäne **oder** eines Objekts gilt und, daß ein bestimmtes Objekt bezüglich <u>einer</u> Domäne **oder** eines anderen Objekts gilt (siehe Kapitel 3.1.3.4 Eigenschaft „bezüglich). In der aktuellen Version des Prototyps ist folgendes nicht vorgesehen:

- eine Domäne kann **nicht** bezüglich einer anderen Domäne **und** eines Objekts definiert werden
- eine Domäne kann **nicht** bezüglich **zwei oder mehr** Domänen definiert werden
- ein Objekt kann **nicht** bezüglich einer Domäne **und** eines anderen Objekts definiert werden
- ein Objekt kann **nicht** bezüglich **zwei oder mehr** Objekten definiert werden

5.3.1.4 Mehrere Domänenbäume

Das Domänenkonzept behandelt nur einen Domänenbaum. In einer Folgeversion sollten die Ausführungen im Domänenkonzept auch für die Existenz von mehreren Domänenbäumen formuliert werden.

5.3.2 Datenbankkonzept

5.3.2.1 Tabelle Vererbung

Nach einer Überprüfung des Prototyps stellte sich heraus, daß die Spalte richtung in der Tabelle entfallen kann.

5.3.2.2 Stored Procedures

Das für diesen Prototyp gewählte Datenbankverwaltungssystem (DBMS) MySQL verfügt nicht über die Möglichkeit, **stored procedures** zu programmieren. Daher mußte im Client-Server-Konzept ein 3-Ebenen-Modell mit einem Applikationsserver eingeführt werden. Durch die Wahl eines anderen DBMS können die umfangreichen Konsistenzprüfungen, die im Applikationsserver implementiert wurden, dort entfallen und vom DBMS durch Definition entsprechender stored procedures übernommen werden.

5.3.3 Client-Server-Konzept

5.3.3.1 Graphische Oberfläche

Für die Benutzer des Rechner-Administrierungssystems sollte eine graphische Oberfläche zur Verfügung stehen.

5.3.4 Sicherheitskonzept

5.3.4.1 Partielle Sicherung der Datenbank

Im Sicherheitskonzept ist nur die Möglichkeit vorgesehen, die gesamte Datenbank zu sichern. Mit wachsender Größe der Datenbank, ist es auch aus Zeitgründen sinnvoll, zusätzlich eine Sicherung von auswählbaren Teilen der Datenbank einzuführen.

5.3.4.2 Sicherung der Datenbank nach Update-Operationen

Nach jeder schreibenden Operation auf der Datenbank sollte eine partielle Sicherung des Bereichs der Datenbank, der verändert wurde, angestoßen werden.

5.3.4.3 Programm zur Prüfung der Konsistenz der Datenbank

Um die Konsistenz einer Datenbank zu erhalten, gehört eine regelmäßige Pflege der Datenbank zu den Aufgaben eines Datenbankverwalters. Im Prototyp kann der Verwalter des Rechner-Administrierungssystems nur manuell überprüfen, ob sich die Datenbank in einem konsistenten Zustand befindet. Ab einer gewissen Größe der Datenbank ist dies kaum mehr möglich. Dann ist ein Programm zur Prüfung der Konsistenz des Domänenbaumes und der Datenbank erforderlich.

6 Abkürzungsverzeichnis

ACL	Access Control List, Zugriffkontroll-Liste
ANSI	American National Standards Institute
AT	Attributstyp
AW	Attributswert
DAP	Directory Access Protocol
DBMS	Database Management System, Datenbank-Verwaltungssystem
DES	Data Encryption Standard
DNA	Domain Naming Service
DSA	Digital Signature Algorithm
ER-Diagramm	Entity Relationship-Diagramm
ER-Modell	Entity Relationship-Modell
JDBC	Java Database Connectivity
JDK	Java Developer’s Kit
LAN	Local Area Network
LDAP	Lightweight Directory Access Protocol
LDAPv2	Version 2 des LDAP
MAD	Microsoft Active Directory
NDS	Novell Directory Services
NIS	Network Information System
NIS+	Network Information System Plus
OSI	Open System Interconnect
PGP	Pretty Good Privacy
RFC	Request for Comment
SNMP	Simple Network Management Protocol
SQL	Standard Query Language
SSL	Secure Socket Layer
TCP/IP	Transmission Control Protocol/Internet Protocol
WAN	Wide Area Network

7 Literaturverzeichnis

[CoHo 96] Gary Cornell, Cay S. Horstmann. Core Java. SunSoft Press, 1996.

[HAN 99] Heinz-Gerd Hegering, Sebastian Abeck, Bernhard Neumair. Integriertes Management vernetzter Systeme: Konzepte, Architekturen und deren betrieblicher Einsatz. dpunkt-Verlag, 1999.

[HeGr 94] Matthias Hein, David Griffiths. SNMP: Simple Network Management Protocol Version 2. International Thomson Publishing, 1994.

[JDBC 99] http://java.sun.com/products/jdbc/jdbc.drivers.html JDBC™ Drivers. Stand: Januar 1999.

[JDK1 99] http://java.sun.com/products/jdk/1.2/ Java Development Kit 1.2. Stand: Januar 1999.

[KeIv 98] David Kearns, Brian Iverson. The Complete Guide to Novell Directory Services. SYBEX, 1998.

[KeEi 96] Alfons Kemper, André Eickler. Datenbanksysteme: eine Einführung. R. Oldenbourg Verlag, 1996.

[Li 98] Liwu Li. Java: Data Structures and Programming. Springer-Verlag, 1998.

[Meie 95] Andreas Meier. Relationale Datenbanken: eine Einführung für die Praxis. Springer-Verlag, 2. Auflage 1995.

[MyS1 99] http://www.tcx.se/ MySQL by T.c.X DataKonsultAB. Stand: Januar 1999.

[MyS2 99] http://www.worldserver.com/mm.mysql/ MM Mysql Drivers. Stand: Januar 1999.

[NIS1 99] http://sunsite.unc.edu/mdw/HOWTO/NIS-HOWTO.html The Linux NIS(YP)/NYS/NIS+ HOWTO. Stand: Januar 1999.

[Rees 97] George Reese. Database Programming with JDBC and Java. O'Reilly & Associates, June 1997.

[Schn 96] Bruce Schneier. Angewandte Kryptographie: Protokolle, Algorithmen, Sourcecode in C. Addison-Wesley, 1996.

[Schu 97] Hans Herbert Schulze. Computer-Englisch Ein Fachwörterbuch. Rowohlt Taschenbuch Verlag, 1997.

[Twz1 99] http://www.voicenet.com/ellert/tjFM JDBC for MySQL. Stand: Januar 1999.

[Voss 94] Gottfried Vossen. Datenmodelle, Datenbanksprachen und Datenbank-Management-Systeme. Addison-Wesley, 2.Auflage 1994.

[Weis 98] Mark Allen Weiss. Data Structures & Problem Solving Using Java. Addison Wesley Longman, 1998.

[Zack 99] http://may3d.com Zack's RSA in Java. Stand: Januar 1999.

8 Anhang A Benutzerhandbuch

Das Benutzerhandbuch für den Prototyp des Rechner-Administrierungssystems umfaßt die Sortware-Voraussetzungen, Erläuterungen zum Entpacken der in dieser Arbeit beigefügten Diskette, zur Umgebungsvariablen CLASSPATH, zur Konfigurationsdatei des Datenbanktreibers twz1, zu Konstanten des Prototyps und zum Starten des Prototyps. Die Dokumentation zum Prototyp befindet sich im Anhang B.

8.1 Software-Voraussetzungen

Der Prototyp braucht folgende Umgebung:

- Programm ZIP (zum Entpacken der Diskette)
- Java Development Kit (JDK) Version 1.1.7
- Datenbank MySQL Version 3.21 oder höher

8.2 Quellcode entpacken

Der Quellcode zum Prototyp des Rechner-Administrierungssystems steht zusammen mit dem Datenbanktreiber twz1 in gepackter Form auf einer Diskette. Die Diskette befindet sich in einer Diskettentasche, welche auf der letzten Seite der vorliegenden Arbeit angebracht ist.

Auf der Diskette befinden sich folgende Dateien:

- **proto.zip** (enthält den Quellcode zum Prototyp)
- **twz1jdbcForMysql-1.0.3-GA.tar.gz** (enthält den Datenbanktreiber twz1)

Folgende Schritte sind nötig, um beide Dateien zu entpacken:

Ein Verzeichnis aussuchen bzw. erstellen, in welches der Datenbanktreiber kopiert werden soll und den Datenbanktreiber mit folgendem Befehl entpacken:

- unter Windows 95/NT: auf die Datei **twz1jdbcForMysql-1.0.3-GA.tar.gz** „doppelklicken“; das Programm WinZip ruft sich auf; nun alle Dateien mit Pfadangabe in das gewünschte Verzeichnis extrahieren;
- unter Linux: **gunzip -c twz1jdbcForMysql-1.0.3-GA.tar.gz|tar -xvf -**

Ein Verzeichnis aussuchen bzw. erstellen, in welches der Quellcode zum Prototyp kopiert werden soll. Den Quellcode mit folgendem Befehl entpacken:

- unter Windows 95/NT: auf die Datei **proto.zip** „doppelklicken“; das Programm WinZip ruft sich auf; nun alle Dateien in das gewünschte Verzeichnis extrahieren;
- unter Linux: **unzip proto.zip**

Die Umgebungsvariable CLASSPATH entsprechend setzen (siehe nächster Abschnitt); ggf. die Konfigurationsdatei des Datenbanktreibers ändern (siehe Abschnitt 8.4); die Konstanten des Moduls **db** ggf. ändern (siehe Abschnitt 8.5);

Prototyp mit dem Befehl **java domtree** im Verzeichnis, in das er kopiert wurde, starten (siehe Abschnitt 8.6).

8.3 Umgebungsvariable CLASSPATH

In die Umgebungsvariable **CLASSPATH** muß der Pfad eingetragen werden, in der sich der Prototyp und der Datenbanktreiber befinden. Im folgenden Beispiel befindet sich das Verzeichnis für den Datenbanktreiber (twz1) mit seinen Unterverzeichnissen im Unterverzeichnis *test* der Benutzerkennung *mamblona*, die wiederum ein Unterverzeichnis der Kennung *stud* und der Kennung *home* ist. Die Dateien zum Prototyp befinden sich in einem Unterverzeichnis des Verzeichnisses *test*. Dann lautet der Eintrag für die Umgebungsvariable CLASSPATH:

setenv CLASSPATH .:/home/stud/mamblona/test (Linux)

Für Windows 95/NT muß ein entsprechender Eintrag in der Datei **autoexec.bat** eingetragen und diese danach ausgeführt werden.

8.4 Konfigurationsdatei des Datenbanktreibers twz1

In der Konfigurationsdatei des Datenbanktreibers twz1 sind folgende Änderungen an den Parametern vorzunehmen: Name der Datenbank, Rechner, der die Datenbank enthält, Name des Datenbankkatalogs (Name des Datenbanktreibers)

Die Konfigurationsdatei befindet sich im Verzeichnis, in dem der Quellcode kopiert wurde, in der Datei **.jdbcMysql.properties** (das erste Zeichen ist ein „.“).

8.5 Konstanten im Quellcode

Im Modul **db** befinden sich folgende datenbankspezifische Konstanten: katalog, treibername und url. Diese müssen ggf. geändert werden.

8.6 Starten des Prototyps

Es gibt zwei Varianten, wie der Prototyp gestartet werden kann: über die Kommandozeilen-Schnittstelle oder ein Menü.

8.6.1.1 Kommandozeilen-Schnittstelle

Die Kommandozeilen-Schnittstelle wird mit folgendem Befehl gestartet, der in dem Verzeichnis, in das der Quellcode kopiert wurde, ausgeführt werden muß:

java domtree *Kommandozeilen-Datei*

Der Parameter ***Kommandozeilen-Datei*** muß durch den Namen der Datei, welche die auszuführenden Kommandos enthält, ersetzt werden. Auf der Diskette befinden sich im Verzeichnis **proto.zip** folgende Kommandozeilen-Dateien: **test_ins.txt** und **test_del.txt**. Die Dateinamen von Kommandozeilen-Dateien können beliebig gewählt werden.

8.6.1.2 Menü

Der Befehl zum Starten des Menüs, der in dem Verzeichnis, in das der Quellcode kopiert wurde, ausgeführt werden muß, lautet:

java domtree

Anhang B Dokumentation des Prototyps

Die Dokumentation des Prototyp wurde mit Javadoc erstellt. Sie umfaßt die folgenden Klassen: domtree, dom, obj, attr und db.

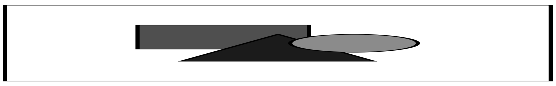
Class domtree

```
java.lang.Object
   |
   +----domtree
```

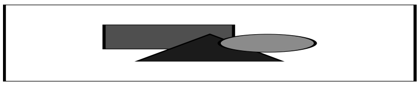
---

public class **domtree**
extends Object
Die Klasse domtree verfügt über ein Menü und eine Kommandozeilenschnittstelle, mit der alle Operationen des Domänenbaums getestet werden können

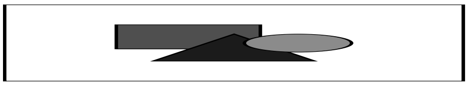
---



domtree()

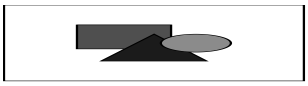


main(String[])
Die Methode main öffnet die Datenbank und ruft das über das Menü oder die Kommandozeilenschnittstelle gewählte Kommando...



domtree

```
public domtree()
```



main

```
public static void main(String args[])
```

Die Methode main öffnet die Datenbank und ruft das über das Menü oder die Kommandozeilenschnittstelle gewählte Kommando bzw. die Operation in der Klasse dom, object oder attr auf; der Parameter ist optional; wenn eine Verarbeitung über eine Kommandozeilenschnittstelle erfolgen soll, enthält er die Eingabedatei mit den Kommandos; wird kein Parameter angegeben, kann der Benutzer das Menü über die Tastatur bedienen

Parameters:
args - Datei für die Kommandozeilenschnittstelle (entfällt bei Eingabe über die Tastatur)

Class dom

```
java.lang.Object
   |
   +----dom
```

---

public class **dom**
extends Object Die Klasse dom verfügt über Methoden, um Domänen in den Domänenbaum einzuhängen, umzuhängen, sich aus dem Domänenbaum ausgeben zu lassen oder zu entfernen. Es können der Name einer Domäne, die Namen der Eigenschaften oder die Werte von Eigenschaften geändert werden.

Variable Index

- **n**
  Die Konstante n gibt die Vektorgröße an

Constructor Index

- **dom**()

Method Index

- **del_attr**(Statement, Statement, String, Vector)
  Die Methode del_attr löscht die im Vektor attr_name angegebenen Attribute aus der mit dem Parameter d_name spezifizierten Domäne und ihren Subdomänen, sofern diese nicht geerbt wurden
- **del_attr_no_inh**(Statement, Statement, String, Vector)
  Die Methode del_attr_no_inh löscht die im Vektor attr_name angegebenen Attribute aus der mit dem Parameter d_name spezifizierten Domäne und ihren Subdomänen, sofern diese nicht geerbt wurden
- **del_dom**(Statement, Statement, String)
  Die Methode del_dom löscht die mit d_name spezifizierte Domäne aus dem Domänenbaum, falls sie keine Unterdomänen und keine Objekte besitzt
- **del_inh_attr**(Statement, Statement, String)
  Die Methode del_inh_attr löscht die geerbten Attribute der mit dem Parameter d_name spezifizierten Domäne
- **del_refer**(Statement, Statement, String, String)
  Die Methode del_refer löscht die Eigenschaft bezüglich bei der Domäneneigenschaft attr_name der Domäne d_name
- **del_refer_no_inh**(Statement, Statement, String, String)
  Die Methode del_refer_no_inh löscht die Eigenschaft bezüglich bei der Domäneneigenschaft attr_name der Domäne d_name
- **exist_dom**(Statement, String)
  Die Methode exist_dom prüft, ob es die mit d_name spezifizierte Domäne gibt
- **get_attr**(Statement, String)
  Die Methode get_attr liefert die Attributsnamen der mit d_name spezifizierten Domäne
- **get_attr_value**(Statement, String, String)

Die Methode get_attr_value liefert den Attributwert der mit d_name spezifizierten Domäne zum mit attr_name angegebenen Attribut

- **get_attr_values**(Statement, String)
  Die Methode get_attr_values liefert die Attributwerte der mit d_name spezifizierten Domäne zu allen Attributen
- **get_attr_values**(Statement, String, Vector)
  Die Methode get_attr_values liefert die Attributwerte der mit d_name spezifizierten Domäne zu den mit attr_name angegebenen Attributen
- **get_dom**(Statement, Statement, String)
  Die Methode get_dom liefert die id, den Namen der Oberdomäne, Subdomänen, Objekte, Attribute und Attributwerte der mit d_name spezifizierten Domäne
- **get_id**(Statement, String)
  Die Methode get_id liefert die id der mit d_name spezifizierten Domäne oder -1, falls es diese nicht gibt
- **get_inh_attr**(Statement, String)
  Die Methode get_inh_attr liefert die Namen der geerbten Attribute zu der mit d_name spezifizierten Domäne
- **get_name**(Statement, int)
  Die Methode get_name liefert den Namen der Domäne zu dem mit id übergebenen Parameter
- **get_oberdom**(Statement, String)
  Die Methode get_oberdom liefert den Namen der Oberdomäne
- **get_refer**(Statement, String, String)
  Die Methode get_refer liefert den Namen des Objektes oder der Domäne bezüglich der mit dem Parameter attr_name angegebenen Domäneneigenschaft
- **get_subdom**(Statement, Statement, String)
  Die Methode get_subdom liefert die Subdomänen
- **has_obj**(Statement, String)
  Die Methode has_obj liefert true, falls die mit dem Parameter d_name spezifizierte Domäne Objekte besitzt; sonst false
- **has_subdom**(Statement, String)
  Die Methode has_subdom liefert true, falls die mit dem Parameter d_name spezifizierte Domäne Subdomänen besitzt; sonst false
- **ins_attr**(Statement, Statement, Connection, String, Vector, int, Vector, Vector)
  Die Methode ins_attr fügt die Attribute und ihre Eigenschaften, wie die id der Oberdomänen, die Art der Vererbung und, ob die Eigenschaft bzgl.
- **ins_attr**(Statement, Statement, Connection, String, Vector, Vector, Vector)
  Die Methode ins_attr fügt die Attribute und ihre Eigenschaften, die Art der Vererbung, ob die Eigenschaft bzgl.
- **ins_attr_no_inh**(Statement, Statement, Connection, String, Vector, int, Vector, Vector)
  Die Methode ins_attr_no_inh wird nur von der Methode ins_inh_attr aufgerufen.
- **ins_attr_values**(Statement, Connection, Vector, int, Vector)
  Die Methode ins_attr_values fügt für die mit dem im Parameter attr_name enthaltenen Attributsnamen die im Parameter attr_value spezifizierten Attributswerte in die Attributstabellen mit der mit dem Parameter id übergebenen id ein
- **ins_dom**(Statement, Statement, Connection, String, String)
  Die Methode ins_dom fügt die mit dem Parameter d_name spezifizierte Domäne in den Domänenbaum unter die mit ober_d_name angegebene Oberdomäne an
- **ins_inh_attr**(Statement, Statement, Connection, String)
  Die Methode ins_inh_attr fügt in die mit d_name spezifizierte Domäne die geerbten

Attribute ein

- **is_subdom**(Statement, Statement, String, String)
  Die Methode is_subdom liefert true, falls die mit dem Parameter d_name spezifizierte Domäne eine Subdomäne der mit dem Parameter ober_d_name angegebenen Oberdomäne ist; sonst false
- **mov_dom**(Statement, Statement, Connection, String, String)
  Die Methode mov_dom entfernt die mit dem Parameter d_name spezifizierte Domäne aus der bisherigen Position samt geerbten Attributen aus dem Domänenbaum und fügt sie unter der mit new_oberd_name angegebenen Oberdomäne samt deren vererbten Eigenschaften in den Domänenbaum ein; die ehemaligen Subdomänen der Domäne d_name werden mit umgehängt, wobei die Vererbung beachtet wird
- **mov_dom**(Statement, Statement, Connection, String, String, String)
  Die Methode mov_dom entfernt die mit dem Parameter d_name spezifizierte Domäne aus der bisherigen Position samt geerbten Attributen aus dem Domänenbaum und fügt sie unter der mit new_oberd_name angegebenen Oberdomäne samt deren vererbten Eigenschaften in den Domänenbaum ein; die mit new_subd_name angegebene Subdomäne der mit new_oberd_name spezifizierten Oberdomäne wird Subdomäne der Domäne d_name
- **new_attr**(Statement, Connection, Vector)
  Die Methode new_attr legt die im Vektor attr_name angegebenen Attributsnamen mittels der Methode als neue Tabellen an
- **new_dom**(Statement, Connection, String)
  Die Methode new_dom fügt den mit root_name übergebenen Domänennamen in die Tabelle Domänen ein und legt eine Tabelle für die Attribute der Domäne an, die die Spalten id, attr, odid, at_aw, bzgl und bzglid besitzt
- **new_dom**(Statement, Statement, Connection, String, String)
  Die Methode new_dom fügt den mit d_name übergebenen Domänennamen in die Tabelle Domänen, die mit ober_d_name spezifizierte Oberdomäne und die Domäne d_name in die Tabelle vererbung ein und legt eine Tabelle für die Attribute der Domäne an, die die Spalten id, attr, odid, at_aw, bzgl und bzglid besitzt
- **new_dom**(Statement, Statement, Connection, String, String, String)
  Die Methode new_dom fügt den mit d_name übergebenen Domänennamen in die Tabelle Domänen, die mit ober_d_name spezifizierte Oberdomäne und die Domäne d_name in die Tabelle vererbung ein und legt eine Tabelle für die Attribute der Domäne an, die die Spalten id, attr, odid, at_aw, bzgl und bzglid besitzt; die mit subd_name spezifizierte Subdomäne der Oberdomäne wird mittels der Methode mov_dom im Domänenbaum als Subdomäne der neuen Domäne d_name umgehängt
- **set_at_aw**(Statement, String, String, String)
  Die Methode set_at_aw ändert die Vererbungseigenschaft attr_name der Domäne d_name um
- **set_attr**(Statement, Statement, String, Vector, Vector)
  Die Methode set_attr ändert den Namen der im Vektor old_attr_name spezifizierten Attribute in die Namen der im Vektor new_attr_name angegebenen Namen um
- **set_attr_values**(Statement, Statement, String, Vector, Vector)
  Die Methode set_attr_values ändert die Werte der mit den Namen der im Vektor attr_name spezifizierten Attribute in die Werte der im Vektor attr_values angegebenen um
- **set_dom**(Statement, String, String)
  Die Methode set_dom ändert den Namen der mit dem Parameter old_d_name spezifizierte Domäne in den mit dem Parameter new_d_name angegebenen Namen
- **set_refer**(Statement, Statement, String, String, String)

Die Methode set_refer fügt die Eigenschaft bezüglich in die Domäneneigenschaft attr_name der Domäne d_name ein.

set_refer_no_inh(Statement, Statement, String, String, String)

Die Methode set_refer_no_inh fügt die Eigenschaft bezüglich in die Domäneneigenschaft attr_name der Domäne d_name ein.

Variables

n

```
static final int n
```

Die Konstante n gibt die Vektorgröße an

Constructors

dom

```
public dom()
```

Methods

exist_dom

```
static boolean exist_dom(Statement st,
                         String d_name)
```

Die Methode exist_dom prüft, ob es die mit d_name spezifizierte Domäne gibt

Parameters:

st - der zur Datenbank sendende SQL-Befehl

d_name - der Name der Domäne

get_id

```
static int get_id(Statement st,
                  String d_name)
```

Die Methode get_id liefert die id der mit d_name spezifizierten Domäne oder -1, falls es diese nicht gibt

Parameters:

st - der zur Datenbank sendende SQL-Befehl

d_name - der Name der Domäne

new_dom

```
static boolean new_dom(Statement st,
                       Connection verbindung,
                       String root_name)
```

Die Methode new_dom fügt den mit root_name übergebenen Domänennamen in die Tabelle Domänen ein und legt eine Tabelle für die Attribute der Domäne an, die die Spalten id, attr, odid, at_aw, bzgl und bzglid besitzt

Parameters:

st - der zur Datenbank sendende SQL-Befehl

verbindung - die Verbindung zur Datenbank

root_name - der Name der Domäne

new_dom

```
static boolean new_dom(Statement st,
                       Statement st2,
```

```
                      Connection verbindung,
                      String d_name,
                      String ober_d_name)
```

Die Methode new_dom fügt den mit d_name übergebenen Domänennamen in die Tabelle Domänen, die mit ober_d_name spezifizierte Oberdomäne und die Domäne d_name in die Tabelle vererbung ein und legt eine Tabelle für die Attribute der Domäne an, die die Spalten id, attr, odid, at_aw, bzgl und bzglid besitzt

Parameters:

st - der zur Datenbank sendende SQL-Befehl
verbindung - die Verbindung zur Datenbank
d_name - der Name der Domäne
ober_d_name - der Name der Oberdomäne

new_dom

```
static boolean new_dom(Statement st,
                      Statement st2,
                      Connection verbindung,
                      String d_name,
                      String ober_d_name,
                      String subd_name)
```

Die Methode new_dom fügt den mit d_name übergebenen Domänennamen in die Tabelle Domänen, die mit ober_d_name spezifizierte Oberdomäne und die Domäne d_name in die Tabelle vererbung ein und legt eine Tabelle für die Attribute der Domäne an, die die Spalten id, attr, odid, at_aw, bzgl und bzglid besitzt; die mit subd_name spezifizierte Subdomäne der Oberdomäne wird mittels der Methode mov_dom im Domänenbaum als Subdomäne der neuen Domäne d_name umgehängt

Parameters:

st - der zur Datenbank sendende SQL-Befehl
st2 - ein weiterer zur Datenbank sendender SQL-Befehl
verbindung - die Verbindung zur Datenbank
d_name - der Name der Domäne
ober_d_name - der Name der Oberdomäne
sub_d_name - der Name der Subdomäne

del_dom

```
static boolean del_dom(Statement st,
                      Statement st2,
                      String d_name)
```

Die Methode del_dom löscht die mit d_name spezifizierte Domäne aus dem Domänenbaum, falls sie keine Unterdomänen und keine Objekte besitzt

Parameters:

st - der zur Datenbank sendende SQL-Befehl
st2 - ein weiterer zur Datenbank sendender SQL-Befehl
d_name - der Name der Domäne

new_attr

```
static boolean new_attr(Statement st,
                       Connection verbindung,
                       Vector attr_name)
```

Die Methode new_attr legt die im Vektor attr_name angegebenen Attributsnamen mittels der Methode als neue Tabellen an

Parameters:

st - der zur Datenbank sendende SQL-Befehl
verbindung - die Verbindung zur Datenbank
attr_name - der Name des Vektors, der die Attributsnamen enthält

ins_attr_no_inh

```
static boolean ins_attr_no_inh(Statement st,
                               Statement st2,
                               Connection verbindung,
                               String d_name,
                               Vector attr_name,
                               int ober_did,
                               Vector attr_at_aw,
                               Vector attr_bzgl)
```

Die Methode ins_attr_no_inh wird nur von der Methode ins_inh_attr aufgerufen. Sie unterscheidet sich von der Methode ins_attr, indem sie die Methode ins_inh_attr nicht aufruft; es werden die mit dem im Parameter attr_name angegebenen Attribute und ihre Eigenschaften in die mit dem Parameter d_name spezifizierte Domänentabelle eingefügt

Parameters:

st - der zur Datenbank sendende SQL-Befehl
st2 - ein weiterer zur Datenbank sendender SQL-Befehl
verbindung - die Verbindung zur Datenbank
d_name - der Name der Domäne
attr_name - der Name des Vektors, der die Attributsnamen enthält
ober_did - die id der Oberdomäne
attr_at_aw - der Name des Vektors, der für jedes Attribut die Art der Vererbung at, aw oder Leerzeichen (blank) enthält
attr_bzgl - der Name des Vektors, der für jedes Attribut einen leeren String, den Namen des Objektes oder den Namen der Domäne enthält bzgl. dieser das Attribut gilt

ins_attr

```
static boolean ins_attr(Statement st,
                        Statement st2,
                        Connection verbindung,
                        String d_name,
                        Vector attr_name,
                        int ober_did,
                        Vector attr_at_aw,
                        Vector attr_bzgl)
```

Die Methode ins_attr fügt die Attribute und ihre Eigenschaften, wie die id der Oberdomänen, die Art der Vererbung und, ob die Eigenschaft bzgl. einer anderen Domäne bzw. Objektes gilt, in die mit dem Parameter d_name spezifizierte Domänentabelle ein; es wird die Methode ins_inh_attr aufgerufen, die alle Attribute von der Oberdomäne an die Domäne erbt

Parameters:

st - der zur Datenbank sendende SQL-Befehl
st2 - ein weiterer zur Datenbank sendender SQL-Befehl
verbindung - die Verbindung zur Datenbank
d_name - der Name der Domäne
attr_name - der Name des Vektors, der die Attributsnamen enthält
ober_did - die id der Oberdomäne
attr_at_aw - der Name des Vektors, der für jedes Attribut die Art der Vererbung at, aw oder o enthält
attr_bzgl - der Name des Vektors, der für jedes Attribut einen leeren String, den Namen des Objektes oder den Namen der Domäne enthält bzgl. dieser das Attribut gilt

ins_attr

```
static boolean ins_attr(Statement st,
                        Statement st2,
                        Connection verbindung,
```

```
                              String d_name,
                              Vector attr_name,
                              Vector attr_at_aw,
                              Vector attr_bzgl)
```

Die Methode ins_attr fügt die Attribute und ihre Eigenschaften, die Art der Vererbung, ob die Eigenschaft bzgl. einer anderen Domäne bzw. Objektes gilt und -1 als odid, welches die Attribute als nicht geerbte ausweist, in die mit dem Parameter d_name spezifizierte Domänentabelle ein es wird die Methode ins_inh_attr aufgerufen, die alle Attribute von der Oberdomäne an die Domäne erbt

Parameters:

st - der zur Datenbank sendende SQL-Befehl
st2 - ein weiterer zur Datenbank sendender SQL-Befehl
verbindung - die Verbindung zur Datenbank
d_name - der Name der Domäne
attr_name - der Name des Vektors, der die Attributsnamen enthält
attr_at_aw - der Name des Vektors, der für jedes Attribut die Art der Vererbung at, aw oder o enthält
attr_bzgl - der Name des Vektors, der für jedes Attribut einen leeren String, den Namen des Objektes oder den Namen der Domäne enthält bzgl. dieser das Attribut gilt

ins_attr_values

```
static boolean ins_attr_values(Statement st,
                               Connection verbindung,
                               Vector attr_name,
                               int did,
                               Vector attr_values)
```

Die Methode ins_attr_values fügt für die mit dem im Parameter attr_name enthaltenen Attributsnamen die im Parameter attr_value spezifizierten Attributswerte in die Attributstabellen mit der mit dem Parameter id übergebenen id ein

Parameters:

st - der zur Datenbank sendende SQL-Befehl
verbindung - die Verbindung zur Datenbank
attr_name - der Name des Vektors, der die Attributsnamen enthält
did - die id der Domäne
attr_values - der Name des Vektors, der für jedes Attribut den einzufügenden Wert enthält

has_subdom

```
static boolean has_subdom(Statement st,
                          String d_name)
```

Die Methode has_subdom liefert true, falls die mit dem Parameter d_name spezifizierte Domäne Subdomänen besitzt; sonst false

Parameters:

st - der zur Datenbank sendende SQL-Befehl
d_name - der Name der Domäne

has_obj

```
static boolean has_obj(Statement st,
                       String d_name)
```

Die Methode has_obj liefert true, falls die mit dem Parameter d_name spezifizierte Domäne Objekte besitzt; sonst false

Parameters:

st - der zur Datenbank sendende SQL-Befehl
d_name - der Name der Domäne

del_attr

```
static boolean del_attr(Statement st,
                        Statement st2,
                        String d_name,
                        Vector attr_name)
```

Die Methode del_attr löscht die im Vektor attr_name angegebenen Attribute aus der mit dem Parameter d_name spezifizierten Domäne und ihren Subdomänen, sofern diese nicht geerbt wurden

Parameters:

st - der zur Datenbank sendende SQL-Befehl
st2 - ein weiterer zur Datenbank sendender SQL-Befehl
d_name - der Name der Domäne
attr_name - der Name des Vektors, der die Namen der zu löschenden Attribute enthält

del_attr_no_inh

```
static boolean del_attr_no_inh(Statement st,
                               Statement st2,
                               String d_name,
                               Vector attr_name)
```

Die Methode del_attr_no_inh löscht die im Vektor attr_name angegebenen Attribute aus der mit dem Parameter d_name spezifizierten Domäne und ihren Subdomänen, sofern diese nicht geerbt wurden

Parameters:

st - der zur Datenbank sendende SQL-Befehl
st2 - ein weiterer zur Datenbank sendender SQL-Befehl
d_name - der Name der Domäne
attr_name - der Name des Vektors, der die Namen der zu löschenden Attribute enthält

is_subdom

```
static boolean is_subdom(Statement st,
                         Statement st2,
                         String d_name,
                         String ober_d_name)
```

Die Methode is_subdom liefert true, falls die mit dem Parameter d_name spezifizierte Domäne eine Subdomäne der mit dem Parameter ober_d_name angegebenen Oberdomäne ist; sonst false

Parameters:

st - der zur Datenbank sendende SQL-Befehl
st2 - ein weiterer zur Datenbank sendender SQL-Befehl
d_name - der Name der Domäne
ober_d_name - der Name der Oberdomäne

ins_dom

```
static boolean ins_dom(Statement st,
                       Statement st2,
                       Connection verbindung,
                       String d_name,
                       String ober_d_name)
```

Die Methode ins_dom fügt die mit dem Parameter d_name spezifizierte Domäne in den Domänenbaum unter die mit ober_d_name angegebene Oberdomäne an

Parameters:

st - der zur Datenbank sendende SQL-Befehl
st2 - ein weiterer zur Datenbank sendender SQL-Befehl
verbindung - die Verbindung zur Datenbank
d_name - der Name der Domäne
ober_d_name - der Name der Oberdomäne

mov_dom

```
static boolean mov_dom(Statement st,
                       Statement st2,
                       Connection verbindung,
                       String d_name,
                       String new_oberd_name)
```

Die Methode mov_dom entfernt die mit dem Parameter d_name spezifizierte Domäne aus der bisherigen Position samt geerbten Attributen aus dem Domänenbaum und fügt sie unter der mit new_oberd_name angegebenen Oberdomäne samt deren vererbten Eigenschaften in den Domänenbaum ein; die ehemaligen Subdomänen der Domäne d_name werden mit umgehängt, wobei die Vererbung beachtet wird

Parameters:

st - der zur Datenbank sendende SQL-Befehl
st2 - ein weiterer zur Datenbank sendender SQL-Befehl
verbindung - die Verbindung zur Datenbank
d_name - der Name der Domäne
new_oberd_name - der Name der neuen Oberdomäne

mov_dom

```
static boolean mov_dom(Statement st,
                       Statement st2,
                       Connection verbindung,
                       String d_name,
                       String new_oberd_name,
                       String new_subd_name)
```

Die Methode mov_dom entfernt die mit dem Parameter d_name spezifizierte Domäne aus der bisherigen Position samt geerbten Attributen aus dem Domänenbaum und fügt sie unter der mit new_oberd_name angegebenen Oberdomäne samt deren vererbten Eigenschaften in den Domänenbaum ein; die mit new_subd_name angegebene Subdomäne der mit new_oberd_name spezifizierten Oberdomäne wird Subdomäne der Domäne d_name

Parameters:

st - der zur Datenbank sendende SQL-Befehl
st2 - ein weiterer zur Datenbank sendender SQL-Befehl
verbindung - die Verbindung zur Datenbank
d_name - der Name der Domäne
new_oberd_name - der Name der neuen Oberdomäne
new_subd_name - der Name der neuen Subdomäne

set_dom

```
static boolean set_dom(Statement st,
                       String old_d_name,
                       String new_d_name)
```

Die Methode set_dom ändert den Namen der mit dem Parameter old_d_name spezifizierte Domäne in den mit dem Parameter new_d_name angegebenen Namen

Parameters:

st - der zur Datenbank sendende SQL-Befehl
old_d_name - der alte Name der Domäne
new_d_name - der neue Name der Domäne

set_attr

```
static boolean set_attr(Statement st,
                        Statement st2,
                        String d_name,
                        Vector old_attr_name,
                        Vector new_attr_name)
```

Die Methode set_attr ändert den Namen der im Vektor old_attr_name spezifizierten Attribute in die Namen der im Vektor new_attr_name angegebenen Namen um

Parameters:

st - der zur Datenbank sendende SQL-Befehl
st2 - ein weiterer zur Datenbank sendender SQL-Befehl
d_name - der Name der Domäne
old_attr_name - der Name des Vektors, der die Namen der alten Attribute enthält
new_attr_name - der Name des Vektors, der die Namen der neuen Attribute enthält

set_attr_values

```
static boolean set_attr_values(Statement st,
                               Statement st2,
                               String d_name,
                               Vector attr_name,
                               Vector attr_values)
```

Die Methode set_attr_values ändert die Werte der mit den Namen der im Vektor attr_name spezifizierten Attribute in die Werte der im Vektor attr_values angegebenen um

Parameters:

st - der zur Datenbank sendende SQL-Befehl
st2 - ein weiterer zur Datenbank sendender SQL-Befehl
d_name - der Name der Domäne
attr_name - der Name des Vektors, der die Namen der Attribute enthält
attr_values - der Name des Vektors, der die neuen Attributwerte enthält

del_inh_attr

```
static boolean del_inh_attr(Statement st,
                            Statement st2,
                            String d_name)
```

Die Methode del_inh_attr löscht die geerbten Attribute der mit dem Parameter d_name spezifizierten Domäne

Parameters:

st - der zur Datenbank sendende SQL-Befehl
st2 - ein weiterer zur Datenbank sendender SQL-Befehl
d_name - der Name der Domäne

ins_inh_attr

```
static boolean ins_inh_attr(Statement st,
                            Statement st2,
                            Connection verbindung,
                            String d_name)
```

Die Methode ins_inh_attr fügt in die mit d_name spezifizierte Domäne die geerbten Attribute ein

Parameters:

st - der zur Datenbank sendende SQL-Befehl
st2 - ein weiterer zur Datenbank sendender SQL-Befehl
verbindung - die Verbindung zur Datenbank
d_name - der Name der Domäne

get_attr

```
static Vector get_attr(Statement st,
                       String d_name)
```

Die Methode get_attr liefert die Attributsnamen der mit d_name spezifizierten Domäne

Parameters:

st - der zur Datenbank sendende SQL-Befehl
d_name - der Name der Domäne

get_dom

```
static Vector get_dom(Statement st,
                      Statement st2,
                      String d_name)
```

Die Methode get_dom liefert die id, den Namen der Oberdomäne, Subdomänen, Objekte, Attribute und Attributwerte der mit d_name spezifizierten Domäne

Parameters:

st - der zur Datenbank sendende SQL-Befehl
st2 - ein weiterer zur Datenbank sendender SQL-Befehl
d_name - der Name der Domäne

get_attr_value

```
static String get_attr_value(Statement st,
                             String d_name,
                             String attr_name)
```

Die Methode get_attr_value liefert den Attributwert der mit d_name spezifizierten Domäne zum mit attr_name angegebenen Attribut

Parameters:

st - der zur Datenbank sendende SQL-Befehl
d_name - der Name der Domäne
attr_name - der Name des Attributs

get_attr_values

```
static Vector get_attr_values(Statement st,
                              String d_name,
                              Vector attr_name)
```

Die Methode get_attr_values liefert die Attributwerte der mit d_name spezifizierten Domäne zu den mit attr_name angegebenen Attributen

Parameters:

st - der zur Datenbank sendende SQL-Befehl
d_name - der Name der Domäne
attr_name - der Name des Attributs

get_attr_values

```
static Vector get_attr_values(Statement st,
                              String d_name)
```

Die Methode get_attr_values liefert die Attributwerte der mit d_name spezifizierten Domäne zu allen Attributen

Parameters:

st - der zur Datenbank sendende SQL-Befehl
d_name - der Name der Domäne

get_inh_attr

```
static Vector get_inh_attr(Statement st,
                           String d_name)
```

Die Methode get_inh_attr liefert die Namen der geerbten Attribute zu der mit d_name spezifizierten Domäne

Parameters:

st - der zur Datenbank sendende SQL-Befehl
d_name - der Name der Domäne

get_name

```
static String get_name(Statement st,
                       int did)
```

Die Methode get_name liefert den Namen der Domäne zu dem mit id übergebenen

Parameter

Parameters:

st - der zur Datenbank sendende SQL-Befehl
id - die id der Domäne

get_oberdom

```
static String get_oberdom(Statement st,
                          String d_name)
```

Die Methode get_oberdom liefert den Namen der Oberdomäne

Parameters:

st - der zur Datenbank sendende SQL-Befehl
d_name - der Name der Domäne

get_subdom

```
static Vector get_subdom(Statement st,
                         Statement st2,
                         String d_name)
```

Die Methode get_subdom liefert die Subdomänen

Parameters:

st - der zur Datenbank sendende SQL-Befehl
d_name - der Name der Domäne

get_refer

```
static String get_refer(Statement st,
                        String d_name,
                        String attr_name)
```

Die Methode get_refer liefert den Namen des Objektes oder der Domäne bezüglich der mit dem Parameter attr_name angegebenen Domäneneigenschaft

Parameters:

st - der zur Datenbank sendende SQL-Befehl
d_name - der Name der Domäne
attr_name - der Name der Eigenschaft

del_refer

```
static boolean del_refer(Statement st,
                         Statement st2,
                         String d_name,
                         String attr_name)
```

Die Methode del_refer löscht die Eigenschaft bezüglich bei der Domäneneigenschaft attr_name der Domäne d_name

Parameters:

st - der zur Datenbank sendende SQL-Befehl
st2 - ein weiterer zur Datenbank sendender SQL-Befehl
d_name - der Name der Domäne
attr_name - der Name der Eigenschaft

del_refer_no_inh

```
static boolean del_refer_no_inh(Statement st,
                                Statement st2,
                                String d_name,
                                String attr_name)
```

Die Methode del_refer_no_inh löscht die Eigenschaft bezüglich bei der Domäneneigenschaft attr_name der Domäne d_name

Parameters:

st - der zur Datenbank sendende SQL-Befehl
st2 - ein weiterer zur Datenbank sendender SQL-Befehl

d_name - der Name der Domäne
attr_name - der Name der Eigenschaft

set_refer

```
static boolean set_refer(Statement st,
                         Statement st2,
                         String d_name,
                         String attr_name,
                         String refer_name)
```

Die Methode set_refer fügt die Eigenschaft bezüglich in die Domäneneigenschaft attr_name der Domäne d_name ein. Der Parameter refer_name enthält das Objekt oder die Domäne, bezüglich der die Domäneneigenschaft gelten soll

Parameters:

st - der zur Datenbank sendende SQL-Befehl
st2 - ein weiterer zur Datenbank sendender SQL-Befehl
d_name - der Name der Domäne
attr_name - der Name der Eigenschaft
refer_name - der Name des Objektes oder der Domäne bezüglich der die Domäneneigenschaft gelten soll

set_refer_no_inh

```
static boolean set_refer_no_inh(Statement st,
                                Statement st2,
                                String d_name,
                                String attr_name,
                                String refer_name)
```

Die Methode set_refer_no_inh fügt die Eigenschaft bezüglich in die Domäneneigenschaft attr_name der Domäne d_name ein. Der Parameter refer_name enthält das Objekt oder die Domäne, bezüglich der die Domäneneigenschaft gelten soll. Diese Methode wird nur für die Vererbung von Domäneneigenschaften benötigt

Parameters:

st - der zur Datenbank sendende SQL-Befehl
st2 - ein weiterer zur Datenbank sendender SQL-Befehl
d_name - der Name der Domäne
attr_name - der Name der Eigenschaft
refer_name - der Name des Objektes oder der Domäne bezüglich der die Domäneneigenschaft gelten soll

set_at_aw

```
static boolean set_at_aw(Statement st,
                         String d_name,
                         String attr_name,
                         String at_aw)
```

Die Methode set_at_aw ändert die Vererbungseigenschaft attr_name der Domäne d_name um

Parameters:

st - der zur Datenbank sendende SQL-Befehl
d_name - der Name der Domäne
attr_name - der Name der Eigenschaft
at_aw - aw, falls der Wert der Eigenschaft geändert werden darf sonst at

Class obj

```
java.lang.Object
   |
   +----obj
```

---

public class **obj**
extends Object Die Klasse obj verfügt über Methoden, um Objekte einer Domäne zu erstellen, sie umzuhängen, sich ausgeben zu lassen oder zu entfernen. Es können der Name eines Objektes, die Namen der Eigenschaften oder die Werte von Eigenschaften geändert werden.

Variable Index

n
Die Konstante n gibt die Vektorgröße an

Constructor Index

obj()

Method Index

del_attr(Statement, String, Vector)
Die Methode del_attr löscht die im Vektor attr_name angegebenen Attribute aus dem mit dem Parameter o_name spezifizierten Objekt

del_mult(Statement, Statement, String, String)
Die Methode del_mult löst das mit o_name spezifizierte Objekt von der Domäne d_name, falls das Objekt mehrfach vorkommt, d.h.

del_obj(Statement, String)
Die Methode del_obj löscht das mit o_name spezifizierte Objekte aus dem Domänenbaum

del_refer(Statement, String, String)
Die Methode del_refer löscht die Eigenschaft bezüglich bei der Objekteigenschaft attr_name des Objektes o_name

exist_obj(Statement, String)
Die Methode exist_obj prüft, ob es das mit o_name spezifizierte Objekt gibt

get_attr(Statement, String)
Die Methode get_attr liefert die Attributsnamen des mit o_name spezifizierten Objektes

get_attr_values(Statement, String)
Die Methode get_attr_values liefert die Attributwerte des mit o_name spezifizierten Objektes zu allen Attributen

get_id(Statement, String)
Die Methode get_id liefert die id des mit o_name spezifizierten Objektes oder -1, falls es dieses nicht gibt

get_mult(Statement, String)
Die Methode get_mult liefert zum mit o_name spezifizierten Objekt die ids der

zugehörigen Domänen;

- **get_name**(Statement, int)
  Die Methode get_name liefert den Namen des Objektes zu dem mit id übergebenen Parameter
- **get_obj**(Statement, String)
  Die Methode get_obj liefert die id, die Namen der Domänen, zu denen es gehört, Attribute und Attributwerte des mit o_name spezifizierten Objektes
- **get_obj_from_dom**(Statement, Statement, String)
  Die Methode get_obj_from_dom liefert den Namen der Objekte zur der mit d_name spezifizierten Domäne
- **get_refer**(Statement, String, String)
  Die Methode get_refer liefert den Namen des Objektes oder der Domäne bezüglich der mit dem Parameter attr_name angegebenen Objekteigenschaft
- **ins_attr**(Statement, Connection, String, String, Vector, Vector)
  Die Methode ins_attr fügt die Attribute und, ob die Eigenschaft bzgl.
- **ins_attr**(Statement, Statement, Statement, Connection, String, String, Vector, Vector, int)
  Die Methode ins_attr fügt die Attribute und ihre Eigenschaften, ob die Eigenschaft bzgl.
- **ins_attr_values**(Statement, Connection, String, Vector, Vector)
  Die Methode ins_attr_values fügt die mit dem im Parameter attr_name enthaltenen Attributsnamen die im Parameter attr_values spezifizierten Attributswerte in die Attributstabellen für das Objekt o_name ein
- **ins_attr_values**(Statement, Connection, Vector, int, Vector)
  Die Methode ins_attr_values fügt für die mit dem im Parameter attr_name enthaltenen Attributsnamen die im Parameter attr_values spezifizierten Attributswerte in die Attributstabellen mit der mit dem Parameter id übergebenen id ein
- **ins_mult**(Statement, Statement, Connection, String, String)
  Die Methode ins_mult fügt das mit dem Parameter o_name spezifizierte Objekt zur Domäne new_d_name hinzu
- **ins_obj**(Statement, Statement, Connection, String, String)
  Die Methode ins_obj fügt das mit dem Parameter o_name spezifizierte Objekt zur Domäne d_name ein
- **is_mult**(Statement, String)
  Die Methode is_mult liefert true, falls das mit o_name spezifizierte Objekte mehrfach vorkommt, d.h.
- **mov_obj**(Statement, Statement, Connection, String, String, String)
  Die Methode mov_obj entfernt das mit dem Parameter o_name spezifizierte Objekt von der bisherigen Domäne old_d_name und fügt es zur Domäne new_d_name ein
- **new_attr**(Statement, Connection, Vector)
  Die Methode new_attr legt die im Vektor attr_name angegebenen Attributsnamen mittels der Methode als neue Tabellen an
- **new_obj**(Statement, Connection, String, String)
  Die Methode new_obj fügt das mit o_name übergebene Objekt zur Domäne d_name ein und legt eine Tabelle für die Attribute des Objektes an, die die Spalten id, attr, did, odid, bzgl und bzglid besitzt
- **set_attr**(Statement, String, Vector, Vector)
  Die Methode set_attr ändert den Namen der im Vektor old_attr_name spezifizierten Attribute in die Namen der im Vektor new_attr_name angegebenen Namen um
- **set_attr_values**(Statement, String, Vector, Vector)
  Die Methode set_attr_values ändert die Werte der mit den Namen der im Vektor attr_name spezifizierten Attribute in die Werte der im Vektor attr_values angegebenen um

set_obj(Statement, String, String)
Die Methode set_obj ändert den Namen des mit dem Parameter old_o_name spezifizierten Objektes in den mit dem Parameter new_o_name angegebenen Namen

set_refer(Statement, String, String, String)
Die Methode set_refer fügt die Eigenschaft bezüglich in die Objekteigenschaft attr_name des Objektes o_name ein.

Variables

n

```
static final int n
```

Die Konstante n gibt die Vektorgröße an

Constructors

obj

```
public obj()
```

Methods

get_name

```
static String get_name(Statement st,
                       int id)
```

Die Methode get_name liefert den Namen des Objektes zu dem mit id übergebenen Parameter

Parameters:
st - der zur Datenbank sendende SQL-Befehl
id - die id des Objektes

exist_obj

```
static boolean exist_obj(Statement st,
                         String o_name)
```

Die Methode exist_obj prüft, ob es das mit o_name spezifizierte Objekt gibt

Parameters:
st - der zur Datenbank sendende SQL-Befehl
o_name - der Name des Objektes

get_id

```
static int get_id(Statement st,
                  String o_name)
```

Die Methode get_id liefert die id des mit o_name spezifizierten Objektes oder -1, falls es dieses nicht gibt

Parameters:
st - der zur Datenbank sendende SQL-Befehl
o_name - der Name des Objektes

del_obj

```
static boolean del_obj(Statement st,
                       String o_name)
```

Die Methode del_obj löscht das mit o_name spezifizierte Objekte aus dem Domänenbaum

Parameters:

st - der zur Datenbank sendende SQL-Befehl
o_name - der Name des Objektes

del_attr

```
static boolean del_attr(Statement st,
                        String o_name,
                        Vector attr_name)
```

Die Methode del_attr löscht die im Vektor attr_name angegebenen Attribute aus dem mit dem Parameter o_name spezifizierten Objekt

Parameters:

st - der zur Datenbank sendende SQL-Befehl
o_name - der Name des Objektes
attr_name - der Name des Vektors, der die Namen der zu löschenden Attribute enthält

get_attr

```
static Vector get_attr(Statement st,
                       String o_name)
```

Die Methode get_attr liefert die Attributsnamen des mit o_name spezifizierten Objektes

Parameters:

st - der zur Datenbank sendende SQL-Befehl
o_name - der Name des Objektes

get_obj

```
static Vector get_obj(Statement st,
                      String o_name)
```

Die Methode get_obj liefert die id, die Namen der Domänen, zu denen es gehört, Attribute und Attributwerte des mit o_name spezifizierten Objektes

Parameters:

st - der zur Datenbank sendende SQL-Befehl
o_name - der Name des Objektes

get_obj_from_dom

```
static Vector get_obj_from_dom(Statement st,
                               Statement st2,
                               String d_name)
```

Die Methode get_obj_from_dom liefert den Namen der Objekte zur der mit d_name spezifizierten Domäne

Parameters:

st - der zur Datenbank sendende SQL-Befehl
st2 - ein weiterer zur Datenbank sendender SQL-Befehl
d_name - der Name der Domäne

get_attr_values

```
static Vector get_attr_values(Statement st,
                              String o_name)
```

Die Methode get_attr_values liefert die Attributwerte des mit o_name spezifizierten Objektes zu allen Attributen

Parameters:

st - der zur Datenbank sendende SQL-Befehl
o_name - der Name des Objektes

get_mult

```
static Vector get_mult(Statement st,
                       String o_name)
```

Die Methode get_mult liefert zum mit o_name spezifizierten Objekt die ids der zugehörigen Domänen;

Parameters:

st - der zur Datenbank sendende SQL-Befehl
o_name - der Name des Objektes

is_mult

```
static boolean is_mult(Statement st,
                       String o_name)
```

Die Methode is_mult liefert true, falls das mit o_name spezifizierte Objekte mehrfach vorkommt, d.h. zu mehreren Domänen gehört

Parameters:

st - der zur Datenbank sendende SQL-Befehl
o_name - der Name des Objektes

del_mult

```
static boolean del_mult(Statement st,
                        Statement st2,
                        String o_name,
                        String d_name)
```

Die Methode del_mult löst das mit o_name spezifizierte Objekt von der Domäne d_name, falls das Objekt mehrfach vorkommt, d.h. zu mehreren Domänen gehört

Parameters:

st - der zur Datenbank sendende SQL-Befehl
st2 - ein weiterer zur Datenbank sendender SQL-Befehl
o_name - der Name des Objektes
d_name - der Name der Domäne

ins_attr_values

```
static boolean ins_attr_values(Statement st,
                               Connection verbindung,
                               String o_name,
                               Vector attr_name,
                               Vector attr_values)
```

Die Methode ins_attr_values fügt die mit dem im Parameter attr_name enthaltenen Attributsnamen die im Parameter attr_values spezifizierten Attributswerte in die Attributstabellen für das Objekt o_name ein

Parameters:

st - der zur Datenbank sendende SQL-Befehl
verbindung - die Verbindung zur Datenbank
o_name - der Name des Objekte
attr_name - der Name des Vektors, der die Attributsnamen enthält
attr_values - der Name des Vektors, der für jedes Attribut den einzufügenden Wert enthält

ins_attr_values

```
static boolean ins_attr_values(Statement st,
                               Connection verbindung,
                               Vector attr_name,
                               int id,
                               Vector attr_values)
```

Die Methode ins_attr_values fügt für die mit dem im Parameter attr_name enthaltenen Attributsnamen die im Parameter attr_values spezifizierten Attributswerte in die Attributstabellen mit der mit dem Parameter id übergebenen id ein

Parameters:

st - der zur Datenbank sendende SQL-Befehl
verbindung - die Verbindung zur Datenbank
attr_name - der Name des Vektors, der die Attributsnamen enthält

id - die id des Objektes
attr_values - der Name des Vektors, der für jedes Attribut den einzufügenden Wert enthält

ins_mult

```
static boolean ins_mult(Statement st,
                        Statement st2,
                        Connection verbindung,
                        String o_name,
                        String new_d_name)
```

Die Methode ins_mult fügt das mit dem Parameter o_name spezifizierte Objekt zur Domäne new_d_name hinzu

Parameters:
st - der zur Datenbank sendende SQL-Befehl
st2 - ein weiterer zur Datenbank sendender SQL-Befehl
verbindung - die Verbindung zur Datenbank
o_name - der Name des Objektes
new_d_name - der Name der Domäne

set_obj

```
static boolean set_obj(Statement st,
                       String old_o_name,
                       String new_o_name)
```

Die Methode set_obj ändert den Namen des mit dem Parameter old_o_name spezifizierten Objektes in den mit dem Parameter new_o_name angegebenen Namen

Parameters:
st - der zur Datenbank sendende SQL-Befehl
old_o_name - der alte Name des Objektes
new_o_name - der neue Name des Objektes

set_attr

```
static boolean set_attr(Statement st,
                        String o_name,
                        Vector old_attr_name,
                        Vector new_attr_name)
```

Die Methode set_attr ändert den Namen der im Vektor old_attr_name spezifizierten Attribute in die Namen der im Vektor new_attr_name angegebenen Namen um

Parameters:
st - der zur Datenbank sendende SQL-Befehl
o_name - der Name des Objektes
old_attr_name - der Name des Vektors, der die Namen der alten Attribute enthält
new_attr_name - der Name des Vektors, der die Namen der neuen Attribute enthält

set_attr_values

```
static boolean set_attr_values(Statement st,
                               String o_name,
                               Vector attr_name,
                               Vector attr_values)
```

Die Methode set_attr_values ändert die Werte der mit den Namen der im Vektor attr_name spezifizierten Attribute in die Werte der im Vektor attr_values angegebenen um

Parameters:
st - der zur Datenbank sendende SQL-Befehl
o_name - der Name des Objektes
attr_name - der Name des Vektors, der die Namen der Attribute enthält
attr_values - der Name des Vektors, der die neuen Attributwerte enthält

ins_obj

```
static boolean ins_obj(Statement st,
                       Statement st2,
                       Connection verbindung,
                       String o_name,
                       String d_name)
```

Die Methode ins_obj fügt das mit dem Parameter o_name spezifizierte Objekt zur Domäne d_name ein

Parameters:

st - der zur Datenbank sendende SQL-Befehl
st2 - ein weiterer zur Datenbank sendender SQL-Befehl
verbindung - die Verbindung zur Datenbank
o_name - der Name des Objektes
d_name - der Name der Domäne

ins_attr

```
static boolean ins_attr(Statement st,
                        Connection verbindung,
                        String o_name,
                        String d_name,
                        Vector attr_name,
                        Vector attr_bzgl)
```

Die Methode ins_attr fügt die Attribute und, ob die Eigenschaft bzgl. einer anderen Domäne bzw. Objektes gilt, in die mit dem Parameter o_name spezifizierte Objekttabelle ein

Parameters:

st - der zur Datenbank sendende SQL-Befehl
verbindung - die Verbindung zur Datenbank
o_name - der Name des Objektes
d_name - der Name der Domäne
attr_name - der Name des Vektors, der die Attributsnamen enthält
attr_bzgl - der Name des Vektors, der für jedes Attribut einen leeren String, den Namen des Objektes oder den Namen der Domäne enthält bzgl. dieser das Attribut gilt

ins_attr

```
static boolean ins_attr(Statement st,
                        Statement st2,
                        Statement st3,
                        Connection verbindung,
                        String o_name,
                        String d_name,
                        Vector attr_name,
                        Vector attr_bzgl,
                        int ober_did)
```

Die Methode ins_attr fügt die Attribute und ihre Eigenschaften, ob die Eigenschaft bzgl. einer anderen Domäne bzw. Objektes gilt, in die mit dem Parameter o_name spezifizierte Objekttabelle ein

Parameters:

st - der zur Datenbank sendende SQL-Befehl
st2 - ein weiterer zur Datenbank sendender SQL-Befehl
st3 - ein weiterer zur Datenbank sendender SQL-Befehl
verbindung - die Verbindung zur Datenbank
o_name - der Name des Objektes
d_name - der Name der Domäne
attr_name - der Name des Vektors, der die Attributsnamen enthält
attr_bzgl - der Name des Vektors, der für jedes Attribut einen leeren String, den Namen des Objektes oder den Namen der Domäne enthält bzgl. dieser das Attribut gilt

ober_did - die id der Oberdomäne

new_attr

```
static boolean new_attr(Statement st,
                        Connection verbindung,
                        Vector attr_name)
```

Die Methode new_attr legt die im Vektor attr_name angegebenen Attributsnamen mittels der Methode als neue Tabellen an

Parameters:

st - der zur Datenbank sendende SQL-Befehl
verbindung - die Verbindung zur Datenbank
attr_name - der Name des Vektors, der die Attributsnamen enthält

mov_obj

```
static boolean mov_obj(Statement st,
                       Statement st2,
                       Connection verbindung,
                       String o_name,
                       String old_d_name,
                       String new_d_name)
```

Die Methode mov_obj entfernt das mit dem Parameter o_name spezifizierte Objekt von der bisherigen Domäne old_d_name und fügt es zur Domäne new_d_name ein

Parameters:

st - der zur Datenbank sendende SQL-Befehl
st2 - ein weiterer zur Datenbank sendender SQL-Befehl
verbindung - die Verbindung zur Datenbank
o_name - der Name des Objektes
old_d_name - der Name der alten Domäne
new_d_name - der Name der neuen Domäne

new_obj

```
static boolean new_obj(Statement st,
                       Connection verbindung,
                       String o_name,
                       String d_name)
```

Die Methode new_obj fügt das mit o_name übergebene Objekt zur Domäne d_name ein und legt eine Tabelle für die Attribute des Objektes an, die die Spalten id, attr, did, odid, bzgl und bzglid besitzt

Parameters:

st - der zur Datenbank sendende SQL-Befehl
verbindung - die Verbindung zur Datenbank
o_name - der Name des Objektes
d_name - der Name der Domäne

get_refer

```
static String get_refer(Statement st,
                        String o_name,
                        String attr_name)
```

Die Methode get_refer liefert den Namen des Objektes oder der Domäne bezüglich der mit dem Parameter attr_name angegebenen Objekteigenschaft

Parameters:

st - der zur Datenbank sendende SQL-Befehl
o_name - der Name des Objektes
attr_name - der Name der Eigenschaft

del_refer

```
static boolean del_refer(Statement st,
                         String o_name,
                         String attr_name)
```

Die Methode del_refer löscht die Eigenschaft bezüglich bei der Objekteigenschaft attr_name des Objektes o_name

Parameters:

st - der zur Datenbank sendende SQL-Befehl
o_name - der Name der Objektes
attr_name - der Name der Eigenschaft

set_refer

```
static boolean set_refer(Statement st,
                         String o_name,
                         String attr_name,
                         String refer_name)
```

Die Methode set_refer fügt die Eigenschaft bezüglich in die Objekteigenschaft attr_name des Objektes o_name ein. Der Parameter refer_name enthält das Objekt oder die Domäne, bezüglich der die Objekteigenschaft gelten soll

Parameters:

st - der zur Datenbank sendende SQL-Befehl
o_name - der Name des Objektes
attr_name - der Name der Eigenschaft
refer_name - der Name des Objektes oder der Domäne bezüglich der die Objekteigenschaft gelten soll

Class attr

```
java.lang.Object
   |
   +----attr
```

---

public class **attr**
extends Object Die Klasse attr verfügt über alle Informationen, die für Attribute benötigt werden

---

Constructor Index

attr()

Method Index

del_attr(Statement, String, int)
Die Methode del_attr löscht aus einer Attributstabelle den mit dem Parameter id spezifizierten Wert des Attributs

del_attr(Statement, String, String)
Die Methode del_attr löscht aus einer Attributstabelle den mit dem Parameter a_value spezifizierten Wert des Attributs

exist_table(Statement, String)
Die Methode exist_table prüft, ob es die mit dem Parameter table_name spezifizierte Tabelle gibt

get_attr(Statement, String, int)
Die Methode get_attr ruft die Methode get_attr_value auf, die den mit dem Parameter id spezifizierten Wert des Attributs liefert

get_attr_value(Statement, String, int)
Die Methode get_attr_value liefert aus einer Attributstabelle den mit dem Parameter id spezifizierten Wert des Attributs

ins_attr(Statement, Connection, String, String, int)
Die Methode ins_attr fügt in eine Attributstabelle den mit dem Parameter a_value übergebenen Wert des Attributs und die id ein

is_empty(Statement, String)
Die Methode is_empty prüft, ob die mit dem Parameter table_name spezifizierte Tabelle leer ist

new_attr(Statement, String)
Die Methode new_attr legt eine Tabelle für ein Attribut an.

set_attr(Statement, String, String, int)
Die Methode set_attr ruft die Methode set_attr_value auf, die den mit dem Parameter id spezifizierten Wert des Attributs in den mit dem Parameter a_value übergebenen Wert ändert

set_attr_value(Statement, String, String, int)
Die Methode set_attr_value ändert den mit dem Parameter id spezifizierten Wert des Attributs in den mit dem Parameter a_value übergebenen Wert

Constructors

attr

```
public attr()
```

Methods

exist_table

```
static boolean exist_table(Statement st,
                           String table_name)
```

Die Methode exist_table prüft, ob es die mit dem Parameter table_name spezifizierte Tabelle gibt

Parameters:

st - der zur Datenbank sendende SQL-Befehl
table_name - der Name der Tabelle

is_empty

```
static boolean is_empty(Statement st,
                        String table_name)
```

Die Methode is_empty prüft, ob die mit dem Parameter table_name spezifizierte Tabelle leer ist

Parameters:

st - der zur Datenbank sendende SQL-Befehl
table_name - der Name der Tabelle

new_attr

```
static boolean new_attr(Statement st,
                        String a_name)
```

Die Methode new_attr legt eine Tabelle für ein Attribut an. Diese Tabelle besteht dann aus den zwei Spalten id und der mit dem Tabellennamen übereinstimmenden Spalte

Parameters:

st - der zur Datenbank sendende SQL-Befehl
a_name - der Name der Tabelle

ins_attr

```
static boolean ins_attr(Statement st,
                        Connection verbindung,
                        String a_name,
                        String a_value,
                        int id)
```

Die Methode ins_attr fügt in eine Attributstabelle den mit dem Parameter a_value übergebenen Wert des Attributs und die id ein

Parameters:

st - der zur Datenbank sendende SQL-Befehl
verbindung - die Verbindung zur Datenbank
a_name - der Name der Tabelle
a_value - der Wert des Attributs
id - die id des Objektes oder der Domäne

del_attr

```
static boolean del_attr(Statement st,
                        String a_name,
                        int id)
```

Die Methode del_attr löscht aus einer Attributstabelle den mit dem Parameter id spezifizierten Wert des Attributs

Parameters:

st - der zur Datenbank sendende SQL-Befehl
a_name - der Name der Tabelle
id - die id des Objektes oder der Domäne

del_attr

```
static boolean del_attr(Statement st,
                        String a_name,
                        String a_value)
```

Die Methode del_attr löscht aus einer Attributstabelle den mit dem Parameter a_value spezifizierten Wert des Attributs

Parameters:

st - der zur Datenbank sendende SQL-Befehl
a_name - der Name der Tabelle
a_value - der Wert des Attributs

get_attr

```
static String get_attr(Statement st,
                       String a_name,
                       int id)
```

Die Methode get_attr ruft die Methode get_attr_value auf, die den mit dem Parameter id spezifizierten Wert des Attributs liefert

Parameters:

st - der zur Datenbank sendende SQL-Befehl
a_name - der Name der Tabelle
id - die id des Objektes oder der Domäne

get_attr_value

```
static String get_attr_value(Statement st,
                             String a_name,
                             int id)
```

Die Methode get_attr_value liefert aus einer Attributstabelle den mit dem Parameter id spezifizierten Wert des Attributs

Parameters:

st - der zur Datenbank sendende SQL-Befehl
a_name - der Name der Tabelle
id - die id des Objektes oder der Domäne

set_attr

```
static boolean set_attr(Statement st,
                        String a_name,
                        String a_value,
                        int id)
```

Die Methode set_attr ruft die Methode set_attr_value auf, die den mit dem Parameter id spezifizierten Wert des Attributs in den mit dem Parameter a_value übergebenen Wert ändert

Parameters:

st - der zur Datenbank sendende SQL-Befehl
a_name - der Name der Tabelle
a_value - der neue Wert des Attributs
id - die id des Objektes oder der Domäne

set_attr_value

```
static boolean set_attr_value(Statement st,
                              String a_name,
                              String a_value,
                              int id)
```

Die Methode set_attr_value ändert den mit dem Parameter id spezifizierten Wert des Attributs in den mit dem Parameter a_value übergebenen Wert

Parameters:

st - der zur Datenbank sendende SQL-Befehl
a_name - der Name der Tabelle
a_value - der neue Wert des Attributs
id - die id des Objektes oder der Domäne

Class db

```
java.lang.Object
   |
   +----db
```

---

public class **db**
extends Object Die Klasse db enthält Methoden zur Kommunikation mit der Datenbank

---

Variable Index

katalog
Die Konstante katalog enthält den Namen des Datenbankkatalogs
treibername
Die Konstante treibername enthält den Namen des Datenbanktreibers
url
Die Konstante url enthält die Adresse für die Datenbankverbindung

Constructor Index

db()

Method Index

alle_tabellen_loeschen(Connection)
Die Methode alle_tabellen_loeschen entlädt und löscht alle Tabellen der mit dem Parameter spezifizierten Datenbank
befehl_anlegen(Connection)
Die Methode befehl_anlegen legt einen Datenbankbefehl an.
initialisierung(Connection)
Die Methode initialisierung erzeugt alle für den Prototyp benötigten Tabellen
katalog_setzen(Connection)
Die Methode katalog_setzen setzt den Katalog auf den Inhalt der Konstanten katalog und gibt den Namen aus.
tabelle_anlegen(int, Statement, String, String)
Die Methode tabelle_anlegen erzeugt eine neue Tabelle in der Datenbank.
tabelle_entladen(int, Statement, String)
Die Methode tabelle_entladen entlädt eine bereits existierende Tabelle, sonst wird eine Fehlermeldung ausgegeben
tabelle_loeschen(int, Statement, String)
Die Methode tabelle_loeschen löscht eine bereits existierende Tabelle, sonst wird eine Fehlermeldung ausgegeben
treiber_laden()
Die Methode treiber_laden lädt den Datenbank-Treiber, der in der Konstanten treibername steht.
verbindung_oeffnen()
Die Methode verbindung_oeffnen öffnet die Verbindung zur Datenbank.
verbindung_schliessen(Connection)

Die Methode verbindung_schliessen schließt die Datenbankverbindung.

Variables

treibername

```
static final String treibername
```

Die Konstante treibername enthält den Namen des Datenbanktreibers

url

```
static final String url
```

Die Konstante url enthält die Adresse für die Datenbankverbindung

katalog

```
static final String katalog
```

Die Konstante katalog enthält den Namen des Datenbankkatalogs

Constructors

db

```
public db()
```

Methods

treiber_laden

```
static void treiber_laden()
```

Die Methode treiber_laden lädt den Datenbank-Treiber, der in der Konstanten treibername steht. Beim Auftreten einer Ausnahme wird diese abgefangen und ausgegeben

verbindung_oeffnen

```
static Connection verbindung_oeffnen()
```

Die Methode verbindung_oeffnen öffnet die Verbindung zur Datenbank. Dazu wird der Inhalt der Konstanten url benutzt. Beim Auftreten einer Ausnahme wird diese abgefangen und ausgegeben

Returns:

die Verbindung zur gewünschten Datenbank

katalog_setzen

```
static void katalog_setzen(Connection verbindung)
```

Die Methode katalog_setzen setzt den Katalog auf den Inhalt der Konstanten katalog und gibt den Namen aus. Beim Auftreten einer Ausnahme wird diese abgefangen und ausgegeben

Parameters:

verbindung - die Verbindung zur gewünschten Datenbank

befehl_anlegen

```
static Statement befehl_anlegen(Connection verbindung)
```

Die Methode befehl_anlegen legt einen Datenbankbefehl an. Beim Auftreten einer Ausnahme wird diese abgefangen und ausgegeben

Parameters:

verbindung - die Verbindung zur gewünschten Datenbank

Returns:

den initialisierten Befehl für die Datenbank

tabelle_anlegen

```
static void tabelle_anlegen(int xrc,
                            Statement st,
                            String tabname,
                            String tabparams)
```

Die Methode tabelle_anlegen erzeugt eine neue Tabelle in der Datenbank. Beim Auftregen einer Ausnahme wird diese abgefangen und ausgegeben

Parameters:

xrc - wird bei jedem erfolgreichen Befehl ausgegeben
st - der initialisierte Datenbankbefehl
tabname - der Name der zu erzeugenden Tabelle
tabparams - die Anzahl und Namen der zu erzeugenden Spalten

tabelle_entladen

```
static void tabelle_entladen(int xrc,
                             Statement st,
                             String tabname)
```

Die Methode tabelle_entladen entlädt eine bereits existierende Tabelle, sonst wird eine Fehlermeldung ausgegeben

Parameters:

xrc - wird bei jedem erfolgreichen Befehl ausgegeben
st - der initialisierte Datenbankbefehl
tabname - der Name der zu erzeugenden Tabelle

tabelle_loeschen

```
static void tabelle_loeschen(int xrc,
                             Statement st,
                             String tabname)
```

Die Methode tabelle_loeschen löscht eine bereits existierende Tabelle, sonst wird eine Fehlermeldung ausgegeben

Parameters:

xrc - wird bei jedem erfolgreichen Befehl ausgegeben
st - der initialisierte Datenbankbefehl
tabname - der Name der zu erzeugenden Tabelle

verbindung_schliessen

```
static void verbindung_schliessen(Connection verbindung)
```

Die Methode verbindung_schliessen schließt die Datenbankverbindung. Beim Auftreten einer Ausnahme wird diese abgefangen und ausgegeben

Parameters:

verbindung - gibt die gewünschte Datenbank an

initialisierung

```
static void initialisierung(Connection verbindung)
```

Die Methode initialisierung erzeugt alle für den Prototyp benötigten Tabellen

Parameters:

verbindung - gibt die gewünschte Datenbankverbindung an

alle_tabellen_loeschen

```
static void alle_tabellen_loeschen(Connection verbindung)
```

Die Methode alle_tabellen_loeschen entlädt und löscht alle Tabellen der mit dem Parameter spezifizierten Datenbank

Parameters:

verbindung - spezifiziert die gewünschte Datenbank