TECHNISCHE
UNIVERSITÄT
MÜNCHEN

INSTITUT FÜR INFORMATIK

Diplomarbeit

Integration des Managements des Domain-Name-Systems in die vorhandene Managementumgebung des Leibniz-Rechenzentrums (LRZ)

Heiko Hauck

TECHNISCHE
UNIVERSITÄT
MÜNCHEN

INSTITUT FÜR INFORMATIK

Diplomarbeit

Integration des Managements des Domain-Name-Systems in die vorhandene Managementumgebung des Leibniz-Rechenzentrums (LRZ)

Bearbeiter:	Heiko Hauck
Aufgabensteller:	Prof. Dr. Heinz-Gerd Hegering
Betreuer:	Dipl. Inform. Kirsten Heiler
	Dr. Robert Valta
Abgabetermin:	15. November 1994

Erklärung

Hiermit erkläre ich, daß ich die vorliegende Arbeit selbständig angefertigt und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 9. November 1994

____

(Heiko Hauck)

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

Nach dem Bau des ersten Personalcomputers in den 70er Jahren kam es im darauffolgenden Jahrzehnt zum allgemeinen Durchbruch des Mikrocomputers. Seine Vertreter waren kaum noch aus Büros, Labors, Banken, aber auch privaten Wohnzimmern mehr wegzudenken.

Was diesen Systemen am Anfang aber weitgehend noch fehlte, war die Fähigkeit, miteinander zu kommunizieren. Es handelte sich oft um isolierte Systeme, bei denen die Daten von Hand eingegeben oder mittels zumeist magnetischer Datenträger geladen werden mußten. Dadurch war ein manuelles Eingreifen seitens des Anwenders bei Kommunikationsvorgängen erforderlich, was einen der wesentlichen Vorteile der elektronischen Datenverarbeitung, die (vollständige) Automatisierung von Arbeitsvorgängen, zwar nicht zunichte machte, aber dennoch deutlich einschränkte.

Um einen Datenaustausch zwischen Rechnern ohne umständlichen und langwierigen Transport eines phyischen Mediums (Band, Diskette, ...) zu ermöglichen, wurden Kommunikationsnetze geschaffen. Deren Bedeutung hat in den vergangenen Jahren enorm zugenommen. Auch wenn derzeit noch zwischen lokalen (LAN) und Weitverkehrsnetzen (WAN) unterschieden wird, so tragen moderne Hochgeschwindigkeitstechnologien wie ATM (Asynchronous Transfer Mode) zu deren Vermengung bei. Damit wird es möglich, die in letzter Zeit viel zitierten „Datenautobahnen“ zu realisieren, mit denen viele heute erst begrenzt möglichen Multimedia-Anwendungen der breiten Masse zugänglich gemacht werden sollen, da hierfür eine enorm hohe Bandbreite erforderlich ist.

Auf dem Weg dorthin müssen wir uns aber noch mit den Problemen existierender und erprobter Netze auseinandersetzen, um dann mit der daraus gewonnenen

Erfahrung Neues schaffen zu können. Zu den genannten Netzen gehört als einer der bekanntesten WAN-Vertreter das ARPA-Internet, das ursprünglich auf ein Forschungsprojekt des amerikanischen Verteidigungsministeriums (DoD) zurückgeht, sich in den vergangenen Jahren aber stark als Forschungs- und Hochschulnetz etablierte und in jüngster Zeit auch immer mehr für kommerzielle Zwecke genutzt wird.

Die vorliegende Arbeit beschäftigt sich mit dem Domain Name System (DNS), einer Anwendung auf dem Internet. Dabei handelt es sich um ein System, mit dessen Hilfe jedem Rechner im gesamten Netz ein Name zugeordnet wird, durch den er global eindeutig identifiziert werden kann. Damit wird die Verwendung umständlicher Adressen (Zahlen) überflüssig.

Derartige verteilte Anwendungen – das Domain Name System wird durch Rechner realisiert, die im ganzen Netz verteilt sind – sind aber sehr komplex und damit fehlerträchtig. Deshalb müssen Möglichkeiten geschaffen werden, das System „in den Griff" zu bekommen, d.h. das System muß gemanaget werden.

Für das Management bieten sich einerseits isolierte, proprietäre und auf der anderen Seite integrierte Verfahren an. Während erstere öfters verfügbar und billiger sind, bietet integriertes Management weitaus mehr Sicherheit[1] und Effizienz und ist deshalb vorzuziehen. Um letzteres zu realisieren, bieten sich grundsätzlich zwei verschiedene Wege an:

- Anbindung isolierter Werkzeuge und Verfahren an eine Plattform für integriertes Management. Hierzu sind Module zu entwickeln, die auf dem vorhandenen Werkzeug aufsetzen und eine Schnittstelle zur Plattform schaffen.
- Entwicklung neuer Werkzeuge und Verfahren, die direkt in die Managementplattform eingebunden werden. Dabei muß die Schnittstelle des zu managenden Objektes berücksichtigt werden, deren proprietäre Spezifikationen dem Manager an der Plattform aber nach Möglichkeit verborgen bleiben sollen, sofern er sie nicht für Spezialzwecke benötigt.

Die letztgenannte Forderung läßt sich u.a. mit Hilfe eines Dokumentationssystems in Form einer Netzdokumentationsdatenbank erfüllen, die die Informationen allgemein und unabhängig von bestimmten Spezifikationen enthält, die manche Komponenten oder Anwendungen mitbringen. Dazu ist eine Anbindung der Plattform bzw. des (darin enthaltenen) Werkzeuges an die Datenbank erforderlich.

Am Leibniz-Rechenzentrum ist eine Netzdokumentationsdatenbank als Bestandteil der lokalen Managementumgebung installiert. Die Aufgabe der vorliegenden

[1] Sicherheit in Bezug auf Korrektheit und Konsistenz aller Managementaufgaben.

Arbeit liegt nun darin, ein Verfahren zu entwickeln, das ein Werkzeug enthält, durch das das Domain Name System an die genannte Datenbank angebunden wird, und so sein Management ermöglicht wird. Davon ist in erster Linie das Konfigurationsmanagement betroffen, da die Datenbank u.a. sämtliche zur Konfiguration notwendigen Informationen bereitstellen soll.

1.2 Vorgehensweise

In Kapitel 2 wird zunächst eine allgemeine Einführung in das Domain Name System, den Gegenstand dieser Arbeit, gegeben. Dabei wird gezeigt, welches Problem zu lösen war, das letztlich zur Einführung des DNS führte, wie dieses vor und nach dessen Einführung gelöst wurde, und welche Vorteile das DNS mit sich brachte.

Mit dem anschließenden 3. Kapitel wird eine Beschreibung der Konfiguration des Domain Name Systems eingeschoben. Neben einer systematischen Unterteilung der gesamten vom DNS zur Verfügung gestellten Information wird anhand von kleinen Beispielkonfigurationsdateien aufgezeigt, wie man eine eigene DNS-Umgebung zum Laufen bringt.

Auch wenn deren Spezifikationen dem Netzverwalter letzlich verborgen bleiben soll (vgl. Abschnitt 1.1), so ist deren Kenntnis für die vorliegende Arbeit unerläßlich, um das neu zu implementierende Datenbankschema und Werkzeug verstehen zu können. Die Dateien stellen die systemnäheste vom Netzverwalter beeinflußbare Schicht dar.

Mit dem 4. Kapitel wird diesbezüglich ein Schnitt gemacht. Nach der Bedeutung des Managements allgemein und speziell für das Domain Name System wird dessen Management aus Betreibersicht, d.h. in diesem Falle aus der Sicht des DNS-Administrators, präsentiert. Neben seinen Aufgaben und den damit verbundenen Anforderungen werden Möglichkeiten aufgezeigt, die Aufgaben umzusetzen, die allerdings isolierter Natur sind.

Deren Nachteile führen zum integrierten Management. Hierfür werden zwei richtungsweisende Ansätze präsentiert, einer aus dem Bereich des Internet-Managements und ein anderer mit Hilfe eines Dokumentationssystems, das am Leibniz-Rechenzentrum u.a. als Netzdokumentationsdatenbank realisiert ist. Der letztgenannte Ansatz wird bzgl. des Konfigurationsmanagements in der Arbeit weiterverfolgt, weshalb am Schluß dieses Kapitels noch Anforderungen, die sich damit an die Managementumgebung stellen, genannt werden.

Die Umsetzung dieses Verfahrens ist Gegenstand des 5. Kapitels. Da bereits eine Netzdokumentationsdatenbank vorhanden ist, in der neuartige Informationen abzulegen sind, muß man sich zunächst mit der vorhandenen auseinandersetzen.

Unabhängig davon werden die Anforderungen an das Design der neuen (zusätzlichen) Datenbank gestellt, bevor unter deren Berücksichtigung das Design selbst entworfen wird. Anschließend wird versucht, die sich ergebenden neuen Relationen in die alten zu integrieren. Schließlich gilt es noch einige organisatorische Probleme zu lösen, bevor das Verfahren eingesetzt werden kann.

Das 6. Kapitel beschreibt schließlich die Implementierung des Verfahrens. Hierzu zählt zum einen das exakte Relationenschema mit einer Beschreibung sämtlicher Attribute und zum anderen das Werkzeug, an das wiederum eigene Anforderungen gestellt werden.

Das Werkzeug wird in Form eines Perl-Skripts realisiert. Bei der Programmbeschreibung wird zunächst eine Übersicht präsentiert, bevor auf die einzelnen Module, aus denen es sich zusammensetzt, eingegangen wird. Ein Beispiel zur Demonstration der Ein- und Ausgabe des Programms rundet das Kapitel ab.

Das 7. Kapitel gibt schließlich einen Ausblick über einen Teil von dem, was noch zu tun ist. Dazu zählen zum einen mögliche Erweiterungen und Verbesserungen des implementierten Verfahrens und zum anderen dessen Einbindung in eine Managementplattform.

Kapitel 2

Einführung in das Domain Name System

Dieses Kapitel beschäftigt sich mit dem Domain Name System, dessen Management in dieser Arbeit in eine vorhandene Mangementumgebung integriert werden soll. Dabei handelt es sich um eine Anwendung aus der Internet-Welt. Andere bekannte Anwendungen sind z.B. NIS (Network Information Service) oder NFS (Network File System).

Zunächst wird in Abschnitt 2.1 ein Problem vorgestellt, das allgemein im Zusammenhang mit Kommunikationsnetzen auftritt. Dabei handelt es sich um die Identifizierung der einzelnen teilnehmenden Komponenten. Während Maschinen mit Zahlen arbeiten, ist es für den Menschen leichter, sich Namen zu merken.

Ein Lösungsansatz zu diesem Problem, wie er vor Einführung des Domain Name Systems zum Einsatz kam, wird in Abschnitt 2.2 präsentiert. Dabei werden auch die Probleme dieses Verfahrens aufgezeigt, aufgrund deren eine andere Lösung gefunden werden mußte.

Hierbei handelt es sich um das Domain Name System. Abschnitt 2.3 gibt einen Überblick über die damit einhergehende Informationsverteilung, den hierarchischen Namensraum sowie die Art und Weise, wie die zur Verfügung gestellten Informationen zu den jeweilgen Anwendungen gelangen.

2.1 Motivation

Komponenten eines elektronischen Kommunikationsnetzes werden durch einen numerischen Wert identifiziert und adressiert. Das hierfür allgemein bekannteste Beispiel ist das Telefonnetz der Telekom. Hier erfolgt die Auswahl des Partnertelefons durch Eingabe von dessen Telefonnummer.

Auch Computernetze kennen Adressen, die je nach verwendetem Protokoll unterschiedlich ausfallen. So verwendet zum Beispiel das im LAN-Bereich sehr verbreitete Ethernet eine 48Bit-Adresse, die von den Herstellern der Adapterkarten bei deren Produktion eindeutig festgelegt wird. Damit ist dann sichergestellt, daß unter ein und derselben Adresse nur ein einziger Rechner erreichbar ist, womit Verwechslungen ausgeschlossen werden.

Im ARPA-Internet vernetzte Rechner verwenden das IP-Protokoll. Die zugehörigen IP-Adressen haben einen Umfang von 32 Bit und werden in der Regel als durch Punkte getrennte dezimale Bytewerte dargestellt. So besitzt zum Beispiel der Rechner, auf dem gerade dieser Text eingegeben wird, die IP-Adresse **129.187.10.20**.

Numerische Adressen werden eingesetzt, da sie vom Computer sehr einfach verarbeitet werden können. Dagegen sind sie für den Benutzer nur schwer zu merken, da der Mensch im allgemeinen ein besseres Namens- wie Zahlengedächtnis besitzt. Um den Bedürfnissen des Anwenders entgegenzukommen, hat man sich deswegen entschlossen, für Rechner auch alphanumerische Namen einzuführen (Beispiel: **sun1**).

Damit ein solcher Service angeboten werden kann, ist ein Mechanismus notwendig, der die Namen auf die numerischen Adressen abbildet. Ein solches Verfahren wird als Namensauflösung (Address Resolution) bezeichnet, die zugehörigen Routinen im Betriebssystem der Rechner als Resolver.

Ein weiterer Vorteil dieser Abbildung besteht darin, daß die Namen einzelner Rechner erhalten bleiben können, wenn sich deren IP-Adressen ändern (beispielsweise beim Umzug eines Rechners in ein anderes Subnetz).

2.2 Früheres Verfahren

Um die Address Resolution durchführen zu können, wurde eine Datei namens *HOSTS.TXT* kreiert [Sant 93], die zeilenweise die Abbildung der Namen auf die IP-Adressen enthielt. Diese Datei wurde zentral verwaltet auf einem Rechner des Network Information Centers (NIC) und mußte auf jedem Rechner vorhanden sein, auf dem unter Verwendung von Namen anstatt von IP-Adressen eine Verbindung zu einem anderen Rechner aufgebaut werden sollte. Der netzweite Verteilungsprozeß wurde mit Hilfe des File Transfer Protocols (FTP) durchgeführt (siehe Abbildung 2.1).

Lokal wurde die Datei (bei UNIX-Systemen) als */etc/hosts* abgelegt. Damit konnten dann die bereits erwähnten Resolverroutinen die notwendige Abbildung vornehmen.

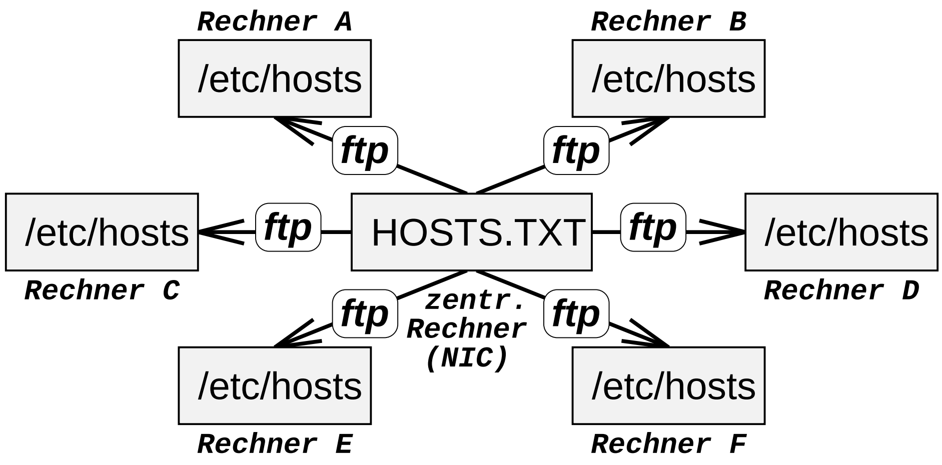

Abbildung 2.1: Verteilung von *HOSTS.TXT*

Dieses Verfahren hatte aber entscheidende Nachteile [Geor 93]:

- *Hohe Netzlast durch Verteilungsprozeß*
  Durch den Verteilungsprozeß wurde das Netz beträchtlich belastet. Als besonders ungünstig war die Tatsache zu werten, daß die Netzlast mit quadratischer Komplexität $O(n^2)$ in Bezug auf die Zahl der Rechner stieg, da bei einer Verdoppelung der Anzahl der Rechner die Datei nicht nur doppelt so oft über das Netz geschickt werden mußte, sondern sich auch ihr Umfang verdoppelte.

- *Langsame Registrierung von Modifikationen*
  Alle Daten wurden bei diesem Verfahren zentral in einer einzigen Datei verwaltet. Das bedeutete, daß jede Änderung dort gemeldet und verarbeitet werden mußte und erst nach einer erneuten Verteilung von *HOSTS.TXT* im ganzen Netz bekannt wurde. Betrachtet man die große Zahl der Rechner im Internet (im Jahre 1993 ca. 2 Millionen), so läßt sich daraus leicht ableiten, daß der notwendige Verwaltungsaufwand heute nicht mehr in angemessener Zeit erledigt werden könnte.

- *Kollisionen durch flachen Namensraum*
  Das Verfahren sah eine Abbildung: Rechnername $\rightarrow$ IP-Adresse vor. Das bedeutete, daß der Rechnername eindeutig sein mußte, also kein Name doppelt oder mehrfach auftauchen durfte. Diesbezügliche Abstimmungsprobleme konnten in der Praxis nicht vollständig ausgeräumt werden.

2.3 Das Domain Name System

Um den oben genannten Problemen zu entgegnen, wurde im Jahre 1984 das Domain Name System (DNS) eingeführt ([RFC 881], [RFC 882] und [RFC 883]). Dabei handelt es sich um einen Verzeichnisdienst, bei dem die Informationen nicht zentral, sondern verteilt gespeichert werden, mit einem hierarchischen statt flachen Namensraum.

Auf diese Punkte sowie auf das Verfahren, wie diese Information schließlich dort verfügbar gemacht wird, wo sie benötigt wird (Anwendung), wird in den folgenden Abschnitten eingegangen. Dabei wird auch der sehr wichtige Begriff des Nameservers erläutert.

2.3.1 Verteilung der Information

Beim DNS liegt ein über das ganze Netz verteiltes Datenbanksystem vor. Die Information, welcher Name auf welche Adresse abzubilden ist, wird (in der Regel) von derjenigen Organisation verwaltet und den anderen zur Verfügung gestellt, bei der sie auch tatsächlich anfällt. Dadurch wird das zweite Problem, die langsame Registrierung von Änderungen der DNS-Daten, gelöst.

Beispielsweise verwaltet das Leibniz-Rechenzentrum (LRZ) sämtliche DNS-relevanten Informationen bezüglich der eigenen Rechner selbst. Daneben übernimmt das LRZ aber auch diesen Dienst für einen Teil der Bayerischen Hochschulen. Es ist also möglich, diese Aufgabe an andere zu übertragen, wenn man sich selbst nicht damit beschäftigen kann bzw. will.

2.3.2 Hierarchischer Namensraum

Das Problem der Namenskollsionen läßt sich dadurch eliminieren, indem die Rechnernamen nicht flach, sondern in einer Baumstruktur hierachisch angeordnet werden (Teilbaum siehe Abbildung 2.2).

Die Namen setzen sich aus dem Hostnamen, also dem Namen des jeweiligen Rechners (zum Beispiel: **sun1**), und dem Domainnamen, der angibt, zu welcher Institution (Bereich) der Rechner gehört (im Beispiel: **lrz-muenchen.de**), zusammen. Die Kombination von beidem wird als Fully Qualified Domain Name (FQDN) bezeichnet (im Beispiel also: **sun1.lrz-muenchen.de**).

Die Domainnamen setzen sich aus den Namen der Knoten im Baum zusammen. Der Domainname eines Knotens besteht aus dessen Name sowie den Namen der im selben Pfad darüberliegenden Knoten – jeweils durch einen Punkt getrennt (Beispiel: **nm.informatik.uni-muenchen.de**). Der Wurzelknoten stellt

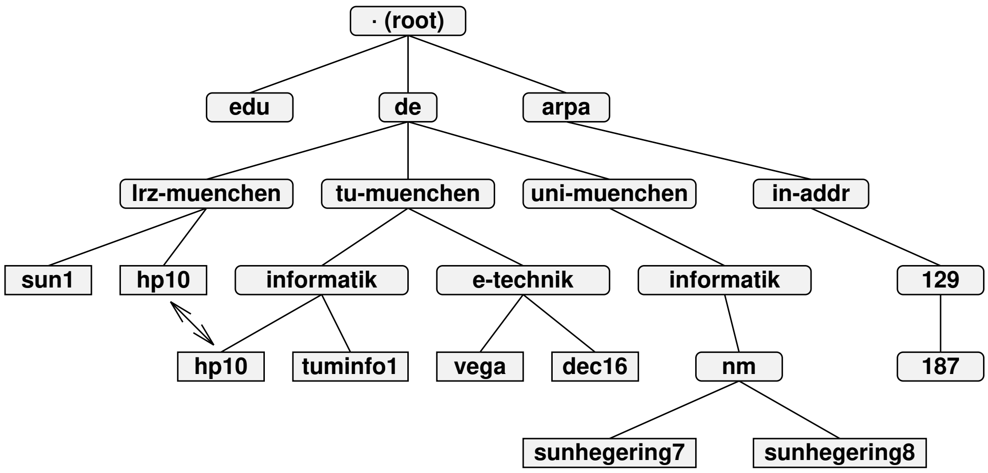

Abbildung 2.2: Hierarchischer Namensraum (Teilbaum)

die Root-Domain **(.)** dar, die darunterliegenden heißen Top-Level-Domains **(edu, de, ...)**, die der nächstniedrigeren Schicht entsprechend Second-Level-Domains **(lrz-muenchen.de, tu-muenchen.de, ...)**. Modalitäten zur Namensvergabe sind in [RFC 1032] beschrieben.

Alle Teilbäume eines Knotens bilden zusammen mit diesem eine Domain, der die darunterliegenden Subdomains und Rechner angehören. Zur Informationaufteilung (verteiltes Datenbanksystem) wird der Baum in Zonen eingeteilt, die von den einzelnen Institutionen in Eigenverantwortung verwaltet werden. Eine Zone umfaßt eine (Sub-)Domain abzüglich der delegierten (in einen anderen Verantworungsbereich übertragenen) Subdomains, die wiederum einer eigenen Zone angehören.

Die Blätter im Baum stellen die Rechner mit ihren Namen, also den Hostnamen dar. Darin ist dann auch die entsprechende IP-Adresse (gegebenenfalls auch mehrere) gespeichert. Wie am Beispiel des Hostnamen **hp10** zu sehen ist, ist es jetzt auch möglich, ein und denselben Hostnamen[1] mehrmals zu verwenden, wenn der Rechner einer anderen Domain angehört. Dasselbe gilt auch für Knoten innerhalb zweier verschiedener Domainnamen (in der Abbildung: **informatik.tu-muenchen.de** $\neq$ **informatik.uni-muenchen.de**).

Eine Besonderheit stellt die Domain **in-addr.arpa** dar. Mit ihr ist es möglich, eine inverse Namensabbildung (Reverse Mapping): IP-Adresse $\rightarrow$ Name (FQDN) vorzunehmen. Dazu wird die IP-Adresse als dezimale Bytewerte ohne führende

[1] **hp10.informatik.tu-muenchen.de** ist ein Aliasname für **hphalle10.informatik.tu-muenchen.de**. Aliasnamen und Hostnamen sind aber gleichbedeutend, weshalb dies als real existierendes Beispiel gewählt wurde.

Nullen und Leerzeichen in umgekehrter Reichenfolge geschrieben, um den Postfix **in-addr.arpa** erweitert und als Name verwendet (Beispiel: **20.10.187.129.in-addr.arpa**). In dem damit referenzierten Blatt findet man dann einen entsprechenden Eintrag, der den Namen enthält (im Beispiel: **sun1.lrz.muenchen.de**). Der in der Abbildung abgeschnittene Teilbaum enthält alle Einträge für ein Class-B-Netz (**129.187.␣␣␣.␣␣␣**) des LRZ.

Neben dem Forward Mapping (Abbildung: Name → IP-Adresse) und dem soeben behandelten Reverse Mapping bietet das Domain Name System noch weitere Dienste/Informationen an. Der wichtigste ist das Mail-Routing [RFC 974]. Daneben können aber auch Informationen über CPU und Betriebssystem der Rechner abgefragt werden, sofern sie bereitgestellt werden. Weitere Informationstypen haben (derzeit) eine untergeordnete Bedeutung und sollen deshalb auch an dieser Stelle nicht erwähnt werden.

2.3.3 Zugriff einer Anwendung auf DNS-Informationen

Im folgenden wird gezeigt, wie eine Anwendung (Programm) auf Informationen, die das Domain Name System zur Verfügung stellt, zugreiffen kann. Abbildung 2.3 gibt eine Übersicht über ein mögliches Szenario.

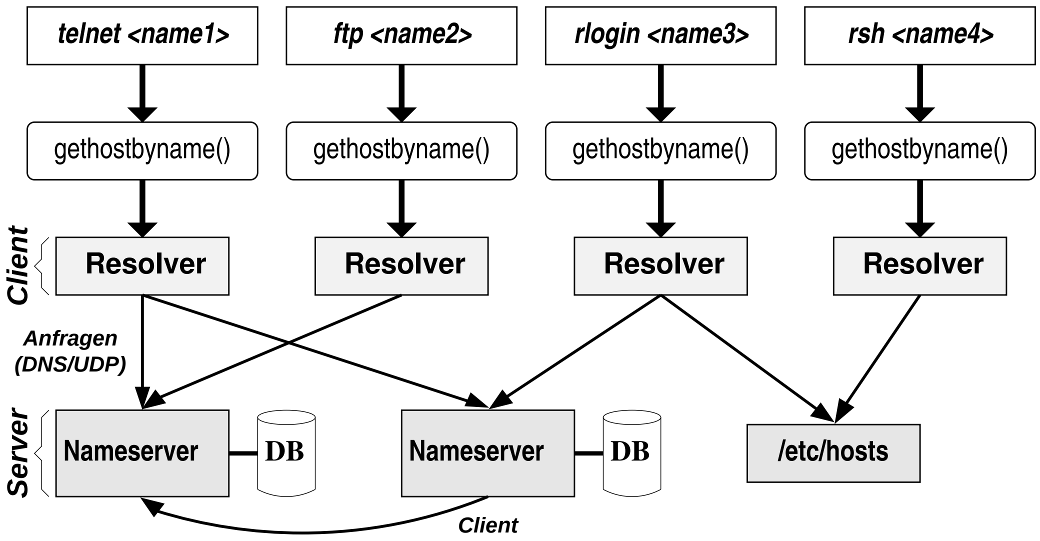

Abbildung 2.3: Informationsfluß (DNS-Anfragen)

Den Anwendungsprotokollen (telnet, ftp, ...) wird anstelle der IP-Adresse der Hostname bzw. der Fully Qualified Domain Name (FQDN) übergeben. Wird nur der Hostname angegeben[2], so wird er um den Domainnamen erweitert. Das aufgerufene Programm bedient sich beim Verbindungsaufbau der Bibliotheksfunktion gethostbyname(). Damit werden die Resolverroutinen des Betriebssystems aufgerufen.

Die Resolverroutinen schicken die Anfragen an den/die nächstgelegenen Nameserver[3]. Je nach Konfiguration der Resolver (siehe Manpage *gethostent* [UCBK]) kann die Anfrage auch an mehrere Nameserver geschickt oder können vorhandene Einträge in */etc/hosts* zur Namensauflösung verwendet werden.

Ist der Nameserver nicht in der Lage, eine Anfrage selbst zu beantworten, sondern nur auf einen diesbezüglich kompetenteren Nameserver zu verweisen, so muß die Anfrage an letzteren geschickt werden. Diese evtl. rekursive Abfrageprozedur kann der Resolver dem Anwendungsprogramm abnehmen[4].

Ein Nameserver ist ein Prozeß, der Anfragen beantwortet, die er mittels des DNS-Protokolls [RFC 1035] zumeist als Datagramme (UDP/IP) erhalten hat. Zu diesem Zweck kann auch eine TCP/IP-Verbindung aufgebaut werden[5]. Der Nameserver läuft als Hintergrundprozeß (Daemon) auf ausgewählten Rechnern eines Instituts oder einer Organisation.

Als eigene Informationsquelle besitzt er eine interne Datenbank, die mit Hilfe von Konfigurationsdateien geladen wird. Daneben besteht aber auch die Möglichkeit, Teile einer Datenbank anderer Nameserver zu kopieren, um dann seinerseits Informationen daraus liefern zu können. Für diesen Bereich dient ein Nameserver dann als Secondary Nameserver, wohingegen er für Bereiche, für die er seine Information aus lokalen Konfigurationsdateien (Zonendateien) erhält, als Primary Nameserver agiert. Diese Spiegelung ist notwendig, da zum einen der Primary Nameserver ausfallen kann und zum anderen ein „näher gelegener“ Secondary Nameserver eine geringere Netzlast und Antwortzeit ausmachen kann. Als Grundsatz gilt, daß für jede Zone mindestens zwei Nameserver vorhanden sein sollen [Quar 94].

Daneben speichert ein Nameserver seine Resultate zwischen, damit er für den Fall, daß die gleiche Anfrage erneut gestellt wird, das Ergebnis sofort zur Verfügung hat. Dieses als Caching bezeichnete Verfahren führt dazu, daß manche Nameserver (Cache Nameserver) nur über das an Informationen verfügen, was sie von anderen Nameservern zwischengespeichert haben. Dies ist sinnvoll, wenn die anderen Nameserver entlastet werden sollen.

[2]Dies wird in der Praxis dadurch festgestellt, daß kein Punkt im Namen enthalten ist (siehe Manpage *resolver* [UCBK], Option RES_DEFNAMES).

[3]Beim Domain Name System wird eine Client/Server Architektur eingesetzt. Der Transport der Information zwischen dem Nameserver (Informationsquelle) und dem Anwendungsprogramm (Client, Informationssenke) erfolgt über das Internet.

[4]siehe Manpage *resolver* [UCBK], Option RES_RECURSE

[5]siehe Manpage *resolver* [UCBK], Option RES_USEVC

An dieser Stelle muß darauf hingewiesen werden, daß die meisten Nameserver sowohl als Primary und Secondary Nameserver arbeiten (jeweils für eine oder mehrere verschiedene Zonen) als auch als Cache Nameserver für die Root-Domain. Letztere dient dazu, im Baum an Zweige zu gelangen, die nicht zu den selbst verwalteten gehören.

Kapitel 3

Konfiguration einer DNS-Umgebung

Beim Domain Name System handelt es sich um einen Verzeichnisdienst, der diejenigen Informationen zur Verfügung stellt, die den Nameservern beim Konfigurationsvorgang mitgegeben werden. Dies geschieht mit Hilfe von Konfigurationsdateien.

Wegen ihres großen Umfangs müssen die Informationen aufgeteilt werden. Die Unterteilung erfolgt in zwei verschiedenen Dimensionen. Beide werden in Abschnitt 3.1 vorgestellt.

In Abschnitt 3.2 werden die Konfigurationsdateien für die Nameserver, die von den Nameservern verwalteten Zonen sowie die Resolver, jeweils anhand von Beispielen, präsentiert. Deren Kenntnis ist Voraussetzung, um ein im Rahmen dieser Arbeit zu erstellendes Datenbankschema und Werkzeug verstehen zu können, mit deren Hilfe entsprechende Dateien generiert werden sollen.

Die Konfigurationsdateien können von Hand erstellt und damit eine funktionstüchtige DNS-Umgebung zum Laufen gebracht werden. Der Vorteil einer manuellen Konfiguration liegt darin, daß keine aufwendigen speziellen Werkzeuge notwendig sind, sondern ein einfacher ASCII-Editor ausreicht. Demgegenüber stehen schwerwiegende Nachteile, die im folgenden Kapitel angesprochen werden und dort ein gezieltes Management motivieren.

3.1 Unterteilung der DNS-Gesamtinformation

Die Unterteilung sämtlicher DNS-Informationen erfolgt in zwei verschiedenen Dimensionen (siehe Abbildung 3.1).

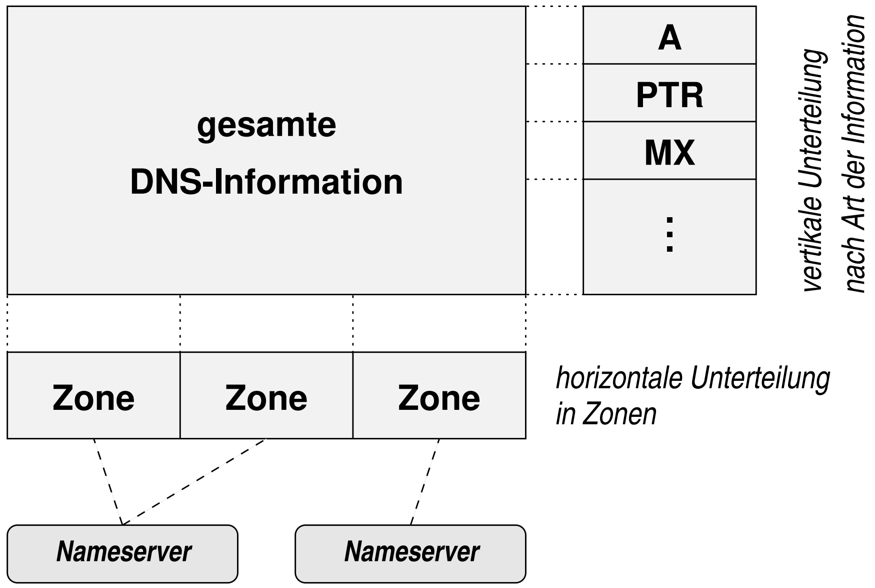

Abbildung 3.1: Unterteilung der DNS-Information

1. *Horizontale Unterteilung in Zonen*
   Bei einer verteilten Anwendung, wie sie das Domain Name System darstellt, ist es notwendig, dafür zu sorgen, daß die Aktivitäten der einzelnen diensterbringenden Komponenten (Nameserver) genau aufeinander abgestimmt werden, damit es zu keinen widersprüchlichen Resultaten und unzureichend geregelten Zuständigkeiten kommen kann.

   Es muß festgelegt werden, welche Information von welchem Nameserver geliefert werden soll, und zwar ob als Primary-, Secondary- oder Cache-Nameserver.

2. *Vertikale Unterteilung nach Art der Information*
   Hier wird festgelegt, welche Art von Information zur Verfügung gestellt wird. Die einzelnen Informationen werden den Nameservern in den Konfigurationsdateien (Zonendateien) als Record-Einträge übergeben.

   Beipiel: `localhost IN A 127.0.0.1`
   Hier steht in der ersten Spalte der Hostname, in der zweiten die Record-Klasse, in der dritten der Recordtyp und in der vierten die IP-Adresse, auf die der Hostname im Zuge des Foreward Mappings abgebildet wird.

   Die genaue Syntax sowie eine vollständige Liste der verschiedenen Record-Typen kann [RFC 1034] entnommen werden. Die wichtigsten davon sind in

Tabelle 3.1 aufgeführt.

Typ	Bedeutung
A	Name → IP-Adresse (Foreward Mapping)
PTR	IP-Adresse → Name (Reverse mapping)
MX	Mail Routing (siehe [RFC 974])
HINFO	Information über CPU und OS eines Rechners
WKS	Information über Netzdienste (vgl. */etc/services*)

Tabelle 3.1: Wichtige Record-Typen

3.2 Konfigurationsdateien

Im folgenden wird jeweils anhand eines Beispiels aufgezeigt, welche Dateien zur Konfiguration der Nameserver (Abschnitte 3.2.1 und 3.2.2) bzw. der Resolver (Abschnitt 3.2.3) benötigt werden.

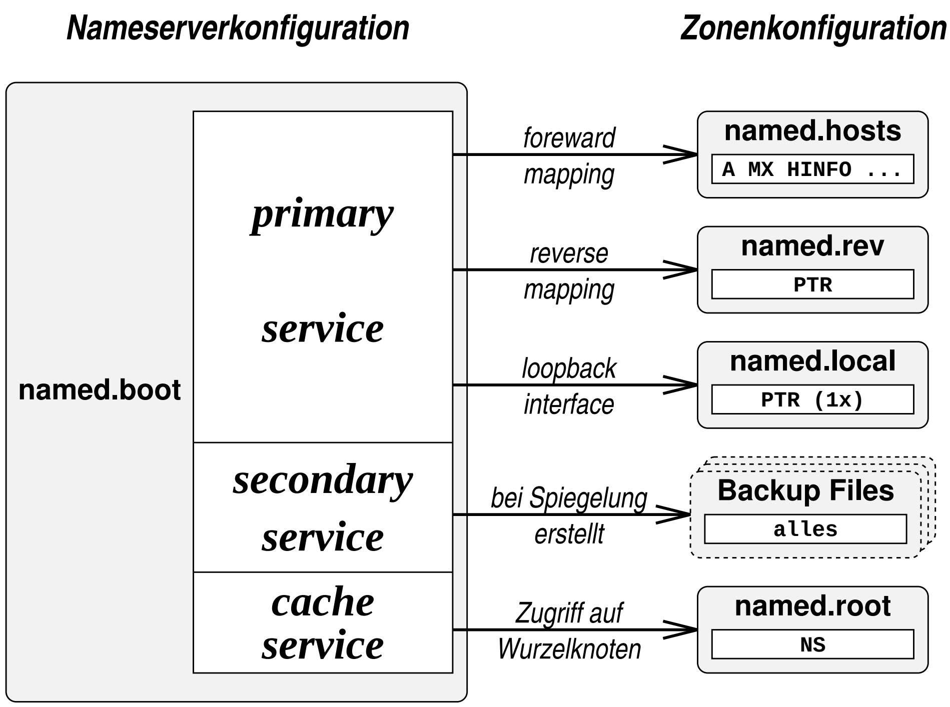

Abbildung 3.2: Konfigurationsdateien: Bootdatei und Zonendateien

Wie aus Abbildung 3.2 ersichtlich ist, teilen sich die für einen Nameserver notwendigen Konfigurationsdateien dabei in eine Bootdatei (Abschnitt 3.2.1) und mehrere Zonendateien (Abschnitt 3.2.2) auf.

3.2.1 Nameserverkonfiguration

In der Bootdatei (*named.boot*) werden die Zonen festgelegt, für die der zu konfigurierende Nameserver DNS-Informationen bereitstellen soll. Dabei werden im wesentlichen drei verschiedene Dienstarten unterschieden:

1. *Primary Service*
   Für jede Zone ist genau ein Primary Nameserver zur Verfügung zu stellen. Ein Nameserver kann diese Funktion für mehrere Zonen übernehmen. Die Konfiguration erfolgt über die noch zu beschreibenden Zonendateien. Diese sind vom DNS-Administrator zu erstellen.

2. *Secondary Service*
   Spiegelt ein Nameserver eine oder mehrere Zonen anderer Nameserver, um seinerseits deren DNS-Informationen bereitzustellen, so dient er hierfür als Secondary Nameserver. Die Information wird vom Nameserver aus in Backup-Files geladen. Letztere müssen also nicht vom DNS-Administrator erstellt werden, und werden deshalb im folgenden bei den Zonendateien auch nicht beschrieben.

3. *Cache Service*
   Werden keine ganzen Zonen gespiegelt, sondern nur einzelne Informationen daraus nach einer Anfrage zwischengespeichert, um sie bei späteren identischen Anfragen schneller weitergeben zu können, so dient der Nameserver hierfür als Cache Nameserver.

   Dieses Verfahren wird in der Praxis vor allem dazu verwendet, um an die Root-Domain zu gelangen, ohne diese komplett spiegeln zu müssen. Der Zugriff auf die Root-Domain ist deshalb unerläßlich, da es sonst keine Möglichkeit gibt, auf alle Domains zugreifen zu können. Sonst hätte man nur die selbst verwalteten Domains (und deren Subdomains) zur Verfügung.

Das folgende Beispiel zeigt, wie eine Bootdatei aussehen könnte. Es erhebt keinen Anspruch auf Vollsändigkeit. In der ersten Spalte steht die Dienstart, in der zweiten die Zone, in der dritten die Zonendateien bzw. die IP-Adresse der zugehörigen Primary Nameserver und in der vierten der Namen der Backup-Files. Die genaue Syntax und Semantik kann [Vixi 93] bzw. [RFC 1034] entnommen werden (dies gilt auch für die nachfolgenden Zonendateien).

```
directory  /etc/namedb
; type     domain                source/host   bak-file
cache      .                     named.cache
primary    lrz-muenchen.de       named.hosts
primary    128.129.in-addr.arpa  named.rev
primary    0.0.127.in-addr.arpa  named.local
secondary  eg                    193.227.1.1   eg.BF
secondary  gov.eg                193.227.1.1   gov.eg.BF
```

3.2.2 Zonenkonfiguration

In den Zonendateien werden die Informationen festgelegt, die vom Domain Name System für die jeweilige Zone zur Verfügung gestellt werden (Zonenkonfiguration). Dabei ist für jede Zone eine eigene Datei zu erstellen. Im folgenden wird eine typische Auswahl vorgestellt. Diese bezieht sich auf obige Bootdatei.

- *named.cache*
  Diese Datei enthält die Nameserver für die Root Domain. Die hierfür zuständigen Nameserver werden von der Netzbetreiberorganisation festgelegt. Die Datei kann mittels FTP (Rechner: *RS.INTERNIC.NET*, Dateiname: */domain/named.cache*) bezogen werden. Eine Beschreibung ihres Aufbaus erübrigt sich an dieser Stelle, da sie nicht modifiziert werden sollte.

- *named.hosts*
  Die folgende Datei ist eine Beispieldatei für das Foreward Mapping. Im Kopf (SOA-Record) werden neben der E-Mail-Adresse des DNS-Administrators und der Seriennummer, die bei jeder Modifikation zu erhöhen ist, eine Reihe von Zeitspannen angegeben. Anschließend werden die Nameserver für die betroffene Domain und deren Subdomain(s), ein Eintrag zum Mailrouting sowie Einträge für die Namensabbilung auf Adressen aufgelistet. Neben den im Beispiel aufgeführten können noch weitere Record-Typen (HINFO, WKS, ...) verwendet werden.

```
$ORIGIN  lrz-muenchen.de.
@                IN SOA   dfvgate postmaster.dfvgate (
                           400338 ; Serial
                            21600 ; Refresh {6 Stunden}
                             1800 ; Retry {1/2 Stunde}
                          3600000 ; Expire {1000 Stunden}
                            86400); Minimum {24 Stunden}
                 IN NS    dfvgate
                 IN NS    hp10
```

```
ppp                IN NS    dfvgate
                   IN NS    hp10
@                  IN MX    140 cd1
localhost          IN A     127.0.0.1
sun1               IN A     129.187.10.20
sun2               IN A     129.187.10.21
dfvgate            IN A     129.187.10.25
hp10               IN A     129.187.13.22
cd1                IN A     129.187.13.3
```

- *named.rev*
  Bei der folgenden Datei Beispiel für das Reverse Mapping für ein Class-B-Netz (**129.187.␣␣␣.␣␣␣**, unvollständig).

```
$ORIGIN  187.129.in-addr.arpa.
@                  IN SOA   dfvgate.lrz-muenchen.de. ...
                   IN NS    dfvgate.lrz-muenchen.de.
                   IN NS    hp10.lrz-muenchen.de.
20.10              IN PTR   sun1.lrz-muenchen.de.
20.11              IN PTR   sun2.lrz-muenchen.de.
195.15             IN PTR   a2824ch.ppp.lrz-muenchen.de.
```

- *named.local*
  Die folgende Datei dient dem Reverse Mapping für das Loopback Interface, also der Abbildung der IP-Adresse **127.1** auf den Namen **localhost**. Die zugehörige Abbildung für das Foreward Mapping wurde in die Datei *named.hosts* mit eingebaut[1].

```
$ORIGIN  0.0.127.in-addr.arpa.
@                  IN SOA   dfvgate.lrz-muenchen.de. ...
                   IN NS    dfvgate.lrz-muenchen.de.
                   IN NS    hp10.lrz-muenchen.de.
1                  IN PTR   localhost.
```

3.2.3 Resolverkonfiguration

Neben den Nameservern und den davon verwalteten Zonen müssen auch die Resolver konfiguriert werden. Dies geschieht mit Hilfe der Datei *resolv.conf*, die bei Unix-Rechnern im */etc*-Verzeichnis abzulegen ist.

[1] Das im weiteren Verlauf dieser Arbeit zu erstellende Werkzeug verwendet für hierfür eine eigene Zone/Datei.

Diese Datei kann je nachdem, ob auf dem jeweiligen Rechner ein Nameserver installiert ist oder nicht, unterschiedlich ausfallen:

- *Mit lokalem Nameserver*
  Für den Fall, daß auf dem Rechner, für den die Datei zu erstellen ist, ein Nameserver läuft, genügt es, nur die Domain anzugeben, der der Rechner angehört[2]. Dieser Domainnamen wird an den Hostnamen angefügt, bevor die Namensauflösung durchgeführt wird, so daß der Anwender bei lokalen Rechnern nur der Hostnamen anzugeben braucht.

  Beispielsweise genügt es, den Hostnamen **dfvgate** anzugeben, wenn man von dem Rechner **sun1.lrz-muenchen.de** aus den Rechner **dfvgate.lrz-muenchen.de** erreichen will.

  Die Konfigurationsdatei hat damit beispielsweise folgende Gestalt:

  ```
  domain          lrz-muenchen.de
  ```

- *Ohne lokalem Nameserver*
  Ist dagegen kein lokaler Nameserver vorhanden, so muß die Anfrage durch den Resolver an einen Nameserver auf einem anderen Rechner im Netz geschickt werden. Dazu sind die IP-Adressen derjenigen Rechner anzugeben, auf denen ein geeigneter Nameserver läuft.

  Also muß die obige Datei noch um die Angabe von Nameservern erweitert werden. Beispiel:

  ```
  domain          lrz-muenchen.de
  nameserver      129.187.10.25     # dfvgate
  nameserver      129.187.16.1      # hp10
  ```

[2] Das im weiteren Verlauf dieser Arbeit zu erstellende Werkzeug gibt wie im zweiten Fall den/die Nameserver an, auch wenn ein Nameserver lokal läuft, ggf. also mit der Adresse **127.0.0.1**.

Kapitel 4

Anforderungen an das Management eines Nameserver-Verbundes

Dieses Kapitel beschäfigt sich mit dem Management einer DNS-Umgebung, die einen oder mehrere Nameserver umfaßt.

Der einführende Abschnitt 4.1 soll einen groben Überblick darüber vermitteln, welche Bedeutung das Management bei Rechnersystemen und Rechnernetzen besitzt, und wie bei dessen Realisierung vorgegangen werden kann.

Im folgenden Abschnitt 4.2 wird die Notwendigkeit des Managements speziell für das Domain Name System herausgestrichen. Anhand von Beispielen wird gezeigt, daß Management hierfür unerläßlich ist.

Anschließend wird das Management aus Sicht des DNS-Administrators dargelegt (Abschnitt 4.3). Dies umfaßt Aufgaben, die unter Berücksichtigung gegebener Anforderungen zu erledigen sind. Zur deren Umsetzung stehen isolierte Verfahren und Werkzeuge bereit.

Deren Nachteile werden in Abschnitt 4.4 dargelegt. Abhilfe schafft hier das integrierte Management. Dabei werden zwei Integrationsansätze für das Domain Name System vorgestellt, von denen einer den weiteren Verlauf der vorliegenden Arbeit bestimmt: Auf der Grundlage einer Anbindung des Domain Name Systems an ein Dokumentationssystem wird ein Verfahren für das zugehörige Konfigurationsmanagement entwickelt.

Einige Anforderungen, die sich in diesem Zusammenhang an die Managementumgebung des DNS-Administrators stellen, um diesen bei seiner Arbeit zu unterstützen, sind Gegenstand des Abschnitts 4.5, bevor im nächsten Kapitel mit der Realisierung des Verfahrens begonnen wird.

4.1 Bedeutung des Managements

Bei modernen Rechnersystemen und Rechnernetzen genügt es nicht, sich nur auf die Aufgaben, die sie zu erbringen haben, zu konzentrieren. Um beispielsweise eine Buchung von einem Bankkonto in Hamburg auf ein anderes in München zuverlässig durchführen zu können, ist weitaus mehr erforderlich als reine Buchhaltungskenntnisse und entsprechende Aktivitäten.

Es muß gewährleistet werden, daß die zugrundeliegenden elektronischen Datenverarbeitungssysteme ihre Aufgaben stets korrekt und im vollen Umfang erbringen, was aufgrund ihrer schnell anwachsenden Komplexität nicht als Selbstverständlichkeit erachtet werden kann. Statt dessen ist es erforderlich, dies nicht nur einmal, sondern ständig von neuem sicherzustellen. Die Systeme sind also zu *managen*:

Netz- bzw. Systemmanagement umfaßt die Gesamtheit der Vorkehrungen und Aktivitäten zur Sicherstellung des effektiven und effizienten Einsatzes eines Rechnernetzes bzw. eines verteilten Systems[1].

Die damit verbundenen Managementaufgaben sind von den Aufgaben der Anwender (im Bsp.: Bankkaufmann am Rechner) zu trennen und füllen i.d.R. eine oder mehrere eigenständige Rollen aus.

Die Durchführung des Managements läßt sich im wesentlichen wie folgt einteilen:

- Es ist stets ein *Ziel* vorzugeben. Man muß wissen, warum und wozu man Management betreibt.
- Um ein Ziel zu erreichen, sind eine oder mehrere *Aufgaben* zu erledigen.
- An jede Aufgabe werden diverse *Anforderungen* gestellt, die angeben, was bei der Erfüllung der Aufgabe zu berücksichtigen ist. Damit im Zusammenhang stehen auch die Dienstgüteparameter (Quality of Service – QoS).
- Zur *Umsetzung* der Aufgabe dienen unterschiedliche Verfahren, die sich verschiedener Werzeuge bedienen. Letztere können isoliert, koordiniert oder integriert sein [Hege 93].

Ferner läßt sich das Management auch in Dimensionen wie Funktionalität oder Zeit einteilen. Der interessierte Leser sei wiederum auf [Hege 93] verwiesen, was auch für viele weitere Aspekte im Zusammenhang mit Managementfragen gilt.

[1] Zitat aus [Hege 93].

4.2 Notwendigkeit des Managements bzgl. des Domain Name Systems

Im vorangehenden Kapitel wurde ein einfaches Beispiel gezeigt, wie sich eine DNS-Umgebung konfigurieren läßt. Dabei wurden ein Nameserver und einige wichtige Zonen berücksichtigt. Wird ein entsprechender Nameserver gestartet, so verfügt man über eine lauffähige DNS-Umgebung.

In der Praxis ist es aber nicht damit getan, ein paar einfache Konfigurationsdateien zu erstellen und dann einen Nameserver laufen zu lassen. Vielmehr treten eine Reihe von Problemen auf, mit denen man sich *vor* und *während* des Betriebs auseinanderzusetzen hat.

Einige wenige dieser Probleme werden im folgenden exemplarisch angesprochen, jedoch nicht alle, da die Gesamtproblematik im Zusammenhang mit dem Domain Name System ein sehr weites Feld umspannt und deren Kenntnis jahrelange praktische Erfahrungen voraussetzt.

Klagen seitens der Anwender, die bei ihrer Arbeit mit dem Domain Name System (indirekt) konfrontiert werden, können z.B. wie folgt lauten:

- „Mein Programm kennt den Hostnamen nicht, obwohl ich mir sicher bin, daß dieser existiert, und ich den richtigen Namen eingegeben habe.“
- „Mein Programm benötigt für einen Verbindungsaufbau bei Angabe von Host- und Domainnamen entscheidend länger, als wie wenn ich die schwer zu merkende IP-Adresse angebe.“
- „Während des Ausfalls unseres Mailhosts wurde meine E-Mail gar nicht oder an einen falschen Host weitergeleitet.“
- „Ich erhalte noch immer die IP-Adresse eines Rechners, den es schon seit einer Woche nicht mehr gibt.“

Derartige Probleme können vielschichtige Ursachen haben. Dies allein anhand der von Hand erstellten Konfigurationsdateien und durch Nachstellen einer möglichst ähnlichen Problemsituation klären zu wollen, ist nicht nur mühsam, sondern in entsprechend komplexen Fehlerfällen nicht mehr zu realisieren.

Was in diesem Zusammenhang auch viel zu wenig berücksichtigt wird, ist der Umstand, daß streßgeplagte Anwender in Fehlersituationen, die sie nicht selbst zu verantworten haben (bzw. hiervon überzeugt sind), sehr ungeduldig werden können und auf eine *schnelle* Lösung des Problems drängen. Es sind Fälle vorstellbar, in denen eine derartige Ungeduld berechtigt ist.

Neben Problemen, mit denen der Anwender konfroniert wird, gibt es auch noch andere, die z.B. die Betreiber von Nameservern untereinander betreffen, wie deren Abstimmung bei der Zonenverteilung bzw. -delegierung oder bei der (gegenseitigen) Spiegelung ihrer Zonen (Secondary Service).

Aus den aufgeführten und vielen weiteren, hier nicht genannten Problemen läßt sich erkennen, daß die Verwaltung einer DNS-Umgebung ein komplexes Aufgabengebiet darstellt, zu deren Bewältigung eine *Managementumgebung* erforderlich ist.

Dem Betreiber einer DNS-Umgebung, dem DNS-Administrator, ist eine Reihe von Verfahren und Werkzeugen zur Verfügung zu stellen, die es ihm mit möglichst geringem (bewältigbaren) manuellen Aufwand erlauben, mittels des/der von ihm verwalteten Nameserver(s) stets *korrekte*, *vollständige*, *aktuelle* und *überall im Netz zugängliche* DNS-Informationen bereitzustellen.

4.3 DNS-Management aus Betreibersicht

In diesem Abschnitt soll erläutert werden, wie sich dem DNS-Administrator seine Managementaufgaben präsentieren. Das Ziel seiner Aufgaben wurde am Ende des vorangehenden Abschnitts bereits vorgestellt.

4.3.1 Aufgaben

Zu den Aufgaben eines DNS-Administrators zählen u.a.:

- Sinnvolle Aufteilung der Rechner einer Organisation auf Domains und Subdomains.
- Aufteilung der Domains und Subdomains auf eine oder mehrere Verwaltungseinheiten, die sog. Zonen.
- Festlegung, auf welchen Rechnern im Netz Nameserver zu installieren sind.
- Zuordnung von Zonen zu Nameservern unter Berücksichtigung der Serviceart (Primary, Secondary, Cache, Slave).

Die obengenannten Aufgaben sind Bestandteil der *Planungsphase* einer DNS-Umgebung. Sie sind zu erbringen, bevor eine entsprechende Struktur realisiert werden kann.

Dagegen werden im folgenden Aufgaben genannt, die in die *Betriebsphase* überführen bzw. in dieser von Bedeutung sind:

- Bereitstellen und Aktivieren von Nameservern.
- Anpassung der Nameserver- bzw. Zonenkonfiguration im Zuge von Aktualisierungen der Netzumgebung (neue Hosts, Domains, Nameserver, Modifikation oder Abbau bestehender Einrichtungen etc.).
- Überwachung der laufenden Nameserverprozesse.
- Korrektur der Nameserver- bzw. Zonenkonfiguration im Falle von aufgetretenen Fehlern.

Während sich die erstgenannten Punkte weitgehend auf das Konfigurationsmanagement beschränken, sind die letzten beiden v.a. dem Fehlermanagement zuzuordnen.

4.3.2 Anforderungen

An die Realisierung von Aufgaben werden diverse Anforderungen gestellt. Mögliche Anforderungen, die sich im Zusammenhang mit den oben genannten Aufgaben ergeben, sind beispielsweise:

- Bei der Aufteilung der Rechner einer Organisation auf Domains und Subdomains ist die Struktur der Organisation zu berücksichtigen. Dies bedeutet z.B. für eine Universität, daß für ihre Fakultäten Subdomains geschaffen werden können und darunter für die einzelnen Lehrstühle wiederum eigene Subdomains. Darin befinden sich dann die Hostnamen der einzelnen Rechner.
- Bei der Aufteilung der Domains und Subdomains auf eine oder mehrere Zonen spielt es eine große Rolle, welche Subdomains/Hosts jeweils gemeinsam miteinander oder z.B. von unterschiedlichen Nameservern verwaltet werden sollen (Primary Service).
- Bei der Verteilung von Nameservern auf Rechner im Netz sind Aspekte wie die Netztopologie, d.h. in diesem Zusammenhang Zugriffsmöglichkeiten von innen (Hosts der Organisation, für die der DNS bereitgestellt wird) und von außen (Hosts im übrigen Internet), Ersatznameserver (Secondary) für den Fall des Ausfalls eines Nameservers oder das allg. Netzverkehrsaufkommen[2] (Lastbalancierung) zu berücksichtigen.

[2] Während der Anteil der DNS-Pakete im Vergleich zu den anderen Anwendungsprotokollen bei den virtuellen Verbindungen (TCP) gering ist, liegt er i.a. bei Datagrammen (UDP) an erster Stelle.

- Beim Bereitstellen von Nameservern ist festzulegen, welche Dienste jeder einzelne anzubieten hat. In der Regel verfügt jeder Nameserver neben dem Service für die ihm zugeteilten Zonen über einige Grundaufgaben wie den Primary Service für das Loopback Interface (Forward und Reverse Mapping) und den Cache Service für die Root-Domain. Diese Aufgaben sind genau zu spezifizieren.

- Bei der Anpassung der DNS-Konfiguration im Zuge von Aktualisierungen läßt sich eine maximale Zeitspanne vorgeben, die vom Augenblick t_0 der Mitteilung der Modifikation an den DNS-Administrator verstreichen darf, bis die neuen Informationen von den zuständigen Nameservern korrekt weitergegeben werden.

  Dabei muß unterschieden werden, ob es sich um einen Primary (Zeitspanne t_{1P}) oder um einen Secondary Nameserver (Zeitspanne t_{1S}) handelt, da letzterer nicht direkt rekonfiguriert werden kann, sondern nur über eine erneute Zonenspiegelung. Vergleichbares gilt für einen Cache Nameserver. In beiden Fällen spielt die Gültigkeitdauer von Einträgen eine wichtige Rolle, da die alten Informationen noch so lange verbreitet werden, wie sie von den Secondary bzw. Cache Nameservern für gültig gehalten werden (siehe auch Abschnitt 4.5.2).

  Die Zeitspannen (Δt_P für Primary bzw. Δt_S für Secondary Nameserver) setzen sich fogendermaßen zusammen:

  $$\begin{array}{lllll} \Delta t_P & = & t_{1P} - t_0 & = & \Delta t_r + \Delta t_a + \Delta t_k \\ \Delta t_S & = & t_{1S} - t_0 & = & \Delta t_r + \Delta t_a + \Delta t_k + \Delta t_g \end{array}$$

  Die noch nicht angegebenen Zeitwerte haben folgende Bedeutung:

  Δt_r: Reaktionszeit des DNS-Administrators.
  Diese ist zum einen zeitabhängig (Tageszeit, Wochentag, Urlaubszeit) und hängt zum anderen von der momentanten Auslastung des DNS-Administrators ab. Im Falle einer Überbelastung sind Prioritäten zu setzen.

  Δt_a: Bearbeitungszeit für die neuen Konfigurationsdateien.
  Hier kommt vor allem zum Tragen, wielange es dauert, bis die zu ändernde Stelle gefunden wird. Dazu ist der DNS-Administrator durch Werkzeuge zu unterstützen. Ein solches Werkzeug wird im nächsten Abschnitt vorgestellt und ein weiteres im Rahmen dieser Arbeit entwickelt. Beide dienen der Generierung von Konfigurationsdateien aus einem besser handhabbaren Datenbestand.

  Δt_k: Zeit für den Rekonfigurationsvorgang der Nameserver.
  Die Zeit für eine (dynamische) Rekonfiguration eines Nameservers ist im Vergleich zu den anderen Zeiten sehr klein und kann i.a. deshalb vernachlässigt werden.

Δt_g: Restgültigkeitsdauer eines gespiegelten Datenbestandes.
Erst nach Ablauf dieser Zeit fragt ein Secondary Nameserver beim zuständigen Primary Nameserver nach, ob seine Daten noch gültig sind. Hinzu kommt noch die erforderliche Zeit für die Spiegelung (Übertragen der Zonendaten).

Mit den beiden angegebenen Zeitspannen werden zwei Dienstgüteparameter (QoS) definiert. Dabei ist vom Worst Case, also der längsten möglichen Dauer, die die Aufgaben in Anspruch nehmen, auszugehen.

- Bei der Korrektur von Fehlern besteht der meiste Aufwand i.d.R. darin, diese zu lokalisieren und zu diagnostizieren. Handelt es sich um einen Konfigurationsfehler, so kommt zu den im letzten Punkt genannten Zeitspannen also noch die Zeit für die Fehlersuche hinzu. Für andere Fehlerursachen lässen sich dagegen nur schwer Zeitangaben machen, da diese zu unterschiedlicher Natur sein können. Diese reichen von Übertragungsfehlern im Netz bis hin zu Bugs in den Nameservern oder Resolvern selbst.

  Um die Fehlersuche zu vereinfachen und damit die hierfür notwendige Zeit zu verkürzen, gibt es hierfür eine Reihe von Werkzeugen. Einige werden im folgenden Abschnitt kurz vorgestellt.

4.3.3 Umsetzung

Gegenstand dieses Abschnitts sind einige existierende isolierte Werkzeuge, die dem DNS-Administrator zur Umsetzung seiner Managementaufgaben dienen. Neben einem Verfahren mit zugehörigem Werkzeug zur Konfiguration von Zonen wird eine Auswahl an Werkzeugen zur Fehlersuche vorgestellt.

Zonenkonfiguration mittels *df2hf*

Im folgenden Verfahren wird nur eine einzige Konfigurationsdatei (Datenbasis) für die Zonenkonfiguration benötigt, in der alle hierfür relevanten Informationen enthalten sind.

Aus dieser „Multi Domain Nameserver Database"-Datei (DOMAINS) werden die Zonendateien und eine HOSTS-Datei mit Hilfe eines Perl[3]-Skripts *df2hf* [4] automatisch generiert (siehe Abbildung 4.1).

[3] *Perl* ist eine sehr mächtige Skriptensprache, die für die wichtigsten Systeme *(MS-DOS, UNIX)* verfügbar ist [Wall 91].

[4] Das Skript stammt von T. Eckert [Ecke 89].

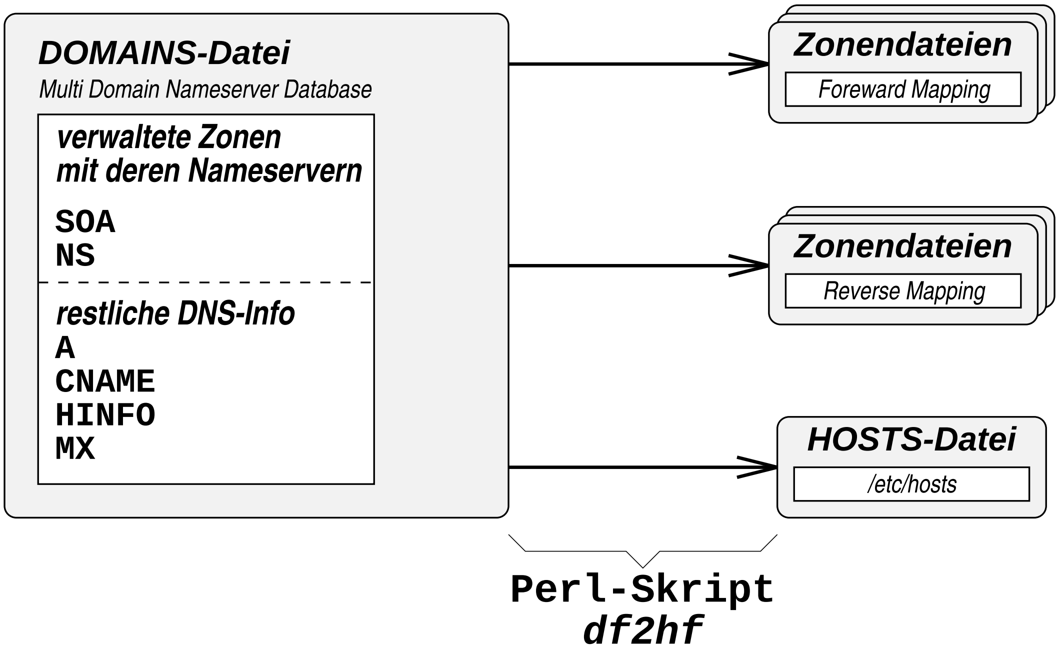

Abbildung 4.1: Einsatz des Generierungsskripts *df2hf*

In der vom DNS-Administrator zu erstellenden DOMAINS-Datei werden die DNS-Informationen (Records) in der gleichen Notation angegeben wie bei den in Kapitel 3.2.2 angegebenen Zonendateien.

Das Perl-Skript erzeugt aus der DOMAINS-Datei mehrere Zonendateien, jeweils eine pro Zone. Dabei werden Zonendateinamen verwendet, die aus dem jeweiligen Domainnamen hervorgehen. Daneben wird eine HOSTS-Datei erzeugt, die für lokale Namensauflösung (/etc/hosts) verwendet werden kann.

Das Verfahren bietet bietet folgende Vorteile:

- *Zentrale Datenhaltung*
  Die von einem Nameserver verwalteten DNS-Informationen aus *mehreren* Zonen werden komplett in *einer* Datei an *einer* Stelle gehalten. Damit können relevante Stellen schneller aufgefunden werden als in den einzelnen Zonendateien, wodurch z.B. die Effizienz einer Fehlersuche gesteigert werden kann.

- *Automatisches Generieren der PTR-Records*
  Erstellt man die Zonendateien von Hand, so muß für jeden Rechner neben dem A-Record für das Foreward Mapping jedesmal auch ein PTR-Record für das zugehörige Reverse Mapping erstellt werden. Da beide Einträge

direkt voneinander abhängen, kann man sich doppelte Arbeit sparen. Das Skript erstellt die PTR-Records automatisch, wenn man die hierfür notwendigen Zonen angibt. Damit wird nicht nur unnötige Handarbeit vermieden, sondern auch eine zusätzliche mögliche Fehlerquelle in Bezug auf die Konsistenz der beiden Mapping-Verfahren, die sich gegenseitig aufheben sollen, eliminiert.

Dem gegenüber stehen aber auch einige Nachteile:

- *Sonderregeln müssen beachtet werden*
  In der Quelldatei (DOMAINS) müssen Alias-Records (CNAME) stets vor den Address-Records (A) der zugehörigen Rechner stehen. Diese Bedingung gilt nur hier und stellt eine unnatürliche Einschränkung der Freiheit des DNS-Administrators dar. Erstellt man die Zonendateien dagegen von Hand, so hat man diesbezüglich freie Wahl. Beim Umsteigen auf dieses Verfahren ist also ein Gewöhnungsprozeß an diese Besonderheit erforderlich.

- *Beschränkung auf eine Klasse*
  Das Domain Name System ist ursprünglich dazu konzipiert, neben den Internet-Adressen (IN) auch noch andere Protokollfamilien zuzulassen (siehe [RFC 1034], Kapitel 3.6), wohingegen das Skript nur die Klasse IN unterstützt.

  Dies ist in der Praxis derzeit irrelevant, da die anderen praktisch nicht eingesetzt werden. Die Klasse HS steht für „MIT/Athena Hesiod Information" (siehe [Vixi 93], Kapitel 2.3). Dabei handet es sich um einen Verzeichnisdienst vergleichbar mit NIS, der aber keine sicherheitssenitiven Daten (z.B. *passwd*) verwaltet.

- *Nachbesserungen von Hand*
  Ein Nachteil, der nicht dem Werkzeug anzulasten ist, sondern dem freizügigen Umgang damit im Rahmen des Verfahrens, ist in der Billigung von Kompromissen in der DOMAINS-Datei zu sehen, wie der folgende Ausschnitt daraus (am LRZ) belegt:

  ```
  ;;; Der folgende Eintrag ist nur noch fuer /etc/hosts drin
  ;;; (eigentlich falsch, weil localhost nur in die top domain gehoert.)

  localhost       IN      CNAME   loghost
  loghost         IN      A       127.0.0.1
  ```

  Werden bewußt Fehler beim Konfigurieren eingebaut, so kann einem anderen DNS-Administrator die Fehlersuche dadurch erheblich erschwert werden, wenn er über derartige Besonderheiten nicht explizit aufgeklärt wird. Um Managementaufgaben erledigen zu können, benötigt er mindestens eine Informationsbasis, die als *korrekt* anzusehen ist.

- *Unterstützung von nur einem Nameserver*
  Aus der DOMAINS-Datei werden nur Zonendateien für einen Nameserver generiert. Will man zwei oder mehrere Nameserver als Primary Nameserver für darin enthaltene Zonen verwenden, so muß man die Aufteilung der Zonen und die Verteilung der zugehörigen Zonendateien von Hand vornehmen. Dabei können sich wiederum leicht Fehler einschleichen, da dann manche Zonen zwei Primary Nameservern oder gar keinem zugeordnet werden könnten.

  Eine Alternative wäre die Verwendung mehrerer DOMAINS-Dateien, jeweils einer pro Nameserver. Damit wäre die gewünschte zentrale Verwaltung sämtlicher Zonendaten nicht mehr gegeben.

- *Bootkonfigurationsdatei wird nicht erstellt*
  Die Bootkonfigurationsdatei (named.boot) wird von dem Skript nicht generiert, sondern muß noch immer von Hand erstellt werden. Da aber die Zonen in der DOMAINS-Datei auch angegeben werden müssen (zumindest für den Primary und Secondary Service), ist es nicht zu rechtfertigen, daß sich der DNS-Administrator fast die gleiche Arbeit zweimal machen soll.

Werkzeuge zur Fehlersuche

Zur Fehlersuche stehen u.a. die folgenden Werkzeuge bereit[5]. Die ersten drei dienen dem Information Retrieval und machen so das Starten eines Anwenderprogramms, das den DNS verwendet, überflüssig.

host: Einfaches Resolverwerkzeug, um von der Kommandozeile aus DNS-Informationen für einen Host abzufragen.

nslookup: Resolverwerkzeug zur Abfrage von DNS-Informationen. Die Anfragen können entweder in der Kommandozeile mit angegeben oder während des Programmlaufs formuliert werden.

dig (domain information groper): Resolverwerkzeug, ebenfalls zur Abfrage von DNS-Informationen. Die Anfragen können wiederum in der Kommandozeile mit angegeben oder in eine Stapeldatei geschrieben werden.

doc (domain obscenity control): Werkzeug zur Überprüfung von fehlerhaften Domains. Dabei werden eine Reihe von Anfragen an alle betroffenen Nameserver geschickt und die Ergebnisse analysiert.

[5] Die meisten Werkzeuge werden in der Newsgroup *comp.networks.noctools.tools* kurz vorgestellt. Genauere Informationen können den zugehörigen Manual Pages entnommen werden.

dnswalk: Werkzeug, das von einzelnen Domains Zonentransfers durchführt und die Ergebnisse auf interne Konsistenz und Sorgfalt überprüft.

dnsstats: Werkzeug zur Analyse von Syslog-Dateien, in denen Anfragen von den Nameservern mitprotokolliert werden können.

Daneben existieren noch weitere Werkzeuge, wie z.B. **ninit**, das der Überwachung eines Nameserverprozesses dient, und im Falle eines Ausfalls einen neuen Prozeß startet [Golk 91].

4.4 Integrationsansätze

Aus der Liste der aufgeführten Werkzeuge wird erkennbar, daß sie sich in ihren Funktionalitäten überschneiden. Es ist aber keines vorhanden, mit welchem sich alle relevanten Managementaspekte abdecken lassen. Vielmehr handelt es sich meist um Shellskripten, die ein einzelner DNS-Administrator erstellt hat, um ein ganz spezielles Problem zu lösen, mit dem er sich (gegenwärtig) konfrontiert sieht. Hinzu kommen oft noch mangelhafte Bedienerschnittstellen, die ebenfalls auf rein individuelle Bedürfnisse zugeschnitten sind.

Derartige Werkzeuge lassen auch keine globalen Betrachtungsweisen zu. Das Domain Name System wird nicht mit einem, sondern mit einer ganzen Reihe von Nameservern im Netz realisiert, weshalb es nicht genügt, nur einen einzigen zu managen. Dasgleiche gilt auch für die Domains bzw. Zonen, die eng miteinander verflochten sind, und deshalb nicht losgelöst voneinander betrachtet werden können.

Ein weiterer Nachteil dieser Werkzeuge liegt darin, daß sie eine globale Datenbasis, die sämtliche Funktionsbereiche des Netz- und Systemmanagements berücksichtigt, nicht unmittelbar unterstützen. Statt dessen ist ein individuell aufbereiteter Datenbestand erforderlich (Bsp.: DOMAINS-Datei bei *df2hf*).

Damit wird für beispielsweise für das Domain Name System ein Datenbestand *Hosts* ↔ *IP-Adressen* angelegt, während daneben für das Routing eine Zuordnung *Hosts* ↔ *Subnetze* ↔ *IP-Adressen* existiert. Beide zusammen sind redundant. Dies hat nicht nur doppelte Arbeit beim Aufbau und bei der Verwaltung zur Folge, sondern birgt auch die Gefahr schwerer Konsistenzprobleme, falls beide Datenbestände nicht stets gleichzeitig und aufeinander abgestimmt modifiziert werden.

Die beschriebenen Mängel machen integriertes Management erforderlich. Dabei geht es im Rahmen dieser Arbeit nicht um die Problematik mit Komponenten verschiedener Hersteller mit unterschiedlichen Protokollen und Schnittstellen in einem heterogenen Netz, da die eingesetzten Nameserver, Protokolle und Schnittstellen jeweils homogen sind, sondern vielmehr um die Anpassung der einzelnen Managementverfahren des Domain Name Systems untereinander sowie deren Integration in das gesamte Netz- und Systemmanagement[6].

Im folgenden werden zwei Ansätze präsentiert, die einen Schritt in Richtung *Integration des DNS-Managements in das Netz- und Systemmanagement* darstellen.

Der erste Ansatz (Abschnitt 4.4.1) verwendet die weitverbreitete Internet-Managementarchitektur und stellt eine einheitliche Schnittstelle zu Resolvern und Nameservern her. Allerdings wurden die Festlegungen (RFC's) seitens des Internet Activities Boards (IAB) hierzu erst im Laufe dieser Arbeit veröffentlicht, weshalb das Internet-Management darin noch keine Rolle spielen wird.

Der zweite Ansatz (Abschnitt 4.4.2) basiert auf einem Dokumentationssystem, das als zentrale Informationsbasis für integriertes Management dient. Ein solches ist in Form einer Netzdokumentationsdatenbank am LRZ installiert. Dieser Ansatz soll im Rahmen dieser Arbeit für das Konfigurationsmanagement des Domain Name Systems realisiert werden.

4.4.1 Internet-Management für das DNS

Mit Hilfe des Simple Network Management Protocols (SNMP) wird im Rahmen des Internet-Managements ein Zugriff auf Instanzen der zu managenden Objekttypen ermöglicht. Dazu ist ein Informationsmodell erforderlich, das festlegt, wie die einzelnen Objekttypen identifiziert werden können. Dazu dient der Internet-Registrierungsbaum. Ein wesentlicher Bestandteil davon ist die MIB-2 (Management Information Base) als Standard-MIB, in der allgemeine, herstellerunabhängige Managementinformation abgelegt ist.

Hierzu gehören mit der Veröffentlichung von [RFC 1611] und [RFC 1612] auch Informationen über das DNS-Management (siehe Abbildung 4.2). Während im ersten RFC Managementinformationen für Nameserver spezifiziert werden, geschieht dies im zweiten für die Resolver. Den MIB-Definitionen liegt dabei die zweite Version des Managementprotokolls (SNMPv2) zugrunde.

Wie aus der Abbildung hervorgeht, werden die Managementinformationen für Nameserver in vier Gruppen unterteilt:

dnsServConfig: Nameserverkonfiguration. Die im Bild angegebene Variable `dnsServConfigImplementIdent` dient beispielsweise der Identifikation des eingesetzten Servers.

[6] Das DNS-Management fällt mit dem DNS als verteilt arbeitendes System in den Bereich des Systemmanagements. Betrachtet man die Nameserver als Managed Objects und das DNS-Protokoll als Anwendungsprotokoll der TCP/IP-Familie, so läßt es sich zudem noch in das Anwendungsmanagement einordnen.

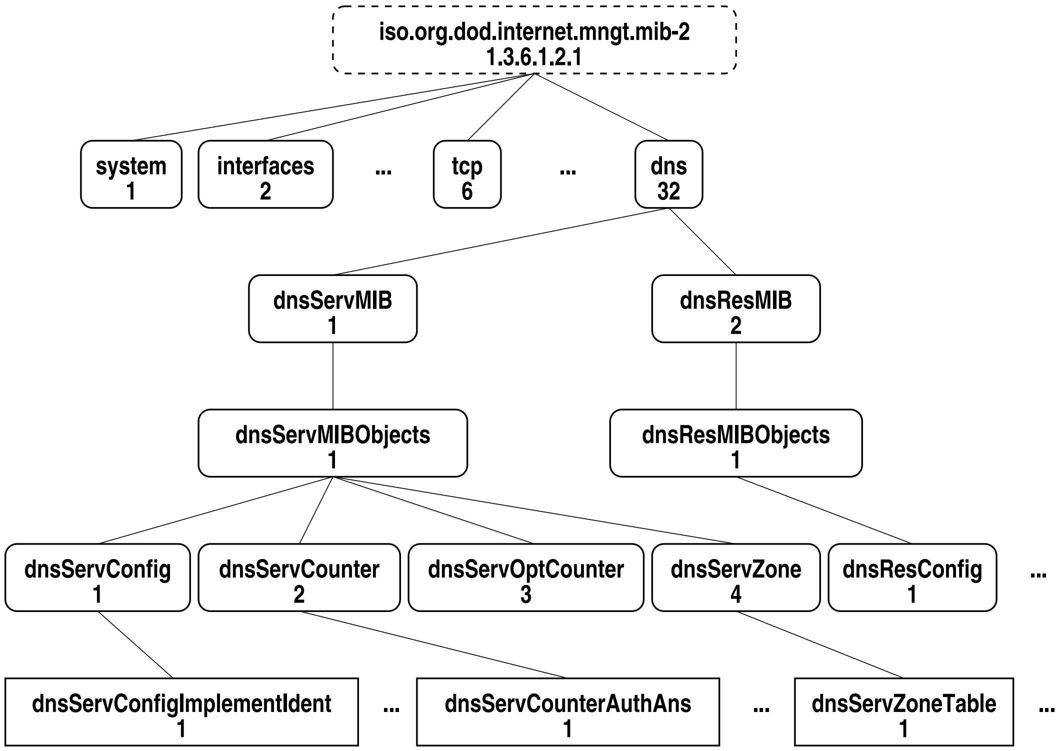

Abbildung 4.2: DNS in der MIB-2 des Internet-Registrierungsbaumes

dnsServCounter: Statistische Angaben über Anfragen, die an einen Nameserver gestellt wurden. Die Variable `dnsServCounterAuthAns` gibt z.B. die Anzahl der DNS-Anfragen an, die von einem Nameserver als Primary oder Secondary beantwortet wurden.

dnsServOptCounter: Im Gegensatz zur letzten Gruppe wird hier noch nach der Herkunft der Anfragen unterschieden.

dnsServZone: Zonenkonfiguration. Die in der Abbildung als Beispiel aufgeführte Variable `dnsServZoneTable` enthält als Tabelle alle Zonen, für die ein Nameserver entsprechende Informationen bereithält.

Bei den Resolvern erfolgt eine Unterteilung in sechs Gruppen, die aber hier mit dem Verweis auf den entsprechenden RFC nicht mehr angegeben werden sollen.

Da weder die aktuellen Nameserver noch die Resolver eine SNMP-Schnittstelle bereitstellen, besteht eine der zukünfig zu erledigenden Aufgaben darin, entsprechende Agenten zu programmieren. Zusammen mit den verfügbaren Managern, sei es als Kommandozeilenversion (**snmpget**, **snmpset**, ...) oder in einer Managementplattform (**HP OpenView**, **Cabletron Spectrum**, ...), kann das

Domain Name System dann im Rahmen des Internet-Managements verwaltet werden.

4.4.2 Anbindung des DNS an ein Dokumentationssystem

Ein Dokumentationssystem dient der Erfassung, Darstellung und Verwaltung der physischen Infrastruktur eines Kommunikationsnetzes [Hege 93]. Betroffen davon ist vor allem das Longterm-Management, also die Verwaltung von Daten, die nur selten geändert werden müssen wie z.B. die Topologie eines Netzes.

Da auch die DNS-Informationen als *statisch* zu betrachten sind und mit den anderen darin enthaltenen Daten teilweise korrelieren, bietet sich deren Integration in ein Dokumentationssystem an.

Am Leibniz-Rechnenzentrum wurde in Form einer Netzdokumentationsdatenbank (neben einem CAD-System) ein Dokumentationssystem realisiert [Valt 92]. Die Aufgabe besteht dabei darin, sämtliche Daten, die die Komponenten, Anwendungen und Systeme in einem Kommunikationsnetz beschreiben, in der Datenbank abzulegen (siehe Abbildung 4.3) mit dem Ziel, ein und dieselbe Information nur an einer Stelle verwalten zu müssen. Dies dient der Vermeidung von Redundanz, einem wesentlichen Vorteil des integrierten Managements.

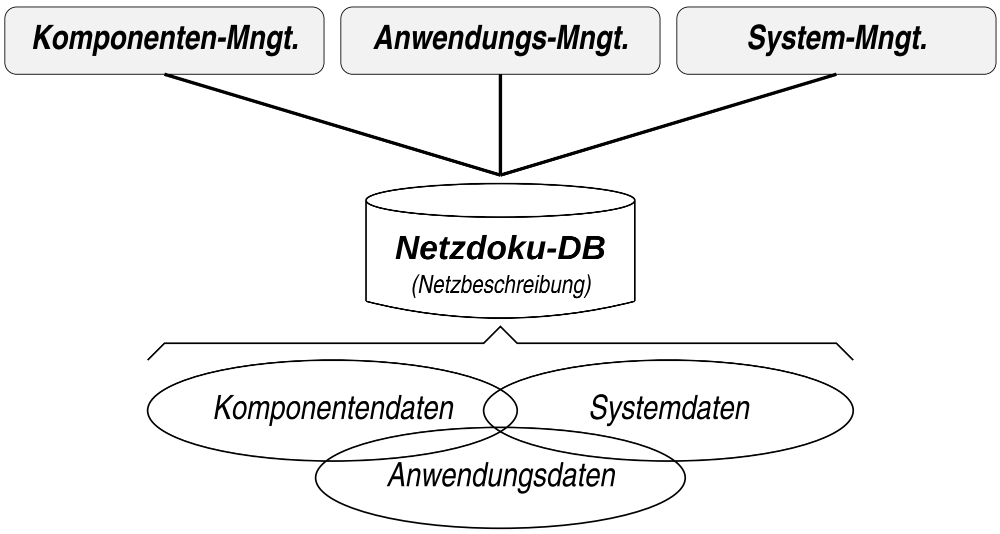

Abbildung 4.3: Netzdokumentationsdatenbank

Die vorliegende Arbeit befaßt sich im weiteren Verlauf mit der angesprochenen

Integration[7], auf deren Grundlage dann ein Verfahren zum DNS-Konfigurationsmanagement erarbeitet wird.

Daneben können dann auch andere Anwendungen davon profitieren. Wird z.B. ein Trouble Ticket System (TTS) im Rahmen des Fehlermanagements eingesetzt, und erfolgt dabei eine Anbindung an das Datenbanksystem der Netzdokumentation, so können auf Basis der darin enthaltenen Information deutlich aussagekräftigere Fehler-Tickets erstellt werden, da neben der reinen Fehlerinformation noch Informationen über das damit verbundene Umfeld präsentiert werden können.

4.5 Anforderungen an eine Umgebung für DNS-Konfigurationsmanagement

Im folgenden werden Forderungen an eine Managementumgebung gestellt, durch die der DNS-Administrator bei seiner Arbeit möglichst gut unterstützt werden soll. Dabei wird auch auf Probleme hingewiesen, sofern sie einer Umsetzung der Forderungen im Wege stehen.

4.5.1 Zentrale Konfiguration verteilter Nameserver

Oftmals betreiben einzelne Organisationen mehrere Nameserver in ihrem Netz. Dabei sollte es dem zuständigen DNS-Administrator ermöglicht werden, daß er den gesamten Nameserver-Verbund von seiner Managementstation aus zentral konfigurieren kann. Ein explizites Remote Login auf den Rechner, auf dem der jeweilige Nameserver läuft, sollte nicht notwendig sein.

Die Nameserver auf den verschiedenen Rechnern benötigen alle lokale Konfigurationsdateien. Diese sollten zentral erstellt und anschließend mittels FTP oder einem anderen geeigneten Verfahren auf die relevanten Rechner verteilt werden. Im Anschluß daran müssen die Nameserverprozesse gestartet bzw. rekonfiguriert werden. Auch dieser Vorgang sollte zentral gesteuert werden können.

4.5.2 Validierung von Änderungen, Lebenszeit

Werden DNS-Informationen kopiert (Secondary Nameserver) oder zwischengespeichert (Cache Nameserver), so ist es günstig, im voraus zu wissen, wielange die von diesen Nameservern ihrerseits zur Verfügung gestellte Information gültig sein werden.

[7] Diese wird auch seitens der Anwender der Netzdokumentationsdatenbank gewünscht [Valt 94].

In den Einträgen der Zonenkonfigurationsdateien sind Zeitinformationen anzugegeben, wielange die jeweiligen Einträge gültig sein sollen (Lebenszeit) und wielange sie darüber hinausgehend maximal verwendet werden dürfen. Diese Zeitspannen werden für jede Zone im SOA-Record festgelegt [RFC 1034]. Daneben kann für jeden sonstigen Record-Eintrag dieser Zonen eine eigene Zeitspanne in Bezug auf seine Gültigkeit angegeben werden (TTL-Parameter).

Beim Management kommt es nun darauf an, dafür geeignete Werte zu finden, wobei man zwischen zwei konkurrierenden Zielen abzuwägen hat:

1. *Kleine Zeitspannen*
   Wählt man kleine Zeitspannen, so wird die Information des Primary Nameservers bei dessen Rekonfiguration in kurzer Zeit an die anderen Nameserver (Slave, Secondary) weitergegeben, so daß auch diese schnell wieder auf dem aktuellen Stand sind und richtige Informationen zur Verfügung stellen.

   Der Nachteil der Verwendung zu kleiner Zeitspannen liegt darin, daß das Netz durch das Domain Name System stärker belastet wird, da zum einen die Secondary Nameserver öfters bei den Primary Nameservern anfragen, ob ihre gespiegelten Zoneninformationen noch aktuell sind (Seriennummer als Zeitstempel), und zum anderen die Cache Nameserver ihre zwischengespeicherten Anfrageergebnisse schneller wieder verwerfen, anstatt damit wiederholt gestellte Anfragen zu beantworten.

2. *Große Zeitspannen*
   Durch größere Zeitspannen läßt sich die durch das Domain Name System verursachte Netzlast deutlich mindern. Schließlich handelt es sich, wie oben schon erwähnt, um weitgehend statische Informationen.

   Sämtliche DNS-Informationen werden von den Primary Nameservern bereit gestellt, und können von den anderen nur über diese bezogen werden, also nicht über Zonendateien, mit denen die Konfiguration von außen her stattfindet. Man hat also unmittelbaren Einfluß nur auf die Primary Nameserver, was das Modifizieren von DNS-Informationen anbetrifft.

   Folglich ergeben sich Probleme bei zu groß gewählten Zeitspannen, wenn Informationen kurzfristig geändert werden müssen, da die alten (inkorrekten) Informationen von den Secondary und Cache Nameservern weiterhin im Netz verteilt werden.

Zu dem genannten Problem gibt es keine optimale Lösung, sondern lediglich Erfahrungswerte, die sich in der Praxis bewährt haben (meistens Zeitspannen von einigen Stunden oder Tagen bis zu einer Woche).

Wünschenswert wäre es, wenn dem DNS-Administrator ein Werkzeug (evtl. innerhalb einer Managamentplattform) zur Verfügung stehen würde, das ihm anhand von Statistiken gute Vorgabewerte aufzeigt, nach denen er sich dann richten kann.

Noch besser wäre es darüberhinaus, wenn aus diesen Werten automatisch die Zeitspannenwerte in den Konfigurationsdateien (Zonendateien) generiert werden würden, möglichst ohne daß der DNS-Administrator sich explizit darum kümmern muß.

4.5.3 Automatische Rekonfiguration bei Änderungen

Hat der DNS-Administrator bzw. das Managementsystem (z.B. nach einer DB-Manipulation) Modifikationen an den Konfigurationsdateien vorgenommen, so müssen die Nameserver rekonfiguriert werden[8].

Bei der Manipulation einer Zone muß deren Seriennummer im SOA-Record in der zugehörigen Zonendatei erhöht und der Primary Nameserver rekonfiguriert werden. An der erhöhten Seriennummer erkennt dann ein Secondary Nameserver, daß er die Zone neu spiegeln muß, wenn die Zeitspanne der Gültigkeit ihrer Daten verstrichen ist.

Im Zusammenhang mit dem geplanten Einsatz der Netzdokumentation liegt das Bestreben nahe, bei Manipulationen in der Datenbank die relevanten Konfigurationsdateien automatisch neu zu erzeugen und die davon betroffenen Nameserver ebenfalls automatisch zu rekonfigurieren, nachdem die Konfigurationsdateien auf die entsprechenden Rechner verteilt wurden.

Die Verwirklichung dieses Bestrebens wirft aber erhebliche Probleme auf, da mit der Datenmanipulation des verwendeten relationalen Datenbanksystems (ORACLE) keine Seiteneffekte (Anstoß eines geeigneten Prozesses) möglich sind.

Dieser Nachteil könnte mit dem Einsatz objektorientierter Datenbankensysteme behoben werden, wenn man dort entsprechende Methoden programmieren kann. Objektorientierte Datenbanksysteme [Heue 92] sollen u.a. beim OSI-Management zum Einsatz kommen. Derzeit sind aber die Voraussetzungen hierfür noch nicht erfüllt. Es existieren noch keine normierten Standards wie zum Beipiel die Abfragesprache SQL im relationalen Bereich.

Eine Lösung, die mit dem jetzigen System realisierbar ist, ist eine Manipulation der Datenbank über ein geeignetes Werkzeug innerhalb einer Managementplattform, das als Nebeneffekt den gewünschten Prozeß anstößt. Dann muß aber sichergestellt werden, daß eine Manipulation durch andere Stellen (Beispiel: *sql*plus* als

[8] Die Nameserver rekonfigurieren ihre interne Datenbank dynamisch, wenn man den betroffenen Nameserverprozessen ein SIGHUP-Signal schickt [Vixi 93].

interaktives Abfrage- und Manipulationswerkzeug von ORACLE) unterbleibt, da sonst Inkonsistenzen zwischen der externen Datenbank auf der einen Seite und den Konfigurationsdateien bzw. den Nameserver-internen Datenbanken auf der anderen Seite auftreten können.

Dabei sollte aber auch bedacht werden, daß die Netzdokumentationsdatenbank nicht nur für das Domain Name System, sondern auch für viele andere Bereiche eingesetzt wird (Komponentenmanagement etc.), wobei jedesmal komplexe Werkzeuge zur Datenbankmanipulation zu erstellen wären, die alle möglichen Nebeneffekte berücksichtigen.

Der zu erwartende Aufwand steht damit in keinem Verhältnis mehr zu dem Vorteil, den eine automatische Rekonfiguration mit sich zieht. Deshalb wird diese Forderung in der vorliegenden Arbeit nicht weiter berücksichtigt.

Kapitel 5

DNS-Konfiguration mit der Netzdokumentationsdatenbank

Im folgenden soll der zweite im vorangehenden Kapitel präsentierte Ansatz für die Integration des DNS-Managements in das Netz- und Systemmanagement, die Anbindung des Domain Name Systems an ein Dokumentationssystem (Kapitel 4.4.2), zum Zwecke der Konfiguration realisiert werden. Dazu dient die am Leibniz-Rechenzentrum installierte Netzdokumentationsdatenbank.

Das im Rahmen dieser Arbeit neu zu entwickelnde Verfahren stellt eine Alternative zur manuellen Konfiguration (vgl. Kapitel 3.2) bzw. zu dem isolierten Werkzeug *df2hf* (Kapitel 4.3.3) dar. Dabei werden sämtliche relevanten DNS-Informationen in der Datenbank abgelegt und hieraus die erforderlichen Konfigurationsdateien automatisch generiert.

Alle anfallenden Neueintragungen/Änderungen sind damit an *einer* Stelle in der Netzdokumentationsdatenbank vorzunehmen. Dadurch wird Redundanz vermieden, und gleichzeitig eine größere Konsistenz jener Informationen, die mehrere Managementbereiche betreffen, gewährleistet. Es müssen insgesamt weniger Managementinformationen verwaltet werden, wodurch nicht zuletzt auch der Komfort, den eine Managementumgebung bieten soll, gesteigert wird.

Bevor damit begonnen werden kann, ein geeignetes Datenbankschema für das Domain Name System zu entwickeln, muß man sich zunächst mit dem schon vorhandenen und den darin enthaltenen Relationen auseinandersetzen, um eine möglichst gute Integration gewährleisten zu können (Abschnitt 5.1).

Unabhängig davon werden im nächsten Abschnitt 5.2 die Anforderungen, die sich an das Design der Datenbank für die neue Aufgabe richten, anhand der vom Management betroffenen Objektklassen Hosts, Zonen, Nameserver und Resolver analysiert.

Im folgenden Abschnitt 5.3 wird ein Entity/Relationship Modell präsentiert, das das aus den Anforderungen entwickelte Design wiedergibt. Daraus wird das neue Relationenschema hergeleitet, noch immer unabhängig von der vorhandenen Datenbank.

Eine Migration der alten und neuen Datenbank findet erst auf Relationenebene statt (Abschnitt 5.4). Würde man diesen Schritt schon auf Designebene vollziehen, so wären die Auswirkungen auf die Struktur der vorhandenen Relationen so gravierend, daß alle existierenden Anwendungen nicht mehr verwendet bzw. weitgehend neu erstellt werden müßten. Aber auch auf Relationenebene ist noch darauf zu achten, daß an den alten Relationen keine tiefgreifenden Änderungen vorgenommen werden dürfen, und bei den neuen die Erfordernisse des DNS nicht außer acht gelassen werden.

Schließlich werden in Abschnitt 5.5 noch einige organisatorische Probleme diskutiert, die mit dem neuen Verfahren einhergehen. Damit muß man sich auseinandersetzten, bevor man daran geht, das Verfahren zu implementieren, was im nächsten Kapitel geschehen soll.

5.1 Die vorhandene Netzdokumentationsdatenbank

Die Netzdokumentationsdatenbank am Leibniz-Rechenzentrum wird in [Valt 92] beschrieben. Diese Beschreibung bezieht sich auf eine nunmehr über zwei Jahre alte Version. Eine Beschreibung der aktuellen Version ist derzeit nicht verfügbar. Da sich aber die zwischenzeitlichen Modifikationen im Bezug auf die für unsere Zwecke relevanten Stellen in Grenzen halten, kann sie hierfür herangezogen werden.

5.1.1 Übersicht

Aus der Beschreibung kann entnommen werden, daß bisher insbesondere Informationen für das Komponentenmanagement als auch allg. Verwaltungsdaten (räumlich und personell) zur Verfügung gestellt werden. Beispielsweise werden Datenendeinrichtungen wie Terminals, PC's, Workstations, Datenübertragungseinrichtungen wie Repeater, Bridges, Modems, Tranceiver oder auch „passive" Elemente wie Koaxsegmente und Cheapernetsegmente als Komponenten unterstützt. Daneben stehen verschiedene Leitungs- und Schnittstellentypen zur Verfügung. Die Festlegung der Typen erfolgt in Strukturtabellen.

Die Abbildung 5.1[1] gibt einen Überblick über die Informationstabellen. Die eingezeichneten Pfeile stellen die Beziehungen zwischen den einzelnen Relationen (Fremdschlüssel → Primärschlüssel) dar.

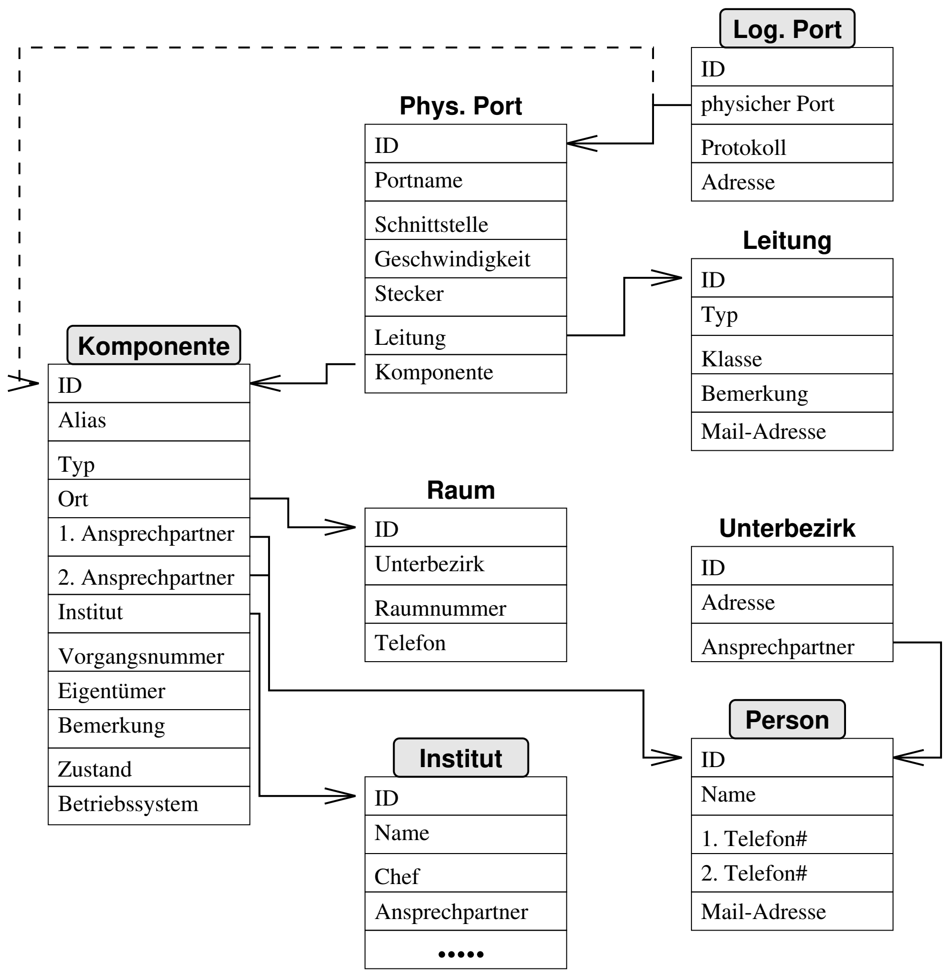

Abbildung 5.1: Schema der bestehenden Datenbank

5.1.2 DNS-relevante Relationen

Bei den für das Domain Name System relevanten Relationen sind die Namen in der Abbildung grau hinterlegt. Auf diese wird im folgenden eingegangen:

KOMPONENTE:

Einige der verwalteten Komponentenklassen wurden bereits erwähnt. Für

[1] Entwurf: Dr. Robert Valta, LRZ München

das Domain Name System sind nur solche interessant, die über Schnittstellen mit IP-Protokoll verfügen:

```
select distinct kl.klasse, kl.name
from komponentenklasse kl, komponente ko, logport lp
where kl.klasse=ko.klasse and
      ko.name=lp.komponente and lp.protokoll='IP';
```

Als Ergebnis dieser Anfrage erhalten wir derzeit[2] folgende Klassen:

```
KL NAME
-- ------------------------------
AS ATM-Switch
BI Bridge
BR Brouter
DI CDCNET-DI
DR Drucker (incl. Laserdrucker)
FK FDDI-Konzentrator
MC Macintosh
PC PC
PL Plotter (incl. DIA-Geraet)
RE Rechner
RO Router
SK Sternkoppler
TH Twisted Pair Hub
TS sonstige Terminalserver
UV X.25-Knoten
WS Workstation
XW XWindow Terminal
```

Dabei ist zu beachten, daß in der Praxis nicht alle Komponenten der hier aufgeführten Klassen einen Host- und Domainnamen erhalten, sondern vor allem die Datenendeinrichtungen wie PC's oder Workstations, aber auch die Koppelelemente der IP-Ebene.

LOGPORT:

Bei den logischen Ports (Adressen) interessieren uns jene der Klasse IP. Interessant dabei ist, wie im Attribut ADRESSE die IP-Adresse dargestellt wird.

Optimal hierzu wäre ein 32-Bit Wort (ohne Vorzeichen), das sich auch byteweise adressieren läßt. Damit hätte man die IP-Adresse sowohl als ganzes

[2] Stand: 27.07.1994

(nützlich für Joins, da dann nur jeweils ein Vergleichsattribut zu bearbeiten ist) als auch komponentenweise (wichtig für Subnetzbildung, partieller Vergleich statt Maskenbildung, und Bildung der umgedrehten Adresse für das Reverse Mapping).

Ein derartiger wahlweiser Zugriff auf das Attribut stellt in der Programmiersprache C (und in einigen anderen) kein Problem dar, auch nicht bei der Verwendung hierarchischer Datenbanken[3].

Hier wird aber das relationale Datenbanksystem ORACLE eingesetzt. Im Bereich der relationalen Datenbanken fordert bereits die erste Normalform (1NF), daß sämtliche Attribute atomar sein müssen. Ein Zugriff auf Teile eines Attributs ist damit ausgeschlossen, ebenso zusammengesetzte Attribute. Geeignete Selektorfunktionen sind nicht vorhanden im Datenbanksystem.

Was an Möglichkeiten zur Darstellung einer IP-Adresse also bleibt, sind zum einen die Verwendung eines Strings oder einer „langen" Zahl und zum anderen die Aufspaltung auf vier Attribute, jeweils eins für jedes Byte. Desweiteren kommt als Einschränkung bei den Zahlen hinzu, daß nur n-stellige Dezimalzahlen als Typen bei ORACLE zur Verfügung stehen.

In der derzeitigen Implementierung wird folgendes Verfahren angewendet: Verwendung eines ASCII-Strings mit maximaler Länge 50. Darin wird linksbündig die IP-Adresse eingetragen, wobei jeweils genau drei Zeichen ein Byte (dezimal) darstellen und durch einen Punkt getrennt werden. Ist ein Bytewert kleiner als 100 bzw. als 10, so sind die führenden Stellen durch Leerzeichen (Spaces) aufzufüllen (Bsp.: 131.159.␣␣4.␣10).

Dieses Verfahren bietet den Vorteil, daß die IP-Adresse komplett als ein Attribut dargestellt wird (wenngleich dies durch die Länge wieder relativiert wird), und daß wahlweise unterschiedliche Adreßtypen in ein und demselben Attribut aufgenommen werden können.

Es stößt aber auch bei vielen Anwendern auf Kritik, da sie zum einen die IP-Adressen oft nicht finden (gemischte Darstellung mit anderen Adreßtypen), zum anderen die strengen Konventionen als umständlich empfinden [Valt 94].

Als Alternative würde es sich anbieten, für jedes Byte der IP-Adresse je ein Attribut mit einer dreistelligen Dezimalzahl (ohne Nachkommastellen, in ORACLE also NUMBER(3,0))zu verwenden. Damit wären dann keine besonderen, strengen Konventionen verbunden, sondern man müßte lediglich darauf achten, daß tatsächlich auch ein Bytewert (0...255) vorliegt. Diese Bedingung müßte bei der implementierten ORACLE-Version 6 manuell

[3] Hierarchische Datenbanken (z.B. IMS von IBM) lassen einen Zugriff an zwei verschiedenen Stellen auf das gleiche Attribut zu (überlappende Fields), was bei der Migration in die relationale Welt erhebliche Probleme bereitet [Baye 94].

überwacht werden.

Ein entscheidendes Problem bei Neuentwicklungen und Erweiterungen von Produkten in Bezug auf die Akzeptanz des Anwenders nicht nur in der Informatik stellt die Kompatibilität zu Vorhandenem dar. Aus diesem Grund soll im Rahmen dieser Arbeit nicht die letzgenannte, neue Methode zur Darstellung von IP-Adressen verwendet werden, sondern an der bisherigen festgehalten werden.

INSTITUT:
Unterschiedliche Institute können ihre Rechner auf entsprechende Subdomains (Zonen) aufteilen. Bei der Implementierung des Domain Name Systems in die Netzdokumentation ist also darauf zu achten, daß eine solche Zuordnung ermöglicht wird.

PERSON:
In dieser Relation sind die Mitarbeiter und wichtige Ansprechpartner des Leibniz-Rechenzentrums mit ihren E-Mail Adressen – sofern vorhanden – gespeichert, also auch die Verantwortlichen für das Netzmanagement. Sie wird benötigt, da bei den DNS-Zonenkonfigurationsdateien im SOA-Record die E-Mail Adresse des Verantwortlichen anzugeben ist.

Dabei sollten neben natürlichen Namen auch Verwaltungsnamen (admin, postmaster etc.) in die Relation eingetragen werden, da die Zuständigkeit wechseln kann. Das hat den positiven Nebeneffekt, daß man z.B. den DNS-Administrator erreichen kann, ohne seinen Namen zu kennen (Forderung in [Valt 94]).

5.1.3 Kritik an den vorhandenen Relationen

Was bei den Relationen im allgemeinen auffällt, ist die häufig darin vorhandene Redundanz. Beispielsweise werden in der Relation PERSON Vor- und Nachnamen nicht nur im Schlüssel NAME, sondern auch nochmal getrennt in den Attributen VORNAME und NACHNAME gespeichert.

Diese Doppelspeicherung kostet nicht nur unnötig Speicherplatz, sondern birgt auch die Gefahr schwerer Konsistenzprobleme, da sich leicht unbeabsichtigt widersprüchliche Tupel generieren lassen, und sollte deshalb in zukünftigen Versionen entfernt werden.

Was in diesem Zusammenhang als besonders unangenehm zu werten ist, ist der aus der Geräteklasse (1. und 2. Zeichen), dem Unterbezirk (3. und 4. Zeichen) sowie der Gerätenummer (5. bis 8. Zeichen) zusammengesetzte Primärschlüssel[4] NAME der Relation KOMPONENTE.

Nun ist ein weiteres Attribut (KLASSE) der gleichen Relation, dessen Belegung mit der des obigen Attributs Geräteklasse identisch ist, vom letztgenannten funktional abhängig (Identität).

Damit ist ein Attribut vorhanden, das vom Schlüssel nicht voll funktional abhängig ist, sondern bereits von einem Teil davon. Folglich genügt die Relation KOMPONENTE nicht der zweiten Normalform (2NF). Damit ist ferner auch die Erfüllung der dritten Normalform (3NF) ausgeschlossen [Frei 93].

5.2 Anforderungen an das Datenbankdesign

Abbildung 5.2 gibt einen Überblick über die vier Objektklassen, die Gegenstand des Konfigurationsmanagements sind. Dafür sind in der Datenbank die notwendigen Mangementinformationen abzulegen.

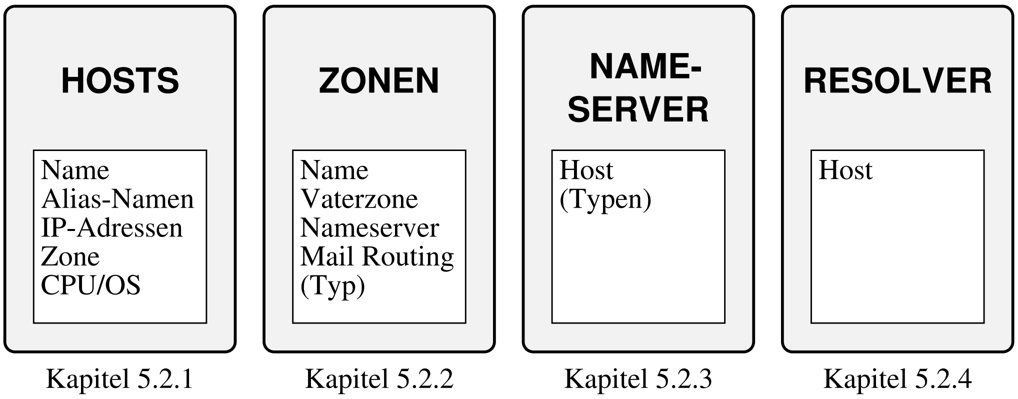

Abbildung 5.2: Zu managende Objektklassen

In den folgenden Unterkapiteln werden spezielle Anforderungen an diese Klassen gestellt, auf denen das Datenbankdesign aufbaut.

[4] Da die Zusammensetzung orthogonal erfolgt (laut Definition), handelt es sich um einen zusammengesetzten Schlüssel, wenngleich er auch in der SQL-Definition als nur ein Attribut festgelegt wird.

5.2.1 Hosts

Unter „Host“ wird im folgenden eine Rechnereinheit verstanden, die *genau einen* Namen (Hostnamen) besitzt[5].

Anforderungen, die sich an das Objekt „Host“ stellen:

Zonenzuordnung:
: Jeder Host *muß* genau einer Zone zugeordnet sein.

IP-Adressen:
: Jeder Host *kann* mehrere IP-Adressen besitzen, *muß* jedoch mindestens eine besitzen.

Alias-Namen:
: Jeder Host *kann* einen oder mehrere Alias-Namen besitzen.

CPU und OS:
: Angaben über CPU und Betriebssystem eines Hosts *können* zur Verfügung gestellt werden (nur paarweise).

Services:
: Angaben über Netzdienste (vgl. */etc/services*) *können* zur Verfügung gestellt werden.

5.2.2 Zonen

Der Begriff „Zone“ wurde bereits in Kapitel 2.3.2 definiert.

Nun muß unterschieden werden, ob es sich um eine selbst- oder fremdverwaltete Zone handelt. Angaben über fremdverwaltete Zonen sind erforderlich, wenn Subdomains eigener Domains delegiert werden sollen und somit eigenständige Zonen darstellen[6], oder wenn fremde Zonen gespiegelt werden sollen[7]. Die folgenden Anforderungen beziehen sich auf selbstverwaltete Zonen (Ausnahme: Siehe Text).

---

[5] Es ist möglich, daß eine Zentraleinheit über zwei oder mehrere Hostnamen verfügt (Bsp. am LRZ: dfvgate/suncollect). In diesem Fall werden im Rahmen der vorliegenden Arbeit dafür zwei der genannten Objekte verwendet. Aliasnamen werden bei der Definition nicht berücksichtigt.

[6] Bsp.: **tu-muenchen.de** wird vom LRZ verwaltet, während **informatik.tu-muenchen.de** von der Rechnerbetriebsgruppe der Informatik an der TUM verwaltet wird. Zum Delegieren muß das LRZ aber neben dem Namen auch die Nameserver dafür kennen.

[7] In diesem Fall ist neben dem Namen die Kenntnis des Primary Nameservers erforderlich.

Anforderungen, die sich an das Objekt „Zone“ stellen:

Vaterzone:
: Zu jeder Zone (auch die delegierten fremdverwalteten Subdomains) *muß* explizit die Vaterzone angegeben werden, sofern diese selbstverwaltet wird. Dies allein anhand des Namens erkennen zu wollen, wäre riskant oder zumindest äußerst umständlich.

  Sind z.B. die drei Zonen **vater**, **sohn.vater** und **enkel.sohn.vater** vorhanden, so erfolgt das Delegieren gewöhnlich stufenweise. Fehlt nun bei dieser Form des Delegierens die Zone **sohn.vater**, so kann die Zone **enkel.sohn.vater** nie erreicht werden. Läßt man dagegen ein Delegieren über mehr als einen Knoten zu, also **enkel.sohn.vater** als delegierte Zone von **vater**, so wäre **enkel.sohn.vater** zwar wieder erreichbar, im Falle des Vorhandenseins von **sohn.vater** wäre Baumstruktur des Namensraumes aber zerstört.

  Es wäre also erforderlich, zwei unterschiedliche Strategien beim Delegieren anzuwenden, je nachdem, ob dazwischen noch eine Zone vorhanden ist oder nicht. Durch die explizite Angabe der Vaterzone kann dem Generierungsprogramm dagegen eine solide Entscheidungshilfe beim Delegieren zur Verfügung gestellt werden.

Primary Nameserver:
: Jede Zone *muß* von einem lokalen Primary Nameserver verwaltet werden.

Mail Routing:
: Für jede Zone *können* Einträge für das Mail Routing erstellt werden[8]. Die hierzu notwendige Information wird manuell in die Datenbank eingegeben. Als Mail Exchanger *muß* einer der verwalteten Hosts dienen.

Desweiteren soll es möglich sein, daß im Falle der Angabe von Zonen, die die Endung **in-addr.arpa** besitzen, die zugehörigen Resource Records für das Reverse Mapping automatisch generiert werden. Für den Fall, daß eine IP-Adresse zu den Masken zweier verschiedener Zonen paßt, ist sie der spezielleren (delegierten) der beiden zuzuordnen.

Eine Ausnahme hiervon stellt die Möglichkeit dar, daß eine IP-Adresse eines lokalen Hosts in die fremdverwaltete Zone eines Subnetzes gelangen könnte. In diesem Fall könnte also nicht für das Vorhandensein eines korrekten Eintrages garantiert werden. Deshalb muß letzterer (auch) in die selbstverwaltete (Vater-) Zone eingetragen werden.

[8] Das Mail Routing [RFC 974] stellt ein eigenständiges Themengebiet dar. Dessen Integration in die Netzdokumentation könnte zum Gegenstand einer seperaten Arbeit gewählt werden.

Mit dem Reverse Mapping eines Internet Class-A Netzes ergibt sich bei Verwendung einer 16Bit Subnetzmaske auch ein Beispiel für die oben erwähnte direkte Delegation der Zone **vater** zur Zone **enkel.sohn.vater**, wobei **vater** sich aus dem Bytewert der Class-A Netzes Maske und der Endung **in-addr.arpa** zusammensetzt, während **enkel.sohn** aus den zwei Bytewerten der Subnetzmaske (in umgekehrter Reihenfolge) besteht.

Im Zusammenhang mit den Zonen stellt sich natürlich auch die Frage, ob es innerhalb *einer* Managementumgebung, für die das Konzept ausgelegt ist, Sinn macht, mehrere Zonen (jeweils für das Forward bzw. Reverse Mapping) zu verwalten. Beispielsweise wird die Domain **uni-erlangen.de** mit all ihren Subdomains als eine einzige Zone verwaltet, es findet also kein Delegieren statt.

Ein wichtiges Argument für die Notwendigkeit, mehrere Zonen verwalten zu können, ist die Aufgabe, disjunkte Domains verwalten zu müssen. Als Beispiel kann das Leibniz-Rechenzentrum in München aufgeführt werden, das die Domains **fh-muenchen.de**, **lrz-muenchen.de**, **tu-muenchen.de** (teilweise) sowie **uni-muenchen.de** verwaltet. Dagegen wird die Domain **de**, die neben der Root Domain als einzige alle genannten Domains einschließen könnte, von einer anderen Organisation verwaltet. Es ist dem LRZ also nicht möglich, mit nur einer Zone auszukommen.

Weiterer Vorteile, die für eine feinere Granularität bei der Zonenbildung sprechen:

- „Kleinere“[9] Zonen sind übersichtlicher.
- Es erweist sich oftmals als günstig, „größere“ Subdomains an die entsprechenden Organisationen selbst zu delegieren, da dann im Falle häufiger Änderungen permanente Absprachen entfallen.
- Schließlich ist die Netzlast beim Spiegeln „kleinerer“ Zonen (nach Manipulationen darin) geringer, als wie wenn jedesmal alle vorhandenen Daten übertragen werden müssen.

Trotz der genannten Vorteile der Bildung mehrerer Zonen muß die Möglichkeit bestehen, auch innerhalb einer Zone Subdomains zu bilden, denen die einzelnen Hosts angehören. Folglich werden von der zu erstellenden Datenbank *beide* Vorgehensweisen unterstützt.

5.2.3 Nameserver

Unter „Nameserver“ wird im folgenden ein entsprechender Prozeß verstanden, der als Daemon (meistens: *named*) auf einem Host läuft und DNS-Informationen auf Anfrage bereitstellt.

[9] Mit „kleiner“ (bzw. „größer“) ist hier der Umfang der Einträge in den Zonen gemeint.

Anforderungen, die sich an das Objekt „Nameserver“ stellen:

Lokaler Host:
: Ein lokaler[10] Nameserver *muß* auf einem lokalen Host laufen. Es macht keinen Sinn, einen Nameserver verwalten zu wollen, der auf einem Host läuft, auf den man keinen Zugriff hat.

Primary Service:
: Ein Nameserver *kann* eine oder mehrere lokale Zonen als Primary Nameserver verwalten.

Secondary Service:
: Ein Nameserver *kann* eine oder mehrere Zonen (lokale und fremde) als Secondary Nameserver verwalten.

Loopback Interface:
: Die Abbildung des Namens **localhost** auf die IP-Adresse **127.0.0.1** und umgekehrt ist sets zu implementieren, weshalb hierfür in der Datenbank kein eigener Eintrag vorzusehen ist.

Root-Domain Unterstützung:
: Dasselbe gilt auch für den Cache Service der Root Domain, der auf jedem Nameserver benötigt wird. Allerdings soll bei der Auswahl der Quelle der benötigten *named.boot*-Datei, die mittels Anoymous-FTP bezogen wird, freie Wahl bestehen. Hierfür sind Einträge in der Datenbank vorzusehen.

Slave Nameserver Unterstützung:
: Es soll möglich sein, einen Nameserver als Slave Nameserver zu betreiben. Neben dem Attribut, ob ein Nameserver als Slave arbeitet, sind die hierzu notwenigen Einträge für die IP-Adressen, die die Nameserver angeben, an die die Anfragen weitergeleitet werden (Forwarding), in der Datenbank vorzusehen.

Auch bei den Nameservern muß man sich mit der Frage auseinandersetzen, ob es sinnvoll ist, mehr als einen innerhalb einer Managementumgebung zu verwalten. Für den Secondary Service muß aber mindestens ein weiterer Nameserver zur Verfügung gestellt werden. Auch wenn dieser nicht unbedingt Bestandteil der lokalen Managementumgebung sein muß, so sollte zumindest die Möglichkeit dazu bestehen.

Da für Nameserver, die neben dem Secondary Service nur den trivialen Primary Service (für das Loopback Interface) und den Cache Service (Root-Domain) zur

[10] Unter „lokal“ wird hier verstanden, daß das jeweilige Objekt innerhalb der eigenen Managementumgebung verwaltet wird.

Verfügung stellen bzw. als Slave Nameserver arbeiten, ein sehr viel geringerer Managementaufwand vonnöten ist, was die Konfiguration betrifft, läßt sich die Überlegung anstellen, ob für diese überhaupt Einträge in der Netzdokumentationsdatenbank vorzusehen sind. Würde man darauf verzichten, so hätte man nur einen Nameserver in der vorliegenden Managementumgebung (Konfiguration).

Folglich kann man die gestellte Frage auch darauf beschränken, ob der Primary Service für die zu verwaltenden Zonen stets auf einem Nameserver konzentriert oder ggf. auch auf mehrere aufgeteilt werden soll. Es soll dem jeweiligen DNS-Administrator selbst überlassen werden, wie er im Einzelfall vorgehen möchte. Dazu sind von Seiten der Datenbank wiederum *beide* Vorgehensweisen zu unterstützen.

5.2.4 Resolver

Unter „Resolver" werden im folgenden Betriebsystemroutinen verstanden, die als Clients auf jedem Host DNS-Anfragen an die Nameserver weiterschicken.

Für jeden der vorkommenden Hosts wird eine Resolverkonfigurationsdatei erstellt. Dabei sind folgende Einschränkungen zu beachten:

Betriebssystem des Hosts:
: Die erzeugte Konfigurationsdatei wird für Resolver auf Unix-Systemen (u.a. SunOS, HP-UX, AIX, EPIX, BSD, Linux) ausgelegt. Resolverroutinen anderer Betriebssysteme (ggf. auch anderer Unix-Varianten) *könnten* hiervon abweichende Konfigurationsverfahren erfordern. Die erzeugte Konfigurationsdatei kann hierzu aber eine Hilfestellung geben.

Verteilung auf die einzelnen Hosts:
: Damit die jeweiligen Dateien auch tatsächlich auf die einzelnen Hosts verteilt werden können, muß der Zugriff auf die hierfür notwendige Stelle im jeweiligen Dateisystem (meistens: Datei */etc/resolv.conf*) gewährleistet sein. Dazu müssen entsprechende Rechte verliehen werden. Kann dieser Forderung nicht nachgekommen werden, so kann die Datei als E-Mail verschickt und von dem jeweiligen Systemverwalter installiert werden.

Keine vollständige Resolverkonfiguration:
: Die Konfigurationsdatei dient nur dem Zweck, den DNS-Domainnamen[11] und die anzusprechenden Nameserver festzulegen. Spezielle Konfigurationsdaten für die Resolver (z.B. Ob und wenn ja wann die Datei */etc/hosts* zur Namensauflösung verwendet wird) werden nicht in die Datenbank integriert und können folglich auch nicht daraus gewonnen werden.

[11]Dieser entspricht *nicht* unbedingt demjenigen, der unter Verwendung von NIS von dem Kommando *domainname* ausgegeben wird.

5.3 Das Datenbankdesign

5.3.1 Entity/Relationship Modell

Abbildung 5.3 zeigt das unter Berücksichtigung der genannten Anforderungen entwickelte Entity/Relationship Modell.

Zentraler Gegenstand des Modells sind die grau hinterlegten Entities, die im wesentlichen die zuvor genannten Objektklassen widerspiegeln. Für die Objektklasse Resolver wird kein eigenständiges Entity benötigt, da für jeden Host eine entsprechende Konfigurationsdatei erstellt wird.

Die Symbole (Beziehungstypen) in der Abbildung sind an die in [Teor 90] verwendeten angelehnt, weichen aber in dem Punkt ab, daß die obligatorische Mitgliedschaft zu einer Beziehung durch einen Punkt bei dem betroffenen Objekt (Entity) dargestellt wird (siehe Legende).

Im folgenden wird der Aufbau des Modells, d.h. die einzelnen Objekte und Beziehungen beschrieben. Dazu sind sie in der Abbildung entsprechend durchnumeriert. Gleichzeitig sollen bereits die Strukturen für die Umsetzung des Modells in ein Relationenschema in dem Maße angegeben werden, wie sich die Grundlagen hierfür *unmittelbar* aus der Beschreibung ergeben[12].

Die endgültigen, vollständigen Tabellen mit den darin enthaltenen Attributen (SQL-Syntax) werden im Rahmen der Implementierung angegeben. Die dortigen Angaben richten sich vor allem auch an den DNS-Administrator, der das zu erstellende Generierungswerkzeug zusammen mit der Datenbank einsetzen will.

5.3.2 Die Objekte (Entities) und deren Umsetzung in Tabellen

Die in Klammern angegebenen Tabellennamen beziehen sich auf die spätere Implementierung.

1. Hosts
   Die zugehörige Objektklasse wurde bereits beschrieben. Hierfür ist eine Tabelle (HOSTS) zu erstellen.

2. IP-Adressen
   Zu einem Host gehörige IP-Adressen. Hierfür ist eine Tabelle (IPADDR) zu erstellen.

[12] Würde man dies getrennt angeben, so müßte man dafür das E/R-Modell ein zweites Mal beschreiben.

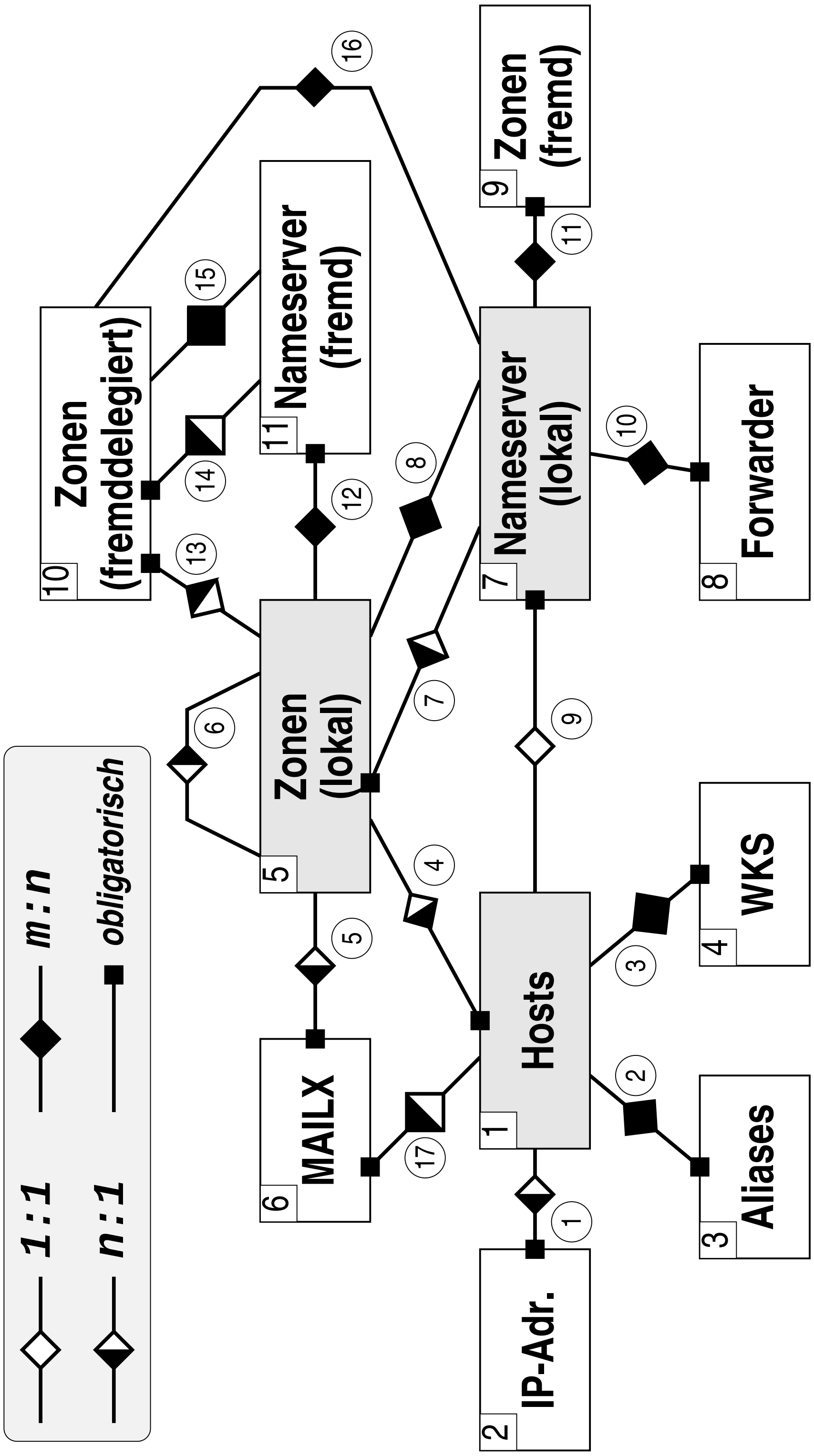

Abbildung 5.3: Entity/Relationsship Modell

3. ALIAS-NAMEN
   Zu einem Host gehörige Alias-Namen. Hierfür ist eine Tabelle (ALIASES) zu erstellen.

4. WELL KNOWN SERVICES
   Von einem Host angebotene Netz-Dienste (vgl. */etc/services*). Hierfür ist eine Tabelle (WKS) zu erstellen.

5. LOKALE ZONEN
   Zonen, die innerhalb der eigenen Managementumgebung verwaltet werden. Hierfür ist eine Tabelle (ZONES) zu erstellen.

6. MAIL EXCHANGER
   Mail-Routing Einträge für eine Zone. Hierfür ist eine Tabelle (MAILX) zu erstellen.

7. LOKALE NAMESERVER
   Nameserver, die innerhalb der eigenen Managementumgebung verwaltet werden. Hierfür ist eine Tabelle (NAMESERVERS) zu erstellen. Weiterhin ist eine eigene Tabelle (NAMESERVERS_SLAVE) zu erstellen, die für alle Nameserver, die als Slave arbeiten, deren Identifikatoren enthält.

8. FORWARDER IP-ADRESSEN
   IP-Adressen, um DNS-Anfragen von einem (Slave) Nameserver an andere weiterzuleiten. Hierfür ist eine eigene Tabelle (FORWARDERS) zu erstellen.

9. FREMDE ZONEN (NICHT DELEGIERT)
   Fremde Zonen, die von einem lokalen Nameserver gespiegelt werden, aber *nicht* von lokalen Zonen delegiert werden. Hierfür ist eine eigene Tabelle (SECONDARY_LF) zu erstellen.

10. FREMDE ZONEN (DELEGIERT)
    Fremde Zonen, die von lokalen Zonen delegiert werden. Hierfür ist eine eigene Tabelle (DEL_ZONES) zu erstellen.

11. FREMDE NAMESERVER
    Fremde Nameserver, die eigene Zonen spiegeln und/oder delegierte Zonen verwalten. Die Einträge werden auf verschiedene Tabellen (DEL_ZONES, DEL_ZONES_SEC und SECONDARY_FL) aufgeteilt, je nachdem, ob der Nameservice Primary oder Secondary für eine delegierte Zone bzw. Secondary für eine lokale Zone ist.

5.3.3 Die Beziehungen (Relationships) und deren Integration in die Tabellen

1. IP-ADRESSEN ↔ HOSTS (**N:1**)
   Jeder Host hat eine oder mehrere IP-Adressen. Die Tabelle IPADDR wird um den Primärschlüssel der Tabelle HOSTS als Fremdschlüssel erweitert.

2. ALIAS-NAMEN ↔ HOSTS (**M:N**)
   Jeder Host kann einen oder mehrere Alias-Namen haben. Da ein Alias-Name für mehrere Hosts (in verschiedenen Domains) verwendet werden kann, liegt eine **M:N**-Beziehung vor. Trägt man den gleichen Alias-Namen aber mehrmals in die Tabelle ALIASES ein, so genügt es, diese um den Primärschlüssel der Tabelle HOSTS als Fremdschlüssel zu erweitern.

3. WELL KNOWN SERVICES ↔ HOSTS (**M:N**)
   Für jeden Host können mehrere Services angegeben werden. Mit dieser **M:N**-Beziehung verhält es sich analog zu der eben genannten. Die Tabelle WKS wird um den Primärschlüssel der Tabelle HOSTS als Fremdschlüssel erweitert.

4. HOSTS ↔ LOKALE ZONEN (**N:1**)
   Jeder Host muß einer lokalen Zone angehören. Zu einer lokalen Zone können mehrere Hosts gehören. Die Tabelle HOSTS wird um den Primärschlüssel der Tabelle ZONES als Fremdschlüssel erweitert.

5. MAIL EXCHANGER ↔ LOKALE ZONEN (**N:1**)
   Für jede Zone können mehrere Mail-Routing Einträge angegeben werden. Die Tabelle MAILX wird um den Primärschlüssel der Tabelle ZONES als Fremdschlüssel erweitert.

6. LOKALE ZONEN ↔ LOKALE ZONEN (SOHN-VATER-BEZIEHUNG) (**N:1**)
   Jede lokale Zone (Domain) kann mehrere deligierte lokale Zonen haben. Die Tabelle ZONES wird um ihren eigenen Primärschlüssel als Fremdschlüssel (Attribut *parent*) erweitert. Hat eine Zone keine lokale Vaterzone, so wird *parent* mit NULL belegt.

7. LOKALE ZONEN ↔ LOKALE NAMESERVER (PRIMARY SERVICE) (**N:1**)
   Lokale Zonen werden durch einen lokalen (Primary) Nameserver verwaltet. Die Tabelle ZONES wird um den Primärschlüssel der Tabelle NAMESERVERS als Fremdschlüssel erweitert.

8. LOKALE ZONEN ↔ LOKALE NAMESERVER (SECONDARY SERVICE) (**M:N**)
   Lokale Zonen können von einem oder mehreren lokalen (Secondary) Nameservern gespiegelt werden. Für die Realisierung dieser **M:N**-Beziehung ist

eine eigene Tabelle (SECONDARY_LL) zu erstellen. Diese enthält die Primärschlüssel der Tabellen HOSTS und NAMESERVERS als Fremdschlüssel.

9. Lokale Nameserver ↔ Hosts (**1:1**)
   Jeder lokale Nameserver muß auf einem (lokalen) Host laufen. Die Tabelle NAMESERVERS wird um den Primärschlüssel der Tabelle HOSTS als Fremdschlüssel erweitert.

10. Forwarder IP-Adressen ↔ Lokale Nameserver (**M:N**)
    Für jeden Nameserver können mehrere Forwarder IP-Adressen angegeben werden. Mit dieser **M:N**-Beziehung verhält es sich analog zu der unter Punkt 2 genannten. Die Tabelle FORWARDERS wird um den Primärschlüssel der Tabelle NAMESERVERS als Fremdschlüssel erweitert.

11. Fremde Zonen (nicht delegiert) ↔ Lokale Nameserver (**M:N**)
    Jeder lokale Nameserver kann mehrere fremde, nicht delegierte Zonen spiegeln (Secondary Service). Mit dieser **M:N**-Beziehung verhält es sich analog zu der unter Punkt 2 genannten. Die Tabelle SECONDARY_LF wird um den Primärschlüssel der Tabelle NAMESERVERS als Fremdschlüssel erweitert.

12. Fremde Nameserver ↔ Lokale Zonen (**M:N**)
    Jeder fremde Nameserver kann mehrere lokale Zonen spiegeln (Secondary Service). Mit dieser **M:N**-Beziehung verhält es sich analog zu der unter Punkt 2 genannten. Die Tabelle SECONDARY_FL wird um den Primärschlüssel der Tabelle ZONES als Fremdschlüssel erweitert.

13. Fremde Zonen (delegiert) ↔ Lokale Zonen (Sohn-Vater-Beziehung) (**N:1**)
    Von jeder lokalen Zone (Domain) können mehrere fremde Zonen delegiert werden. Die Tabelle DEL_ZONES wird um den Primärschlüssel der Tabelle ZONES als Fremdschlüssel erweitert.

14. Fremde Zonen (delegiert) ↔ Fremde Nameserver (Primary Service) (**N:1**)
    Fremde, delegierte Zonen werden durch einen fremden (Primary) Nameserver verwaltet. Dieser wird in der Tabelle DEL_ZONES eingetragen.

15. Fremde Zonen (delegiert) ↔ Fremde Nameserver (Secondary Service) (**M:N**)
    Fremde, delegierte Zonen können von einem oder mehreren fremden (Secondary) Nameservern gespiegelt werden. Für die Realisierung dieser **M:N**-Beziehung ist eine eigene Tabelle (DEL_ZONES_SEC) zu erstellen. Diese enthält den Primärschlüssel der Tabelle DEL_ZONES als Fremdschlüssel und Angaben über den entsprechenden Nameserver.

16. FREMDE ZONEN (DELEGIERT) ↔ LOKALE NAMESERVER (SECONDARY SERVICE) (**M:N**)
    Fremde, delegierte Zonen können von einem oder mehreren lokalen (Secondary) Nameservern gespiegelt werden. Für die Realisierung dieser **M:N**-Beziehung werden die hiervon betroffenen Zonen zusätzlich in die Tabelle SECONDARY_LF eingetragen.

17. MAIL EXCHANGER ↔ HOSTS (**N:1**)
    Als Mail Exchanger dient ein lokaler Host. Die Tabelle MAILX wird um den Primärschlüssel der Tabelle HOSTS als Fremdschlüssel erweitert.

5.4 Integration in vorhandene Datenbank

In Kapitel 5.1 wurde die vorhandene Datenbank in Hinblick auf die für das Domain Name System relevanten Stellen untersucht. Im vorangehenden Abschnitt wurde ein Datenbankschema entworfen, auf dessen Basis DNS-Konfigurationsmanagement möglich ist.

Die nun folgende Aufgabe besteht darin, beides in Einklang zu bringen, d.h. das neue Schema möglichst gut in das alte zu integrieren.

Eine *wesentliche* Forderung besteht dabei darin, am vorhandenen Modell keine so tiefgreifenden Modifikationen vorzunehmen, daß eine Reihe vorhandener Applikationen, die auf der Netzdokumentationsdatenbank aufsetzen, nicht mehr lauffähig wären, auf die andere Anwender angewiesen sind. Dies würde ein großes Hindernis in Bezug auf die Akzeptanz des neuen Verfahrens bedeuten.

Deshalb ist es nun vorrangiges Ziel, daß die alte Struktur erhalten bleibt und lediglich um das, was zur Anbindung notwendig und sinnvoll ist, ergänzt wird.

5.4.1 HOSTS (neu) → KOMPONENTE (alt)

Während beim Domain Name System Hosts im Vordergrund stehen, sind es in der Netzdokumentationsdatenbank die Komponenten.

Die ursprünglich vorgesehene Integration beider Objektklassen in eine Tabelle würde eine **1:1**-Beziehung zwischen beiden voraussetzen. Dies scheitert aber daran, daß ein und derselben Komponente mehrere Hostnamen zugeordnet werden können. Ein Beispiel hierfür am LRZ ist die Komponente **WSC10018**, die die beiden Hostnamen **dfvgate** und **suncollect** besitzt.

Es ist also eine zusätzliche Tabelle heranzuziehen, die eine Zuordnung von einem oder mehreren Hosts (Definition siehe Kapitel 5.3.1) zu einer Komponente leistet.

Hierfür soll die Tabelle LOGPORT (alt) herangezogen werden, die u.a. die IP-Adressen enthält, was aber zu Problemen führt, wie das folgende Beispiel zeigt:

Hostname	IP-Adresse
dfvgate	129.187.10.25
suncollect	129.187.9.241

Tabelle 5.1: Zuordnung Hostname → IP-Adresse

Mittels KOMPONENTE und LOGPORT soll die in Tabelle 5.1 dargestellte Abbildung hergeleitet werden.

name	alias
WSC10018	*dfvgate/suncollect*

Tabelle 5.2: Tabelle KOMPONENTE - Ausschnitt

komponente	protokoll	adresse
WSC10018	*IP*	*129.187.10.25*
WSC10018	*IP*	*129.187.9.241*

Tabelle 5.3: Tabelle LOGPORT - Ausschnitt

Die Tabellen 5.2 und 5.3 enthalten die entsprechenden Tupel mit den relevanten Attributen der Datenbank.

Wie man sehr schnell erkennen kann, ist die gewünschte Abbildung damit nicht möglich. Eine genauere Analyse der Netzdokumentationsdatenbank ergibt, daß der notwendige Hinweis, welcher Name zu welcher Adresse gehört, zusammen mit beliebiger anderer Information als Bemerkung in der Tabelle LOGPORT untergebracht werden. Es ist also keine systematische Zuordnung gegeben, die aber unabdingbare Voraussetzung für ein automatisches Auslesen der Datenbank ist.

5.4.2 HOSTS (neu) → LOGPORT (alt)

Da die Tabelle LOGPORT also nicht geeignet ist, eine Anbindung von HOSTS an KOMPONENTE zu schaffen, soll nun versucht werden, HOSTS an LOGPORT zu binden. Dazu soll LOGPORT genauer betrachtet werden:

```
SQL> desc logport
 Name                            Null?    Type
 ------------------------------- -------- ----
 KOMPONENTE                               CHAR(8)
 PORT                                     CHAR(10)
 PROTOKOLL                       NOT NULL CHAR(3)
 ADRESSE                         NOT NULL CHAR(50)
 BEMERKUNG                                CHAR(35)
```

Die vier Attribute KOMPONENTE, PORT, PROTOKOLL und BEMERKUNG sind zusammen nicht eindeutig, da eine Komponente mehrere logische Ports mit dem gleichen Protokoll haben kann, und die Felder PORT und BEMERKUNG meistens unbelegt bzw. aussagelos sind.

Diese Attribute bilden also keinen Schlüsselkandidaten, weshalb das Attribut ADRESSE Bestandteil des Schlüssels sein muß. Die Attribute PORT und BEMERKUNG stehen aus dem soeben genannten Grund nicht für die Bildung eines minimalen Schlüssels zur Verfügung.

Als mögliche Schlüsselkandidaten bleiben also (ADRESSE, KOMPONENTE), (ADRESSE, PROTOKOLL) bzw. (ADRESSE, KOMPONENTE, PROTOKOLL) übrig. Da eine Komponente bei unterschiedlichen Protokollen für einen logischen Port mehrere gleiche Adressen haben kann, fällt auch KOMPONENTE als Schlüsselattribut weg. Es bleibt also (ADRESSE, PROTOKOLL) als minimaler Schlüssel übrig. Was das Domain Name System angeht, so reduziert er sich auf (ADRESSE, 'IP').

Folgende zwei Vorgehensweisen bieten sich nun für unsere eigentliche Aufgabe, die Anbindung der Tabelle HOSTS an LOGPORT, an:

- ADRESSE wird als Fremdschlüssel in HOSTS eingefügt:
  In diesem Fall könnte einem HOST nur eine einzige IP-Adresse zugeordnet werden, da laut erster Normalform (1NF) Attribute atomar sein müssen. Ein und derselbe Host kann aber mehrere IP-Adressen besitzen, was diese Methode unbrauchbar macht.

- Attribute der Tabelle HOSTS werden in LOGPORT ergänzt:
  Dies würde den Zweck der Tabelle LOGPORT untergraben, logische Ports *unterschiedlicher* Protokolle bereitzustellen, da dann z.B. ein Host auch einer Ethernet- oder X.25-Adresse zugeordnet werden kann, was überhaupt keinen Sinn macht. Die neuen Felder bleiben also im Falle eines solchen Protokolls unbelegt, was eine unnötige Ressourcenverschwendung darstellt.

Als Ergebnis aller bisher genannten Anbindungsversuche der Tabelle HOSTS an die beiden vorhandenen Tabellen läßt sich feststellen, daß es hier keine enge Anbindung geben kann. Die einzige Möglichkeit, die übrig bleibt, besteht darin, die Tabelle HOSTS um den Primärschlüssel der Tabelle KOMPONENTE als Fremdschlüssel (optionales Attribut) zu erweitern. Dies wird im Rahmen der Implementierung auch geschehen, so daß zumindest eine lose Kopplung gegeben ist.

5.4.3 ZONES (neu) → PERSON (alt)

Bei der Beschreibung der Tabellen der vorhandenen Netzdokumentation wurde bereits darauf hingewiesen, daß für den SOA-Record einer Zone die E-Mail-Adresse des DNS-Administrators anzugeben ist.

In der Tabelle PERSON sind nur natürliche Namen mit deren E-Mail-Adressen gespeichert. Eine Erweiterung auf Verwaltungsnamen ist (derzeit) nicht vorgesehen. Was hier auch dagegen spricht, ist die Tatsache, daß damit oft sog. Mailing-Listen verbunden sind, die wiederum eine völlig andere Tabellenstruktur erforderlich machen würden.

Deshalb wird keine Anbindung der Tabelle ZONES an PERSON vorgenommen. Im Rahmen dieser Arbeit wird auch keine Ersatztabelle für Verwaltungsnamen geschaffen, da dies kein DNS-spezifisches Problem ist, sondern nur ein Attribut SOA_RESPONSIBLE eingeführt, in dem die erforderliche E-Mail-Adresse direkt abzulegen ist.

5.4.4 ZONES (neu) → INSTITUT (alt)

Jedem Institut kann eine eigene Subdomain (Zone) zugeordnet werden. Dazu wird die Tabelle INSTITUT um den Primärschlüssel der Tabelle ZONES als Fremdschlüsselattribut erweitert. Die Zuordnung ist optional.

Bestehende Applikationen, die auf der Netzdokumentationsdatenbank aufbauen, werden von dieser Modifikation nicht beeinflußt, sofern sie sauber programmiert sind (kein Einlesen von Daten in Listen via *select * from institut*, ...).

5.5 Organisatorische Probleme

Die folgenden organisatorischen Probleme sind noch zu lösen, bevor das Konfigurationsverfahren implementiert und damit letztlich angewendet werden kann.

Während das erste Problem das Verfahren nur intern betrifft und im folgenden durch einen Kompromiß gelöst wird, muß sich der DNS-Administrator mit der Benutzer- und Systemverwaltung seiner Organisation arrangieren, um eine Lösung für die beiden anderen zu erzielen.

5.5.1 Eingabe der DNS-Daten in die Datenbank

Zwischen den drei Entities Hosts, lokale Zonen und lokale Nameserver (siehe Abbildung 5.3) bestehen folgende Abhängigkeiten:

1. Hosts
   Jeder Host muß einer Zone angehören. Dazu muß der Fremdschlüssel ZONE in HOSTS auf einen gültigen Primärschlüssel in ZONES zeigen. Folglich ist die Zone vor dem Host in die Datenbank einzugeben.

2. Zonen
   Jede Zone benötigt einen Primary Nameserver. Dazu muß der Fremdschlüssel NAMESERVER in ZONES auf einen gültigen Primärschlüssel in NAMESERVERS zeigen. Folglich ist der Nameserver vor der Zone in die Datenbank einzugeben.

3. Nameserver
   Jede Nameserver läuft auf einem Host. Dazu muß der Fremdschlüssel HOST in NAMESERVERS auf einen gültigen Primärschlüssel in HOSTS zeigen. Folglich ist der Host vor dem Nameserver in die Datenbank einzugeben.

Allen drei Forderungen kann nicht gleichzeitig entsprochen werden. Obwohl ORACLE in der verwendeten Version 6 keine Direktiven bereitstellt, die die referentielle Integrität erzwingen, womit man das Problem in der Praxis damit lösen könnte, daß man zunächst an einer Stelle einen falschen Fremdschlüssel angibt und diesen nachträglich korrigiert, wenn man den Rest eingegeben hat, ist nach einer saubereren Lösung zu suchen. Neuere Versionen und andere relationale Datenbanksysteme unterstützen die genannte Integritätsforderung, was im Falle eines Updates oder Umstiegs diesen Lösungsansatz unbrauchbar macht.

Da es keine echte Auflösung für die Verklemmung gibt, kommt man nicht umhin, an einer von drei möglichen Stellen einen Einschnitt zu machen. Im Rahmen dieser Arbeit wird die Forderung fallen gelassen, daß jedem Host *sofort* eine Zone zugeordnet werden muß.

Damit ergibt sich folgende Eingabereihenfolge für die oben genannten Entities: Zuerst die Hosts, dann die Nameserver und danach die Zonen. Schließlich darf nicht vergessen werden, jedem Host seine entsprechende Zone zuzuordnen.

Weitere Forderungen, die sich im Zusammenhang mit der Eingabereihenfolge ergeben, bestehen darin, daß z.B. die IP-Adressen, Alias-Namen und Well Known Services nach den Hosts oder die Mail Routing Einträge nach den Hosts und den lokalen Zonen (letzteres wird impliziert durch die oben genannte Forderung) einzugeben sind. Derartige Forderungen sind aber trivial, da hier keine Verklemmungen wie oben auftreten können. Deshalb soll darauf auch nicht im einzelnen eingegangen werden.

5.5.2 Verteilung der Konfigurationsdateien

Hat man mit Hilfe eines Generierungstools die für einen Rechnerverbund notwendigen Resolver- und Nameserver-Konfigurationsdateien erzeugt, so liegen diese auf dem Rechner, auf dem sie erzeugt wurden, und noch nicht auf den jeweiligen Rechnern, wo sie tatsächlich benötigt werden.

Man muß sich also mit der Frage auseinandersetzen, wie der Verteilungsprozeß vonstatten gehen soll, *bevor* entschieden wird, das Verfahren einzusetzen. Die im folgenden aufgezeigten Lösungsansätze beschränken sich auf UNIX-Systeme, da in anderen Umgebungen andere unterschiedlichste Probleme auftauchen können, die hier nicht alle erfaßt werden können. Dem DNS-Administrator sei empfohlen, sich bei heterogenen Umgebungen über individuele Lösungen (evtl. Teillösungen) kundig zu machen.

Bei den genannten Dateien handelt es sich um Systemdateien, die zumeist unter *root*-Rechten verwaltet werden. In der Praxis ist aber nicht jeder Systemverwalter bereit, dem DNS-Administrator solche Rechte auf allen Rechnern (Resolver- und Nameserverkonfiguration) einzuräumen. Der Zugriff auf die betroffenen Dateien seitens des DNS-Administrators ist aber zwingend, will er nicht seine Konfigurationsdateien per E-Mail verschicken und den Systemverwalter jedesmal auffordern, diese entsprechend zu installieren. Dies wäre auch unter dem Gesichtspunkt eine schlechte Lösung, da er dann nicht immer wüßte, ob die (Re-)Konfiguration zu einem bestimmten Zeitpunkt bereits erfolgt ist.

Eine bessere Alternative besteht darin, die Rechte an den betroffenen Dateien und Verzeichnissen (innerhalb des Systemverzeichnisses) einem eigenen User *dnsadmin* zu übertragen, der den DNS-Administrator darstellt. Die Zugriffsmöglichkeiten seitens des Systemverwalters (*root*) bleiben davon unberührt.

Für den Verteilungsvorgang, der nach Möglichkeit automatisiert werden sollte, stehen Anwendungsprotokolle wie FTP, RCP, NFS oder AFS (auch gemischt) zur Verfügung.

5.5.3 Zugriff auf Nameserver-Prozeß

Ein Problem, das mit der erfolgreichen Verteilung der Konfigurationsdateien noch nicht gelöst ist, ist die Prozeßsteuerung des Nameservers (named), die mit Hilfe von Signalen erfolgt [Vixi 93]. Beispielsweise erfolgt die dynamische Rekonfiguration durch ein SIGHUP-Signal.

Da der Nameserver auf TCP/UDP-Sockets mit der Portnummer 53 kommuniziert, muß er unter der *root*-Kennung laufen. Dies bedeutet aber, daß die notwendigen Signale an ihn grundsätzlich nur mit dieser Kennung abgeschickt werden können, die der DNS-Administrator aus oben genannten Grund nicht immer besitzt.

Als Lösung bietet sich hier an, die Rechte am Nameserver-Prozeß dem DNS-Administrator zu übertragen, so daß dieser ihn starten und steuern kann, gleichzeitig aber mittels dem SUID-Bit (set user ID on execution) dafür zu sorgen, daß der Prozeß unter der *root*-Kennung läuft (*chmod*-Kommando).

Kapitel 6

Implementierung

Dieses Kapitel beschäftigt sich schließlich mit der Implementierung der Datenbankrelationen und des Werkzeuges, für die im vorangehenden Kapitel die Grundlagen – aus Sicht der Datenbank und aus organisatorischer Sicht – geschaffen wurden. Damit eignet sich Verfahren dann für die praktische Anwendung, die das Ziel der bisherigen Überlegungen darstellt.

Zunächst werden in Abschnitt 6.1 die bereits strukturell hergeleiteten Datenbankrelationen vervollständigt und mit den verwendeten SQL-Datentypen präsentiert. Diese Beschreibung ist Voraussetzung für den Einsatz, da der DNS-Administrator wissen muß, welche Informationen (Semantik) er in der Netzdokumentationsdatenbank ablegen kann bzw. muß, und wie die Daten (Syntax) anzugeben sind.

Damit ist die Datenbankbeschreibung abgeschlossen. Im Anschluß geht es um das Werkzeug, das als Programm *ora2dns*[1] in Form eines Oraperl-Skripts realisiert wird. Das Programm unterliegt den in Abschnitt 6.2 gezeigten Anforderungen.

Der Programmaufbau wird in Abschnitt 6.3 vorgestellt. Neben einer Übersicht wird vor allem das Hauptprogramm erläutert, während die einzelnen Module, die für die Generierung der verschiedenen Typen von Konfigurationsdateien zuständig sind, Gegenstand eines eigenen Abschnitts 6.4 sind.

Ein Beispielprogrammlauf in Abschnitt 6.5, für den ein kleiner Teil des realen DNS-Informationsbestandes in die Datenbank eingegeben wird, rundet das Kapitel ab.

[1] Zum Namen: Aus einer Datenbank, die mit dem RDBMS *ORA*CLE verwaltet wird, werden *DNS*-Konfigurationsdateien generiert.

6.1 Aufbau der Relationen

Im vorangehenden Kapitel wurden mit Hilfe eines Entity/Relationship Modells die notwendigen DB-Tabellen hergeleitet bzw. eine vorhandene modifiziert. Diese Herleitung beschränkte sich auf wesentliche Strukturmerkmale (Schlüssel etc.).

Nun werden die vollständigen Tabellen angegeben, mit denen der DNS-Administrator, der das Verfahren anwendet, arbeiten kann. Die folgende Beschreibung versteht sich daher auch als Hilfe zum Integrieren der notwendigen Daten in die Datenbank.

Für die wichtigsten Tabellen werden künstliche Primärschlüssel (Identifikatoren) eingeführt. Mit Hilfe eines Index kann auch bei der verwendeten ORACLE-Version 6 die Integrität dieses Schlüssels erzwungen werden.

An einigen Stellen werden optionale Attribute eingeführt, die die Lebenszeit (TTL) bzw. Klasse der Resource Records bei deren Erstellung betreffen. Diese sind jedoch optional und brauchen in der Regel nicht berücksichtigt werden[2]. Wie für die meisten optionalen Attribute stehen Standardwerte zur Verfügung.

Zunächst wird die modifizierte Tabelle vorgestellt. Anschließend werden die neuen Tabellen beschrieben, wobei sie in drei Klassen eingeteilt werden, je nachdem, ob sie vorwiegend Host-, Zonen- oder Nameserverdaten enthalten.

[2]Der Grund für deren Implementierung ist darin zu sehen, eine über das gewöhnlich notwendige Minimum hinausgehende Flexibilität zu schaffen. Es soll auch ein Anreiz sein, neue Attribute in das Verfahren zu integrieren, wenn dies notwendig werden soll.

6.1.1 Modifizierte Tabelle

Die in der Netzdokumentationsdatenbank vorhandene Tabelle INSTITUT wird wie folgt erweitert:

	Attributname	Wertebereich	NOT NULL
1	id	char(4)	×
2	organisation	char(60)	×
3	name	char(180)	
4	chef	char(60)	
5	telefon	char(20)	
6	strasse	char(50)	
7	ort	char(50)	
8	knoten	char(3)	
9	dv1name	char(60)	
10	dv1tel	char(20)	
11	dv2name	char(60)	
12	dv2tel	char(20)	
13	dv3name	char(60)	
14	dv3tel	char(20)	
15	zone	number(5,0)	

Das hinzugekommene Attribut ZONE (Nummer 15) gibt die Zone (Subdomain) an, die dem Institut zugeordnet wird (Fremdschlüssel auf ZONES.ID).

6.1.2 Host-Tabellen

- HOSTS

	Attributname	Wertebereich	NOT NULL
1	id	number(5,0)	×
2	name	char(63)	×
3	subdomain_name	char(240)	
4	zone	number(5,0)	(siehe Text)
5	komponente	char(8)	
6	record_ttl	number(10,0)	
7	record_class	char(2)	
8	cpu	char(20)	
9	os	char(20)	

1. 5-stellige Host-Nummer (Primärschlüssel).
2. Hostname (max. 63 Zeichen, siehe [RFC 1035], Kapitel 2.3.4).

3. Subdomainname, falls der Host einer Subdomain innerhalb seiner Zone angehört. [optional]
4. Zone, der der Host angehört (Fremdschlüssel auf ZONES.ID). *Achtung:* Dies ist kein optinales Attribut, sondern muß später ergänzt werden (siehe Kapitel 5.5.1).
5. Komponente, zu der der Host gehört (Fremdschlüssel auf KOMPONENTE.NAME). Beschreibung siehe [Valt 92]. [optional]
6. TTL-Wert für Host-spezifische Resource-Records (32Bit-Wert mit VZ, siehe [RFC 1035], Kapitel 3.2.1). [optional]
7. Klasse der Host-spezifischen Resource-Records (2 Zeichen, die die Klassenbezeichnung enthalten, siehe [RFC 1035], Kapitel 3.2.1). [optional, standardmäßig: 'IN']
8. Bezeichnung der CPU (max. 20 Zeichen). [optional]
9. Bezeichnung des Betriebssystems (max. 20 Zeichen). [optional]

Werden die letzten beiden Attribute zusammen angegeben, so wird für den Host ein HINFO-RR generiert (Standardwerte siehe [RFC 1010]).

Mit der folgenden Anweisung wird die Eindeutigkeit des Primärschlüssels gewährleistet:
`create unique index hosts_id on hosts(id);`

- IPADDR

	Attributname	Wertebereich	NOT NULL
1	host	number(5,0)	×
2	ip_addr	char(15)	×

1. Host (Fremdschlüssel auf HOSTS.ID).
2. Internet-Adresse (String mit 15 Zeichen). Die Darstellungsweise entspricht aus Kompatibilitätsgründen der bisher in der Netzdokumentationsdatenbank verwendeten (siehe auch Kapitel 5.1.2).

- ALIASES

	Attributname	Wertebereich	NOT NULL
1	host	number(5,0)	×
2	canonical_name	char(63)	×

1. Host (Fremdschlüssel auf HOSTS.ID).
2. Alias-Name (max. 63 Zeichen, siehe [RFC 1035], Kapitel 2.3.4).

- WKS

	Attributname	Wertebereich	NOT NULL
1	host	number(5,0)	×
2	service	char(240)	×

1. Host (Fremdschlüssel auf HOSTS.ID).
2. Angabe von IP-Adresse, Transportproktokoll und Services in einem String, durch Leerzeichen getrennt (max. 240 Zeichen, max. Stringlänge bei ORACLE). Beispiel aus [Vixi 93], Kapitel 6.5.7:
   `128.32.0.10 UDP who route timed domain`

Well Known Services werden im Rahmen dieser Arbeit nur in der angegebenen Form unterstüzt, da sie in der Praxis selten angegeben werden. Es wäre aber auch denkbar, die entsprechenden Einträge automatisch aus der Datei */etc/services* des jeweiligen Rechners generieren zu lassen.

6.1.3 Zonen-Tabellen

- ZONES (lokale Zonen)

	Attributname	Wertebereich	NOT NULL
1	id	number(5,0)	×
2	name	char(240)	×
3	prim_nameserver	number(5,0)	×
4	parent	number(5,0)	
5	sourcefile	char(64)	
6	soa_responsible	char(240)	×
7	soa_class	char(2)	
8	soa_serial	number(10,0)	×
9	soa_refresh	number(10,0)	
10	soa_retry	number(10,0)	
11	soa_expire	number(10,0)	
12	soa_minimum	number(10,0)	

1. 5-stellige Zonen-Nummer (Primärschlüssel).
2. Domainname (max. 240 Zeichen, max. Stringlänge bei ORACLE).
3. Primary Nameserver für die Zone (Fremdschlüssel auf NAMESERVERS.ID).
4. Vaterzone (Fremdschlüssel auf ZONES.ID). Wird die Vaterzone nicht selbst verwaltet, so ist das Attribut mit dem Wert NULL zu belegen.

5. Name der zu erstellenden Zonendatei. [optional, standardmäßig: Domainname + Suffix]
6. E-Mail Adresse des DNS-Administrators für den SOA-RR dieser Zone.
7. Klasse des SOA-RR (2 Zeichen, die die Klassenbezeichnung enthalten, siehe [RFC 1035], Kapitel 3.2.1). [optional, standardmäßig: 'IN']
8. Seriennummer des SOA-RR (32Bit-Wert ohne VZ, siehe [RFC 1035], Kapitel 3.3.13). Diese muß bei jeder Manipulation von DNS-Informationen in der Zone erhöht werden.
9. Refresh-Intervall (in Sek.) im SOA-RR (32Bit-Wert, siehe [RFC 1035], Kapitel 3.3.13). [optional, standardmäßig: 21600]
10. Retry-Intervall (in Sek.) im SOA-RR (32Bit-Wert, siehe [RFC 1035], Kapitel 3.3.13). [optional, standardmäßig: 3600]
11. Expire-Intervall (in Sek.) im SOA-RR (32Bit-Wert, siehe [RFC 1035], Kapitel 3.3.13). [optional, standardmäßig: 604800]
12. Minimale TTL (in Sek.) im SOA-RR (32Bit-Wert ohne VZ, siehe [RFC 1035], Kapitel 3.3.13). [optional, standardmäßig: 172800]

Die Standardzeitwerte entsprechen einem Vorschlag in [Golk 91].

Mit der folgenden Anweisung wird die Eindeutigkeit des Primärschlüssels gewährleistet:
`create unique index zones_id on zones(id);`

- DEL_ZONES (lokal delegierte fremde Zonen)

	Attributname	Wertebereich	NOT NULL
1	id	number(5,0)	×
2	name	char(240)	×
3	pns_ip_name	char(240)	×
4	pns_ip_addr	char(15)	(siehe Text)
5	parent	number(5,0)	×

1. 5-stellige Zonen-Nummer (Primärschlüssel).
2. Domainname (max. 240 Zeichen, max. Stringlänge bei ORACLE).
3. Fully Qualified Domain Name des Primary Nameservers für die Zone.
4. IP-Adresse des Primary Nameservers für die Zone (Darstellung wie IPADDR.IP_ADDR). Diese *muß* angegeben werden, wenn sein Name in der zur Vaterzone gleichnamigen Domain liegt (für die Erzeugung eines Glue-Records).
5. Vaterzone (Fremdschlüssel auf ZONES.ID).

Mit der folgenden Anweisung wird die Eindeutigkeit des Primärschlüssels gewährleistet:
`create unique index del_zones_id on del_zones(id);`

- DEL_ZONES_SEC

	Attributname	Wertebereich	NOT NULL
1	del_zone	number(5,0)	×
2	sns_ip_name	char(240)	×
3	sns_ip_addr	char(15)	(siehe Text)

1. Zone (Fremdschlüssel auf DEL_ZONES.ID).
2. Fully Qualified Domain Name des Secondary Nameservers für die Zone.
3. IP-Adresse des Secondary Nameservers für die Zone (Darstellung wie IPADDR.IP_ADDR). Diese *muß* angegeben werden, wenn sein Name in der zur Vaterzone gleichnamigen Domain liegt (für die Erzeugung eines Glue-Records).

- MAILX

	Attributname	Wertebereich	NOT NULL
1	zone	number(5,0)	×
2	name	char(240)	
3	record_ttl	number(10,0)	
4	record_class	char(2)	
5	pref_value	number(5,0)	×
6	host	number(5,0)	×

1. Zone (Fremdschlüssel auf ZONES.ID).
2. Name des Mail-Exchangers innerhalb der Domain. Wird kein Name angegeben, so wird dieser für die Domain (Zone) selbst erstellt. Hier können auch Wildcards ('$*$') angegeben werden (siehe [RFC 1034], Kapitel 4.3.3). [optional]
3. TTL-Wert für den MX-RR (32Bit-Wert mit VZ, siehe [RFC 1035], Kapitel 3.2.1). [optional]
4. Klasse des MX-RR (2 Zeichen, die die Klassenbezeichnung enthalten, siehe [RFC 1035], Kapitel 3.2.1). [optional, standardmäßig: 'IN']
5. Preference Value (16Bit-Wert). Dieser gibt im Falle mehrerer passender Einträge an, wohin Mails bevorzugt zugestellt werden sollen, wobei ein niedrigerer Wert eine höhere Priorität bedeutet (siehe [RFC 1035], Kapitel 3.3.9).
6. Ziel-Host (Fremdschlüssel auf HOSTS.ID).

Mail Routing (über MX-RRs) wird im Rahmen dieser Arbeit nur in der angegebenen Form unterstützt (siehe Kapitel 5.2.2).

6.1.4 Nameserver-Tabellen

- NAMESERVERS

	Attributname	Wertebereich	NOT NULL
1	id	number(5,0)	×
2	host	number(5,0)	×
3	filename_exe	char(64)	
4	filename_pid	char(64)	
5	soa_responsible	char(240)	
6	soa_class	char(2)	
7	soa_serial	number(10,0)	
8	soa_refresh	number(10,0)	
9	soa_retry	number(10,0)	
10	soa_expire	number(10,0)	
11	soa_minimum	number(10,0)	
12	files_dir	char(128)	
13	filename_lo	char(64)	
14	filename_lb	char(64)	
15	filename_ca	char(64)	
16	ftp_host	char(240)	
17	ftp_dir	char(128)	
18	ftp_file	char(64)	

1. 5-stellige Nameserver-Nummer (Primärschlüssel).
2. Host, auf dem der Nameserver läuft (Fremdschlüssel auf HOSTS.ID).
3. Dateiname des Nameserver-Prozesses (max. 64 Zeichen). [optional, standardmäßig: '/etc/named']
4. Name der Datei, die die Prozeßnummer des Nameserver-Prozesses enthält (max. 64 Zeichen). [optional, standardmäßig: '/etc/named.pid']
5. E-Mail Adresse des DNS-Administrators für die SOA-RRs des Loopback Interfaces. Wird sie nicht angegeben, so wird die derjenigen Zone verwendet, der der Nameserver-Host angehört. [optional]
6. Klasse der SOA-RRs des Loopback Interfaces (2 Zeichen, die die Klassenbezeichnung enthalten, siehe [RFC 1035], Kapitel 3.2.1). Wird keine Klasse angegeben, so wird die derjenigen Zone verwendet, der der Nameserver-Host angehört. Wird diese (dort) auch nicht angegeben, so wird der Standardwert für Nameserver (Voreinstellung: 'IN') verwendet. [optional]

7. Seriennummer der SOA-RRs des Loopback Interfaces (32Bit-Wert ohne VZ, siehe [RFC 1035], Kapitel 3.3.13). Diese muß bei jeder Manipulation von DNS-Informationen im Loopback Interface erhöht werden. Wird sie nicht angegeben, so wird die derjenigen Zone verwendet, der der Nameserver-Host angehört. [optional]
8. Refresh-Intervall (in Sek.) in den SOA-RRs des Loopback Interfaces (32Bit-Wert, siehe [RFC 1035], Kapitel 3.3.13). [optional]
9. Retry-Intervall (in Sek.) in den SOA-RRs des Loopback Interfaces (32Bit-Wert, siehe [RFC 1035], Kapitel 3.3.13). [optional]
10. Expire-Intervall (in Sek.) in den SOA-RRs des Loopback Interfaces (32Bit-Wert, siehe [RFC 1035], Kapitel 3.3.13). [optional]
11. Minimale TTL (in Sek.) in den SOA-RRs des Loopback Interfaces (32Bit-Wert ohne VZ, siehe [RFC 1035], Kapitel 3.3.13). [optional]
12. Verzeichnis, in dem die Zonendateien abgelegt werden (max. 128 Zeichen). [optional, standardmäßig: '/etc/namedb']
13. Name der Zonendatei für das Loopback Interface (**127.1** → **localhost**) (max. 64 Zeichen). [optional, standardmäßig: 'named.local']
14. Name der Zonendatei für das Loopback Interface (**localhost** → **127.1**) (max. 64 Zeichen). [optional, standardmäßig: 'named.loopback']
15. Name der Zonendatei für die Root-Domain (max. 64 Zeichen). [optional, standardmäßig: 'named.cache']
16. FTP-Host, von dem die letztgenannte Datei bezogen wird. (max. 240 Zeichen). [optional, standardmäßig: 'rs.internic.net']
17. Verzeichnis auf dem FTP-Host, in dem diese Datei abgelegt ist. (max. 128 Zeichen). [optional, standardmäßig: '/domain']
18. Name dieser Datei auf dem FTP-Host. (max. 64 Zeichen). [optional, standardmäßig: 'named.cache']

Das 3. und 4. Attribut werden zur Generierung der Konfigurationsdateien nicht benötigt. Sie dienen der Prozeßsteuerung der Nameserver, die im Rahmen eines erweiterten Konfigurationsverfahrens implementiert werden könnte.

Mit den Zeitwerten in den SOA-RRs des Loopback Interfaces verhält es sich im Falle von deren Nichtangabe analog zum Attribut SOA_CLASS. Die Standardwerte entsprechen denen der Tabelle ZONES.

Mit der folgenden Anweisung wird die Eindeutigkeit des Primärschlüssels gewährleistet:

```
create unique index nameservers_id on nameservers(id);
```

- SECONDARY_LL (lokale Nameserver für lokale Zonen)

	Attributname	Wertebereich	NOT NULL
1	zone	number(5,0)	×
2	nameserver	number(5,0)	×
3	backupfile	char(64)	

1. Zone (Fremdschlüssel auf ZONES.ID).
2. Nameserver (Fremdschlüssel auf NAMESERVERS.ID).
3. Name der zugehörigen Backup-Zonendatei. [optional, standardmäßig: Domainname + Suffix]

- SECONDARY_LF (lokale Nameserver für fremde Zonen)

	Attributname	Wertebereich	NOT NULL
1	nameserver	number(5,0)	×
2	zone_name	char(240)	×
3	ip_addr	char(15)	×
4	backupfile	char(64)	

1. Nameserver (Fremdschlüssel auf NAMESERVERS.ID).
2. Domainname (max. 240 Zeichen, max. Stringlänge bei ORACLE).
3. IP-Adresse des Primary Nameservers für die Zone (Darstellung wie in IPADDR.IP_ADDR).
4. Name der zugehörigen Backup-Zonendatei. [optional, standardmäßig: Domainname + Suffix]

- SECONDARY_FL (fremde Nameserver für lokale Zonen)

	Attributname	Wertebereich	NOT NULL
1	zone	number(5,0)	×
2	ip_name	char(240)	×
3	ip_addr	char(15)	(siehe Text)

1. Zone (Fremdschlüssel auf ZONES.ID).
2. Fully Qualified Domain Name des Secondary Nameservers für die Zone.
3. IP-Adresse des Secondary Nameservers für die Zone (Darstellung wie IPADDR.IP_ADDR). Diese *muß* angegeben werden, wenn sein Name in der zur Vaterzone gleichnamigen Domain liegt (für die Erzeugung eines Glue-Records).

- FORWARDERS

	Attributname	Wertebereich	NOT NULL
1	nameserver	number(5,0)	×
2	ip_addr	char(15)	×

1. Nameserver (Fremdschlüssel auf NAMESERVERS.ID).
2. IP-Adresse des Master Nameservers, an den die DNS-Anfragen weitergeleitet werden (Darstellung wie in IPADDR.IP_ADDR).

- NAMESERVERS_SLAVE

	Attributname	Wertebereich	NOT NULL
1	nameserver	number(5,0)	×

1. Nameserver, der als Slave arbeitet (Fremdschlüssel auf NAMESERVERS.ID). Für einen solchen Nameserver sind Forwarder IP-Adressen unbedingt erforderlich.

6.2 Anforderungen an das Werkzeug

Neben den Anforderungen an die Managementumgebung (Kapitel 4.5) und denen an das Datenbankdesign (Kapitel 5.2) werden auch von dem Programm bestimmte Eigenschaften erwartet. In diesem Abschnitt wird festgelegt, wie und woher das Programm das Programm seine Eingabedaten bezieht.

6.2.1 Informationsfluß

Das Werkzeug (Programm) hat die Aufgabe, seine Eingabedaten in eine Menge von Ausgabedaten zu transformieren. Der hierbei auftretende Informationsfluß ist genau zu spezifizieren (siehe Abbildung 6.1). Dies dient nicht nur dem Verständinis des Programms und eventueller Schwierigkeiten, die bei der Informationsbeschaffung auftreten können, sondern ist vor allem eine unabdingbare Voraussetzung für eine konsequente Fehlerrückverfolgung im Rahmen des Fehlermanagements.

Als Informationquellen für das Programm dienen *ausschließlich* die folgenden Objekte:

- Die Datenbank (Netzdokumentation), deren Tabellen die aktuelle DNS-Informationsbasis darstellen.

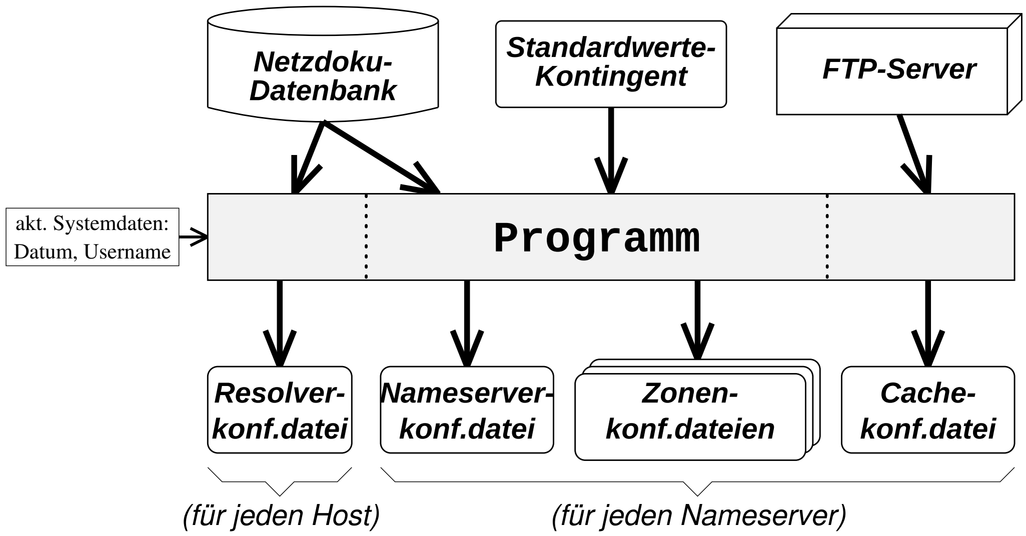

Abbildung 6.1: Informationsquellen und -senken

- Ein Kontingent an Standardwerten, die in der Regel dann eingesetzt werden, wenn die entsprechenden speziellen Angaben in der Datenbank fehlen.
- Ein FTP-Server[3], von dem mittels Anonymous-FTP die Zonendatei für die Root-Domain (Cache Service) bezogen wird.
- Die aktuellen Systemdaten Datum und Username werden als Kommentar in die Konfigurationsdateien geschrieben und spielen deshalb nur eine untergeordnete Rolle. Dennoch seien sie hier der Vollständigkeit wegen erwähnt.

Die zu erzeugenden Konfigurationsdateien stellen die Informationsenken dar. Deren Aufgabe wurde im 3. Kapitel (Konfiguration einer DNS-Umgebung) beschrieben.

Was in diesem Zusammenhang fehlt, ist eine HOSTS-Datei (*/etc/hosts*). Auf deren Generierung wird im Rahmen dieser Arbeit verzichtet. Mit den vorliegenden Informationen (DB-Schema und Programmlistings) sollte es aber kein Problem darstellen, dies noch nachträglich zu integrieren.

Außer einigen Programmablaufmeldungen (Standardausgabe: *stdout*) sind keine weiteren Ausgaben oder Funktionen des Programms vorgesehen. Dies ist vor dem Hintergrund zu sehen, das das Programm für eine spätere Integration in eine Managementplattform konzipiert ist, in der der Programmablauf graphisch aufbereitet zu präsentieren ist. Aus dem gleichen Grund wird auch auf eine interaktive Benutzerschnittstelle in der vorliegenden Textversion verzichtet.

[3] Auswahl des Servers in der Datenbank oder bei den Standardwerten.

6.2.2 Flexibilität und Komfort

In vielen DNS-Managementumgebungen liegt ein sehr umfangreicher Datenbestand vor, der beim Einsatz eines neuen Konfigurationsverfahrens diesem zugänglich gemacht werden muß. In unserem Fall bedeutet dies, das sämtliche betroffenen Daten in die Datenbank eingegeben werden müssen.

Wegen des ohnehin schon großen Arbeitsaufwandes liegt das Bestreben nahe, die Eingabe auf die (im Normalfall) notwendigen Daten zu beschränken. Auf der anderen Seite will man die Konfiguration aber auch an die individuellen Bedürfnisse anpassen können.

Man ist also zwei konkurrierenden Zielen ausgesetzt:

- „Wenige“ Daten (nur das Nötigste) in der Datenbank bedeuten mehr Komfort, da weniger eingegeben und verwaltet werden muß.
- „Viele“ Daten (alles, was sich konfigurieren läßt) in der Datenbank garantieren die (gelegentlich) erforderliche Flexibilität.

Dieser Konfliktsituation wurde beim Design der DB-Tabellen bereits Rechnung getragen mittels einer Reihe optionaler Attribute (siehe Kapitel 6.1).

Nun ist es Aufgabe des Programms, dieses mehrstufige Informationsbeschaffungssystem entsprechend umzusetzen. Dabei ist auch noch eine Verfeinerung vorgesehen, da auch die Standardwerte vom Benutzer frei bestimmt werden können sollen.

Hierfür ist eine Konfigurationsdatei (*.ora2dnsrc*) vorgesehen, die im Home-Verzeichnis des Programmbenutzers abzulegen ist. Ein Beispiel dieser Datei befindet sich im Anhang A.1 mit allen manipulierbaren Attributen und ihren Werten, wie sie im Programm verwendet werden, falls sie nicht (mittels dieser Datei) geändert werden.

Die verschiedenen Stufen der Informationsbeschaffung sollen im folgenden anhand des Refresh-Intervalls im einem SOA-Record einer Zone des Loopback Interfaces eines Nameservers exemplarisch aufgezeigt werden:

1. Das entsprechende Attribut im Nameserver-Tupel der Datenbank ist gesetzt. In diesem Fall wird dessen Wert verwendet.
2. Ist dieses Attribut unbelegt (NULL), aber das entsprechende Attribut im Zonen-Tupel der Zone, der der Nameserver-Host angehört, gesetzt, so wird letzteres verwendet.

3. Ist dieses ebenfalls unbelegt, so wird der in der Konfigurationsdatei hierfür letztgenannte Wert verwendet (NAMESERVERS.SOA_RETRY).

4. Ist auch in der Konfigurationsdatei kein entsprechender Eintrag vorhanden, so wird schließlich seitens des Programms ein fester Wert (21600) vorgegeben.

Damit ist es möglich, die Informationen stets so fein wie nötig und so grob wie möglich anzugeben. Dabei überlagert das Spezielle das Allgemeine (Generische).

6.3 Programmaufbau

Abbildung 6.2 gibt einen Überblick, welche Funktionen im Programm der Reihe nach ausgeführt werden. In den folgenden Abschnitten dieses und des nächsten Unterkapitels wird auf die Einzelheiten eingegangen. Dazu zählen das Hauptprogramm, ein Modul mit verschiedenen Unterprogrammen sowie ein Reihe von Modulen, die die speziellen Generierungsaufgaben der jeweiligen Konfigurationsdateitypen übernehmen. Die zugehörigen kommentierten Listings sind in Anhang A zu finden.

6.3.1 Hauptprogramm

Das Hauptprogramm (*ora2dns*, Listing siehe Anhang A.2) hat die Aufgabe, die einzelnen in Abbildung 6.2 gezeigten Schritte zu koordinieren und die Daten für die einzelnen Unterprogramme soweit wie möglich aufzubereiten.

Die erste Zeile dieser Skript-Datei, die den Interpreter angibt, der das Skript ausführt, muß nach den beiden Zeichen `#!` (Auswahl des Interpreters bei Shellskripten) den vollständigen Dateinamen des Oraperl-Interpreters enthalten. Dieser ist ggf. an die lokale Umgebung anzupassen.

Zunächst werden die einzelnen Bibliotheken eingebunden und die Standardwerte vorbelegt, bevor das Einloggen in das Datenbanksystem erfolgt, damit die Namen und Domainnamen der einzelnen Hosts, die die Namen der im nächsten Schritt zu generierenden Unterverzeichnisse bestimmen, zur Verfügung stehen. Anschließend erfolgt die komplette Resolverkonfiguration in dem dafür vorgesehenen Unterprogramm.

Für die Nameserverkonfiguration wird innerhalb des Hauptprogramms ein größerer Aufwand getrieben, da die in den einzelnen Unterprogrammen notwendigen

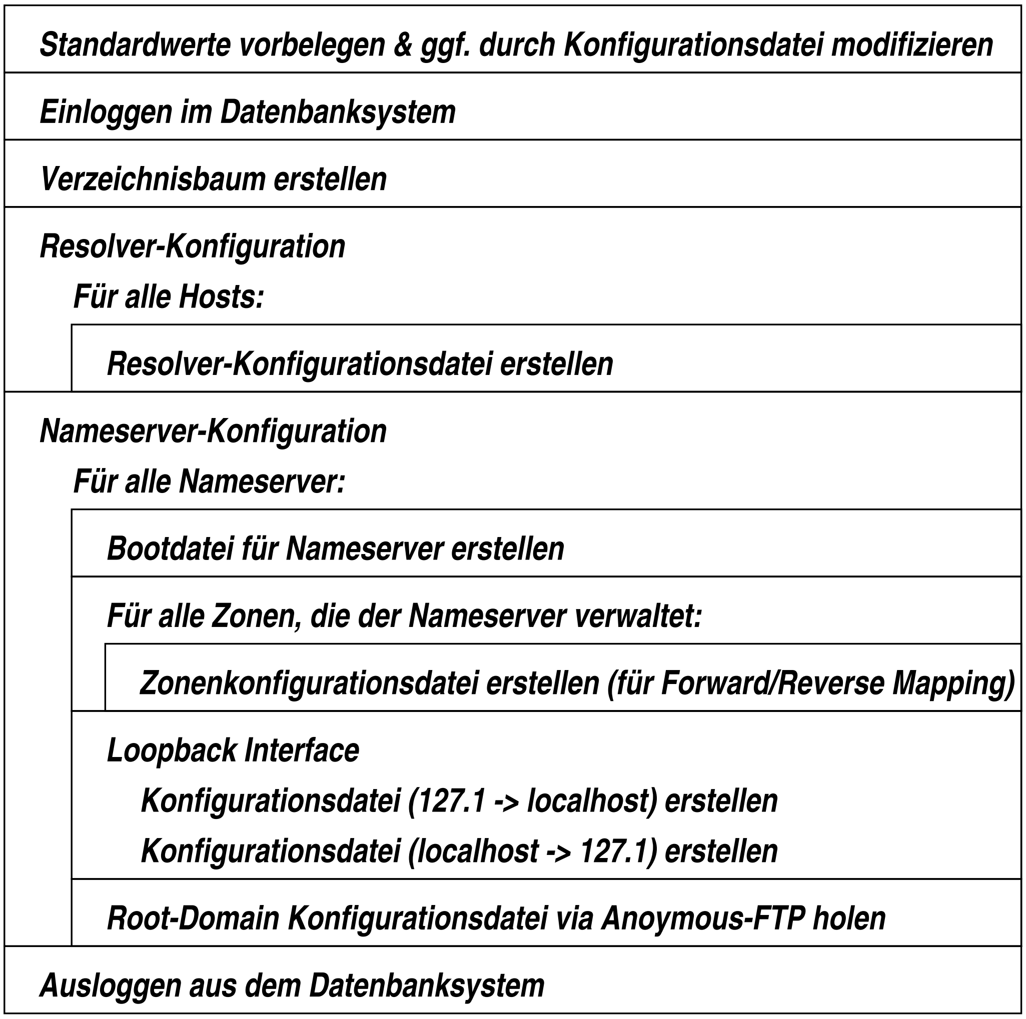

Abbildung 6.2: Programm *ora2dns* – Übersicht

Maßnahmen erheblich umfangreicher sind. Diese sollen durch vom Hauptprogramm aufbereitete Parameter entlastet werden. Der Vorteil dieser Vorgehensweise liegt neben einer ausgeglicheneren Aufgabenverteilung vor allem in der gesteigerten Effizienz, da die gleichen Daten nicht in jedem Unterprogramm neu bestimmt werden müssen, was vor allem in Bezug auf komplexe Datenbankabfragen (Joins über mehrere Attribute) von großer Bedeutung für die Programmlaufzeit[4] ist.

Sämtliche Nameserver werden innerhalb einer Schleife einzeln konfiguriert. Neben ihren eigenen Daten müssen dafür zunächst diejenigen des Hosts, auf dem

[4] Diese Zeit beeinflußt den Parameter Δt_a, der wiederum die QoS-Parameter Δt_P und Δt_S mitbestimmt (siehe Kapitel 4.3.2).

der Namerserver läuft bestimmt werden. Sind einzelne Attribute dabei in der Datenbank nicht spezifiziert, so werden sie im Anschluß mit Standardwerten belegt (vgl. Kapitel 6.2.2).

Sind alle notwendigen Parameter bestimmt, so wird in einem ersten Schritt die Nameserverkonfigurationsdatei (Bootdatei) erstellt, bevor die Zonenkonfigurationsdateien für jede von dem Nameserver verwaltete Zone (Primary Service) generiert werden. Hierbei muß unterschieden werden, ob es sich um eine Zone mit Forward oder Reverse Mapping (Endung: **.in-addr.arpa**) handelt, da beide sich inhaltlich stark von einander unterscheiden. Anschließend werden die beiden für das Loopback Interface notwendigen Zonenkonfigurationsdateien erstellt. Schließlich wird die Konfigurationsdatei für die Root-Domain per Anonymous-FTP geholt.

Am Ende des Programms wird aus dem Datenbanksystem wieder ausgeloggt und eine Meldung über den erfolgreichen Ablauf ausgegeben.

6.3.2 Unterprogramme

Für das Programm werden einige Funktionen und Prozeduren in einem zentralen Modul (*ora2dns.pl*, Listing siehe Anhang A.3) bereitgestellt.

Neben einer Prozedur, die die Belegeung der Standardwerte vornimmt, gehören dazu vor allem Funktionen, die die Datenbankzugriffe steuern. Zunächst ist je eine Prozedur zum Ein- bzw. Ausloggen aus dem Datenbanksystem vorgesehen. Erstere *muß* an die lokale Umgebung angepaßt werden, was das Standardlogin und -paßwort anbetrifft. Außer einer Prozedur, die vollständige Datenbankabfragen vornimmt, sind auch Abfragen nach dem Cursor-Konzept vorgesehen.

Für die Behandlung von NULL-Werten in Attributen ist eine spezielle Variable vorgesehen. Eine Funktion, die ein Attribut auf einen NULL-Wert testet, bzw. eine Prozedur, die das Auftreten eines solchen Wertes untersagt, sind in diesem Zusammenhang ebenfalls erforderlich.

6.4 Programmodule

Jedes dieser Module ist Bestandteil des im vorangehenden Abschnitt vorgestellten Programms und wird mittels einer eigenen Oraperl-Quelltextdatei realisiert. Die Module werden im folgenden entsprechend ihrem Zweck angegeben.

6.4.1 Verzeichnisbaum aufbauen

Das Modul *make_dir.pl* (Listing siehe Anhang A.4) beinhaltet im wesentlichen eine Prozedur, mit deren Hilfe die Unterverzeichnisse, in denen die zu erstellenden Konfigurationsdateien abzulegen sind, generiert werden.

Dazu wird für jeden Host ein Unterverzeichnis mit seinem Fully Qualified Domain Name als Verzeichnisname sowie ein darin enthaltenes *etc*-Verzeichnis erstellt.

6.4.2 Resolverkonfiguration

Zur Resolverkonfiguration dient das Modul *res_conf.pl* (Listing siehe Anhang A.5). In der entsprechenden Prozedur wird für jeden Host eine Konfigurationsdatei in dem oben angegebenen *etc*-Verzeichnis erstellt.

Darin wird neben einem Dateikopf der Domainname (DNS, nicht NIS-Domainname) des entsprechenden Hosts angegeben, der sich aus dem Zonennamen und ggf. dem Subdomainnamen des Hosts zusammensetzt.

In der Datei werden ferner bis zu drei Nameserver angegeben, die der Reihenfolge nach angesprochen werden, bis ein Ergebnis zur Verfügung steht.

Dabei ist zu beachten, daß zuerst ein möglichst „naheliegender“ Nameserver in die Liste eingetragen wird. Daneben ist es vorteilhaft, im Falle mehrerer IP-Adressen der beteiligten Hosts (Resolver- und Nameserverhost) die optimalen auszuwählen.

In der vorliegenden Arbeit wird dazu ein Algorithmus eingesetzt, der anhand eines byteweisen Netzmaskenvergleiches feststellt, welche IP-Adressen auszuwählen sind, um den kürzesten Weg zwischen zwei Hosts zu wählen. Dies funktioniert allerdings nicht immer, wie in Abbildung 6.3 gezeigt wird.

In diesem Beispiel liegt ein geteiltes Ethernet-Subnetz vor, dessen beiden Teile über ein FDDI-Backbone miteinander verbunden sind. Soll nun für den Resolver (Host: **XXX.YYY.50.13**) die IP-Adresse des nächstgelegenen Nameservers ermittelt werden, so erhält man **XXX.YYY.50.6**, obwohl **XXX.YYY.40.8** näher liegt.

6.4.3 Nameserverkonfiguration

Die Nameserver werden über ihre Bootdatei konfiguriert. Zum Erstellen einer solchen Datei dient das Modul *named_boot.pl* (Listing siehe Anhang A.6).

Zunächst wird dazu ein Dateikopf erstellt und das Verzeichnis, in dem die Zonenkonfigurationsdateien für den Nameserver abgelegt werden, angegeben. Anschließend werden alle Zonen aufgeführt, für die der Nameserver einen Primary

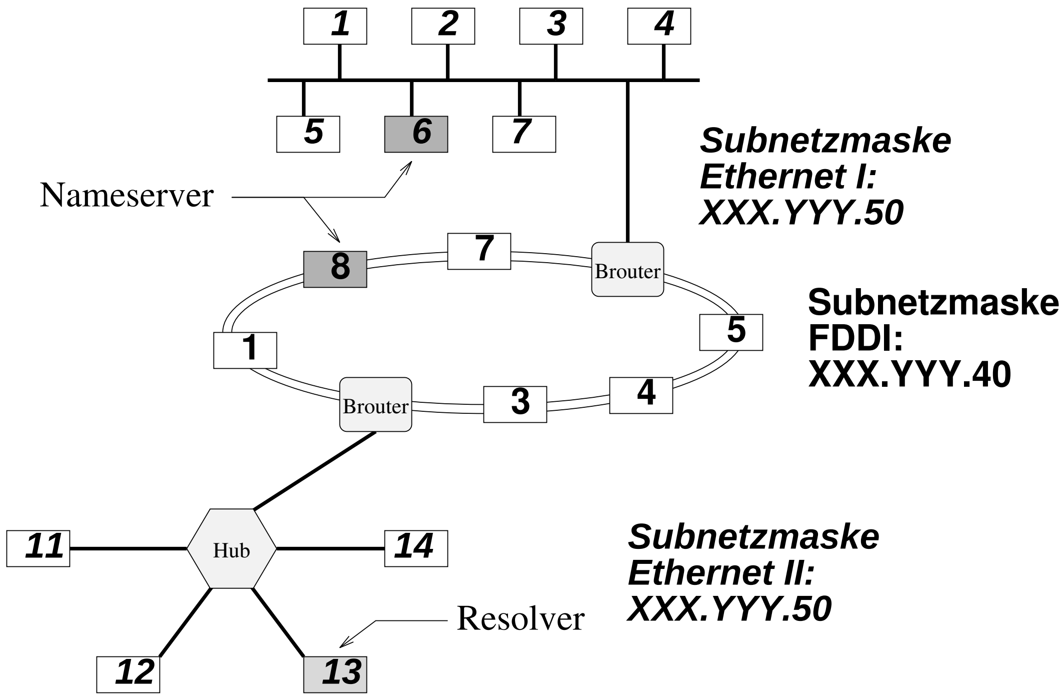

Abbildung 6.3: Geteiltes Subnetz

Service anbietet. Hierzu zählen auch die Standardeinträge für das Loopback Interface. Als nächstes folgen die gespiegelten Zonen (Secondary Service), wobei zuerst die lokalen und dann die fremden Zonen angegeben werden.

Darauf folgen evtl. gegebene Forwarding-Einträge. Sind dergleichen vorhanden und ist der Namerserver als Slave Nameserver aufgeführt, so wird ein entsprechender Eintrag vollzogen. Als letztes ist noch ein Eintrag für den Cache Service der Root-Domain vonnöten, der wie beim Loopback Interface ebenfalls standardmäßig erfolgt.

6.4.4 Zonenkonfiguration für Forward Mapping

Das Modul *named_forward.pl* (Listing siehe Anhang A.7) dient zur Zonenkonfiguration (Forward Mapping).

In die Zonenkonfigurationsdatei werden znächst der Dateikopf und der SOA-Record für die Zone eingetragen.

Anschließend wird der Primary Nameserver der Zone angegeben. Dieser und alle folgenden Namerserver werden nicht mit ihren IP-Adressen, sondern mit ihren

Host- und ggf. Domainnamen bezeichnet. Dadurch können Glue Records erforderlich werden. Alle Kandidaten hierfür werden deshalb in eine Liste eingetragen.

Bei den Secondary Nameservern für die Zone werden zunächst die lokalen und dann die fremden berücksichtigt.

Als nächstes sind alle von der Zone delegierten lokalen Zonen anzugeben mit ihrem Primary Nameserver, ihren lokalen und fremden Secondary Nameservern. Dieser Vorgang wiederholt sich anschließend für die von der Zone delegierten fremden Zonen, wobei diesmal bei den Secondary Nameservern zuerst die fremden berücksichtigt werden.

Nachdem dem Sortieren der Glue-Records werden mit den Mail Routing Einträgen die ersten Zoneninformationen eingetragen. Darauf folgen die Einträge für die einzelnen Hosts, jeweils mit IP-Adresse(n), Alias-Namen, CPU- und Betriebssystemangaben und angebotenen Netzdiensten, sofern diese in der Datenbank angegeben wurden.

Schließlich müssen noch verbleibende Nameserver-Hosts als Glue-Records eingetragen werden, wenn diese in der zur Zone gleichnamigen Domain liegen.

6.4.5 Zonenkonfiguration für Reverse Mapping

Das Modul *named_reverse.pl* (Listing siehe Anhang A.8) stellt das Gegenstück zu dem soeben genannten dar. Es dient ebenfalls der Zonenkonfiguration, allerdings bzgl. des Reverse Mappings.

Hier werden ebenfalls zunächst Dateikopf und SOA-Record generiert. Im folgenden wird für die Zone eine Subnetzmaske erstellt, mit deren Hilfe sich später alle betroffenen IP-Adressen herausfiltern lassen. Anschließend werden wieder der Primary und die Secondary Nameserver für die Zone angegeben. Glue-Records entfallen beim Reverse Mapping.

In der Folge findet wie beim Forward Mapping das Delegieren statt. Hier kommt jedoch dazu, daß man sich die zu den delegierten Zonen gehörenden Subnetzmasken merken muß, da die entsprechenden IP-Adressen dann zu diesen Zonen gehören und nicht zur aktuellen Zone[5].

Schließlich werden die Abbildungen der betroffenen IP-Adressen auf deren Fully Qualified Domain Name (PTR-Records) angegeben, die zur Subnetzmaske passen, aber nicht durch das Delegieren ausgeschlossen werden.

[5]IP-Adressen (von lokalen Hosts), die dadurch in fremd delegierte Zonen „abgeschoben“ würden, werden in der aktuellen Zone dennoch berücksichtigt, um deren Reverse Mapping sicherzustellen. Dieser Fall sollte in der Praxis (bei konsistentem Datenbankinhalt) nicht vorkommen, ist aber bei über verschiedene Zonen verteilte Hosts in einem Subnetz nicht vollkommen auszuschließen.

6.4.6 Loopback Interface (127.1 → localhost)

Das Modul *named_local.pl* (Listing siehe Anhang A.9). ist für den Reverse Mapping Teil des Loopback Interfaces zuständig.

Neben dem Kopf der Konfigurationsdatei und dem SOA-Record werden der aktuelle Nameserver (für die Zone **0.0.127.in-addr.arpa**) und der notwendige PTR-Record angegeben.

6.4.7 Loopback Interface (localhost → 127.1)

Das Modul *named_loopback.pl* (Listing siehe Anhang A.10) ist für den Forward Mapping Teil des Loopback Interfaces zuständig.

Neben dem Kopf der Konfigurationsdatei und dem SOA-Record werden der aktuelle Nameserver (für die Zone **localhost**) und der notwendige A-Record angegeben.

6.4.8 Root-Zonenkonfigurationsdatei holen

Das Modul *named_cache.pl* (Listing siehe Anhang A.11) enthält eine Prozedur, die mittel Anonymous-FTP die Konfigurationsdatei für die Root-Domain (Cache Service) holt.

Dazu ist eine *.netrc*-Datei im Homeverzeichnis des Programmanwenders erforderlich, die das Einloggen beim FTP übernimmt. Diese wird ggf. erstellt bzw. ergänzt. Anschließend erfolgt das Transferieren der Datei.

6.5 Ein- und Ausgabe des Programms anhand eines Beispiels

Nachdem bisher die Grundlagen für das Werkzeug erläutert und eine Programmbeschreibung gegeben worden ist, soll nun anhand eines kleinen Beispiels seine Wirkungsweise demonstriert werden.

Dazu ist in der Datenbank ein konsistenter Datenbestand anzulegen. Der für dieses Beispiel ausgewählte Datenbestand, der einen (kleinen) Teil der echten am Leibniz-Rechenzentrum verwalteten DNS-Informationen wiedergibt, wird in Anhang B angegeben. Er stellt zusammen mit der in Anhang A.1 angegebenen Konfigurationsdatei die Eingabedatenmenge des Programms dar.

Wird das Programm damit zum Laufen gebacht, so erscheint folgende Ausgabe:

```
sun1.(0)./home/a/a2824ch/dipl/perl> ora2dns
Starting execution...
Creating directory '/home/a/a2824ch/dipl/perl/ftp/dfvgate.lrz-muenchen.de'
Creating directory '/home/a/a2824ch/dipl/perl/ftp/dfvgate.lrz-muenchen.de/etc'
[...]
Creating directory '/home/a/a2824ch/dipl/perl/ftp/a2824ch.ppp.lrz-muenchen.de'
Creating directory '/home/a/a2824ch/dipl/perl/ftp/a2824ch.ppp.lrz-muenchen.de/etc'
[...]
Creating directory '/home/a/a2824ch/dipl/perl/ftp/hp11.lrz-muenchen.de'
Creating directory '/home/a/a2824ch/dipl/perl/ftp/hp11.lrz-muenchen.de/etc'
Creating directory '/home/a/a2824ch/dipl/perl/ftp/dfvgate.lrz-muenchen.de/etc/namedb'
(remote-file) (local-file)
Creating directory '/home/a/a2824ch/dipl/perl/ftp/hp10.lrz-muenchen.de/etc/namedb'
(remote-file) (local-file)
Ready.
```

Nach einer Startmeldung, die erscheint, wenn der Oraperl-Interpreter den Quelltext fertig vorüberstetzt hat, erscheinen Meldungen über das Generieren von Unterverzeichnissen, in denen die Konfigurationsdateien, die selbst nicht angegeben werden, abgelegt werden. Die Meldung „(remote-file) (local-file)“ wird vom FTP erzeugt, womit die Zonendateien für die Root-Domain, die nicht selbst erzeugt werden, geholt werden.

Nach erfolgreichem Programmlauf („Ready.“-Meldung) stehen u.a. folgende Konfigurationsdateien bereit:

```
ftp/dfvgate.lrz-muenchen.de/etc/resolv.conf                          (*)
ftp/dfvgate.lrz-muenchen.de/etc/namedb/lrz-muenchen.de.DNF           (*)
ftp/dfvgate.lrz-muenchen.de/etc/namedb/ppp.lrz-muenchen.de.DNF
ftp/dfvgate.lrz-muenchen.de/etc/namedb/187.129.in-addr.arpa.DNF      (*)
ftp/dfvgate.lrz-muenchen.de/etc/namedb/15.187.129.in-addr.arpa.DNF
ftp/dfvgate.lrz-muenchen.de/etc/namedb/tu-muenchen.de.DNF
ftp/dfvgate.lrz-muenchen.de/etc/namedb/named.local
ftp/dfvgate.lrz-muenchen.de/etc/namedb/named.loopback
ftp/dfvgate.lrz-muenchen.de/etc/namedb/named.cache
ftp/dfvgate.lrz-muenchen.de/etc/named.boot                           (*)
ftp/suncollect.lrz-muenchen.de/etc/resolv.conf
ftp/hp10.lrz-muenchen.de/etc/resolv.conf
ftp/hp10.lrz-muenchen.de/etc/namedb/named.local
ftp/hp10.lrz-muenchen.de/etc/namedb/named.loopback
ftp/hp10.lrz-muenchen.de/etc/namedb/named.cache
ftp/hp10.lrz-muenchen.de/etc/named.boot
ftp/sunmanager.lrz-muenchen.de/etc/resolv.conf
[...]
ftp/hp11.lrz-muenchen.de/etc/resolv.conf
```

Die mit einem Stern gekennzeichneten Dateien werden in gekürzter Form in Anhang C wiedergegeben.

Die nun folgende Aufgabe seitens des DNS-Administrators besteht darin, diese Dateien auf die einzelnen Hosts zu verteilen und die Nameserver (evtl. auch die Resolver) zu (re)konfigurieren (siehe Kapitel 5.5).

Kapitel 7

Ausblick

In der vorliegenden Arbeit wurden Möglichkeiten des Managements des Domain Name Systems aufgezeigt. Dabei wurde ein Schema entwickelt, wie sich DNS-Informationen in ein Dokumentationssystem integrieren lassen. Darauf aufbauend wurde ein Verfahren zur Konfiguration der beteiligten Komponenten präsentiert.

Neben dem Konfigurationsmanagement ist für das Domain Name System das Fehlermanagement von großer Bedeutung, da neben Konfigurationsfehlern auch Fehler auftreten können, die nicht oder nur mittelbar mit der Konfiguration zusammenhängen. Als Beispiele hierfür seien Fehler bei der Übertragung von DNS-Informationen im Netz oder der Absturz eines Nameserverhosts genannt.

Andere Managementbereiche wie z.B. das Sicherheitsmanagement[1], spielen speziell im Zusammenhang mit dem Domain Name System gegenüber den beiden genannten eine eher untergeordnete Rolle. Was das Accounting Management anbetrifft, so wird grundsätzlich nicht pro DNS-Anfrage abgerechnet.

In Abschnitt 7.1 werden Möglichkeiten vorgestellt, wie sich das entwickelte Konfigurationsverfahren noch erweitern bzw. verbessern läßt. Einige Aspekte wurden schon im Laufe der Arbeit angedeutet und sollen hier nochmals zusammengefassend erläutert werden. Neu dagegen sind v.a. die Unterstützung einer Dummy Domain und der Aufbau eines DB-Datenbestandes aus Konfigurationsdateien. Besonders der letzte Punkt sollte dabei herausgestrichen werden, da er das Gegenstück zu dem vorgestellten Verfahren (Generierung von Konfigurationsdateien aus der Datenbank) darstellt.

[1] Beispiel für eine existierende Sicherheitsvorkehrung: Einschränkung des Zugriffs auf Informationen einer bestimmten Zone mittels der Anweisung *xfrnets* in der Nameserver-Konfigurationsdatei [Vixi 93]. Dies ist z.B. für Anwender interessant, die kein Interesse daran haben, daß jeder Netzteilnehmer über das DNS Informationen über deren Rechner (Namen, IP-Adressen, ...) beziehen kann.

Das in dieser Arbeit entwickelte Werkzeug wurde nicht für den isolierten Einsatz konzipiert, sondern für die Einbindung in eine Managementplattform. Eine mögliche Einbindung des Werkzeuges in die Plattform ist Gegenstand des Abschnittes 7.2.

7.1 Erweiterungs- und Verbesserungsmöglichkeiten

Die folgenden Erweiterungs- bzw. Verbesserungsvorschläge betreffen neben dem Werkzeug (Programm) ein erweitertes Datenbankschema (Punkte 3 und 7) sowie neue Informationsquellen (Punkte 4 und 5). Punkt 8 stellt dagegen ein völlig neues Verfahren dar, für das ein eigenständiges Werkzeug entwickelt werden muß.

1. Erweiterte Konsistenzüberprüfungen
   Innerhalb des Programms wird die Datenbank bereits an einigen Stellen auf einen möglichen inkonsistenten Zustand hin überprüft. Beispielsweise wird das Programm mit einer Fehlermeldung abgebrochen, falls für einen Host keine IP-Adressen verfügbar sind (Modul 'named_forward.pl'), oder eine IP-Adresse mehreren Hosts zugeordnet werden kann (Modul 'res_conf.pl'). Derartige Tests sind aber nur an Stellen eingebaut, wo sie für den Generierungsprozeß unerlässlich sind.

   Es könnten aber auch noch weitere Inkonsistenzen in der Datenbank seitens des Programms aufgedeckt werden. Als Beispiel sei hier die Forderung genannt, daß innerhalb einer Domain keine zwei gleichen Hostnamen auftauchen dürfen, auch nicht als Aliasnamen. Damit könnte der DNS-Administrator auf Fehler hingewiesen werden, ohne daß dies für ihn in irgendeiner Form zusätzlichen Arbeitsaufwand bedeuten würde.

2. Generierung der HOSTS-Datei
   Steht das Domain Name System nicht zur Verfügung (z.B. Netzausfall), so wird eine HOSTS-Datei zur Namensauflösung benötigt. Die Datei enthält einen Teil der Informationen, wie sie das DNS bereitstellt. Aus diesem Grunde bietet es sich an, sie bei der Generierung der Nameserver- und Resolverkonfigurationsdateien gleich mitzuerzeugen. Dazu ist das Programm um ein entsprechendes Modul zu erweitern.

3. Mail Routing Datenbank
   Das Mail Routing [RFC 974] stellt neben der Namensauflösung die zweite wichtige, in der Praxis eingesetzte Aufgabe des Domain Name Systems dar. Deshalb wird es auch in seiner vollen erforderlichen Funktionalität seitens des erstellten Datenbankschemas berücksichtigt.

Daneben ließe sich aber auch ein eigenständiges Schema entwickeln, das besondere, für das Routing wichtige Aspekte wie die Topologie von Netzen mit dazwischenliegenden Mail Gateways berücksichtigt. Derartige Informationen gehen bei dem implementierten Schema verloren, da eine einfache Zuordnung von Namen innnerhalb Domains auf Mail Exchanger Hosts erfolgt, die für E-Mail-Management kaum ausreichend sein dürfte.

Eine Anbindung beider Datenbankschematas wäre dann ein weiterer Schritt in Richtung integriertes Management unter Einsatz eines Dokumentationssystems.

4. Verbesserte Nameserverauswahl bei der Resolverkonfiguration
   Der für die Auswahl der Nameserver bei der Resolverkonfiguration eingesetzte Algorithmus und das damit verbundenen Problem wurden bereits in Kapitel 6.4.2 erläutert.

   Für eine bessere Lösung sind folgende Aspekte zu beachten:

   - Die Anzahl der Hops (Knotenpunkte) ist zu minimieren.
   - Es sind kostengünstige (Teil-)Strecken auszuwählen.
   - Geschwindigkeit und Auslastung der (Teil-)Strecken sind zu berücksichtigen.

   Dazu sind mehr als die bisher vorhandenen Informationen bereitzustellen. Es bieten sich zwei Lösungsansätze an, die auch miteinander kombiniert werden können:

   Statisches Verfahren: Für das Dokumentationssystem wird ein entsprechendes Datenbankschema entwickelt, das sämtliche notwendigen Routinginformationen enthält. Bei Einsatz eines CAD-Systems innerhalb des Dokumentationssystems ist auf Konsistenz von dessen Daten mit jenen in der Datenbank zu achten.

   Dynamisches Verfahren: Hier werden die erforderlichen Routing-Informationen dynamisch ermittelt, z.B. mit Hilfe des *traceroute*-Kommandos. Dabei ist aber zu beachten, daß im Moment der Konfiguration eine Ausnahmesituation im Netz (z.B. Ausfall, Überlastung von Teilstrecken) vorliegen kann. Deshalb ist es sinnvoll, die Informationen von Zeit zu Zeit immer wieder zu ermitteln, um sich eine Statistik anzulegen, die dann als Grundlage für die Auswahl herangezogen werden kann.

5. Automatisches Generieren von WKS-Records
   Informationen über das Angebot von Netzdiensten eines Hosts (unter einer bestimmten IP-Adresse) spielen in der Praxis des Domain Name Systems

eine untergeordnete Rolle. Deshalb wird dafür seitens des implementierten Verfahrens auch nur ein auf das Notwendigste beschränkter Support geboten (siehe Kapitel 6.1.2).

Hat der DNS-Administrator dagegen Zugriff auf die Datei '/etc/services' der betroffenen Unix-Rechner, so ließen sich die WKS-Records automatisch generieren, da sie die gesuchten Informationen enthalten.

6. Automatische Seriennummernverwaltung für SOA-Records
   In der Datenbank werden an verschiedenen Stellen die erforderlichen Seriennummern für die SOA-Records der einzelnen Zonen explizit abgelegt. Diese müssen bei einer Änderung von Informationen in der entsprechenden Zone von Hand erhöht werden.

   Als bequemere Alternative würde es sich anbieten, für alle Zonen eine globale Seriennummer zu verwenden, die bei jedem Programmablauf automatisch erhöht wird. Dazu müßte diese z.B. in einer Datei abgelegt werden.

   Ein Argument gegen eine globale Seriennummer ist die dann eintretende unnötige Spiegelung von Zonen, bei denen sich außer einer Modifikation der Seriennummer im SOA-Record nichts geändert hat. Deshalb wurde auch bei der Implementierung davon abgesehen.

7. Unterstützung einer Dummy-Domain
   Hierbei handelt es sich um einen Vorgehensweise, die am Leibniz-Rechenzentrum vom zuständigen DNS-Administrator Herrn W. Spirk erfolgreich eingesetzt wird, um folgendens oft auftretendes Problem zu lösen: Oft werden unsinnige DNS-Anfragen (z.B. falsche Domainnamen) ins Netz geschickt, und diese wandern dann durchs Netz, da kein Nameserver dafür zuständig ist.

   Um dies zu unterbinden, hat er hier eine sogenannte Dummy-Domain eingeführt, die derartige Namen auf die IP-Adresse **127.1** (**localhost**) abbildet. Der Benutzer, der eine solche Anfrage initiiert hat, erhält als Ergebnis also die IP-Adresse seines eigenen Rechners, und das Netz wird dadurch nicht unnötig belastet.

   Um dies seitens des in dieser Arbeit präsentierten Verfahrens zu unterstützen, wäre das Datenbankschema um eine Relation und das Programm um ein entsprechendes Modul zu erweitern.

8. Aufbau eines DB-Datenbestandes aus Konfigurationsdateien
   Das vorgestellte Verfahren setzt voraus, daß sämtlichen DNS-Informationen in der Netzdokumentationsdatenbank vorhanden sind bzw. dort eingegeben werden. In den existierenden DNS-Umgebungen sind dagegen zumeist nur die Konfigurationsdateien vorhanden.

Um sich beim Einsatz des Verfahrens eine erhebliche Menge Arbeit zu sparen, die bei der Eingabe der Daten in die Datenbank anfällt, ist ein eigenes Teilverfahren zu entwickeln, bei dem der Eingabeprozeß automatisiert wird. Hierfür stehen die Daten aus den vorhandenen Konfigurationsdateien zur Verfügung.

Es ist also ein Programm *dns2ora* zu entwickeln, das als Werkzeug für das Teilverfahren dient. Seine Aufgabe besteht darin, die gegebenen Konfigurationsdateien auszulesen und die darin vorhandenen Informationen in die Datenbank zu schreiben. *dns2ora* stellt das Gegenstück zu *ora2dns* dar.

7.2 Einbindung in eine Managementplattform

Abbildung 7.1 gibt einen Überblick, wie das entwickelte Werkzeug in eine Managementplattform eingebunden werden kann. Als Plattform soll das Produkt OpenView der Firma Hewlett-Packard zum Einsatz kommen. Im folgenden soll nur die Nameserverkonfiguration betrachtet werden, da die viel seltener vollzogene Resolver(re)konfiguration in ähnlicher Weise abläuft.

Die in der Abbildung eingetragenen Pfeile haben folgende Bedeutung:

1. Der Generierungsprozeß wird von der Plattform aus angestoßen. Dazu ist das anwendungsspezifische Modul als graphisches Symbol auf der Plattformoberfläche darzustellen[2].

2. Den nächsten Schritt stellt die Verteilung der erzeugten Konfigurationsdateien auf die betroffenen Rechner dar. Dieser Prozeß sollte durch Anklicken eines graphischen Symbols – ggf. zusammen mit dem ersten und dem noch folgenden – initiiert werden können. Die zur Verfügung stehenden Ladeprotokolle werden in Kapitel 5.5.2 erwähnt.

3. Der letzte Konfigurationsschritt besteht darin, den Nameservern mitzuteilen, daß sie ihre internen Datenbanken aus den Konfigurationsdateien neu laden sollen. Dazu ist ein Einloggen bzw. ein Remote-Shell-Kommando auf dem jeweiligen Nameserver-Rechner erforderlich.

4. Neben der Konfiguration sind die Nameserver von der Plattform aus auch zu überwachen. Dies betrifft beispielsweise das Ereignismangement: Fällt z.B. ein Nameserver aus, so ist ein Event zu erzeugen, daß den DNS-Administrator auf diesen Mißstand hinweist und u.U. sofort einen neuen Prozeß erzeugt.

[2]Vgl. Konfigurieren mittels herstellerspezifischer Moduln [Hege 93].

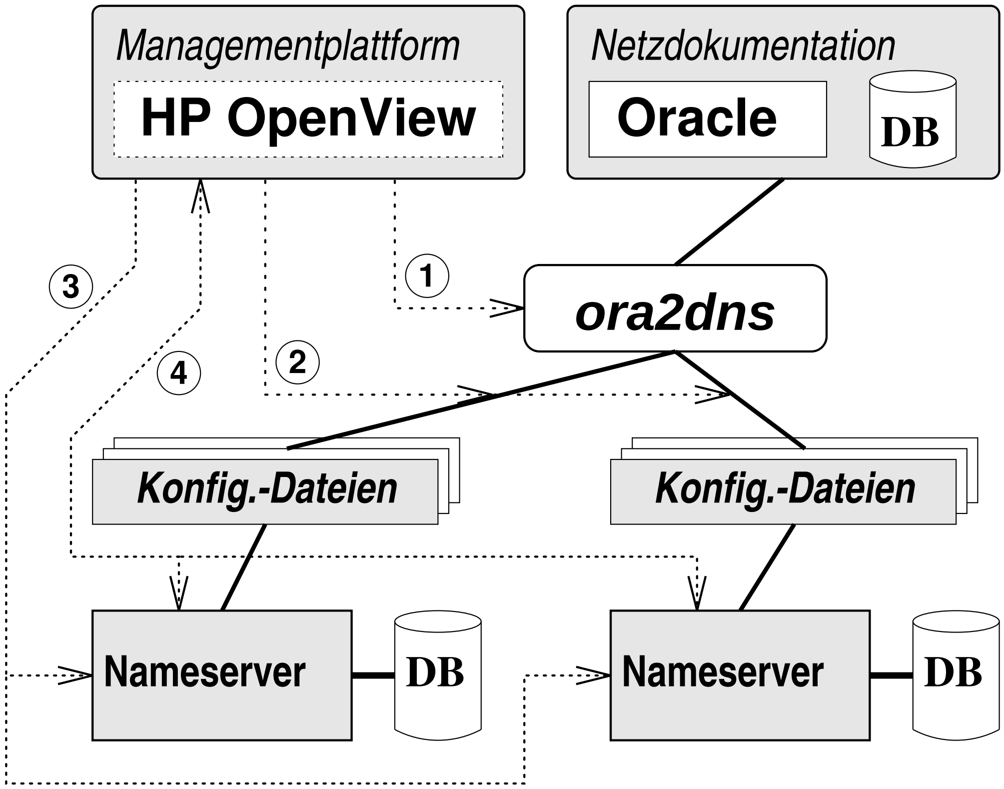

Abbildung 7.1: Einbindung in eine Managementplattorm

Die direkte Anbindung der Managementplattform an das Datenbanksystem bzw. dieser beiden an ein Trouble Ticket System (siehe auch Kapitel 4.4.2) wurde in der Darstellung nicht berücksichtigt.

Der Einsatz einer Managementplattform soll dem Netzverwalter neben der Oberflächenintegration der einzelnen zu erledigenden Aufgaben weitere Vorteile bieten. Einer wichtiger davon sind Views, also verschiedene, auf die jeweiligen Bedürfnisse angepaßten Betrachtungsweisen ein und desselben Problems.

Für das Management des Domain Name Systems sind zwei verschiedene Betrachtungsweisen der zu managenden Objekte vorstellbar:

Komponentensicht: Hier wird der Rechner, auf dem der Nameserver (bzw. Resolver) läuft, als Komponente im Netz (topologische Sicht) betrachtet. Mittels eines Managementprotokolls werden für den dort installierten Nameserver Managementoperationen, die auch die von dem Nameserver verwalteten Zonen betreffen, durchgeführt. Dieser Weg wird im Rahmen der Internet-Managementarchitektur (siehe Kapitel 4.4.1) beschritten.

Domainsicht: Bei der oben genannten Sicht ist es erforderlich, einen Nameserverhost für eine bestimmte Zone zu kennen, um an deren Managementinformationen zu gelangen. Dies widerspricht der Struktur des Domain Name Systems: Zuerst werden die Domains, die Zonen, die hierfür verantworlichen Nameserver und schließlich die Rechner, auf denen die Nameserver laufen, festgelegt. Eine derartige Auswahlstrategie sollte auch seitens der Plattform ermöglicht werden.

Während die erste Sicht den Vorteil bietet, mittels SNMP in bewährter Weise Management betreiben zu können, ist die zweite Sicht für jemanden erforderlich, der sich schon längere Zeit mit dem DNS beschäfigt, die Strukturen kennt und daran festhalten möchte.

Eine Implementierung sollte beide Sichtweisen unterstützen.

Anhang A

Listings

A.1 Konfigurationsdatei .ora2dnsrc

```
# *******************************************************************
# * Integration des Managements des Domain-Name-Systems in die      *
# * vorhandene Managementumgebung des Leibniz-Rechenzentrums (LRZ)  *
# * (Diplomarbeit an der TU-Muenchen, SS 94 - WS 94/95)             *
# * --------------------------------------------------------------- *
# * Autor: Heiko Hauck (hauck@informatik.tu-muenchen.de)            *
# * Datei: .ora2dnsrc (Konfigurationsdatei fuer ora2dns)            *
# *******************************************************************

# Format dieser Datei:
# <Variablenname> [<Leerzeichen> <Variablenbelegung>]
# Wird keine Variablenbelegung angegeben, so wird ein Leerstring verwendet.
# Kommentare muessen mit einem '#'-Zeichen beginnen.
# Leerzeilen sind erlaubt.

# DB-Qualifier, der bei saemtlichen DB-Anfragen vor den Tabellennamen
# eingefuegt wird
ORA2DNS.QUALIFIER
# Verzeichnis, in dem die erstellten Konfigurationsdateien
# (in entsprechenden Unterverzeichnissen) abgelegt werden
ORA2DNS.FTP_DIR /home/a/a2824ch/dipl/perl/ftp

# TTL-Wert fuer A-RRs in den Zonendateien
HOSTS.RECORD_TTL
# Klasse der A-RRs in den Zonendateien
HOSTS.RECORD_CLASS IN

# TTL-Wert fuer MX-RRs in den Zonendateien
MAILX.RECORD_TTL
# Klasse der MX-RRs in den Zonendateien
MAILX.RECORD_CLASS IN
```

```
# Dateiname fuer den Nameserverprozess
NAMESERVERS.FILENAME_EXE /etc/named
# Name der Datei, in der die ID des Nameserverprozesses steht
NAMESERVERS.FILENAME_PID /etc/named.pid
# Klasse des SOA-Records (fuer Loopback-Interface des Nameservers)
NAMESERVERS.SOA_CLASS IN
# Refresh-Intervall des SOA-Records (fuer Loopback-Interface des Nameservers)
NAMESERVERS.SOA_REFRESH 21600
# Retry-Intervall des SOA-Records (fuer Loopback-Interface des Nameservers)
NAMESERVERS.SOA_RETRY 3600
# Expire-Intervall des SOA-Records (fuer Loopback-Interface des Nameservers)
NAMESERVERS.SOA_EXPIRE 604800
# Minimum-Intervall des SOA-Records (fuer Loopback-Interface des Nameservers)
NAMESERVERS.SOA_MINIMUM 172800
# Verzeichnis, in dem die Zonendateien abgelegt werden
NAMESERVERS.FILES_DIR /etc/namedb
# Name der Zonendatei fuer das Loopback-Interface (127.1 ->localhost)
NAMESERVERS.FILENAME_LO named.local
# Name der Zonendatei fuer das Loopback-Interface (localhost -> 127.1)
NAMESERVERS.FILENAME_LB named.loopback
# Name der Root-Zonendatei (Cache Service)
NAMESERVERS.FILENAME_CA named.cache
# Host, von dem diese Datei mittels Anonymous FTP geholt wird
NAMESERVERS.FTP_HOST rs.internic.net
# Verzeichnis auf diesem Host
NAMESERVERS.FTP_DIR /domain
# Dateiname auf diesem Host
NAMESERVERS.FTP_FILE named.cache

# Dateiendung, die an den Zonendateinamen angefuegt wird, falls hierfuer
# der jeweilige Domainname verwendet wird (lokale Zonen - Secondary Service)
SECONDARY_LL.BACKUPFILE_SUFFIX .BF

# Dateiendung, die an den Zonendateinamen angefuegt wird, falls hierfuer
# der jeweilige Domainname verwendet wird (fremde Zonen - Secondary Service)
SECONDARY_LF.BACKUPFILE_SUFFIX .BF

# Dateiendung, die an den Zonendateinamen angefuegt wird, falls hierfuer
# der jeweilige Domainname verwendet wird
ZONES.SOURCEFILE_SUFFIX .DNF
# Klasse des SOA-Records
ZONES.SOA_CLASS IN
# Refresh-Intervall des SOA-Records
ZONES.SOA_REFRESH 21600
# Retry-Intervall des SOA-Records
ZONES.SOA_RETRY 3600
# Expire-Intervall des SOA-Records
ZONES.SOA_EXPIRE 604800
# Minimum-Intervall des SOA-Records
ZONES.SOA_MINIMUM 172800
```

A.2 Hauptprogramm ora2dns

```
#!/home/a/a2824ch/bin/oraperl
# ********************************************************************
# * Integration des Managements des Domain-Name-Systems in die        *
# * vorhandene Managementumgebung des Leibniz-Rechenzentrums (LRZ)     *
# * (Diplomarbeit an der TU-Muenchen, SS 94 - WS 94/95)               *
# * ---------------------------------------------------------------- *
# * Autor: Heiko Hauck (hauck@informatik.tu-muenchen.de)              *
# * Datei: ora2dns (Hauptprogramm)                                    *
# ********************************************************************

# AUFRUF: ora2dns [username password]
# Werden Benutzername und Passwort nicht mit angegeben,
# so werden hierfuer Standardangaben eingesetzt
# (siehe Prozedur login in dem Modul 'ora2dns.pl')

# Patch fuer die lokal installierte Version von 'oraperl'
@INC = ('.', '/sw/PD/perl-4.36/lib');

# Einbinden der Programmodule
require 'ora2dns.pl';
require 'make_dir.pl';
require 'res_conf.pl';
require 'named_boot.pl';
require 'named_cache.pl';
require 'named_forward.pl';
require 'named_reverse.pl';
require 'named_local.pl';
require 'named_loopback.pl';

# Programmstart
print "Starting execution...\n";

# Setzen bzw. Einlesen der Defaultwerte (aus Datei '$HOME/.ora2dnsrc')
&set_default;

# DB-Qualifier (globale Variable)
# wird bei den SQL-Abfragen vor alle Tabellennamen gesetzt
$DB = $def{'ORA2DNS.QUALIFIER'};

# Einloggen in die Datenbank
&login;

# Generierung der Host-Unterverzeichnisse
&make_dir;

# Resolverkonfiguration fuer jeden Host
&res_conf;

# ====================================================================
```

```
# Nameserverkonfiguration

# hierzu notwendige (globale) SQL-Anfragen:

# Auswahl saemtlicher Nameserver mit ihren Daten
&query(<<"END_SQL", *nameservers);
  select id, host, soa_responsible, soa_class, soa_serial, soa_refresh,
         soa_retry, soa_expire, soa_minimum, files_dir, filename_lo,
         filename_lb, filename_ca, ftp_host, ftp_dir, ftp_file
  from ${DB}nameservers
END_SQL

# Auswahl der Daten eines bestimmten Hosts und der zughoerigen Zone
$csr_hosts = &query_open(<<"END_SQL");
  select h.name, h.subdomain_name, h.zone, h.record_ttl, h.record_class,
         z.name, z.soa_responsible, z.soa_class, z.soa_serial,
         z.soa_refresh, z.soa_retry, z.soa_expire, z.soa_minimum
  from ${DB}hosts h, ${DB}zones z
  where h.zone = z.id and h.id = :1
END_SQL

# Auswahl aller IP-Adressen eines bestimmten Hosts
$csr_ipaddr = &query_open(<<"END_SQL");
  select ip_addr from ${DB}ipaddr where host = :1
END_SQL

# Auswahl aller Zonen, die einen bestimmten Primary Nameserver haben
$csr_zones = &query_open(<<"END_SQL");
  select id, name, sourcefile, soa_responsible, soa_class, soa_serial,
         soa_refresh, soa_retry, soa_expire, soa_minimum
  from ${DB}zones
  where prim_nameserver = :1
END_SQL

# ----------------------------------------------------------------------

# Schleife ueber saemtliche Nameserver

for(@nameservers)
{
  # Einlesen der Nameserverdaten

  local($id, $host, $soa_responsible, $soa_class, $soa_serial, $soa_refresh,
   $soa_retry, $soa_expire, $soa_minimum, $files_dir, $filename_lo,
   $filename_lb, $filename_ca, $ftp_host, $ftp_dir, $ftp_file) = split(/$;/);
  # $id, $host koennen nicht NULL sein
  # die restlichen Variablen duerfen NULL sein

  # Einlesen der Daten des zugehorigen Hosts

  local(@hosts);
```

```
&query_bind($csr_hosts, *hosts, $host);
# Ergebnis kann nur aus einem Tupel bestehen, da der Schluessel zones.id
# eindeutig ist (unique index)
local($hostname, $subdomainname, $zone, $record_ttl, $record_class,
 $domainname, $z_soa_responsible, $z_soa_class, $z_soa_serial,
 $z_soa_refresh, $z_soa_retry, $z_soa_expire, $z_soa_minimum)
 = split(/$;/, $hosts[0]);
# $hostname, $zone, $domainname, $z_soa_responsible & $z_soa_serial koennen
# nicht NULL sein
# $subdomainname, $record_ttl, $record_class, $z_soa_class, $z_soa_refresh,
# $z_soa_retry, $z_soa_expire & $z_soa_minimum duerfen NULL sein

# ---------------------------------------------------------------------

# Belegen der optionalen Attribute, falls
# sie in der Datenbank nicht belegt wurden (NULL-Werte)

# Eintraege fuer die SOA-Records der Zonen des Loopback-Interfaces
# (Forward & Reverse Mapping)

# Sind die folgenden beiden Attribute nicht speziell fuer den
# Nameserver angegeben, so werden die Werte der Zone, zu der
# der Nameserver-Host gehoert, verwendet

if(&testnull($soa_responsible))
  { $soa_responsible = $z_soa_responsible; }
if(&testnull($soa_serial))
  { $soa_serial = $z_soa_serial; }

# Im Gegensatz zu den beiden obigen Attributen kommt bei den folgenden
# noch der Default-Wert als letzte Alternative hinzu

if(&testnull($soa_class))
  { $soa_class = &testnull($z_soa_class) ?
                 $def{'NAMESERVERS.SOA_CLASS'} : $z_soa_class; }
if(&testnull($soa_refresh))
  { $soa_refresh = &testnull($z_soa_refresh) ?
                   $def{'NAMESERVERS.SOA_REFRESH'} : $z_soa_refresh; }
if(&testnull($soa_retry))
  { $soa_retry = &testnull($z_soa_retry) ?
                 $def{'NAMESERVERS.SOA_RETRY'} : $z_soa_retry; }
if(&testnull($soa_expire))
  { $soa_expire = &testnull($z_soa_expire) ?
                  $def{'NAMESERVERS.SOA_EXPIRE'} : $z_soa_expire; }
if(&testnull($soa_minimum))
  { $soa_minimum = &testnull($z_soa_minimum) ?
                   $def{'NAMESERVERS.SOA_MINIMUM'} : $z_soa_minimum; }

# Bei den folgenden Attributen stehen neben Nameserver-spezifischen
# Angaben Default-Werte zur Verfuegung
```

```
# Verzeichnis, in dem die Zonendateien abgelegt werden
if(&testnull($files_dir))
  { $files_dir = $def{'NAMESERVERS.FILES_DIR'}; }

# Name fuer Zonendatei des Loopback-Interfaces (127.1 -> localhost)
if(&testnull($filename_lo))
  { $filename_lo = $def{'NAMESERVERS.FILENAME_LO'}; }

# Name fuer Zonendatei des Loopback-Interfaces (localhost -> 127.1)
if(&testnull($filename_lb))
  { $filename_lb = $def{'NAMESERVERS.FILENAME_LB'}; }

# Name fuer Zonendatei fuer Root-Domain (Cache)
if(&testnull($filename_ca))
  { $filename_ca = $def{'NAMESERVERS.FILENAME_CA'}; }

# FTP Host, von dem obige Datei geholt wird
if(&testnull($ftp_host))
  { $ftp_host = $def{'NAMESERVERS.FTP_HOST'}; }

# Quellverzeichnis auf dem FTP-Server
if(&testnull($ftp_dir))
  { $ftp_dir = $def{'NAMESERVERS.FTP_DIR'}; }

# Name der Datei auf dem FTP-Server
if(&testnull($ftp_file))
  { $ftp_file = $def{'NAMESERVERS.FTP_FILE'}; }

# Die folgenden beiden Attribute werden u.U. fuer das Erzeugen
# eines Glue-Records fuer den aktuellen Nameserver-Host beim
# Forward-Mapping benoetigt; neben Host-spezifischen Angaben
# stehen Default-Werte zur Verfuegung

if(&testnull($record_ttl))
  { $record_ttl = $def{'HOSTS.RECORD_TTL'}; }
if(&testnull($record_class))
  { $record_class = $def{'HOSTS.RECORD_CLASS'}; }

# --------------------------------------------------------------------

# Aufbau des Fully Qualified Domain Name fuer den Nameserver-Host

local($fqdn);
if((!&testnull($subdomainname)) && (length($subdomainname) > 0))
{
  $fqdn = $hostname . '.' . $subdomainname . '.' . $domainname;
}
else
{
  $fqdn = $hostname . '.' . $domainname;
}
```

```
# Generierung der zugehoerigen IP-Adresse
# Im Falle mehrerer IP-Adressen wird die "erste" ausgewaehlt
# Die Liste der IP-Adressen wird ausserdem fuer die Boot-Datei benoetigt

local(@ipaddr);
&query_bind($csr_ipaddr, *ipaddr, $host);
# $ipaddr[..] kann nicht NULL sein
local($ipa) = $ipaddr[0];

# Loeschen von Leerzeichen in der Adresse (DB-Konvention)
$ipa =~ s/\x20//g;

# ----------------------------------------------------------------------

# Verzeichnis, in dem Daten fuer den Nameserver-Host abgelegt werden

local($directory) = $def{'ORA2DNS.FTP_DIR'} . '/' . $fqdn;

# Loeschen fuehrender Slashes ('/') in den Namesattributen

$files_dir =~ s/^\///o;
$filename_lo =~ s/^\///o;
$filename_lb =~ s/^\///o;
$filename_ca =~ s/^\///o;

# Generierung des Verzeichnisses fuer die Zonendateien

&make_dir_ns($directory, $files_dir);

# ----------------------------------------------------------------------

# Generierung der Boot-Datei

# Die Standard-Ausgabe wird fuer das Unterprogramm auf die
# entprechende Datei umgelenkt

local($filename) = $directory . '/etc/named.boot';
open(NF, ">$filename") || die "Can't open $filename: $!\n";
local($OLD_HANDLE) = select(NF);
&named_boot($ipa, $fqdn, $id, $filename_ca, *ipaddr, $files_dir,
            $filename_lo, $filename_lb);
select($OLD_HANDLE);
close(NF);

# ----------------------------------------------------------------------

# Verzeichnis fuer Zonendateien

$directory .= '/' . $files_dir. '/';
```

```
# ---------------------------------------------------------------------

# Generierung der Zonen-Dateien fuer Forward- & Reverse-Mapping

local(@zones);
&query_bind($csr_zones, *zones, $id);

# Schleife ueber saemtliche Zonen, die von dem aktuellen Nameserver
# als Primary Nameserver verwaltet werden

for(@zones)
{
  local($id, $name, $sourcefile, $soa_responsible, $soa_class,
   $soa_serial, $soa_refresh, $soa_retry, $soa_expire, $soa_minimum)
   = split(/$;/);
  # $id, $name, $soa_responsible & $soa_serial koennen nicht NULL sein
  # $sourcefile, $soa_class, $soa_refresh, $soa_retry, $soa_expire &
  # $soa_minimum duerfen NULL sein

  # Belegen der optionalen Attribute, falls
  # sie in der Datenbank nicht belegt wurden (NULL-Werte)

  # Sind die folgenden Attribute nicht speziell fuer Zone angegeben,
  # so werden Default-Werte verwendet

  # Endung des Namens der Zonendatei

  if(&testnull($sourcefile))
  {
    $sourcefile = $name . $def{'ZONES.SOURCEFILE_SUFFIX'};
  }

  # Eintraege fuer den SOA-Record der Zone

  if(&testnull($soa_class))
    { $soa_class = $def{'ZONES.SOA_CLASS'}; }
  if(&testnull($soa_refresh))
    { $soa_refresh = $def{'ZONES.SOA_REFRESH'}; }
  if(&testnull($soa_retry))
    { $soa_retry = $def{'ZONES.SOA_RETRY'}; }
  if(&testnull($soa_expire))
    { $soa_expire = $def{'ZONES.SOA_EXPIRE'}; }
  if(&testnull($soa_minimum))
    { $soa_minimum = $def{'ZONES.SOA_MINIMUM'}; }

  # -------------------------------------------------------------------

  # Die Standard-Ausgabe wird fuer das jeweilige Unterprogramm auf die
  # entprechende Datei umgelenkt

  $filename = $directory . $sourcefile;
```

```
  open(NF, ">$filename") || die "Can't open $filename: $!\n";
  $OLD_HANDLE = select(NF);

  # ------------------------------------------------------------------

  # Forward- & Reverse-Mapping Zonendateien erfordern sehr unterschiedliche
  # Generierungsverfahren; Feststellung des Typs anhand des Zonennamens

  if(!($name =~ /\.in-addr\.arpa$/i))
  {
    &named_forward('/' . $files_dir. '/' . $sourcefile, $fqdn, $ipa,
             $soa_class, $soa_responsible, $soa_serial,
             $soa_refresh, $soa_retry, $soa_expire, $soa_minimum,
             $name, $id, $record_ttl, $record_class);
  }
  else
  {
    &named_reverse('/' . $files_dir. '/' . $sourcefile, $fqdn, $ipa,
             $soa_class, $soa_responsible, $soa_serial,
             $soa_refresh, $soa_retry, $soa_expire, $soa_minimum,
             $name, $id);
  }

  # ------------------------------------------------------------------

  select($OLD_HANDLE);
  close(NF);
}

# -------------------------------------------------------------------

# Generierung der Zonen-Datei fuer das Loopback-Interface (127.1 -> localhost)

# Die Standard-Ausgabe wird fuer das Unterprogramm auf die
# entprechende Datei umgelenkt

$filename = $directory . $filename_lo;
open(NF, ">$filename") || die "Can't open $filename: $!\n";
$OLD_HANDLE = select(NF);
&named_local('/' . $files_dir. '/' . $filename_lo, $ipa,
             $soa_class, $fqdn, $soa_responsible, $soa_serial,
             $soa_refresh, $soa_retry, $soa_expire, $soa_minimum);
select($OLD_HANDLE);
close(NF);

# -------------------------------------------------------------------

# Generierung der Zonen-Datei fuer das Loopback-Interface (localhost -> 127.1)

# Die Standard-Ausgabe wird fuer das Unterprogramm auf die
# entprechende Datei umgelenkt
```

```
  $filename = $directory . $filename_lb;
  open(NF, ">$filename") || die "Can't open $filename: $!\n";
  $OLD_HANDLE = select(NF);
  &named_loopback('/' . $files_dir. '/' . $filename_lb, $ipa,
                  $soa_class, $fqdn, $soa_responsible, $soa_serial,
                  $soa_refresh, $soa_retry, $soa_expire, $soa_minimum);
  select($OLD_HANDLE);
  close(NF);

  # --------------------------------------------------------------------

  # Holen der Zonen-Datei fuer die Root-Domain (via Anonymous FTP)
  # Als Passwort wird die E-Mail-Adresse des DNS-Administrators verwendet

  $filename = $directory . $filename_ca;
  &named_cache($filename, $ftp_host, $ftp_dir, $ftp_file, $soa_responsible);
}

# ----------------------------------------------------------------------

# Schliessen der Cursor globaler SQL-Anfragen

&query_close($csr_hosts);
&query_close($csr_ipaddr);
&query_close($csr_zones);

# ======================================================================

# Ausloggen aus der Datenbank
&logout;

# Programmende
print "Ready.\n";
```

A.3 Modul ora2dns.pl

```
# ********************************************************************
# * Integration des Managements des Domain-Name-Systems in die        *
# * vorhandene Managementumgebung des Leibniz-Rechenzentrums (LRZ)    *
# * (Diplomarbeit an der TU-Muenchen, SS 94 - WS 94/95)               *
# * ---------------------------------------------------------------- *
# * Autor: Heiko Hauck (hauck@informatik.tu-muenchen.de)              *
# * Datei: ora2dns.pl (Oraperl Bibliothek fuer ora2dns)               *
# ********************************************************************

# *globale* Variable, auf deren Wert ein DB-Attribut gesetzt wird (vor
# dem Packen), wenn es undefiniert (NULL) ist

$NULL="\0";

# ********************************************************************

# PROZEDUR zum Einlesen der Standard-/Default-Werte (keine Parameter)
# Diese koennen mit Hilfe der Datei $HOME/.ora2dnsrc abgeaendert werden.
# Ergebnis: *globales* assoziatives Array %def

sub set_default
{
  # Standardvorbelegung des Arrays
  %def =
  (
    'ORA2DNS.QUALIFIER', '',
    'ORA2DNS.FTP_DIR', '/home/a/a2824ch/dipl/perl/ftp',
    'HOSTS.RECORD_TTL', '',
    'HOSTS.RECORD_CLASS', 'IN',
    'MAILX.RECORD_TTL', '',
    'MAILX.RECORD_CLASS', 'IN',
    'NAMESERVERS.FILENAME_EXE', '/etc/named',
    'NAMESERVERS.FILENAME_PID', '/etc/named.pid',
    'NAMESERVERS.SOA_CLASS', 'IN',
    'NAMESERVERS.SOA_REFRESH', '21600',
    'NAMESERVERS.SOA_RETRY', '3600',
    'NAMESERVERS.SOA_EXPIRE', '604800',
    'NAMESERVERS.SOA_MINIMUM', '172800',
    'NAMESERVERS.FILES_DIR', '/etc/namedb',
    'NAMESERVERS.FILENAME_LO', 'named.local',
    'NAMESERVERS.FILENAME_LB', 'named.loopback',
    'NAMESERVERS.FILENAME_CA', 'named.cache',
    'NAMESERVERS.FTP_HOST', 'rs.internic.net',
    'NAMESERVERS.FTP_DIR', '/domain',
    'NAMESERVERS.FTP_FILE', 'named.cache',
    'SECONDARY_LL.BACKUPFILE_SUFFIX', '.BF',
    'SECONDARY_LF.BACKUPFILE_SUFFIX', '.BF',
    'ZONES.SOURCEFILE_SUFFIX', '.DNF',
    'ZONES.SOA_CLASS', 'IN',
```

```
  'ZONES.SOA_REFRESH', '21600',
  'ZONES.SOA_RETRY', '3600',
  'ZONES.SOA_EXPIRE', '604800',
  'ZONES.SOA_MINIMUM', '172800'
);

# Name der Konfigurationsdatei
local($filename) = $ENV{'HOME'} . '/.ora2dnsrc';

# Aktuelle Zeilennummer beim Auslesen (fuer Fehlermeldungen)
local($line) = 1;

# Falls die Datei existiert, ...
if(-e $filename)
{
  # ..., wird sie geoeffnet
  open(RCFILE,$filename) || die "Can't open $filename: $!\n";

  # zeilenweises Einlesen
  while(<RCFILE>)
  {
    # Kommentarzeilen werden ignoriert
    if (/^#/) { $line++; next; }

    # Einlesen von Variablenname & -belegung
    local($var, $value) = ('', '');
    ($var, $value) = /^(\S+)\s+(\S+)/;

    # Ist die Belegung leer, so wird der Name dennoch eingelesen
    if($value eq '') { ($var) = /^(\S+)/; }

    # Variablenname gegeben?
    if($var ne '')
    {
      # Im Falle eines undefinierten Namens wird eine Fehlermeldung
      # ausgegeben (kein Programmabbruch)
      if(!defined($def{$var}))
      {
        print STDERR "Unknown variable $var in '$filename' line $line\n";
      }

      else
      # gueltiger Variablenname
      {
        # Folgende Variablen muessen mit einem Wert belegt sein,
        # sonst Fehlermeldung und keine Modifikation des bisherigen Wertes
        if((length($value) == 0) &&
           ($var ne 'ORA2DNS.QUALIFIER') &&
           ($var ne 'HOSTS.RECORD_TTL') &&
           ($var ne 'MAILX.RECORD_TTL') &&
           ($var ne 'SECONDARY_LL.BACKUPFILE_SUFFIX') &&
```

```
            ($var ne 'SECONDARY_LF.BACKUPFILE_SUFFIX') &&
            ($var ne 'ZONES.SOURCEFILE_SUFFIX'))
          {
            print STDERR
              "Variable $var in '$filename' line $line can't be set to ''\n";
          }

          else
          # Modifikation der Variable
          {
            $def{$var} = $value;
          }
        }
      }

      # neue Zeile
      $line++;
    }

    # Konfigurationsdatei schliessen
    close(RCFILE);
  }
}

# *********************************************************************

# PROZEDUR zum Einloggen in die Datenbank
# (keine eigenen Parameter, sondern ggf. die des Programmaufrufes)
# Ergebnis: *globale* Variable $lda (Datenbank-ID)

sub login
{
  # lokale Variablen
  local($login_sid) = 'oracle';
  local($login_name,$login_pw);

  # Werden beim Programmstart zwei (oder mehrere) Parameter angegeben,
  # so werden diese (die ersten beiden) als Login-Name bzw. Passwort
  # fuer die Datenbank verwendet, ...
  if(@ARGV >= 2)
  {
    $login_name = @ARGV[0];
    $login_pw   = @ARGV[1];
  }

  else
  # ..., sonst werden Standardvorgaben eingesetzt:
  {
    # Die folgenden Zeilen sind an die jeweilige Umgebung anzupassen
    $login_name = `cat /home/a/a2824ch/oracle/LOGIN`;
    $login_pw   = `cat /home/a/a2824ch/oracle/PASSWD`;
```

```
#    $login_name = 'nduser';
#    $login_pw   = 'usernd';
  }

  # Einloggen
  # Misslingt dies, dann Programmabbruch
  $lda = &ora_login($login_sid, $login_name, $login_pw) ||
    die "$ora_errstr\n";
}

# ********************************************************************

# PROZEDUR zum Ausloggen aus der Datenbank (keine Parameter)
# Voraussetzung: *globale* Variable $lda (Datenbank-ID) vorher gesetzt

sub logout
{
  # Ausloggen
  # Misslingt dies, dann Programmabbruch
  do ora_logoff($lda) || die "Can't log off Oracle\n";
}

# ********************************************************************

# PROZEDUR fuer eine vollstaendige Datenbankabfrage
# Parameter: $_[0] = Query-Text
#            $_[1] = Pointer auf die Ergebnisliste mit Tupeln
# Im Falle mehrerer abgefragter Attribute werden diese gepackt.
# Voraussetzung: *globale* Variable $lda (Datenbank-ID) vorher gesetzt
# Rueckgabewert: Anzahl der gefundenen Tupel

sub query
{
  # (lokalen) Cursor oeffnen und Abfrage durchfuehren
  # Misslingt dies, dann Programmabbruch
  local($csr) = &ora_open($lda, $_[0]) || die $ora_errstr;

  # lokale Variablen
  local(@entry);
  local(*entrylist) = $_[1];
  local($entrycount) = 0;

  # Liste am Anfang leer
  @entrylist = ();

  # Einlesen saemtlicher Tupel
  while(@entry = &ora_fetch($csr))
  {
    # Ueberpruefung, ob <NULL>-Werte in den Attributen
    # vorhanden sind; in diesem Fall werden sie durch den Wert der
    # globalen Variable $NULL ersetzt (dieser darf also nicht als
```

```
    # Attributbelegung in der DB vorkommen).

    # Wuerde man auf diese Massnahme verzichten, so wuerde
    # die Information eines <NULL>-Wertes nach join/split
    # dahingehend verloren gehen, dass er als Leerstring
    # bzw. numerische Null interpretiert werden wuerde.

    # Ueberpruefung fuer jedes einzelne Attribut
    foreach(@entry)
    {
      $_ = $NULL unless defined($_);
    }

    # Packen aller Attribute eines Tupels zu einem Listenelement
    # und Eintrag in die Ergebnisliste
    push(@entrylist, join($;, @entry));
    $entrycount++;
  }

  # Warnung, falls ein Fehler aufgetreten ist
  warn $ora_errstr if $ora_errno;

  # Cursor schliessen
  # Misslingt dies, dann Programmabbruch
  do ora_close($csr) || die "Can't close Oracle cursor\n";

  # Anzahl der gefundenen Tupel
  $entrycount;
}

# ********************************************************************

# PROZEDUR zum Eroeffnen einer Datenbankabfrage
# Parameter: $_[0] = Query-Text
# Voraussetzung: *globale* Variable $lda (Datenbank-ID) vorher gesetzt
# Rueckgabewert: Cursor fuer die Datenbankabfrage

sub query_open
{
  # Cursor oeffnen
  # Misslingt dies, dann Programmabbruch
  &ora_open($lda, $_[0]) || die $ora_errstr;
}

# ********************************************************************

# PROZEDUR zum Durchfuehren einer Datenbankabfrage
# Parameter: $_[0]    = Cursor fuer die Datenbankabfrage
#            $_[1]    = Pointer auf die Ergebnisliste mit Tupeln
#            $_[2..] = Werte der zu bindenden Query-Variablen
# Im Falle mehrerer abgefragter Attribute werden diese gepackt.
```

```
# Rueckgabewert: Anzahl der gefundenen Tupel

sub query_bind
{
  # lokale Variablen
  local($csr) = shift @_;
  local(@entry);
  local(*entrylist) = shift @_;
  local($entrycount) = 0;

  # Liste am Anfang leer
  @entrylist = ();

  # Binden der Query-Variablen und Abfrage durchfuehren
  # Misslingt dies, dann Programmabbruch
  &ora_bind($csr,@_) || die $ora_errstr;

  # Einlesen saemtlicher Tupel
  while(@entry = &ora_fetch($csr))
  {
    # Ueberpruefung, ob <NULL>-Werte in den Attributen
    # vorhanden sind; in diesem Fall werden sie durch den Wert der
    # globalen Variable $NULL ersetzt (dieser darf also nicht als
    # Attributbelegung in der DB vorkommen).

    # Wuerde man auf diese Massnahme verzichten, so wuerde
    # die Information eines <NULL>-Wertes nach join/split
    # dahingehend verloren gehen, dass er als Leerstring
    # bzw. numerische Null interpretiert werden wuerde.

    # Ueberpruefung fuer jedes einzelne Attribut
    foreach(@entry)
    {
      $_ = $NULL unless defined($_);
    }

    # Packen aller Attribute eines Tupels zu einem Listenelement
    # und Eintrag in die Ergebnisliste
    push(@entrylist, join($;, @entry));
    $entrycount++;
  }

  # Warnung, falls ein Fehler aufgetreten ist
  warn $ora_errstr if $ora_errno;

  # Anzahl der gefundenen Tupel
  $entrycount;
}

# ***********************************************************************
```

```
# PROZEDUR zum Schliessen einer Datenbankabfrage
# Parameter: $_[0] = Cursor fuer die Datenbankabfrage

sub query_close
{
  # Cursor schliessen
  # Misslingt dies, dann Programmabbruch
  do ora_close($_[0]) || die "Can't close Oracle cursor\n";
}

# *****************************************************************

# FUNKTION zum Testen, ob ein Attribut undefiniert ist
# Parameter: $_[0] = entpacktes Attribut
# Rueckgabewert: TRUE, falls undefiniert, sonst FALSE

sub testnull
{
  $_[0] eq $NULL;
}

# *****************************************************************

# PROZEDUR zum Sicherstellen, dass ein Attribut definiert ist
# (sonst Programmabbruch)
# Parameter: $_[0] = entpacktes Attribut
#            $_[1] = Tabellenname
#            $_[2] = Attributname
# Voraussetzung: *globale* Variable $DB (Datenbank-Qualifier) vorher gesetzt

sub notnull
{
  # Programmabbruch mit Fehlermeldung, falls undefiniert
  die "Illegal entry (NULL) for $_[2] in table ${DB}$_[1]\n"
  if $_[0] eq $NULL;
}

# *****************************************************************
1;
```

A.4 Modul make_dir.pl

```
# *******************************************************************
# * Integration des Managements des Domain-Name-Systems in die        *
# * vorhandene Managementumgebung des Leibniz-Rechenzentrums (LRZ)    *
# * (Diplomarbeit an der TU-Muenchen, SS 94 - WS 94/95)               *
# * ----------------------------------------------------------------- *
# * Autor: Heiko Hauck (hauck@informatik.tu-muenchen.de)              *
# * Datei: make_dir.pl (Oraperl Bibliothek fuer ora2dns)              *
# *******************************************************************

# PROZEDUR zum Erstellen eines einzelnen Verzeichnisses innerhalb des
# Unix-Dateibaumes
# Parameter: $_[0] = Name des Verzeichnisses

sub mk_dir
{
  # Das Verzeichnis wird nur erstellt, wenn es noch nicht vorhanden ist
  if(!(-d $_[0]))
  {
    # Erstellung mit Leserechten fuer jedermann
    if(!mkdir($_[0], 0755))
    {
      # Misslingt dies, dann Programmabbruch
      die "Can't create directory '$_[0]'";
    }

    # Bestaetigung
    print "Creating directory '$_[0]'\n";
  }
}

# *******************************************************************

# PROZEDUR zum Erstellen des fuer die Resolverkonfiguration benoetigten
# Verzeichnisbaumes (keine Parameter)

sub make_dir
{
  # Wurzel des neuen Verzeichnisbaumes
  local($dir) = $def{'ORA2DNS.FTP_DIR'};

  # Falls diese nicht exisiert, dann Programmabbruch
  if(!(-d $dir))
  {
    die "Directory '$dir' (for ftp) does not exist";
  }

  # Auswahl aller Hosts mit Name, Subdomain- und Domainname
  # Hierfuer werden innerhalb des angegebenen Verzeichnisses Unterverzeichnisse
  # angelegt
```

```
  local(@hosts);
  &query(<<"  END_SQL", *hosts);
    select h.name, h.subdomain_name, z.name
    from ${DB}hosts h, ${DB}zones z
    where h.zone = z.id
  END_SQL

  # -------------------------------------------------------------------

  # Schleife ueber alle Hosts
  for(@hosts)
  {
    # Parameter fuer den aktuellen Host
    local($host, $subdomain, $domain) = split(/$;/);
    # $host, $domain koennen nicht NULL sein
    # $subdomain darf NULL sein

    # Aufbau des FQDN, der den Unterverzeichnisnamen bestimmt
    local($fqdn);
    if((!&testnull($subdomain)) && (length($subdomain) > 0))
    {
      $fqdn = $host . '.' . $subdomain . '.' . $domain;
    }
    else
    {
      $fqdn = $host . '.' . $domain;
    }

    # Generieren des Unterverzeichnisses und eines darin enthaltenen
    # '/etc'-Verzeichnisses
    &mk_dir($dir . '/' . $fqdn);
    &mk_dir($dir . '/' . $fqdn . '/etc');
  }
}

# *********************************************************************

# PROZEDUR zum Erstellen des fuer die Zonenkonfigurationsdateien benoetigten
# Verzeichnisses
# Parameter: $_[0] = Verzeichnis, in dem das neue erstellt werden soll
#            $_[1] = Name des zu erstellenden Verzeichnis(-pfades)

sub make_dir_ns
{
  # lokale Parameterkopien
  local($dir,$newdir) = @_;

  # Liste der einzelnen Knoten im Verzeichnispfad
  local(@fdirs) = split(/\//, $newdir);
```

```
  # sukzessiver Aufbau des Verzeichnispfades
  for(@fdirs)
  {
    $dir .= '/' . $_;
    &mk_dir($dir);
  }
}

# ***********************************************************************
1;
```

A.5 Modul res_conf.pl

```
# *******************************************************************
# * Integration des Managements des Domain-Name-Systems in die        *
# * vorhandene Managementumgebung des Leibniz-Rechenzentrums (LRZ)     *
# * (Diplomarbeit an der TU-Muenchen, SS 94 - WS 94/95)               *
# * ---------------------------------------------------------------- *
# * Autor: Heiko Hauck (hauck@informatik.tu-muenchen.de)              *
# * Datei: res_conf.pl (Oraperl Bibliothek fuer ora2dns)              *
# *******************************************************************

# PROZEDUR zur formatierten Ausgabe eines Nameserver-Eintrags
# Parameter: $_[0] = IP-Adresse des Nameservers
#            $_[1] = FQDN des Nameservers
# Pro Konfigurationsdatei max. 3 Nameserver (*globale* Variable $nsc)

sub print_nameserver
{
  # Noch ein Eintrag zulaessig?
  if($nsc < 3)
  {
    # formatierte Ausgabe und Zaehlererhoehung
    print "nameserver @_[0]", " " x (20 - length(@_[0])), "# @_[1]\n";
    $nsc++;
  }
}

# *******************************************************************

# PROZEDUR zur Bestimmung des "optimalen" Weges zwischen zwei Hosts
# mit jeweils einer oder mehreren IP-Adressen; dabei die die guenstigste
# Adresse des Zielhosts ausgewaehlt und aus der Zieladressenliste gestrichen

# Das hierzu verwendete Verfahren basiert auf einem Vergleich der
# Subnetzmasken; diese Prozedur kann evtl. durch eine bessere ersetzt
# werden (siehe Ausarbeitung)

# Parameter: $_[0] = Pointer auf Liste mit IP-Adressen des Quellhosts
#            $_[1] = Pointer auf Liste mit IP-Adressen des Zielhosts
# Rueckgabewert: s. oben

sub best_route
{
  # Lokale Parameterkopien und Variablen
  local(*sourcelist) = $_[0];
  local(*destlist) = $_[1];
  local($fits) = -1;              # Anzahl der bisher max. identischen Bytes
  local($offset);                 # Nummer der zugehoerigen Ziel-IP-Adresse

  # Schleife ueber alle Quellhost-IP-Adressen
  SOURCE:
```

```
for(@sourcelist)
{
  # Extrahieren der Quellhost-IP-Adresse
  local($a,$b,$c,$d) =
  /\x20{0,2}(\d{1,3})\.\x20{0,2}(\d{1,3})\.\x20{0,2}(\d{1,3})\.\x20{0,2}(\d{1,3})/;

  # Nummer der jeweiligen Zielhost-IP-Adresse
  local($o)=0;

  # Schleife ueber alle Zielhost-IP-Adressen
  for(@destlist)
  {
    # Extrahieren der Quellhost-IP-Adresse
    # Programmabbruch im Falle eines syntaktischen Fehlers
    die "Illegal entry for ip_addr in table ${DB}ipaddr\n" unless
    /\x20{0,2}(\d{1,3})\.\x20{0,2}(\d{1,3})\.\x20{0,2}(\d{1,3})\.\x20{0,2}(\d{1,3})/;

    # Stufenweiser Byte-Vergleich zur Bestimmung des Uebereinstimmungsgrades
    # beider Adressen
    local($f);
    if($1 == $a)
    {
      if($2 == $b)
      {
        if($3 == $c)
        {
          if($4 == $d) { $f = 4; } # Identische Adressen
          else { $f = 3; }         # Bytes 1-3 gleich
        }
        else { $f = 2; }           # Bytes 1-2 gleich
      }
      else { $f = 1; }             # Byte 1 gleich
    }
    else { $f = 0; }               # Voellig verschiedene Adressen

    # Verbesserung gegenueber bisher?
    if($fits < $f)
    {
      # Neuer Uebereinstimmungsgrad und Nummer
      $fits = $f;
      $offset = $o;

      # Bei identischen Adressen keine Verbesserung mehr moeglich,
      # deshalb Schleifenabbruch
      last SOURCE if($fits == 4);
    }
    $o++;
  }
}

# Auswahl und Ausschneiden der besten IP-Adresse aus der Zielhostliste
```

```
  splice(@destlist,$offset,1);
}

# ********************************************************************

# PROZEDUR zur Erstellung saemtlicher Resolverkonfigurationsdateien
# Parameter: keine

sub res_conf
{
  # Auswahl aller Hosts mit ihren Identifikatoren, Namen, Subdomain-Namen &
  # Identifikatoren der zugehoerigen Zonen
  local(@hosts);
  &query(<<"  END_SQL", *hosts);
    select id, name, subdomain_name, zone from ${DB}hosts
  END_SQL

  # Auswahl des Domain-Namens & Host-Identifikators, auf dem der
  # zugehoerige Primary Nameserver laeuft, fuer eine bestimmte Zone
  local($csr_1) = &query_open(<<"  END_SQL");
    select z.name, n.host from ${DB}nameservers n, zones z
    where n.id = z.prim_nameserver and z.id = :1
  END_SQL

  # Auswahl aller IP-Adressen eines bestimmten Hosts
  local($csr_2) = &query_open(<<"  END_SQL");
    select ip_addr from ${DB}ipaddr where host = :1
  END_SQL

  # Auswahl der Host-Identifikatoren von lokalen Secondary Nameservern,
  # die eine bestimmte lokale Zone spiegeln
  local($csr_3) = &query_open(<<"  END_SQL");
    select n.host
    from ${DB}nameservers n, ${DB}secondary_ll s
    where n.id = s.nameserver and s.zone = :1
  END_SQL

  # Auswahl der Hosts mit ihren Namen, Subdomain- & Domainnamen,
  # die eine bestimmte IP-Adresse besitzen
  local($csr_4) = &query_open(<<"  END_SQL");
    select h.name, h.subdomain_name, z.name
    from ${DB}hosts h, ${DB}ipaddr i, ${DB}zones z
    where h.id = i.host and h.zone=z.id and i.ip_addr = :1
  END_SQL

  # ------------------------------------------------------------------

  # Schleife ueber saemtliche Hosts
  for(@hosts)
  {
    # Anzahl der Nameserver pro Konfig.Datei: *globale* Variable $nsc
```

```
$nsc = 0;

# Parameter fuer den aktuellen Host
local($id, $name, $subdomain, $zone) = split(/$;/);
# $id, $name koennen nicht NULL sein
# $zone darf nicht NULL sein (&notnull ...)
# $subdomain_name darf NULL sein
&notnull($zone, 'hosts', 'zone');

# -------------------------------------------------------------------

# Auswahl des Primary Nameservers der Zone des aktuellen Hosts
local(@dn_pnsh);
&query_bind($csr_1, *dn_pnsh, $zone);
# Ergebnis = 1 Tupel
local($domain, $pns_host) = split(/$;/,@dn_pnsh[0]);
# $domain, $pns_host koennen nicht NULL sein

# Vollstaendiger Domainname
if((!&testnull($subdomain)) && (length($subdomain) > 0))
{
  $domain = $subdomain . '.' . $domain;
}

# Ausgabedatei oeffnen und Standardausgabe dorthin umlenken
local($filename) = $def{'ORA2DNS.FTP_DIR'} . '/' .
  $name . '.' . $domain . '/etc/resolv.conf';
open(NF,">$filename") || die "Can't open $filename: $!\n";
local($OLD_HANDLE)=select(NF);

# IP-Adresse des aktuellen Hosts bestimmen
local(@ipa_host);
&query_bind($csr_2, *ipa_host, $id);
# $ipa_host[..] kann nicht NULL sein

# Auswahl der "ersten" IP-Adresse
local($ipa) = $ipa_host[0];

# Eliminieren von darin enthaltenen Leerzeichen
$ipa =~ s/\x20//g;

# -------------------------------------------------------------------

# Ausgabe des Dateikopfes
print "# /etc/resolv.conf for $name.$domain [$ipa]\n";
print "# created by ora2dns (08/94 by Heiko Hauck; Diplomarbeit TUM)\n\n";
print "# User: ", `whoami`;
print "# Date: ", `date`, "\n";;
print "domain     $domain\n\n";

# -------------------------------------------------------------------
```

```
# Laeuft der Primary Nameserver auf dem aktuellen Host, so wird dieser
# sofort eingetragen, ...
local(@ipa_nsh);
if($psn_host == $id)
{
  &print_nameserver('127.0.0.1', 'localhost');
}

else
# ..., sonst wird seine "beste" IP-Adresse in eine Liste (@ipa_nsh)
# eingetragen
{
  local(@ipa);
  &query_bind($csr_2, *ipa, $pns_host);
  # $ipa[..] kann nicht NULL sein
  @ipa_nsh  = (&best_route(*ipa_host, *ipa));
}

# ------------------------------------------------------------------

# Auswahl der Secondary Nameserver der Zone des aktuellen Hosts
local(@sns_hosts);
&query_bind($csr_3, *sns_hosts, $zone);
# $sns_hosts[..] kann nicht NULL sein

# Schleife ueber alle Secondary Nameserver
for(@sns_hosts)
{
  # Laeuft der Nameserver auf dem aktuellen Host, so wird dieser
  # sofort eingetragen, ...
  if($_ == $id)
  {
    if ($nsc == 0)
    {
      &print_nameserver('127.0.0.1', 'localhost');
    }
  }

  else
  # ..., sonst wird seine "beste" IP-Adresse in eine Liste (@ipa_nsh)
  # eingetragen
  {
    local(@ipa);
    &query_bind($csr_2, *ipa, $_);
    # $ipa[..] kann nicht NULL sein
    push(@ipa_nsh,&best_route(*ipa_host, *ipa));
  }
}

# ------------------------------------------------------------------
```

```
    # Eintragen der (restlichen) Nameserver in die Datei

    # Schleife ueber die jeweils "besten" IP-Adressen der Nameserverhosts
    while(@ipa_nsh)
    {
      # Auswahl und Ausaschneiden der "besten" Adresse der Liste
      local($ipa) = &best_route(*ipa_host,*ipa_nsh);

      # Auswahl des zugehoerigen Hosts (Name, Subdomain- & Domainname)
      local(@hn_sn_dn);
      if(&query_bind($csr_4, *hn_sn_dn, $ipa) !=1 )
      {
        # Programmabbruch, falls eine IP-Adresse mehreren Hosts zugeordnet
        # werden kann
        die "Multiple hosts for one IP address";
      }
      local($host, $subdomain, $domain) = split(/$;/,$hn_sn_dn[0]);
      # $host, $domain koennen nicht NULL sein
      # $subdomain darf NULL sein

      # Vollstaendiger Domainname
      if((!&testnull($subdomain)) && (length($subdomain) > 0))
      {
        $domain = $subdomain . '.' . $domain;
      }

      # Eliminieren von darin enthaltenen Leerzeichen
      $ipa =~ s/\x20//g;

      # Eintrag
      &print_nameserver($ipa, $host . '.' . $domain);
    }

    # Standardausgabe wieder herstellen und Datei schliessen
    select($OLD_HANDLE);
    close(NF);
  }

  # Schliessen der oben fuer die Datenbankabfrage geoeffneten Cursor
  &query_close($csr_1);
  &query_close($csr_2);
  &query_close($csr_3);
  &query_close($csr_4);
}

# *************************************************************************
1;
```

A.6 Modul named_boot.pl

```
# ******************************************************************
# * Integration des Managements des Domain-Name-Systems in die        *
# * vorhandene Managementumgebung des Leibniz-Rechenzentrums (LRZ)     *
# * (Diplomarbeit an der TU-Muenchen, SS 94 - WS 94/95)               *
# * ---------------------------------------------------------------- *
# * Autor: Heiko Hauck (hauck@informatik.tu-muenchen.de)              *
# * Datei: named_boot.pl (Oraperl Bibliothek fuer ora2dns)            *
# ******************************************************************

# PROZEDUR zur Generierung einer Nameserver-Bootdatei
# Parameter: $_[0] = IP-Adresse des Nameserverhosts (ohne Leerzeichen)
#            $_[1] = FQDN des Nameserverhosts
#            $_[2] = ID des Nameservers
#            $_[3] = Name der Root-Domain-Datei (Cache Service)
#            $_[4] = Pointer auf Liste mit IP-Adressen des Nameserverhosts
#            $_[5] = Verzeichnis, in dem die Zonendateien abgelegt werden
#            $_[6] = Name der Localhost-Zonendatei (127.1 -> localhost)
#            $_[7] = Name der Loopback-Zonendatei (localhost -> 127.1)
# Die Ausgabe erfolgt zur Standard-Ausgabe.

sub named_boot
{
  # Auswahl der Domainnamen und Sourcefile-Namen,
  # deren Zonen einen best. Primary Nameserver haben
  local($csr_prim) = &query_open(<<"  END_SQL");
    select name, sourcefile from ${DB}zones where prim_nameserver = :1
  END_SQL

  # Auswahl des Backupfile-Namens, Domainnamens & lokalen Primary Nameservers
  # fuer Zonen, die einen bestimmten lokalen Secondary Nameserver haben
  local($csr_sec_ll) = &query_open(<<"  END_SQL");
    select s.backupfile, z.name, z.prim_nameserver
    from ${DB}secondary_ll s, ${DB}zones z
    where z.id = s.zone and s.nameserver = :1
  END_SQL

  # Auswahl der IP-Adressen eines bestimmten Primary Nameservers
  local($csr_ipaddr) = &query_open(<<"  END_SQL");
    select i.ip_addr
    from ${DB}ipaddr i, ${DB}nameservers n
    where i.host = n.host and n.id = :1
  END_SQL

  # Auswahl fremder Domainnamen, deren Prim.Nameserver-IP-Adressen und
  # Backupfile-Namen, die einen bestimmten lokalen Secondary Nameserver haben
  local($csr_sec_lf) = &query_open(<<"  END_SQL");
    select zone_name, ip_addr, backupfile
    from ${DB}secondary_lf
    where nameserver = :1
```

```
END_SQL

# Auswahl der Forwarder-IP-Adressen eines bestimmten Slave Nameservers
local($csr_forward) = &query_open(<<"  END_SQL");
  select ip_addr from ${DB}forwarders where nameserver = :1
END_SQL

# Test, ob ein bestimmter Nameserver Slave Nameserver ist
local($csr_slave) = &query_open(<<"  END_SQL");
  select count(*) from ${DB}nameservers_slave where nameserver = :1
END_SQL

# -------------------------------------------------------------------

# Ausgabe des Dateikopfes
print "; /etc/named.boot for $_[1] [$_[0]]\n";
print "; created by ora2dns (08/94 by Heiko Hauck; Diplomarbeit TUM)\n";
print ";\n";
print "; User: ", `whoami`;
print "; Date: ", `date`;
print ";\n";
print "; type\t\tdomain\t\t\tsource file or host\tbackup file\n";
print ";\n";

# Ausgabe des Verzeichnisses, in dem die Zonendateien abgelegt werden
print "directory\t/$_[5]\n";

# -------------------------------------------------------------------

# Zonen-Eintraege fuer den Primary Service des Nameservers

# Hinweis als Kommentarzeile
print "; primary nameserver:\n";

# Endung der Zonendateinamen
local($suffix) = $def{'ZONES.SOURCEFILE_SUFFIX'};

# Zonen bestimmen
local(@prim);
&query_bind($csr_prim, *prim, $_[2]);

# Schleife ueber alle betroffenen Zonen
for(@prim)
{
  # Parameter der aktuellen Zone
  local($domainname, $sourcefile) = split(/$;/);
  # $domainname kann nicht NULL sein
  # $sourcefile darf NULL sein

  # Wird der Name der Zonendatei nicht explizit angegeben, ...
  if(&testnull($sourcefile))
```

```
  {
    # ..., so wird der Domainname + Suffix verwendet
    $sourcefile = $domainname . $suffix;
  }

  # Eintrag in die Bootdatei
  print "primary\t\t$domainname\t$sourcefile\n";
}

# Standardeintraege in der Bootdatei, da jeder Nameserver Primary Nameserver
# fuer das Loopback-Interface (forward & reverse) ist
print "primary\t\t0.0.127.in-addr.arpa\t$_[6]\n";
print "primary\t\tlocalhost\t\t$_[7]\n";

# ----------------------------------------------------------------------

# Zonen-Eintraege fuer den Secondary Service des Nameservers

# Hinweis als Kommentarzeile
print "; secondary nameserver:\n";

# Endung der Zonendateinamen (lokale Zonen)
$suffix = $def{'SECONDARY_LL.BACKUPFILE_SUFFIX'};

# Lokale Zonen bestimmen
local(@sec_ll);
&query_bind($csr_sec_ll, *sec_ll, $_[2]);

# Schleife ueber alle betroffenen Zonen
for(@sec_ll)
{
  # Parameter der aktuellen Zone
  local($backupfile, $domainname, $prim_id) = split(/$;/);
  # $backupfile darf NULL sein
  # $domainname, $prim_id koennen nicht NULL sein

  # IP-Adressen des zugehoerigen Primary Nameservers bestimmen ...
  local(@ipaddr);
  &query_bind($csr_ipaddr, *ipaddr, $prim_id);
  # $ipaddr[..] kann nicht NULL sein

  # ... und die "beste" davon auswaehlen
  local($ipa) = &best_route($_[4], *ipaddr);

  # Eliminieren von darin enthaltenen Leerzeichen
  $ipa =~ s/\x20//g;

  # Wird der Name der Zonendatei nicht explizit angegeben, ...
  if(&testnull($backupfile))
  {
    # ..., so wird der Domainname + Suffix verwendet
```

```
    $backupfile = $domainname . $suffix;
  }

  # Eintrag in die Bootdatei
  print "secondary\t$domainname\t$ipa\t$backupfile\n";
}

# ---------------------------------------------------------------------

# Endung der Zonendateinamen (fremde Zonen)
$suffix = $def{'SECONDARY_LF.BACKUPFILE_SUFFIX'};

# Fremde Zonen bestimmen
local(@sec_lf);
&query_bind($csr_sec_lf, *sec_lf, $_[2]);

# Schleife ueber alle betroffenen Zonen
for(@sec_lf)
{
  # Parameter der aktuellen Zone
  local($domainname, $ipa, $backupfile) = split(/$;/);
  # $domainname, $ipa koennen nicht NULL sein
  # $backupfile darf NULL sein

  # Eliminieren von in der IP-Adresse enthaltenen Leerzeichen
  $ipa =~ s/\x20//g;

  # Wird der Name der Zonendatei nicht explizit angegeben, ...
  if(&testnull($backupfile))
  {
    # ..., so wird der Domainname + Suffix verwendet
    $backupfile = $domainname . $suffix;
  }

  # Eintrag in die Bootdatei
  print "secondary\t$domainname\t$ipa\t$backupfile\n";
}

# ---------------------------------------------------------------------

# Forwarding-Eintraege, Nameserver als Forwarder Nameserver arbeitet

# Forwarder-IP-Adressen bestimmen
local(@forward);
&query_bind($csr_forward, *forward, $_[2]);

# Sind welche vorhanden?
if(@forward > 0)
{
  # Hinweis als Kommentarzeile
  print "; forwarding nameservers:\n";
```

```
    # Eliminieren von in den IP-Adressen enthaltenen Leerzeichen
    for(@forward)
    {
      s/\x20//g;
    }

    # Eintrag in die Bootdatei
    print "forwarders\t", join(' ', @forward), "\n";

    # Laeuft der Nameserver als Slave, ...
    local(@slave);
    &query_bind($csr_slave, *slave, $_[2]);
    if($slave[0] > 0)
    {
      # ..., so gibt es einen entsprechenden Eintrag in die Bootdatei
      print "slave\n";
    }
  }

  # ---------------------------------------------------------------------

  # Standardeintrag in der Bootdatei, da jeder Nameserver Cache Nameserver
  # fuer die Root-Domain ist
  print "; cache nameserver:\n";
  print "cache\t\t.\t\t\t$_[3]\n";

  # Schliessen der oben fuer die Datenbankabfrage geoeffneten Cursor
  &query_close($csr_prim);
  &query_close($csr_sec_ll);
  &query_close($csr_ipaddr);
  &query_close($csr_sec_lf);
  &query_close($csr_forward);
  &query_close($csr_slave);
}

# *********************************************************************
1;
```

A.7 Modul named_forward.pl

```
# *******************************************************************
# * Integration des Managements des Domain-Name-Systems in die       *
# * vorhandene Managementumgebung des Leibniz-Rechenzentrums (LRZ)   *
# * (Diplomarbeit an der TU-Muenchen, SS 94 - WS 94/95)              *
# * --------------------------------------------------------------- *
# * Autor: Heiko Hauck (hauck@informatik.tu-muenchen.de)             *
# * Datei: named_forward.pl (Oraperl Bibliothek fuer ora2dns)        *
# *******************************************************************

# PROZEDUR zur Generierung einer Zonen-Datei fuer das Forward Mapping
# Parameter: $_[0]  = Name der Datei
#            $_[1]  = FQDN des Nameserverhosts
#            $_[2]  = IP-Adresse des Nameserverhosts
#            $_[3]  = Klasse des SOA-Records
#            $_[4]  = E-Mail-Adresse (SOA-Responsible)
#            $_[5]  = Seriennummer im SOA-Record
#            $_[6]  = Refresh-Intervall im SOA-Record
#            $_[7]  = Retry-Intervall im SOA-Record
#            $_[8]  = Expire-Intervall im SOA-Record
#            $_[9]  = Minimum-Intervall im SOA-Record
#            $_[10] = Domainname der Zone
#            $_[11] = Identifikator der Zone
#            $_[12] = TTL fuer Glue-Records
#            $_[13] = Klasse der Glue-Records
# Die Ausgabe erfolgt zur Standard-Ausgabe.

sub named_forward
{
  # Ausgabe des Dateikopfes
  print "; $_[0] for $_[1] [$_[2]]\n";
  print "; created by ora2dns (08/94 by Heiko Hauck; Diplomarbeit TUM)\n";
  print ";\n";
  print "; User: ", `whoami`;
  print "; Date: ", `date`;
  print ";\n";

  # Ausgabe der betroffenen Zone
  print "\$ORIGIN\t\t$_[10].\n";

  # Ausgabe des SOA-Records fuer diese Zone
  print "\@\t\t$_[3]\tSOA\t$_[1]. $_[4]. (\n";
  print "\t\t\t\t\t$_[5]   ; serial\n";
  print "\t\t\t\t\t$_[6]   ; refresh\n";
  print "\t\t\t\t\t$_[7]   ; retry\n";
  print "\t\t\t\t\t$_[8]   ; expire\n";
  print "\t\t\t\t\t$_[9] ) ; minimum\n";

  # ------------------------------------------------------------------
```

```
# Eintrag fuer den Primary Nameserver der Zone
print "\t\tIN\tNS\t$_[1].\n";

# ------------------------------------------------------------------

# Anlegen einer Liste fuer potentielle Glue-Records
# folgende Attribute werden dabei als Listenelement gepackt:
# FQDN, TTL, Class, IP-Adresse (Reihenfolge wie angegeben)

# Primary Nameserver Host ist Kandidat fuer Glue-Record
local(@glue) = (join($;, ($_[1], $_[12], $_[13], $_[2])));

# ------------------------------------------------------------------

# Lokale Secondary Nameserver fuer die Zone
local(@servers);
&query(<<"  END_SQL", *servers);
  select h.id, h.name, h.subdomain_name, h.record_ttl, h.record_class, z.name
  from ${DB}zones z, ${DB}hosts h, ${DB}nameservers n, ${DB}secondary_ll s
  where z.id = h.zone and h.id = n.host and
        n.id = s.nameserver and s.zone = $_[11]
END_SQL

# Auswahl aller IP-Adressen eines bestimmten Hosts
local($csr_ipaddr) = &query_open(<<"  END_SQL");
  select ip_addr from ${DB}ipaddr where host = :1
END_SQL

# Schleife ueber alle betroffenen Nameserver
for(@servers)
{
  # Parameter des aktuellen Nameservers
  local($id, $hostname, $subdomainname, $record_ttl, $record_class,
   $domainname) = split(/$;/);
  # $id, $hostname, $domainname koennen nicht NULL sein
  # $subdomainname, $record_ttl, $record_class duerfen NULL sein

  # Aufbau des Fully Qualified Domain Name fuer den Nameserver-Host
  local($fqdn);
  if((!&testnull($subdomainname)) && (length($subdomainname) > 0))
  {
    $fqdn = $hostname . '.' . $subdomainname . '.' . $domainname;
  }
  else
  {
    $fqdn = $hostname . '.' . $domainname;
  }

  # Eintrag fuer den Nameserver
  print "\t\tIN\tNS\t$fqdn.\n";
```

```
  # ------------------------------------------------------------------

  # Secondary Nameserver Host ist Kandidat fuer Glue-Record

  # Record-TTL und -Class
  if(&testnull($record_ttl))
    { $record_ttl = $def{'HOSTS.RECORD_TTL'}; }
  if(&testnull($record_class))
    { $record_class = $def{'HOSTS.RECORD_CLASS'}; }

  # IP-Adresse des Nameserver-Hosts
  local(@ipaddr);
  &query_bind($csr_ipaddr, *ipaddr, $id);
  # $ipaddr[..] kann nicht NULL sein
  local($ipa) = $ipaddr[0];

  # Loeschen von Leerzeichen in der Adresse (DB-Konvention)
  $ipa =~ s/\x20//g;

  # Eintrag in die Liste
  push(@glue, join($;, ($fqdn, $record_ttl, $record_class, $ipa)));
}

# --------------------------------------------------------------------

# Achtung: 'Perl' ueberschreibt aktuelle Parameter ohne entspr. Zuweisung!
# Deshalb: Kopie in lokale Variable
local($current) = $_[10];

# --------------------------------------------------------------------

# Fremde Secondary Nameserver fuer die Zone

# Auswahl der FQDNs und IP-Adressen der betroffenen Nameserver
&query(<<"  END_SQL", *servers);
  select ip_name, ip_addr from ${DB}secondary_fl where zone = $_[11]
END_SQL

# Schleife ueber alle betroffenen Nameserver
for(@servers)
{
  # Parameter des aktuellen Nameservers
  local($ip_name, $ip_addr) = split(/$;/);
  # $ip_name kann nicht NULL sein
  # $ip_addr darf u.U. NULL sein

  # Eintrag fuer den Nameserver
  print "\t\tIN\tNS\t$ip_name.\n";

  # ------------------------------------------------------------------
```

```
  # Secondary Nameserver Host ist Kandidat fuer Glue-Record,
  # falls sein FQDN in der zur aktuellen Zone gehoerigen Domain liegt
  if($ip_name =~ /\.$current$/i)
  {
    # In diesem Fall muss seine IP-Adresse angegeben werden,
    # sonst Programmabbruch
    &notnull($ip_addr, 'ip_addr', 'secondary_fl');

    # Loeschen von Leerzeichen in der Adresse (DB-Konvention)
    $ip_addr =~ s/\x20//g;

    # Eintrag in die Liste
    push(@glue, join($;, ($ip_name, '', 'IN', $ip_addr)));
  }
}

# --------------------------------------------------------------------

# Nameserver Eintraege fuer delegierte (lokale) Subdomains

# Auswahl aller Zonen, die die aktuelle Zone als Vaterzone haben
local(@domains);
&query(<<"  END_SQL", *domains);
  select id, name, prim_nameserver from ${DB}zones where parent = $_[11]
END_SQL

# Auswahl von ID, Namen, Subdomainnamen, RR-Parameter und Domainnamen
# eines Hosts, auf dem ein bestimmter Primary Nameserver laeuft
local($csr_prim) = &query_open(<<"  END_SQL");
  select h.id, h.name, h.subdomain_name, h.record_ttl, h.record_class, z.name
  from ${DB}zones z, ${DB}hosts h, ${DB}nameservers n
  where z.id = h.zone and h.id = n.host and n.id = :1
END_SQL

# Auswahl von ID, Namen, Subdomainnamen, RR-Parameter und Domainnamen
# von Hosts, auf denen fuer eine bestimmte Zone Secondary Nameserver laufen
local($csr_serv) = &query_open(<<"  END_SQL");
  select h.id, h.name, h.subdomain_name, h.record_ttl, h.record_class, z.name
  from ${DB}zones z, ${DB}hosts h, ${DB}nameservers n, ${DB}secondary_ll s
  where z.id = h.zone and h.id = n.host and
        n.id = s.nameserver and s.zone = :1
END_SQL

# Auswahl von fremden Secondary Nameservern fuer eine bestimmte Zone
local($csr_secf) = &query_open (<<"  END_SQL");
  select ip_name, ip_addr from ${DB}secondary_fl where zone = :1
END_SQL

# --------------------------------------------------------------------

# Schleife ueber alle delegierten lokalen Subdomains
```

```
for(@domains)
{
  # Parameter der akuellen Subdomain
  local($id, $name, $prim_nameserver) = split(/$;/);
  # $id, $name & $prim_nameserver koennen nicht NULL sein

  # Subdomain-Name (innerhalb) der Domain bestimmen
  local($subdomain) = $name;
  $subdomain =~ s/\.$current$//i;

  # Primary Nameserver fuer Subdomain
  local(@prim);
  &query_bind($csr_prim, *prim, $prim_nameserver);
  local($hostid, $hostname, $subdomainname, $record_ttl,
        $record_class, $domainname) = split(/$;/, $prim[0]);
  # $hostid, $hostname, $domainname koennen nicht NULL sein
  # $subdomainname, $record_ttl, $record_class duerfen NULL sein

  # Aufbau des Fully Qualified Domain Name fuer den Primary Nameserver
  local($fqdn);
  if((!&testnull($subdomainname)) && (length($subdomainname) > 0))
  {
    $fqdn = $hostname . '.' . $subdomainname . '.' . $domainname;
  }
  else
  {
    $fqdn = $hostname . '.' . $domainname;
  }

  # Ausgabe des Subdomain-Namens mit Primary Nameserver
  print "$subdomain\t\tIN\tNS\t$fqdn.\n";

  # ------------------------------------------------------------------

  # Primary Nameserver Host ist Kandidat fuer Glue-Record

  # Record-TTL und -Class
  if(&testnull($record_ttl))
    { $record_ttl = $def{'HOSTS.RECORD_TTL'}; }
  if(&testnull($record_class))
    { $record_class = $def{'HOSTS.RECORD_CLASS'}; }

  # IP-Adresse des Nameserver-Hosts
  local(@ipaddr);
  &query_bind($csr_ipaddr, *ipaddr, $hostid);
  # $ipaddr[..] kann nicht NULL sein
  local($ipa) = $ipaddr[0];

  # Loeschen von Leerzeichen in der Adresse (DB-Konvention)
  $ipa =~ s/\x20//g;
```

```
# Eintrag in die Liste
push(@glue, join($;, ($fqdn, $record_ttl, $record_class, $ipa)));

# ------------------------------------------------------------------

# Lokale Secondary Nameserver fuer die Subdomain
local(@serv);
&query_bind($csr_serv, *serv, $id);

# Schleife ueber alle lokalen Secondary Nameserver
for(@serv)
{
  # Host-Parameter des aktuellen Nameservers
  local($id, $hostname, $subdomainname, $record_ttl, $record_class,
   $domainname) = split(/$;/);
  # $id, $hostname, $domainname koennen nicht NULL sein
  # $subdomainname, $record_ttl, $record_class duerfen NULL sein

  # Aufbau des Fully Qualified Domain Name fuer den Secondary Nameserver
  local($fqdn);
  if((!&testnull($subdomainname)) && (length($subdomainname) > 0))
  {
    $fqdn = $hostname . '.' . $subdomainname . '.' . $domainname;
  }
  else
  {
    $fqdn = $hostname . '.' . $domainname;
  }

  # Ausgabe des Secondary Nameservers
  print "\t\tIN\tNS\t$fqdn.\n";

  # ----------------------------------------------------------------

  # Secondary Nameserver Host ist Kandidat fuer Glue-Record

  # Record-TTL und -Class
  if(&testnull($record_ttl))
    { $record_ttl = $def{'HOSTS.RECORD_TTL'}; }
  if(&testnull($record_class))
    { $record_class = $def{'HOSTS.RECORD_CLASS'}; }

  # IP-Adresse des Nameserver-Hosts
  local(@ipaddr);
  &query_bind($csr_ipaddr, *ipaddr, $id);
  # $ipaddr[..] kann nicht NULL sein
  local($ipa) = $ipaddr[0];

  # Loeschen von Leerzeichen in der Adresse (DB-Konvention)
  $ipa =~ s/\x20//g;
```

```
    # Eintrag in die Liste
    push(@glue, join($;, ($fqdn, $record_ttl, $record_class, $ipa)));
  }

  # ------------------------------------------------------------------

  # Fremde Secondary Nameserver fuer die Subdomain
  local(@secf);
  &query_bind($csr_secf, *secf, $id);

  # Schleife ueber alle fremden Secondary Nameserver
  for(@secf)
  {
    # Parameter des aktuellen Nameservers
    local($ip_name, $ip_addr) = split(/$;/);
    # $ip_name kann nicht NULL sein
    # $ip_addr darf u.U. NULL sein

    # Eintrag fuer den Nameserver
    print "\t\tIN\tNS\t$ip_name.\n";

    # ----------------------------------------------------------------

    # Secondary Nameserver Host ist Kandidat fuer Glue-Record,
    # falls sein FQDN in der zur aktuellen Zone gehoerigen Domain liegt
    if($ip_name =~ /\.$current$/i)
    {
      # In diesem Fall muss seine IP-Adresse angegeben werden,
      # sonst Programmabbruch
      &notnull($ip_addr, 'ip_addr', 'secondary_fl');

      # Loeschen von Leerzeichen in der Adresse (DB-Konvention)
      $ip_addr =~ s/\x20//g;

      # Eintrag in die Liste
      push(@glue, join($;, ($ip_name, '', 'IN', $ip_addr)));
    }
  }
}

# Schliessen der fuer delegierte (lokale) Subdomains geoeffneten DB-Cursor
&query_close($csr_prim);
&query_close($csr_serv);
&query_close($csr_secf);

# -------------------------------------------------------------------

# Nameserver Eintraege fuer delegierte (fremde) Subdomains

# Auswahl der fremden Zonen (mit Primary Nameserver), die die aktuelle Zone
# als Vaterzone haben
```

```
local(@del_zones);
&query(<<"  END_SQL", *del_zones);
  select id, name, pns_ip_name, pns_ip_addr
  from ${DB}del_zones where parent = $_[11]
END_SQL

# Auswahl der (fremden) Secondary Nameserver einer bestimmten delegierten
# fremden Zone
local($csr_dzs) = &query_open (<<"  END_SQL");
  select sns_ip_name, sns_ip_addr from ${DB}del_zones_sec where del_zone = :1
END_SQL

# Auswahl der lokalen Secondary Nameserver einer bestimmten fremden Zone
local($csr_seclf) = &query_open (<<"  END_SQL");
  select h.name, h.subdomain_name, z.name
  from ${DB}zones z, ${DB}hosts h, ${DB}nameservers n, ${DB}secondary_lf s
  where z.id = h.zone and h.id = n.host and
        n.id = s.nameserver and s.zone_name = :1
END_SQL

# Schleife ueber alle delegierten fremden Subdomains
for(@del_zones)
{
  # Parameter der aktuellen Subdomain
  local($id, $name, $pns_ip_name, $pns_ip_addr) = split(/$;/);
  # $id, $name & $pns_ip_name koennen nicht NULL sein
  # $pns_ip_addr darf u.U. NULL sein
  local($subdomain) = $name;

  # Subdomain-Name (innerhalb) der Domain bestimmen
  $subdomain =~ s/\.$current$//i;

  # Ausgabe des Subdomain-Namens mit Primary Nameserver
  print "$subdomain\tIN\tNS\t$pns_ip_name.\n";

  # ------------------------------------------------------------------

  # Primary Nameserver Host ist Kandidat fuer Glue-Record,
  # falls sein FQDN in der zur aktuellen Zone gehoerigen Domain liegt
  if($pns_ip_name =~ /\.$current$/i)
  {
    # In diesem Fall muss seine IP-Adresse angegeben werden,
    # sonst Programmabbruch
    &notnull($pns_ip_addr, 'pns_ip_addr', 'del_zones');

    # Loeschen von Leerzeichen in der Adresse (DB-Konvention)
    $pns_ip_addr =~ s/\x20//g;

    # Eintrag in die Liste
    push(@glue, join($;, ($pns_ip_name, '', 'IN', $pns_ip_addr)));
  }
```

```
# ------------------------------------------------------------------

# Fremde Secondary Nameserver
local(@dzs);

# Schleife ueber alle fremden Secondary Nameserver
&query_bind($csr_dzs, *dzs, $id);
for(@dzs)
{
  # Parameter des aktuellen Nameservers
  local($sns_ip_name, $sns_ip_addr) = split(/$;/);
  # $sns_ip_name kann nicht NULL sein
  # $sns_ip_addr darf u.U. NULL sein

  # Ausgabe des Secondary Nameservers
  print "\t\tIN\tNS\t$sns_ip_name.\n";

  # ----------------------------------------------------------------

  # Secondary Nameserver Host ist Kandidat fuer Glue-Record,
  # falls sein FQDN in der zur aktuellen Zone gehoerigen Domain liegt
  if($sns_ip_name =~ /\.$current$/i)
  {
    # In diesem Fall muss seine IP-Adresse angegeben werden,
    # sonst Programmabbruch
    &notnull($sns_ip_addr, 'sns_ip_addr', 'del_zones');

    # Loeschen von Leerzeichen in der Adresse (DB-Konvention)
    $sns_ip_addr =~ s/\x20//g;

    # Eintrag in die Liste
    push(@glue, join($;, ($sns_ip_name, '', 'IN', $sns_ip_addr)));
  }
}

# ------------------------------------------------------------------

# Lokale Secondary Nameserver
local(@seclf);
&query_bind($csr_seclf, *seclf, $name);

# Schleife ueber alle lokalen Secondary Nameserver
for(@seclf)
{
  # Host-Parameter des aktuellen Nameservers
  local($hostname, $subdomainname, $domainname) = split(/$;/);
  # $hostname & $domainname koennen nicht NULL sein
  # $subdomainname darf NULL sein

  # Aufbau des Fully Qualified Domain Name fuer den Secondary Nameserver
```

```
    local($fqdn);
    if((!&testnull($subdomainname)) && (length($subdomainname) > 0))
    {
      $fqdn = $hostname . '.' . $subdomainname . '.' . $domainname;
    }
    else
    {
      $fqdn = $hostname . '.' . $domainname;
    }

    # Ausgabe des Secondary Nameservers
    print "\t\tIN\tNS\t$fqdn.\n";
  }
}

# Schliessen der fuer delegierte (fremde) Subdomains geoeffneten DB-Cursor
&query_close($csr_dzs);
&query_close($csr_seclf);

# ----------------------------------------------------------------------

# Sortieren der Glue-Record Kandidaten, Elimination doppelter Eintraege

local($last) = undef;

# sortierte minimale Liste
local(@sglue) = ();

# Schleife fuer alle Eintraege (innerhalb sind sie bereits sortiert)
for(sort @glue)
{
  # Falls der aktuelle Eintrag schon eingetragen wurde, wird
  # er nicht nochmal eingetragen
  if($_ eq $last) { next; }

  # Eintrag in die neue Liste
  push(@sglue, $_);

  # Merken des aktuellen Eintrags fuer den naechsten Schleifendurchlauf
  $last = $_;
}

# ----------------------------------------------------------------------

# Mail Exchanger (MX-Records) fuer die Zone

# Auswahl der Mail Exchanger Eintraege fuer die aktuelle Zone
local(@mailx);
&query(<<"  END_SQL", *mailx);
  select m.name, m.record_ttl, m.record_class, m.pref_value,
         h.name, h.subdomain_name, z.name
```

```
  from ${DB}zones z, ${DB}hosts h, ${DB}mailx m
  where z.id = h.zone and h.id = m.host and m.zone = $_[11]
END_SQL

# Schleife ueber alle betroffenen Mail Exchanger Eintraege
for(@mailx)
{
  # aktuelle Parameter der Eintraege
  local($name, $record_ttl, $record_class, $pref_value, $hostname,
        $subdomainname, $domainname) = split(/$;/);
  # $name, $pref_value, $hostname, $domainname koennen nicht NULL sein
  # $record_ttl, $record_class, $subdomainname duerfen NULL sein

  # Aufbau des Fully Qualified Domain Name des Ziel-Hosts
  local($fqdn);
  if((!&testnull($subdomainname)) && (length($subdomainname) > 0))
  {
    $fqdn = $hostname . '.' . $subdomainname . '.' . $domainname;
  }
  else
  {
    $fqdn = $hostname . '.' . $domainname;
  }

  # Falls kein bzw. ein leerer Name angegeben wird, dann
  # Eintrag fuer aktuelle Zone
  if((&testnull($name)) || ($name eq '')) { $name = '@'; }

  # Record-TTL und -Class
  if(&testnull($record_ttl))
    { $record_ttl = $def{'MAILX.RECORD_TTL'}; }
  if(&testnull($record_class))
    { $record_class = $def{'MAILX.RECORD_CLASS'}; }

  # Ausgabe des MX-Records
  print "$name\t$record_ttl\t$record_class\tMX\t$pref_value\t$fqdn.\n";
}

# ---------------------------------------------------------------------

# Hosts (A-, CNAME-, HINFO- & WKS-Records) fuer die Zone

# Auswahl der Hosts in der aktuellen Zone
local(@hosts);
&query(<<"  END_SQL", *hosts);
  select id, name, subdomain_name, record_ttl, record_class, cpu, os
  from ${DB}hosts
  where zone = $_[11]
END_SQL

# Auswahl der WKS eines bestimmten Hosts
```

```
local($csr_wks) = &query_open(<<"  END_SQL");
  select service from ${DB}wks where host = :1
END_SQL

# Auswahl der Alias-Namen eines bestimmten Hosts
local($csr_alias) = &query_open(<<"  END_SQL");
  select canonical_name from ${DB}aliases where host = :1
END_SQL

# ---------------------------------------------------------------------

# Schleife ueber alle Hosts in der aktuellen Zone
for(@hosts)
{
  # akutelle Host-Parameter
  local($id, $name, $subdomainname, $record_ttl, $record_class,
        $cpu, $os) = split(/$;/);
  # $id, $name koennen nicht NULL sein
  # $subdomainname, $record_ttl, $record_class, $cpu & $os duerfen NULL sein

  # Name innerhalb der aktuellen Zone
  local($hostname) = $name;
  if((!&testnull($subdomainname)) && (length($subdomainname) > 0))
  {
    $hostname .= '.' . $subdomainname;
  }

  # Record-TTL und -Class
  if(&testnull($record_ttl))
    { $record_ttl = $def{'HOSTS.RECORD_TTL'}; }
  if(&testnull($record_class))
    { $record_class = $def{'HOSTS.RECORD_CLASS'}; }

  # Falls fuer einen Host keine einzige IP-Adresse vorhanden ist, ...
  if(&query_bind($csr_ipaddr, *ipaddr, $id) == 0)
  {
    # ..., dann Programmabbruch
    die "No IP-adress available for $hostname.$current";
  }

  # Auswahl der ersten IP-Adresse
  # $ipaddr[..] kann nicht NULL sein
  local($ipa) = shift(@ipaddr);

  # Loeschen von Leerzeichen in der Adresse (DB-Konvention)
  $ipa =~ s/\x20//g;

  # Ausgabe des Hostnamens (innerhalb der aktuellen Zone) und der
  # (ersten) IP-Adresse als A-Record
  print "$hostname\t$record_ttl\t$record_class\tA\t$ipa\n";
```

```
# Schleife ueber restliche IP-Adressen des Hosts
for(@ipaddr)
{
  # Loeschen von Leerzeichen in der Adresse (DB-Konvention)
  s/\x20//g;

  # Ausgabe des entsprechenden A-Records
  print "\t$record_ttl\t$record_class\tA\t$_\n";
}

# Erscheint der Host in der Glue-Liste, muss er dort geloescht
# (fuer ungueltig erklaert) werden (sonst doppelter Eintrag)
for(@sglue)
{
  if(/^$hostname\.$current$;/i) { $_ = undef; last; }
}

# ------------------------------------------------------------------

# Existieren fuer den Host gueltige Angaben ueber CPU & Betriebssystem, ...
if((!&testnull($cpu)) && (length($cpu) > 0) &&
   (!&testnull($os)) && (length($os) > 0))
{
  # ..., so werden diese als HINFO-Record ausgegeben
  print "\t$record_ttl\t$record_class\tHINFO\t$cpu $os\n";
}

# ------------------------------------------------------------------

# WKS des Hosts
local(@wks);
&query_bind($csr_wks, *wks, $id);
# $wks[..] kann nicht NULL sein

# Schleife ueber alle Eintraege
for(@wks)
{
  # Ausgabe als WKS-Record
  print "\t$record_ttl\t$record_class\tWKS\t$_\n";
}

# ------------------------------------------------------------------

# Aliasnamen des Hosts
local(@alias);
&query_bind($csr_alias, *alias, $id);
# $alias[..] kann nicht NULL sein

# Schleife ueber alle Alias-Namen
for(@alias)
{
```

```
      # ACHTUNG: Alias-Namen werden nicht direkt der Zone zugeordnet,
      # sondern der jeweiligen Subdomain des Hosts
      if((!&testnull($subdomainname)) && (length($subdomainname) > 0))
      {
        $_ .= '.' . $subdomainname;
      }

      # Ausgabe als CNAME-Record
      print "$_\t$record_ttl\t$record_class\tCNAME\t$hostname\n";
    }
  }

  # Schliessen der fuer die Hostausgabe geoeffneten DB-Cursor
  &query_close($csr_wks);
  &query_close($csr_alias);
  &query_close($csr_ipaddr);

  # ---------------------------------------------------------------------

  # Erzeugen der Glue-Records

  # Schleife ueber alle Listeneintraege
  for(@sglue)
  {
    # Wurde der aktuelle Eintrag geloescht (fuer ungueltig) erklaert,
    # so darf er nicht weiter beruecksichtigt werden
    if(defined($_))
    {
      # aktuelle Glueparameter
      local($fqdn, $record_ttl, $record_class, $ipa) = split(/$;/);

      # Falls der Host in der zur aktuellen Zone gehoerenden Domain liegt, ...
      if($fqdn =~ s/\.$current$//i)
      {
        # ..., muss er als A-Record (Glue) eingetragen werden
        print "$fqdn\t$record_ttl\t$record_class\tA\t$ipa\n";
      }
    }
  }
}

# *********************************************************************
1;
```

A.8 Modul named_reverse.pl

```
# *****************************************************************
# * Integration des Managements des Domain-Name-Systems in die        *
# * vorhandene Managementumgebung des Leibniz-Rechenzentrums (LRZ)     *
# * (Diplomarbeit an der TU-Muenchen, SS 94 - WS 94/95)                *
# * ------------------------------------------------------------------ *
# * Autor: Heiko Hauck (hauck@informatik.tu-muenchen.de)               *
# * Datei: named_reverse.pl (Oraperl Bibliothek fuer ora2dns)          *
# *****************************************************************

# PROZEDUR zur Generierung einer Zonen-Datei fuer das Reverse Mapping
# Parameter: $_[0]  = Name der Datei
#            $_[1]  = FQDN des Nameserverhosts
#            $_[2]  = IP-Adresse des Nameserverhosts
#            $_[3]  = Klasse des SOA-Records
#            $_[4]  = E-Mail-Adresse (SOA-Responsible)
#            $_[5]  = Seriennummer im SOA-Record
#            $_[6]  = Refresh-Intervall im SOA-Record
#            $_[7]  = Retry-Intervall im SOA-Record
#            $_[8]  = Expire-Intervall im SOA-Record
#            $_[9]  = Minimum-Intervall im SOA-Record
#            $_[10] = Domainname der Zone
#            $_[11] = Identifikator der Zone
# Die Ausgabe erfolgt zur Standard-Ausgabe.

sub named_reverse
{
  # Ausgabe des Dateikopfes
  print "; $_[0] for $_[1] [$_[2]]\n";
  print "; created by ora2dns (08/94 by Heiko Hauck; Diplomarbeit TUM)\n";
  print ";\n";
  print "; User: ", `whoami`;
  print "; Date: ", `date`;
  print ";\n";

  # Ausgabe der betroffenen Zone
  print "\$ORIGIN\t\t$_[10].\n";

  # Ausgabe des SOA-Records fuer diese Zone
  print "\@\t\t$_[3]\tSOA\t$_[1]. $_[4]. (\n";
  print "\t\t\t\t\t$_[5]   ; serial\n";
  print "\t\t\t\t\t$_[6]   ; refresh\n";
  print "\t\t\t\t\t$_[7]   ; retry\n";
  print "\t\t\t\t\t$_[8]   ; expire\n";
  print "\t\t\t\t\t$_[9] ) ; minimum\n";

  # ------------------------------------------------------------------

  # Achtung: 'Perl' ueberschreibt aktuelle Parameter ohne entspr. Zuweisung!
  # Deshalb: Kopie in lokale Variable
```

```
local($current) = $_[10];

# ----------------------------------------------------------------------

# Aufbau der Maske zur Auswahl der IP-Adressen
# Dabei wird unterschieden, wieviele Bytes der IP-Adresse durch den
# Domainnamen schon vorgegeben sind

local($mask, $replace);
if($current =~ /^(\d{1,3})\.(\d{1,3})\.(\d{1,3})\.in-addr\.arpa$/i)
{
  # 3 Bytes vorgegeben
  # Beispiel: Fuer die Zone '15.187.129.in-addr.arpa' ist die zugehoerige
  # Maske '129.187. 15.'
  # Diese wird spaeter mit dem Begin der IP-Adressen verglichen
  $mask = ' ' x (3 - length($3)) . $3 . '.' .
          ' ' x (3 - length($2)) . $2 . '.' .
          ' ' x (3 - length($1)) . $1 . '.';

  # 1 Byte ist noch auszuwaehlen
  $replace = 1;
}
else
{
  if($current =~ /^(\d{1,3})\.(\d{1,3})\.in-addr\.arpa$/i)
  {
    # 2 Bytes vorgegeben
    $mask = ' ' x (3 - length($2)) . $2 . '.' .
            ' ' x (3 - length($1)) . $1 . '.';

    # 2 Bytes sind noch auszuwaehlen
    $replace = 2;
  }
  else
  {
    if($current =~ /^(\d{1,3})\.in-addr\.arpa$/i)
    {
      # 1 Byte vorgegeben
      $mask = ' ' x (3 - length($1)) . $1 . '.';

      # 3 Bytes sind noch auszuwaehlen
      $replace = 3;
    }
    else
    {
      if($current =~ /^\.in-addr\.arpa$/i)
      {
        # kein Byte vorgegeben
        $mask = '';

        # alle 4 Bytes sind noch auszuwaehlen
```

```
        $replace = 4;
      }
      else
      {
        # Programmabbruch, falls ein fehlerhafter Domainname angegeben wurde
        die "Illegal domain $current";
      }
    }
  }
}

# Liste von Subnetzmasken, deren IP-Adressen nicht zu dieser,
# sondern zu einer delegierten Zone gehoeren (siehe unten)
local(@submasks) = ();

# ---------------------------------------------------------------------

# Eintrag fuer den Primary Nameserver der Zone
print "\t\tIN\tNS\t$_[1].\n";

# ---------------------------------------------------------------------

# Lokale Secondary Nameserver fuer die Zone
local(@servers);
&query(<<"  END_SQL", *servers);
  select h.name, h.subdomain_name, z.name
  from ${DB}zones z, ${DB}hosts h, ${DB}nameservers n, ${DB}secondary_ll s
  where z.id = h.zone and h.id = n.host and
        n.id = s.nameserver and s.zone = $_[11]
END_SQL

# Schleife ueber alle betroffenen Nameserver
for(@servers)
{
  # Parameter des aktuellen Nameservers
  local($hostname, $subdomainname, $domainname) = split(/$;/);
  # $hostname, $domainname koennen nicht NULL sein
  # $subdomainname darf NULL sein

  # Aufbau des Fully Qualified Domain Name fuer den Nameserver-Host
  local($fqdn);
  if((!&testnull($subdomainname)) && (length($subdomainname) > 0))
  {
    $fqdn = $hostname . '.' . $subdomainname . '.' . $domainname;
  }
  else
  {
    $fqdn = $hostname . '.' . $domainname;
  }

  # Eintrag fuer den Nameserver
```

```
  print "\t\tIN\tNS\t$fqdn.\n";
}

# -------------------------------------------------------------------

# Fremde Secondary Nameserver fuer die Zone

# Auswahl der FQDNs der betroffenen Nameserver
&query(<<"  END_SQL", *servers);
  select ip_name from ${DB}secondary_fl where zone = $_[11]
END_SQL

# Schleife ueber alle betroffenen Nameserver
for(@servers)
{
  # $_ kann nicht NULL sein

  # Eintrag fuer den Nameserver
  print "\t\tIN\tNS\t$_.\n";
}

# -------------------------------------------------------------------

# Nameserver Eintraege fuer delegierte (lokale) Subdomains

# Auswahl aller Zonen, die die aktuelle Zone als Vaterzone haben
local(@domains);
&query(<<"  END_SQL", *domains);
  select id, name, prim_nameserver from ${DB}zones where parent = $_[11]
END_SQL

# Auswahl des Namens, Subdomain- und Domainnamens eines Hosts,
# auf dem ein bestimmter Primary Nameserver laeuft
local($csr_prim) = &query_open(<<"  END_SQL");
  select h.name, h.subdomain_name, z.name
  from ${DB}zones z, ${DB}hosts h, ${DB}nameservers n
  where z.id = h.zone and h.id = n.host and n.id = :1
END_SQL

# Auswahl des Namens, Subdomain- und Domainnamens von Hosts,
# auf denen fuer eine bestimmte Zone Secondary Nameserver laufen
local($csr_serv) = &query_open(<<"  END_SQL");
  select h.name, h.subdomain_name, z.name
  from ${DB}zones z, ${DB}hosts h, ${DB}nameservers n, ${DB}secondary_ll s
  where z.id = h.zone and h.id = n.host and
        n.id = s.nameserver and s.zone = :1
END_SQL

# Auswahl von fremden Secondary Nameservern fuer eine bestimmte Zone
local($csr_secf) = &query_open (<<"  END_SQL");
  select ip_name from ${DB}secondary_fl where zone = :1
```

```
END_SQL

# ------------------------------------------------------------------

# Schleife ueber alle delegierten lokalen Subdomains
for(@domains)
{
  # Parameter der akuellen Subdomain
  local($id, $name, $prim_nameserver) = split(/$;/);
  # $id, $name & $prim_nameserver koennen nicht NULL sein

  # Subdomain-Name (innerhalb) der Domain bestimmen
  local($subdomain) = $name;
  $subdomain =~ s/\.$current$//i;

  # Primary Nameserver fuer Subdomain
  local(@prim);
  &query_bind($csr_prim, *prim, $prim_nameserver);
  local($hostname, $subdomainname, $domainname) = split(/$;/, $prim[0]);
  # $hostname, $domainname koennen nicht NULL sein
  # $subdomainname darf NULL sein

  # Aufbau des Fully Qualified Domain Name fuer den Primary Nameserver
  local($fqdn);
  if((!&testnull($subdomainname)) && (length($subdomainname) > 0))
  {
    $fqdn = $hostname . '.' . $subdomainname . '.' . $domainname;
  }
  else
  {
    $fqdn = $hostname . '.' . $domainname;
  }

  # Ausgabe des Subdomain-Namens mit Primary Nameserver
  print "$subdomain\t\tIN\tNS\t$fqdn.\n";

  # ------------------------------------------------------------------

  # Aufbau der Subnetzmaske zum Ausschluss von IP-Adressen, die in
  # die lokale delegierte Subdomain fallen

  local($submask);

  if($subdomain =~ /^(\d{1,3})\.(\d{1,3})\.(\d{1,3})$/i)
  {
    # 3 Bytes vorgegeben

    # nur moeglich, wenn noch 4 Bytes zu besetzen sind,
    # sonst Programmabbruch
    if($replace < 4)
      { die "Illegal subdomain $subdomain of domain $current"; }
```

```
    $submask = ' ' x (3 - length($3)) . $3 . '.' .
               ' ' x (3 - length($2)) . $2 . '.' .
               ' ' x (3 - length($1)) . $1 . '.';
  }
  else
  {
    if($subdomain =~ /^(\d{1,3})\.(\d{1,3})$/i)
    {
      # 2 Bytes vorgegeben

      # nur moeglich, wenn noch mindestens 3 Bytes zu besetzen sind,
      # sonst Programmabbruch
      if($replace < 3)
        { die "Illegal subdomain $subdomain of domain $current"; }

      $submask = ' ' x (3 - length($2)) . $2 . '.' .
                 ' ' x (3 - length($1)) . $1 . '.';
    }
    else
    {
      if($subdomain =~ /^(\d{1,3})$/i)
      {
        # 1 Byte vorgegeben

        # nur moeglich, wenn noch mindestens 2 Bytes zu besetzen sind,
        # sonst Programmabbruch
        if($replace < 2)
          { die "Illegal subdomain $subdomain of domain $current"; }

        $submask = ' ' x (3 - length($1)) . $1 . '.';
      }
      else
      {
        # nicht zulaessige Subdomain, Programmabbruch
        die "Illegal subdomain $subdomain of domain $current";
      }
    }
  }

  # Eintrag der Subnetzmaske in die dafuer vorgesehene Liste
  push(@submasks,$submask);

  # ------------------------------------------------------------------

  # Lokale Secondary Nameserver fuer die Subdomain
  local(@serv);
  &query_bind($csr_serv, *serv, $id);

  # Schleife ueber alle lokalen Secondary Nameserver
  for(@serv)
```

```
  {
    # Host-Parameter des aktuellen Nameservers
    local($hostname, $subdomainname, $domainname) = split(/$;/);
    # $hostname & $domainname koennen nicht NULL sein
    # $subdomainname darf NULL sein

    # Aufbau des Fully Qualified Domain Name fuer den Secondary Nameserver
    local($fqdn);
    if((!&testnull($subdomainname)) && (length($subdomainname) > 0))
    {
      $fqdn = $hostname . '.' . $subdomainname . '.' . $domainname;
    }
    else
    {
      $fqdn = $hostname . '.' . $domainname;
    }

    # Ausgabe des Secondary Nameservers
    print "\t\tIN\tNS\t$fqdn.\n";
  }

  # ------------------------------------------------------------------

  # Fremde Secondary Nameserver fuer die Subdomain
  local(@secf);
  &query_bind($csr_secf, *secf, $id);

  # Schleife ueber alle fremden Secondary Nameserver
  for(@secf)
  {
    # $_ kann nicht NULL sein

    # Ausgabe des Secondary Nameservers
    print "\t\tIN\tNS\t$_.\n";
  }
}

# Schliessen der fuer delegierte (lokale) Subdomains geoeffneten DB-Cursor
&query_close($csr_prim);
&query_close($csr_serv);
&query_close($csr_secf);

# ---------------------------------------------------------------------

# Nameserver Eintraege fuer delegierte (fremde) Subdomains

# Auswahl der fremden Zonen (mit Primary Nameserver), die die aktuelle Zone
# als Vaterzone haben
local(@del_zones);
&query(<<"  END_SQL", *del_zones);
  select id, name, pns_ip_name
```

```
  from ${DB}del_zones where parent = $_[11]
END_SQL

# Auswahl der (fremden) Secondary Nameserver einer bestimmten delegierten
# fremden Zone
local($csr_dzs) = &query_open (<<"  END_SQL");
  select sns_ip_name from ${DB}del_zones_sec where del_zone = :1
END_SQL

# Auswahl der lokalen Secondary Nameserver einer bestimmten fremden Zone
local($csr_seclf) = &query_open (<<"  END_SQL");
  select h.name, h.subdomain_name, z.name
  from ${DB}zones z, ${DB}hosts h, ${DB}nameservers n, ${DB}secondary_lf s
  where z.id = h.zone and h.id = n.host and
        n.id = s.nameserver and s.zone_name = :1
END_SQL

# Schleife ueber alle delegierten fremden Subdomains
for(@del_zones)
{
  # Parameter der aktuellen Subdomain
  local($id, $name, $pns_ip_name,) = split(/$;/);
  # $id, $name & $pns_ip_name koennen nicht NULL sein

  # Subdomain-Name (innerhalb) der Domain bestimmen
  local($subdomain) = $name;
  $subdomain =~ s/\.$current$//i;

  # Ausgabe des Subdomain-Namens mit Primary Nameserver
  print "$subdomain\tIN\tNS\t$pns_ip_name.\n";

  # Fremde Secondary Nameserver
  local(@dzs);
  &query_bind($csr_dzs, *dzs, $id);

  # Schleife ueber alle fremden Secondary Nameserver
  for(@dzs)
  {
    # $_ kann nicht NULL sein

    # Ausgabe des Secondary Nameservers
    print "\t\tIN\tNS\t$_.\n";
  }

  # Lokale Secondary Nameserver
  local(@seclf);
  &query_bind($csr_seclf, *seclf, $name);

  # Schleife ueber alle lokalen Secondary Nameserver
  for(@seclf)
  {
```

```
    # Host-Parameter des aktuellen Nameservers
    local($hostname, $subdomainname, $domainname) = split(/$;/);
    # $hostname & $domainname koennen nicht NULL sein
    # $subdomainname darf NULL sein

    # Aufbau des Fully Qualified Domain Name fuer den Secondary Nameserver
    local($fqdn);
    if((!&testnull($subdomainname)) && (length($subdomainname) > 0))
    {
      $fqdn = $hostname . '.' . $subdomainname . '.' . $domainname;
    }
    else
    {
      $fqdn = $hostname . '.' . $domainname;
    }

    # Ausgabe des Secondary Nameservers
    print "\t\tIN\tNS\t$fqdn.\n";
  }
}

# Schliessen der fuer delegierte (fremde) Subdomains geoeffneten DB-Cursor
&query_close($csr_dzs);
&query_close($csr_seclf);

# --------------------------------------------------------------------

# PTR-Eintraege fuer die Zone

# Auswahl saemtlicher Hosts (Name, Subdomain- & Domainname, IP-Adresse),
# bei denen eine IP-Adresse zu der Maske passt
local(@ptr);
&query(<<"  END_SQL", *ptr);
  select h.name, h.subdomain_name, z.name, i.ip_addr
  from ${DB}ipaddr i, ${DB}hosts h, ${DB}zones z
  where z.id = h.zone and h.id = i.host and i.ip_addr like '$mask\%'
END_SQL

# Schleife ueber alle ausgewaehlten Hosts
PTR: for(@ptr)
{
  # Parameter des aktuellen Hosts
  local($hostname, $subdomainname, $domainname, $ipa) = split(/$;/);
  # $hostname, $domainname & $ipa koennen nicht NULL sein
  # $subdomainname darf NULL sein

  # Ausschluss von IP-Adressen, die einer delegierten Zone (Subnetz)
  # angehoeren
  for(@submasks)
  {
    if($ipa =~ /^$mask$_/) {next PTR;}
```

```
    }

    # Aufbau des Fully Qualified Domain Name
    local($fqdn);
    if((!&testnull($subdomainname)) && (length($subdomainname) > 0))
    {
      $fqdn = $hostname . '.' . $subdomainname . '.' . $domainname;
    }
    else
    {
      $fqdn = $hostname . '.' . $domainname;
    }

    # Zerlegen der IP-Adresse in ihre 4 Bytes
    $ipa =~ /^\x20*(\d+)\.\x20*(\d+)\.\x20*(\d+)\.\x20*(\d+)$/;

    # Das 4. Byte wird immer ausgegeben
    print $4;

    # Byte 1-3 werden nur ausgegeben, wenn deren Auswahl innerhalb
    # der Zone noch frei war
    if($replace >= 2) { print ".$3"; }
    if($replace >= 3) { print ".$2"; }
    if($replace == 4) { print ".$1"; }

    # Ausgabe des restlichen Records (Klasse, Typ, FQDN)
    print "\t\tIN\tPTR\t$fqdn.\n";
  }
}

# ************************************************************************
1;
```

A.9 Modul named_local.pl

```
# ******************************************************************
# * Integration des Managements des Domain-Name-Systems in die       *
# * vorhandene Managementumgebung des Leibniz-Rechenzentrums (LRZ)   *
# * (Diplomarbeit an der TU-Muenchen, SS 94 - WS 94/95)              *
# * -------------------------------------------------------------- *
# * Autor: Heiko Hauck (hauck@informatik.tu-muenchen.de)             *
# * Datei: named_local.pl (Oraperl Bibliothek fuer ora2dns)          *
# ******************************************************************

# PROZEDUR zur Generierung der Zonen-Datei fuer das Loopback-Interface
# (127.1 -> localhost)
# Parameter: $_[0] = Name der Datei
#            $_[1] = IP-Adresse des Nameserverhosts
#            $_[2] = Klasse des SOA-Records
#            $_[3] = FQDN des Nameserverhosts
#            $_[4] = E-Mail-Adresse (SOA-Responsible)
#            $_[5] = Seriennummer im SOA-Record
#            $_[6] = Refresh-Intervall im SOA-Record
#            $_[7] = Retry-Intervall im SOA-Record
#            $_[8] = Expire-Intervall im SOA-Record
#            $_[9] = Minimum-Intervall im SOA-Record
# Die Ausgabe erfolgt zur Standard-Ausgabe.

sub named_local
{
  # Ausgabe des Dateikopfes
  print "; $_[0] for $_[3] [$_[1]]\n";
  print "; created by ora2dns (08/94 by Heiko Hauck; Diplomarbeit TUM)\n";
  print ";\n";
  print "; User: ", `whoami`;
  print "; Date: ", `date`;
  print ";\n";

  # Ausgabe der betroffenen Zone
  print "\$ORIGIN\t0.0.127.in-addr.arpa.\n";

  # Ausgabe des SOA-Records fuer diese Zone
  print "\@\t$_[2]\tSOA\t$_[3]. $_[4]. (\n";
  print "\t\t\t\t$_[5]   ; serial\n";
  print "\t\t\t\t$_[6]   ; refresh\n";
  print "\t\t\t\t$_[7]   ; retry\n";
  print "\t\t\t\t$_[8]   ; expire\n";
  print "\t\t\t\t$_[9] ) ; minimum\n";

  # Ausgabe des Nameservers fuer diese Zone (= aktueller Nameserver)
  print "\tIN\tNS\t$_[3].\n";

  # Ausgabe des PTR-Records, der fuer die eigentliche Abbildung zustaendig ist
  print "1\tIN\tPTR\tlocalhost.\n";
```

```
}

# *******************************************************************
1;
```

A.10 Modul named_loopback.pl

```
# *******************************************************************
# * Integration des Managements des Domain-Name-Systems in die        *
# * vorhandene Managementumgebung des Leibniz-Rechenzentrums (LRZ)    *
# * (Diplomarbeit an der TU-Muenchen, SS 94 - WS 94/95)               *
# * --------------------------------------------------------------- *
# * Autor: Heiko Hauck (hauck@informatik.tu-muenchen.de)              *
# * Datei: named_loopback.pl (Oraperl Bibliothek fuer ora2dns)        *
# *******************************************************************

# PROZEDUR zur Generierung der Zonen-Datei fuer das Loopback-Interface
# (localhost -> 127.1)
# Parameter: $_[0] = Name der Datei
#            $_[1] = IP-Adresse des Nameserverhosts
#            $_[2] = Klasse des SOA-Records
#            $_[3] = FQDN des Nameserverhosts
#            $_[4] = E-Mail-Adresse (SOA-Responsible)
#            $_[5] = Seriennummer im SOA-Record
#            $_[6] = Refresh-Intervall im SOA-Record
#            $_[7] = Retry-Intervall im SOA-Record
#            $_[8] = Expire-Intervall im SOA-Record
#            $_[9] = Minimum-Intervall im SOA-Record
# Die Ausgabe erfolgt zur Standard-Ausgabe.

sub named_loopback
{
  # Ausgabe des Dateikopfes
  print "; $_[0] for $_[3] [$_[1]]\n";
  print "; created by ora2dns (08/94 by Heiko Hauck; Diplomarbeit TUM)\n";
  print ";\n";
  print "; User: ", `whoami`;
  print "; Date: ", `date`;
  print ";\n";

  # Ausgabe der betroffenen Zone
  print "\$ORIGIN\tlocalhost.\n";

  # Ausgabe des SOA-Records fuer diese Zone
  print "\@\t$_[2]\tSOA\t$_[3]. $_[4]. (\n";
  print "\t\t\t\t$_[5]   ; serial\n";
  print "\t\t\t\t$_[6]   ; refresh\n";
  print "\t\t\t\t$_[7]   ; retry\n";
  print "\t\t\t\t$_[8]   ; expire\n";
```

```
  print "\t\t\t\t$_[9] ) ; minimum\n";

  # Ausgabe des Nameservers fuer diese Zone (= aktueller Nameserver)
  print "\tIN\tNS\t$_[3].\n";

  # Ausgabe des A-Records, der fuer die eigentliche Abbildung zustaendig ist
  print "\tIN\tA\t127.0.0.1\n";
}

# ***********************************************************************
1;
```

A.11 Modul named_cache.pl

```
# *****************************************************************
# * Integration des Managements des Domain-Name-Systems in die        *
# * vorhandene Managementumgebung des Leibniz-Rechenzentrums (LRZ)     *
# * (Diplomarbeit an der TU-Muenchen, SS 94 - WS 94/95)                *
# * ------------------------------------------------------------- *
# * Autor: Heiko Hauck (hauck@informatik.tu-muenchen.de)               *
# * Datei: named_cache.pl (Oraperl Bibliothek fuer ora2dns)            *
# *****************************************************************

# PROZEDUR zum Holen der Root-Domain-Datei (Cache) mittels Anonymous FTP
# Parameter: $_[0] = Name der lokalen Datei (inkl. Verzeichnispfad)
#            $_[1] = Name der FTP-Hosts, von dem die Datei geholt wird
#            $_[2] = Verzeichnis, in dem dort die Datei abgelegt ist
#            $_[3] = Name der Datei auf dem FTP-Host
#            $_[4] = E-Mail-Adresse (als Passwort fuer Anonymous FTP)

sub named_cache
{
  # lokale Parameterkopien
  local($local_file, $host, $remote_dir, $remote_file, $email) = @_;

  # Die Login-Name- & Passwort-Ubergabe an das FTP geschieht mittels
  # der '.netrc'-Datei des Anwenders.
  # Ist in dieser Datei noch kein ein Eintrag fuer den angegebenen FTP-Host
  # vorhanden, so wird er eingefuegt (ein vorhandener wird nicht geaendert).
  # Ist die Datei noch nicht vorhanden, wird sie mit dem Eintrag erzeugt.

  # '.netrc' liegt im Home-Verzeichnis des Anwenders
  local($filename) = $ENV{'HOME'}.'/.netrc';

  # Eintrag in '.netrc'
  local($netrc) = "machine $host login ftp password $email\n";

  # Existiert die Datei '.netrc' bereits?
  if(-e $filename)
  {
    # Datei oeffnen zum Lesen
    # Misslingt dies, dann Programmabbruch
    open(NETRC, "<$filename") || die "Can't open $filename: $!\n";

    # Zeilenweises Durchsuchen nach einen entspr. Host-Eintrag
    while(<NETRC>)
    {
      if(/^machine\s+$host\s+/)
      {
        # Wird ein solcher gefunden, braucht kein weiterer mehr gesucht werden
        $netrc = undef; # Markierung fuer "gefunden"
        last;
      }
```

```
  }

  # Datei (vom Lesen) schliessen
  close(NETRC);

  # Falls noch kein entsprechender Eintrag vorhanden ist:
  if(defined($netrc))
  {
    # Datei oeffnen zum Anfuegen
    # Misslingt dies, dann Programmabbruch
    open(NETRC, ">>$filename") || die "Can't append to $filename: $!\n";

    # Eintrag
    print NETRC $netrc;

    # Datei (vom Anfuegen) schliessen
    close(NETRC);
  }
}

else
# Datei '.netrc' existiert noch nicht
{
  # Datei oeffnen zum Schreiben
  # Misslingt dies, dann Programmabbruch
  open(NETRC, ">$filename") || die "Can't write to $filename: $!\n";

  # Eintrag
  print NETRC $netrc;

  # Datei (vom Schreiben) schliessen
  close(NETRC);
}

# Zugriffsrechte auf die Datei ".netrc" (notwendig)
chmod 0600, $filename;

# -------------------------------------------------------------------

# Oeffnen eines Ausgabekanals, der als Eingabe fuer FTP dient
open(FTP,"|ftp $host");

# ASCII-Modus, da eine Text-Datei uebertragen wird
print FTP "ascii\n";

# Wahl des Verzeichnisses auf dem FTP-Host
print FTP "cd $remote_dir\n";

# Datei soll geholt werden
print FTP "get\n";
```

```
  # Angabe des Dateinamens auf dem FTP-Host
  print FTP "$remote_file\n";

  # Angabe des lokalen Dateinamens (inkl. Verzeichnis)
  print FTP "$local_file\n";

  # FTP verlassen
  print FTP "quit\n";

  # Schliessen des Kanals und Durchfuehrung der FTP-Prozedur
  close(FTP);

  print "\n";
}

# ***********************************************************************
1;
```

Anhang B

Datenbank – Beispiel

ALIASES		
host	canonical_name	*Kommentar*
0	nettest	dfvgate
0	dfv	
0	ircserver	
3	x500	sunmanager
3	ntp1	
4	hpmanager	hpman
4	hpmgr	
4	ntp2	
6	mailhost	cd1
7	mlists	sunserver
7	ftp	
7	lrznews	
7	news	
7	idserver	
7	listserv	
7	gopher	
8	sunpop	sun1

DEL_ZONES				
id	name	pns_ip_name	pns_ip_addr	parent
0	informatik.tu-muenchen.de	tuminfo1.informatik.tu-muenchen.de	`131.159.␣␣0.␣␣1`	4

DEL_ZONES_SEC		
del_zone	sns_ip_name	sns_ip_addr
0	tuminfo2.informatik.tu-muenchen.de	`131.159.␣␣0.␣81`
0	hpsystem1.informatik.tu-muenchen.de	`131.159.␣␣0.176`
0	ns.Germany.EU.net	

HOSTS						
id	name	subdomain_name	zone	komponente	cpu	os
0	dfvgate		0	WSC10018		
1	suncollect		0	WSC10018		
2	hp10		0	WSCP0043		
3	sunmanager		0	WSC30000		
4	hpman		0	WSC00016		
5	hpman2		0	WSC30596		
6	cd1		0	RECP0005		
7	sunserver		0	WSC20010		
8	sun1		0	WSC20005		
9	sun2		0	WSC20008		
10	a2824ch		1		PC	Linux/PPP
11	a2824ar	ppp	0		PC	Linux/PPP
12	hp11		0	WSCP0044		

IPADDR		
host	ip_addr	*Kommentar*
0	`129.187.␣10.␣25`	dfvgate
1	`129.187.␣␣9.241`	suncollect
2	`129.187.␣13.␣22`	hp10
2	`129.187.␣16.␣␣1`	
3	`129.187.␣10.␣32`	sunmanager
4	`129.187.␣10.160`	hpman
5	`129.187.␣10.159`	hpman2
6	`129.187.␣13.␣␣3`	cd1
6	`129.187.␣14.␣45`	
7	`129.187.␣10.␣35`	sunserver
7	`129.187.␣13.␣35`	
8	`129.187.␣10.␣20`	sun1
9	`129.187.␣10.␣21`	sun2
10	`129.187.␣15.195`	a2824ch
11	`129.187.␣15.157`	a2824ar
12	`129.187.␣16.␣␣2`	hp11

MAILX				
zone	name	pref_value	host	*Kommentar*
0		140	6	cd1
0	sunmail	10	7	sunserver
0	hpmail	10	2	hp10
1	a2824ch	10	6	cd1
1	a2824ar	10	6	cd1

NAMESERVERS		
id	host	*Kommentar*
0	0	Nameserver auf dfvgate
1	2	Nameserver auf hp10

SECONDARY_LL		
zone	nameserver	*Kommentar*
0	1	lrz-muenchen.de
1	1	ppp.lrz-muenchen.de
2	1	187.129.in-addr.arpa
3	1	15.187.129.in-addr.arpa
4	1	tu-muenchen.de

SECONDARY_LF			
nameserver	zone_name	ip_addr	*Kommentar*
0	informatik.tu-muenchen.de	`131.159.␣␣0.␣␣1`	Nameserver auf dfvgate
1	informatik.tu-muenchen.de	`131.159.␣␣0.␣␣1`	Nameserver auf hp10
0	eg	`193.227.␣␣1.␣␣1`	Nameserver auf dfvgate

SECONDARY_FL			
zone	ip_name	ip_addr	*Kommentar*
0	ns.nic.de		lrz-muenchen.de
0	charly.bl.physik.tu-muenchen.de		
4	ns.nic.de		tu-muenchen.de
4	charly.bl.physik.tu-muenchen.de	`129.187.160.␣10`	
4	tuminfo1.informatik.tu-muenchen.de	`131.159.␣␣0.␣␣1`	

ZONES					
id	name	prim_nameserver	parent	soa_responsible	soa_serial
0	lrz-muenchen.de	0		postmaster@dfvgate.lrz-muenchen.de	1994082900
1	ppp.lrz-muenchen.de	0	0	postmaster@dfvgate.lrz-muenchen.de	1994082901
2	187.129.in-addr.arpa	0		postmaster@dfvgate.lrz-muenchen.de	1994082902
3	15.187.129.in-addr.arpa	0	2	postmaster@dfvgate.lrz-muenchen.de	1994082903
4	tu-muenchen.de	0		postmaster@dfvgate.lrz-muenchen.de	1994082904

Anhang C

Generierte Konfigurationsdateien – Beispiele

Die im folgenden angegebenen Dateien sind gegenüber den Originalen z.T. gekürzt [...] bzw. vom Layout an die vorliegende Arbeit angepaßt. Syntax und Semantik werden dadurch nicht beeinflußt.

C.1 Resolverkonfiguration

```
# /etc/resolv.conf for dfvgate.lrz-muenchen.de [129.187.10.25]
# created by ora2dns (08/94 by Heiko Hauck; Diplomarbeit TUM)

# User: a2824ch
# Date: Wed Oct 26 15:42:48 MET 1994

domain     lrz-muenchen.de

nameserver 127.0.0.1          # localhost
nameserver 129.187.13.22      # hp10.lrz-muenchen.de
```

C.2 Nameserverkonfiguration

```
; /etc/named.boot for dfvgate.lrz-muenchen.de [129.187.10.25]
; created by ora2dns (08/94 by Heiko Hauck; Diplomarbeit TUM)
;
; User: a2824ch
; Date: Wed Oct 26 15:42:56 MET 1994
;
; type          domain                  source file or host     backup file
```

```
;
directory       /etc/namedb
; primary nameserver:
primary         lrz-muenchen.de          lrz-muenchen.de.DNF
primary         ppp.lrz-muenchen.de      ppp.lrz-muenchen.de.DNF
primary         187.129.in-addr.arpa     187.129.in-addr.arpa.DNF
primary         15.187.129.in-addr.arpa 15.187.129.in-addr.arpa.DNF
primary         tu-muenchen.de           tu-muenchen.de.DNF
primary         0.0.127.in-addr.arpa     named.local
primary         localhost                named.loopback
; secondary nameserver:
secondary       informatik.tu-muenchen.de 131.159.0.1   informatik.tu-muenchen.de.BF
secondary       eg                       193.227.1.1    eg.BF
; cache nameserver:
cache           .                        named.cache
```

C.3 Zonenkonfiguration (Forward Mapping)

```
; /etc/namedb/lrz-muenchen.de.DNF for dfvgate.lrz-muenchen.de [129.187.10.25]
; created by ora2dns (08/94 by Heiko Hauck; Diplomarbeit TUM)
;
; User: a2824ch
; Date: Wed Oct 26 15:42:56 MET 1994
;
$ORIGIN         lrz-muenchen.de.
@               IN      SOA     dfvgate.lrz-muenchen.de. postmaster@[...] (
                                        1994082900    ; serial
                                        21600         ; refresh
                                        3600          ; retry
                                        604800        ; expire
                                        172800 )      ; minimum
                IN      NS      dfvgate.lrz-muenchen.de.
                IN      NS      hp10.lrz-muenchen.de.
                IN      NS      ns.nic.de.
                IN      NS      charly.bl.physik.tu-muenchen.de.
ppp             IN      NS      dfvgate.lrz-muenchen.de.
                IN      NS      hp10.lrz-muenchen.de.
@               IN      MX      140     cd1.lrz-muenchen.de.
sunmail         IN      MX      10      sunserver.lrz-muenchen.de.
hpmail          IN      MX      10      hp10.lrz-muenchen.de.
dfvgate         IN      A       129.187.10.25
nettest         IN      CNAME   dfvgate
dfv             IN      CNAME   dfvgate
ircserver       IN      CNAME   dfvgate
suncollect      IN      A       129.187.9.241
hp10            IN      A       129.187.13.22
                IN      A       129.187.16.1
sunmanager      IN      A       129.187.10.32
x500            IN      CNAME   sunmanager
```

```
ntp1            IN      CNAME   sunmanager
[...]
sun1            IN      A       129.187.10.20
sunpop          IN      CNAME   sun1
sun2            IN      A       129.187.10.21
a2824ar.ppp     IN      A       129.187.15.157
                IN      HINFO   PC Linux/PPP
hp11            IN      A       129.187.16.2
```

C.4 Zonenkonfiguration (Reverse Mapping)

```
; /etc/namedb/187.129.in-addr.arpa.DNF for dfvgate.lrz-muenchen.de [129.187.10.25]
; created by ora2dns (08/94 by Heiko Hauck; Diplomarbeit TUM)
;
; User: a2824ch
; Date: Wed Oct 26 15:43:01 MET 1994
;
$ORIGIN         187.129.in-addr.arpa.
@               IN      SOA     dfvgate.lrz-muenchen.de. postmaster@[...] (
                                        1994082902   ; serial
                                        21600        ; refresh
                                        3600         ; retry
                                        604800       ; expire
                                        172800 )     ; minimum
                IN      NS      dfvgate.lrz-muenchen.de.
                IN      NS      hp10.lrz-muenchen.de.
15              IN      NS      dfvgate.lrz-muenchen.de.
                IN      NS      hp10.lrz-muenchen.de.
25.10           IN      PTR     dfvgate.lrz-muenchen.de.
241.9           IN      PTR     suncollect.lrz-muenchen.de.
22.13           IN      PTR     hp10.lrz-muenchen.de.
1.16            IN      PTR     hp10.lrz-muenchen.de.
[...]
20.10           IN      PTR     sun1.lrz-muenchen.de.
21.10           IN      PTR     sun2.lrz-muenchen.de.
2.16            IN      PTR     hp11.lrz-muenchen.de.
```

Literaturverzeichnis

[Baye 94] Rudolf Bayer, *Vorlesung Datenbanksysteme I*, Technische Universität München, Sommersemester 1994.

[Ecke 89] Toerless Eckert, *Unix Manual Page for* df2hf, Friedrich Alexander Universität, Erlangen, Januar 1989.

[Frei 93] B. Freitag, *Vorlesungsskriptum Datenbanken I*, Technische Universität München, Juli 1993.

[Geor 93] Jürgen Georgi und Ulrike Dillmann, „Domain Name System & X.500 Directory“, KOM-93, Rechenzentrum der Universität Stuttgart, 1993.

[Golk 91] Lisa Golka, Ulrike Dillmann und Jürgen Georgi, *Nameserver – Das Domain Name System und die Verwaltung einer Internet-Domain in Deutschland*, Rechenzentrum der Universität Stuttgart, 1991.

[Hege 93] Heinz-Gerd Hegering und Sebastian Abeck, *Integriertes Netz- und Systemmanagement*, Addison-Wesley (Deutschland) GmbH, Bonn, 1993.

[Heue 92] Andreas Heuer, *Objektorientierte Datenbanken: Konzepte, Modelle, Systeme*, Addison-Wesley (Deutschland) GmbH, Bonn, 1992.

[IBM 89] International Business Machines Corporation, *AIX Operating System: TCP/IP Users Guide*, 2. Auflage, Dezember 1989.

[Kern 93] Helmut Kerner, *Rechnernetze nach OSI*, 2. Auflage, Addison-Wesley (Deutschland) GmbH, Bonn, 1993.

[Misg 91] Wolfgang D. Misgeld, *ORACLE für Profis*, Carl Hanser Verlag, München Wien, 1991.

[Quar 94] John S. Quarterman und Smoot Carl-Mitchell, *The Internet connection: system connectivity and configuration*, Addison-Wesley Unix and Open Systems Series, Texas Internet Consulting, 1994.

[RFC 1010] J. Reynolds und J. Postel, „Assigned Numbers“, RFC 1010, USC Information Sciences Institute, Marina del Rey, California, Mai 1987.

[RFC 1032] M. Stahl, „Domain Administrators Guide“, RFC 1032, SRI International, November 1987.

[RFC 1034] P. Mockapetris, „Domain Names – Concepts and Facilities“, RFC 1034, USC Information Sciences Institute, November 1987.

[RFC 1035] P. Mockapetris, „Domain Names – Implementation and Specification“, RFC 1035, USC Information Sciences Institute, November 1987.

[RFC 1611] R. Austein und J. Saperia, „DNS Server MIB Extensions“, RFC 1611, Epilogue Technology Corp. / Digital Equipment Corp., Mai 1994.

[RFC 1612] R. Austein und J. Saperia, „DNS Resolver MIB Extensions“, RFC 1612, Epilogue Technology Corp. / Digital Equipment Corp., Mai 1994.

[RFC 881] J. Postel, „The Domain Names Plan and Schedule“, RFC 881, USC Information Sciences Institute, November 1983.

[RFC 882] P. Mockapetris, „Domain names - Concepts and Facilities“, RFC 882, USC Information Sciences Institute, November 1983.

[RFC 883] P. Mockapetris, „Domain names - Implementation and Specification“, RFC 883, USC Information Sciences Institute, November 1983.

[RFC 974] Craig Partridge, „Mail Routing and the Domain System“, RFC 974, CSNET CIC BBN Laboratories Inc., Januar 1986.

[Sant 93] Michael Santifaller, *TCP/IP und ONC/NFS in Theorie und Praxis*, Addison-Wesley (Deutschland) GmbH, Bonn, 1993.

[Saue 92] Hermann Sauer, *Relationale Datenbanken: Theorie und Praxis*, Addison-Wesley (Deutschland) GmbH, Bonn, 1992.

[Teor 90] Toby J. Teorey, *Database Modeling and Design: The Entity-Relationship Approach*, Morgan Kaufmann Publishers Inc., 1990.

[UCBK] University of California, Berkeley, *Unix Manual Pages*.

[Valt 92] Robert Valta, *Netzdokumentation (Version 1.0)*, Leibniz-Rechenzentrum, München, April 1992.

[Valt 94] Robert Valta, *Auswertung der Umfrage zur aktuellen Netzdokumentation*, Leibniz-Rechenzentrum, München, Januar 1994.

[Vixi 93] Paul Vixie, *Name Server Operations Guide for BIND, Release 4.9*, Digital Equipment Corp., Palo Alto, California, April 1993.

[Wall 91] Larry Wall und Randal L. Schwartz, *Programming Perl*, O'Reilly & Associates Inc., Sebastopol, California, Januar 1991.