Prüfer: Prof. Dr. A. Reuter

Betreuer: Friedemann Schwenkreis

Beginn am: 01.12.1992
Beendet am: 31.05.1993

CR-Nummer: H.2.4, D.4.4, D.4.5

Studienarbeit-Nr. 1231

Erweiterung eines Kommunikationsmechanismus für den rechnerübergreifenden Betrieb

Michael Häuptle

Fakultät für Informatik
Institut für Parallele und verteilte Höchstleistungsrechner (IPVR)
Universität Stuttgart
Breitwiesenstraße 20-22
7000 Stuttgart 80

Inhaltsverzeichnis

1 Einführung 5

2 Anforderungen an das Kommunikationssystem von APRICOTS 8

2.1 Der Resource-Manager-Call (RM_Call) 9

2.2 Rechnerübergreifende Problemstellung 10

3 Bekannte Lösungsansätze 11

3.1 Das Distributed Computing Environment (DCE) 11

3.2 Camelot and Avalon 14

3.3 Domain Name System (DNS) 16

3.4 Bewertung im Hinblick auf die Anforderungen 18

3.4.1 DCE 18

3.4.2 Camelot und Avalon 19

3.4.3 Domain Name System 19

4 Architektur für die Kommunikation 20

4.1 Inselgruppen 20

4.2 Der Nameserver 21

4.3 Briefträger 25

4.4 Rechnerübergreifende Kommunikation 27

5 Realisierung 29

5.1 Netzwerkkommunikation 29

5.1.1 Realisierung des Client und Server 29

5.1.2 Unterprogramme für die Netzwerkkommunikation 30

5.1.3 Zeitüberschreitung und Fehlerbehandlung 31

5.2 Der Briefträger 32

5.2.1 Funktion und Nachrichtenformat des Briefträgers 34

5.2.2 Protokoll der Datenübetragung 35

5.2.3 Bearbeitung einer Nachricht durch den Server 36

5.2.4 Schnittstelle zum Briefträger 38
5.3 Der Nameservice 39
5.3.1 Datenstruktur und Schnittstelle zur Datenbank 39
5.3.2 Der Nameserver 43
5.3.3 Protokoll des Nameservers 45
5.3.4 Schnittstelle zum Nameserver 47
5.3.5 Beispiel für eine Dienstsuche 48
5.4 Betrachtungen zur Portabilität 50

6 Zusammenfassung und Ausblick 51
6.1 Briefträger 51
6.2 Nameservice 53

7 Literatur 54

Anhang 56

Abbildungsverzeichnis

1: APRICOTS Komponenten auf einem Rechner. 7
2: Aufbau der logischen Adresse eines Dienstes. 8
3: Client sucht Dienst D mittels dem CDS und rpcd. 12
4: Verbindung der Cells über den GDS (Global Directory Service). 13
5: Arbeitsweise eines Mach Remote Procedure Calls. 15
6: Auszug aus dem Domainnamensraum. 16
7: Aufbau einer Inselgruppe. 20
8: Verbindung der Inselgruppen über den Nameserver. 21
9: Schema der Kommunikation zwischen entfernten Rechnern. 25
10: Socketfunktionen für eine TCP/IP-Verbindung. 29
11: Unterprogramme für die Kommunikation über das Netzwerk. 30
12: Aufbau des Briefträgers. 33
13: Ablauf des Protokolls zwischen send_message() und receive_message(). 35
14: Schnittstelle zur Nameservice-Datenbank. 41
15: Vernetzung von Inselgruppen. 42
16: Protokoll bei einer LOOKUP Nachricht eines Clients. 45
17: Protokoll einer REQUESTDB Nachricht. 46
18: Ablauf einer Dienstsuche. 48

1 Einführung

Die vorliegende Studienarbeit befaßt sich mit Kommunikationsmechanismen für den rechnerübergreifenden Betrieb im APRICOTS-Projekt. Ziele dieser Studienarbeit sind, die bestehende Schicht des Kommunikationssystem von APRICOTS für die Nachrichtenübermittlung auf entfernte Rechner in einem Netzwerk zu erweitern und ein Protokoll für das Auffinden von Server-Diensten zu entwickeln.
In diesem ersten Kapitel möchte ich APRICOTS kurz vorstellen. Ausführlichere Beschreibungen befinden sich in [3], [4], [17] und [18].

Bei langdauernden Abläufen, wie sie bei Fertigungsprozessen oder großen, verteilten Datenbanksystemen vorkommen, eignet sich das klassische Modell einer ACID[1]-Transaktion für die Überführung des Systems von einem logisch konsistenten Zustand in einen neuen logisch konsistenten Zustand nur bedingt. Zum Beispiel schließt eine ACID-Transaktion aus, Einfluß auf den Kontrollfluß zu nehmen, d.h. es existiert keine Möglichkeit, Beziehungen zwischen den einzelnen Transaktionen zu definieren, um eine bestimmte Transaktion beim Scheitern einer anderen Transaktion nicht auszuführen. Aufgrund des Konzeptes der Atomarität einer ACID-Transaktion wird im Fehlerfall der Ausgangszustand wieder hergestellt. Die unter Umständen nützliche Arbeit, die diese Transaktion bewältigt hat, verschwindet dabei. Die ACID-Transaktion bieten weiter nur gute Performance, wenn die einzelnen Transaktionen wenig Zeit im System verbringen bzw. nur wenige Datensätze berühren müssen.

Aufgrund der Einschränkungen des ACID-Modells und dem Bedarf eines anwendungsneutralen Systemdienstes für die zuverlässige Ausführung langlebiger, verteilter Vorgänge wurde das ConTract-Modell entwickelt [3]. Ein ConTract garantiert dabei die Abarbeitung einer "Ausführungseinheit" in der vorgesehenen Zeit oder aber die Zurücksetzung auf den logischen Ausgangszustand mit Hilfe von Kompensationsaktionen. Der ConTract modelliert also die gleiche Semantik einer Zustandsübergangsbeschreibung wie die ACID-Transaktion, erlaubt aber die Ausführung von sequentiellen oder parallelen Transaktionen über ein Programm, das den Kontrollfluß zwischen diesen Transaktionen steuert (die einzelnen Transaktionen basieren dabei auf dem ACID-Modell). Dieses "modula"[2]-ähnliche Programm wird in der ConTract Terminologie als "Skript" bezeichnet.

Das Skript beschreibt den Ablauf einzelner Ausführungseinheiten und stellt dem Anwendungsprogrammierer Kontrollkonstrukte für bedingte Verzweigungen, Schleifen und parallele Verarbeitung zur Verfügung. Die parallel ablaufenden ConTracts können über sog. Invarianten synchronisiert werden und im Falle einer Verletzung dieser Invarianten entsprechende Konfliktbehandlungsmaßnahmen durchführen.
Durch die Angabe von Abhängigkeiten zwischen Transaktionen läßt sich der Ablauf des ConTracs weiter regeln.

---

[1] ACID – Atomicity, Consistency, Isolation, Durability.

[2] Die Syntax der Programmiersprache ist an Modula angelehnt.

Beispielsweise wird bei einer Kombination von Flugbuchung und gleichzeitiger Hotelreservierung nur dann die Hotelreservierung ausgeführt, wenn eine Flugbuchung zum Zielort möglich ist und umgekehrt.

Die einzelnen Ausführungseinheiten werden "Steps" genannt und laufen unter ACID-Transaktionsschutz ab. Die Steps arbeiten mit einem stabilen und wiederherstellbaren Ausführungskontext in dem ihre erzeugten Zwischenergebnisse verwaltet werden. Diese Zwischenergebnisse können anderen Steps innerhalb desselben ConTracts zugänglich gemacht werden. Weiterhin sind die Steps für die Auswertung der Invarianten und dem Aufrufen der "Objektverwalter" (Resource Manager), die die eigentlichen Funktionen implementieren, zuständig.

Eine Implementation für die Abwicklung von ConTracs wird in dem Prototypsystem APRICOTS[3] am IPVR durchgeführt. Neben den Verarbeitungsmechanismen auf Basis der ConTracts, soll das System eine möglichst breite Anwendungsschicht abdecken und sich auf viele Betriebssystemen leicht portieren lassen. Das Prototypsystem soll dabei auf Standard-Hard- und Software-Komponente aufbauen und vorhandene Datenbanksysteme nutzen. Im Gegensatz zum ACID-Konzept, das in der Regel nur bei Datenbanksystemen verwendet wird, lassen sich mit Hilfe des ConTract System beliebige Anwendungensarten modellieren. Beispielsweise können die Resource-Manager auf verschiedene physische Geräte, wie sie z.B. in Fertigungsprozessen vorkommen, zugreifen.
Ein weiteres Entwurfsziel von APRICOTS ist die Fehlertoleranz und Ausfallsicherheit des Systems. Der Ausfall eines Rechnerknoten oder einer Systemkomponente von APRICOTS darf kein Scheitern des ConTracts verursachen. Das Laufzeitsystem muß in diesem Falle die Ausführungseinheit auf einen anderen Knoten verlagern können (Migration).

Die von den Resource-Manager angebotenen Funktionen werden generell als Dienste oder Service bezeichnet. In APRICOTS lassen sich die Dienste eines oder mehreren Resource-Manager zu sog. "Komponenten" zusammenfassen. Eine Komponente läuft in einem eigenen Prozeß ab. Aufgrund der Zusammenfassung von Resource-Managern zu Komponenten läßt sich relativ teure Interprozeßkommunikation für den Datenaustausch vermeiden. Jede Komponente in APRICOTS kann als Client oder Server arbeiten. Wegen dieser Anforderung wird der klassische Remote Procedure Call (RPC) zu einem sog. Resource-Manager-Call (RM_call; siehe nächstes Kapitel) erweitert, der die Aktivierung eines Dienstes in Form von *Nachrichten* übermittelt. Der RM_call arbeitet im Gegensatz zum RPC sowohl synchron wie auch asynchron. Die Komponenten können also Nachrichten für die Aktivierung eines ihrer Dienste empfangen und selbst über den RM_call Nachrichten an andere Komponenten versenden.
Der Nachrichtentransport zwischen den Komponenten auf einem Rechnerknoten wird über "Mailboxen" realisiert. Die Mailboxen stellen einen stabilen Datenbereich mit einer FIFO-Organisation dar. Die Nachrichten in der Mailbox überleben einen Ausfall des Rechnerknotens oder den Absturz der Komponente.

[3]APRICOTS – A Prototyp Implementation of a ConTract System.

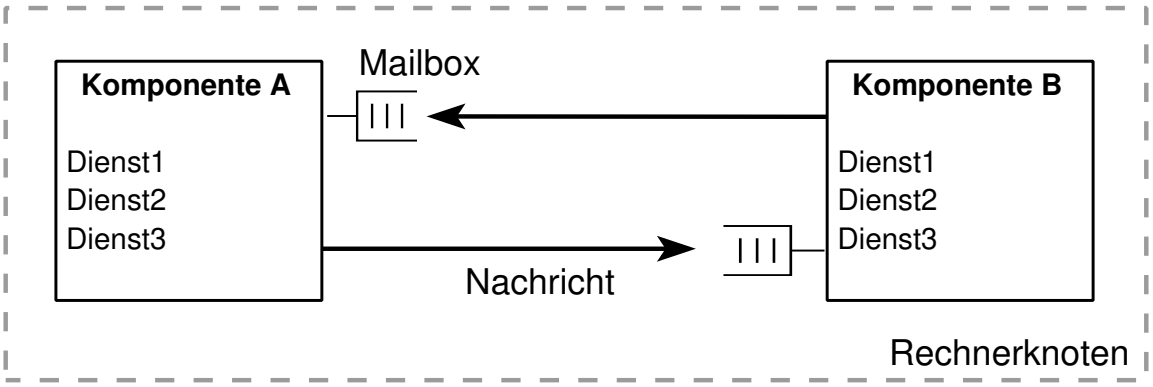

Abbildung 1: APRICOTS Komponenten auf einem Rechner.

Die Abwicklung des RM_Calls und der Transport der Nachrichten zwischen den Komponenten wird vom *Kommunikationssystem* realisiert. Es stellt den Basismechanismus für APRICOTS dar und wird in jede Komponente eingebunden.

Ziel dieser Studienarbeit ist es nun, den Nachrichtentransport über Knotengrenzen in einem heterogenen und verteilten Netzwerk zu realisieren. Weiterhin soll ein Konzept für die Bekanntgabe und Suche von Diensten entwickelt werden.

2 Anforderungen an das Kommunikationssystem von APRICOTS

Aus der Beschreibung der ConTracts und den Entwurfszielen von APRICOTS stellen sich grundsätzlich folgende Anforderungen an das Kommunikationssystem:

- Resource-Manager die einer bestimmten Klasse angehören, lassen sich zu einer Komponente zusammenfassen. Die Komponenten werden jeweils in einen eigenen Prozeß geladen. Das Kommunikationssystem muß Methoden für die Dienstaktivierung zwischen den Komponenten zur Verfügung stellen.

- Eine Komponente kann gleichzeitig Client und Server sein. Es muß deshalb möglich sein, eine Dienstaktivierung asynchron abzuarbeiten. Die Komponente wird in diesem Falle nicht blockiert und kann selbst eigene Dienstaktivierungen von anderen Komponenten ausführen.

- Die Nachrichten, die einen bestimmten Dienstaufruf an eine Komponente übermitteln, dürfen im Falle eines Rechner- oder Komponentenabsturz nicht verloren gehen. Das Kommunikationssystem muß einen stabilen Speicherbereich für die Pufferung der Nachrichten verwalten (Mailbox).

- Ein ConTract garantiert die zuverlässige Abwicklung von Basistransaktionen während seiner Laufzeit. Im Falle eines Knotenausfalls oder Absturz einer Komponente, muß das Kommunikationssystem die Komponente auf einen anderen Knoten migrieren können. Dies erfordert, daß der Dienst nicht mit einer physikalischen Adresse angesprochen wird, da sich diese bei der Migration ändert. Das Kommunikations-system benutzt deshalb eine logische Adressierung von Diensten, die eine Ortstransparenz der Komponente ermöglichen. Eine logische Adresse ist dabei nach einem dreistufigen Konzept aufgebaut:

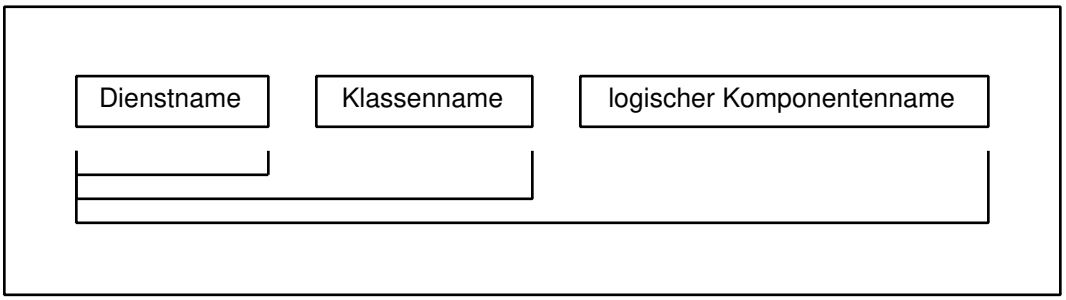

Abbildung 2:Aufbau der logischen Adresse eines Dienstes.

Die allgemeinste Adressierungsmöglichkeit besteht durch Angabe eines Dienstnamens. Dies läßt sich durch hinzufügen eines Klassennamens einschränken. Eine vollständige symbolische Adresse besteht aus dem logischen Komponentennamen mit Klassen- und Dienstname.

Das Kommunikationssystem bietet den Resource-Manager-Call (RM_Call) als eine abstrakte Schnittstelle für die Dienstaktivierung zwischen den Komponenten an.

2.1 Der Resource-Manager-Call (RM_Call)

Der RM_Call implementiert eine einheitliche Schnittstelle für die Kommunikation zwischen den Komponenten. Ein Dienst wird nach dem in Abbildung 2 beschriebenen Schema spezifiziert. Im RM_Call erfolgt anhand des Dienst- und/oder Klassennamens die Umsetzung auf die physikalische Adresse der Komponente. Diese "Namensumsetzung" liefert den Rechnernamen auf dem sich eine Komponente mit dem angegebenen Dienst befindet. Für den Nachrichtentransport ist weiterhin die Mailbox-ID der Komponente notwendig, um in den stabilen Datenbereich der Mailbox die Nachricht zu hinterlegen. Je nachdem wo sich die Komponente befindet, lassen sich drei Fälle unterscheiden:

1. **Prozeßlokaler Fall:**
   Die Implementierung des spezifizierten Dienstes befindet sich in derselben Komponente und damit im gleichen Prozeß. Der Aufruf des Dienstes erfolgt über einen "normalen" Funktionsaufruf.

2. **Rechnerlokaler Fall:**
   Die Komponente mit dem spezifizierten Dienst befindet sich auf demselben Rechnerknoten. Das Kommunikationssystem baut eine Verbindung zur Mailbox der Komponente auf und überträgt die Nachricht für den Dienstaufruf. Die Nachricht wird dann, entsprechend der FIFO-Organisation der Mailbox, von der Empfängerkomponente bearbeitet.

3. **Entfernter Fall:**
   Die Komponente befindet sich auf einem entfernten Rechner. In diesem Fall muß die Nachricht erst zum Zielrechner transferiert und anschließend in die Mailbox der entsprechenden Komponente übertragen werden.

Durch dieses dreistufige Vorgehen läßt sich die Dienstaktivierung im Hinblick auf den Ort der Empfängerkomponente optimieren und teure Interprozeßkommunikation vermeiden.

Die Ausführung eines RM_Calls kann asynchron oder synchron, d.h. blockierend erfolgen. Im 1.Fall wird die Dienstaktivierung wegen der Semantik eines Funktionssaufrufes immer synchron ausgeführt bzw. läßt sich durch Eintragen einer entsprechenden Nachricht über die Dienstaktivierung in die eigene Mailbox umgehen.

Bei einem asynchronen RM_call wird ein sog. callback-service angegeben. Über diesen Dienst wird die Komponente, die den RM_Call ausgeführt hat benachrichtigt, daß die Dienstaktivierung abgeschlossen wurde.

2.2 Rechnerübergreifende Problemstellung

Aufgrund der logischen Adressierung von Diensten ist eine Namensumsetzung auf die physikalische Adresse einer Komponente notwendig. Diese Aufgabe der Umsetzung übernimmt der **Nameservice**.
Das Kommunikationssystem beauftragt den Nameservice, zu einem übergebenen Dienst- und Klassennamen, den Rechnernamen und die Mailbox-ID einer Komponente, die diesen Dienst implementiert, ausfindig zu machen.

Erfolgt die Dienstaktivierung auf einem entfernten Rechnerknoten (Fall 3 aus Unterkapitel 2.1) so wird der **Briefträger** (siehe Kapitel 4.3) mit der Weiterleitung der Nachricht beauftragt. Der Briefträger ist für die rechnerübergreifende Kommunikation zuständig.

Der Nameservice und der Briefträger müssen sich hinsichtlich den Entwurfszielen von APRICOTS wie Ausfallsicherheit, Portabilität, Fehlertoleranz, etc. in das Gesamtsystem integrieren lassen.

3 Bekannte Lösungsansätze

Die beiden vorgestellten Konzepte eines Briefträgers und Nameservice existieren auch bei anderen verteilten Systemen. Bevor nun die eigentliche Beschreibung und Implementierung im Hinblick auf APRICOTS folgt, möchte ich das Distributed Computer Environment, Camelot & Avalon und das Domain Name System als weitere Beispiele für verteilte Systeme behandeln. Sie liefern Grundlagen, die in das Design des Briefträgers und des Nameservice von APRICOTS einflossen. Diese Systeme sind zwar teilweise für allgemeinere Aufgaben entworfen worden (vor allem das DCE), aber im Bereich des Kommunikationssystems haben sie im wesentlichen die gleichen Anforderungen.

3.1 Das Distributed Computing Environment (DCE)

Das DCE von der Open Software Foundation [10] ist eine Entwicklungsumgebung, um verteilte Anwendungen, selbst über einen weltweiten, heterogenen Netzwerkverbund zu ermöglichen.
Kernstück dieses Systems sind die Remote Procedure Calls (RPC). Ein RPC verhält sich wie ein lokaler Prozeduraufruf, mit dem Unterschied, daß die Prozedur auf einem Server-Rechner ausgeführt wird. Es ergibt sich also eine klassische Client-Server Architektur.
Die Kommunikation in DCE zwischen einem Client und dem Server erfolgt ausschließlich über den RPC. Es wird keine Unterscheidung zwischen einem lokalen (d.h. auf demselben Rechner) oder entfernten Aufruf gemacht. Die Server registrieren einen Kommunikationsendpunkt, über den sie RPC-Anforderungen empfangen können. Dieser Kommunikationsendpunkt (Port) und die von einem Server angebotenen Dienste, werden vom sog. Cell Directory Service (CDS) verwaltet. Der CDS liefert einem Client den Server, der ihm den gewünschten Dienst anbietet. Um nun den Port des Servers für die RPC-Kommunikation zu erhalten, wird der rpcd-Prozeß auf dem Rechner des Servers befragt. Dieser Prozeß liefert dem Client den gerade gültigen Port, auf welchem der Server Nachrichten empfangen kann (siehe Abbildung). Mit diesem Konzept lassen sich Server-Prozesse, ohne besondere Konfigurationssmaßnahmen, an einem beliebigen Port starten. Der Server-Prozeß muß nur dem rpcd-Prozeß diesen Port bekanntgeben.

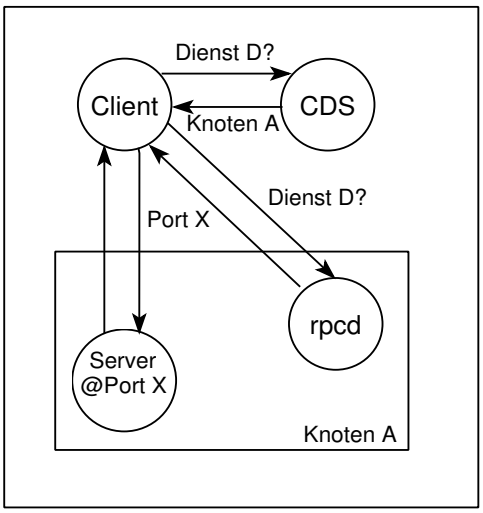

Abbildung 3: Client sucht Dienst D mittels dem CDS und rpcd.

Für die Verwaltung von Benutzern, Maschinen und anderer Ressourcen in einem verteilten System, bietet DCE den Directory Service an. Statt einer lokalen Sammlung von Informationen werden diese Daten in einer hierarchisch organisierten, verteilten Datenbank gehalten. Dies ermöglicht eine schnelle Anpassung der sich ändernden Informationen in einem großen, geographisch verteilten System (z.B. Internet).

Das DCE faßt Rechner, die etwa einer bestimmten Abteilung zugeordnet sind, zu sog. Cells zusammen. Jeder dieser Cells ist ein Cell Directory Server (CDS) für die Verwaltung ihrer Ressourcen (Rechner, Serverprozesse, Namen, etc.) zugeordnet.
Um die Geschwindigkeit für den Zugriff auf Informationen des CDS zu erhöhen, werden die Daten von sog. CDS Clerks "ge-cached". Die Clerks werden auf jedem Rechner im DCE gestartet und arbeiten zwischen dem Client und dem CDS. Die Informationen, die ein Clerk im Laufe der Zeit sammelt, werden in periodischen Abständen abgespeichert, um einen Rechner- oder Prozeßabsturz zu überleben. Bei Inkonsistenzen zwischen den Daten im Clerk und dem CDS, kann der Client den Clerk umgehen und direkt mit dem CDS kommunizieren.

Es besteht die Möglichkeit, die Daten des CDS redundant zu halten. Dazu können mehrere CDS in einer Cell gestartet werden. Dabei bekommt ein CDS das Recht, Änderungen in seiner Datenbank durchzuführen (Master). Die anderen CDS dürfen lediglich lesend auf ihre Daten zugreifen. Um die Datenbanken konsistent zu halten, werden Änderungen am Master sofort an die anderen CDS weitergegeben. Außerdem wird nach einer bestimmten Zeitperiode (z.B. 24 Stunden), die Änderungen der Master-Datenbank auf die übrigen CDS transferiert.

Einzelne Cells werden mittels des Global Directory Service (GDS) untereinander verbunden. Der GDS basiert auf dem CCITT X.500/ISO 9594 [13] Standard. Dieser Standard definiert Schnittstellen und Konzepte für eine verteilte, replizierte Datenbank für die Verwaltung *allgemeiner* Ressourcen. Über den GDS lassen sich weltweite Netzwerkstrukturen für das DCE aufbauen.

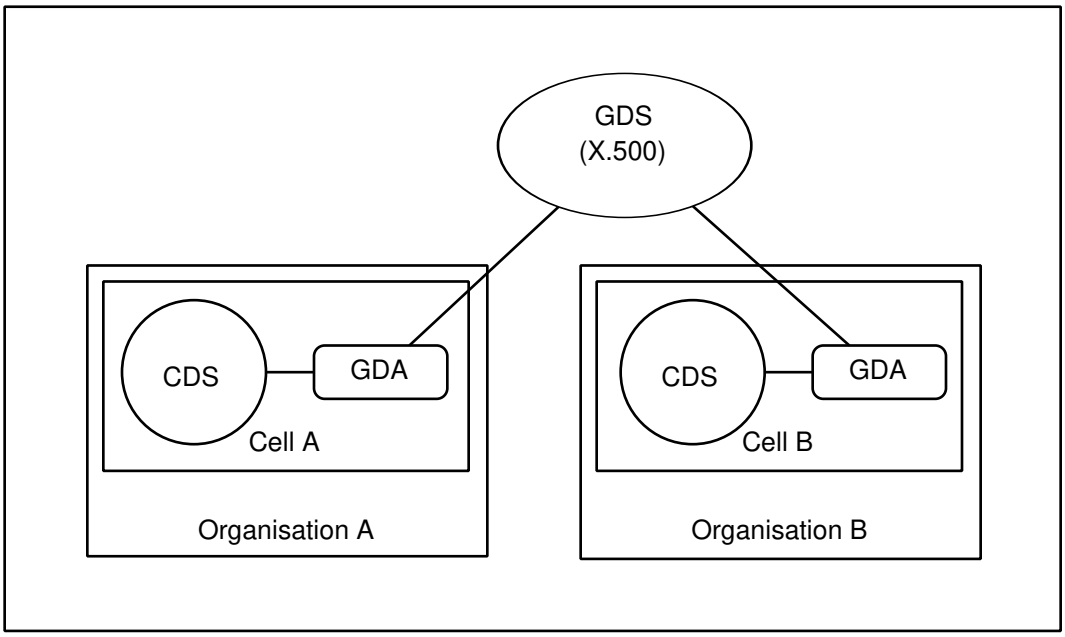

Abbildung 4: Verbindung der Cells über den GDS (Global Directory Service).

Die Kommunikation zwischen CDS und GDS erfolgt über den Global Directory Agent (GDA). Kann der CDS eine Anfrage nach einer Ressource nicht in seiner lokalen Datenbank finden, so kontaktiert er den GDA. Dieser versucht dann, über den GDS die Ressource zu lokalisieren.

Die Global Directory Server können über das in X.500 festgelegte Protokoll miteinander verbunden werden, um so einen weltweiten Directory Service aufzubauen. Die GDS Datenbank ist eine hierarchische, objektorientierte Datenbank.

Beispiel für ein Eintrag im "DCE Global Namespace":

/…/C=US/O=OSF/OU=DCE/my_cell/subsys/my_company/greet_server

Der Erste Teil "/…/C=US/O=OSF/OU=DCE/my_cell" bezeichnet die Cell in X.500 Format (C=Country, O=Organization, ...). Der Rest gibt den Ort eines Servers an. Unter einem Eintrag (wie greet_server) lassen sich dann Werte unter verschiedenen Attributen abspeichern.

Zusammenfassend:
Das DCE benutzt also für das Auffinden von Informationen ein zweistufiges Konzept: den CDS für lokale Anfragen und den GDS für Cell-übergreifende Suchen. Beim GDS wird dabei der X.500 Standard benutzt. Es lassen sich so eine weltweite, verteilte Datenbank für die Ressourcen-Verwaltung aufbauen.

3.2 Camelot and Avalon

Dieses System für verteilte Transaktionen gliedert sich in zwei Teile: Camelot implementiert die Mechanismen für die Synchronisation, Wiederherstellung und Kommunikation, die bei verteilten Transaktionen notwendig sind [11]. Avalon hingegen ist eine Programmiersprache, um zuverlässige, verteilte Anwendungen zu entwickeln (basierend auf einer C++ Klassenbibliothek).

Für den Vergleich zum Kommunikationssystem und Nameservice von APRICOTS, werde ich mich auf die in Camelot und Avalon entsprechenden Komponenten und Mechanismen beschränken.

Camelot baut auf dem Client-Server Prinzip auf. Ein Client ruft mittels eines RPC eine Server-Funktion auf. Der RPC ist dabei Bestandteil der Mach[4]-Interprozeßkommunikation, die auf einem Nachrichtenkonzept (messages) über Ports aufbaut.
Ein RPC wird mittels des Mach Interface Generator (MIG) definiert. Der Generator übersetzt die Implementierung einer Prozedur in spezifische Module (Stub-Code), die vom Client und Server eingebunden werden.
Über den Network Server (Bestandteil von Mach) können Nachrichten (und damit auch die RPCs) über das Netzwerk transportiert werden. Für den Anwender ist es transparent, ob die Interprozeßkommunikation lokal oder über das Netzwerk erfolgt. Der Network Server unterstützt auch das Auffinden eines Server auf dem (lokalen Sub-)Netz mittels broadcast.

Das Beispiel in Abbildung 5 zeigt die Arbeitsweise eines RPC unter Mach: Der Server "Jill" bietet den Dienst `jill_read` an. Der Client "Jack" ruft den Dienst `jill_read` mittels des SERVER_CALL Macro auf. Der Stub-Code für `jill_read` sendet eine Nachricht für die Dienstaktivierung an den "Jill" Server. Auf dem Server wird die Nachricht von der START_SERVER Funktion empfangen und an das lokale `op_jill_read` Unterprogramm weitergeleitet. Das `op_jill_read` Unterprogramm implementiert die `jill_read` Funktion. Wenn dieses lokale Unterprogramm beendet ist, wird das Ergebnis zur Anwendung "Jack" zurückgesandt.

Für die Suche nach Servern die bestimmte Dienste anbieten, werden vom Laufzeitsystem Avalons die Server mit Attributen versehen, um sie näher zu kennzeichnen. Dazu meldet sich ein Server bei dem sog. Catalog Name Server mit einer Attributsliste an (Bsp.: "PRINTSERVER", "HP-LaserJet.10"). Mittels den Attributen kann der Client einen entsprechenden Server lokalisieren und mit ihm Kontakt aufnehmen. Es existiert jedoch keine Möglichkeit, größere Strukturen wie im DCE zu modellieren.

[4]MACH ist ein MicroKernel der Carnegie-Mellon University mit Speicherverwaltungs-, Thread- und Interprozesskommunikationsmechanismen (siehe [11], Seite 13-20]).

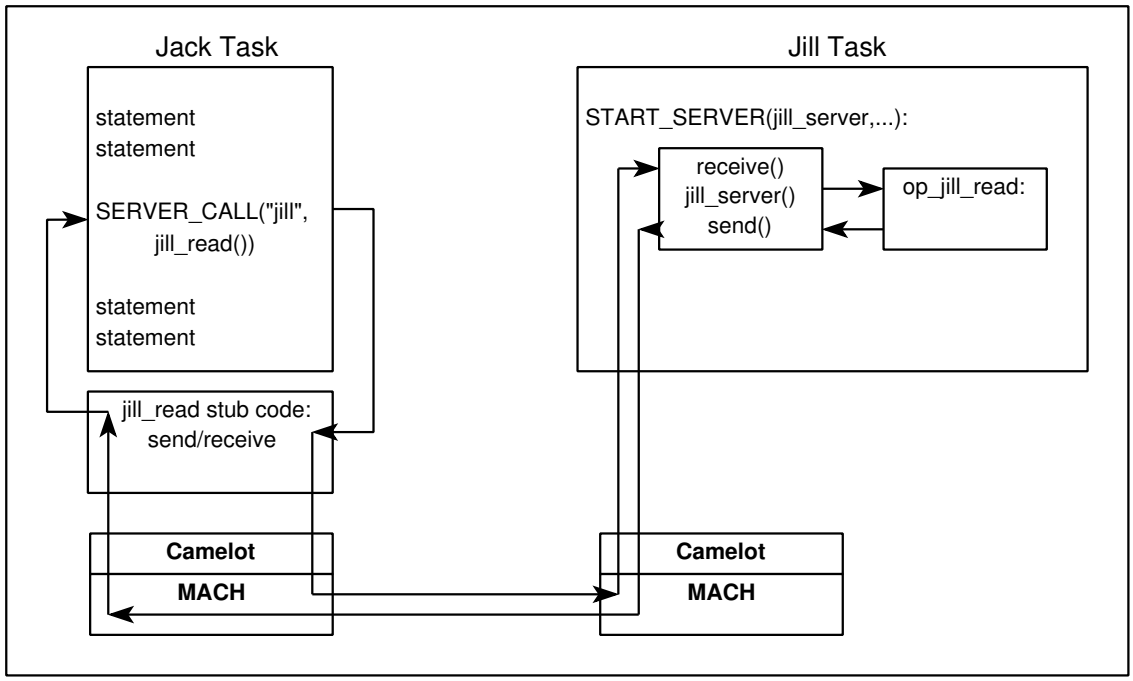

Abbildung 5: Arbeitsweise eines Mach Remote Procedure Calls.

3.3 Domain Name System (DNS)

Das DNS ist ein verteiltes Directory-System. Es verwaltet Informationen über Objekte wie Rechnernamen und zugehörige Internetnummer, Mailboxadressen von Benutzern, etc. Die zentrale Komponente des Systems ist die hierarchische Strukturierung des Domainnamensraumes, welche die dezentrale Verwaltung der Informationen ermöglicht. Eine Domain ist ein Ast dieser baumartigen Struktur. Jedem dieser Äste kann ein Administrator für die Verwaltung zugeordnet werden. Durch diese Baumstruktur wird die Eindeutigkeit von Rechnernamen gewährleistet. Der Administrator einer Domain muß jedoch dafür sorgen, daß die Namen in seiner Domain eindeutig sind. Beispiel eines Namens: rusmv1.rus.uni-stuttgart.de.

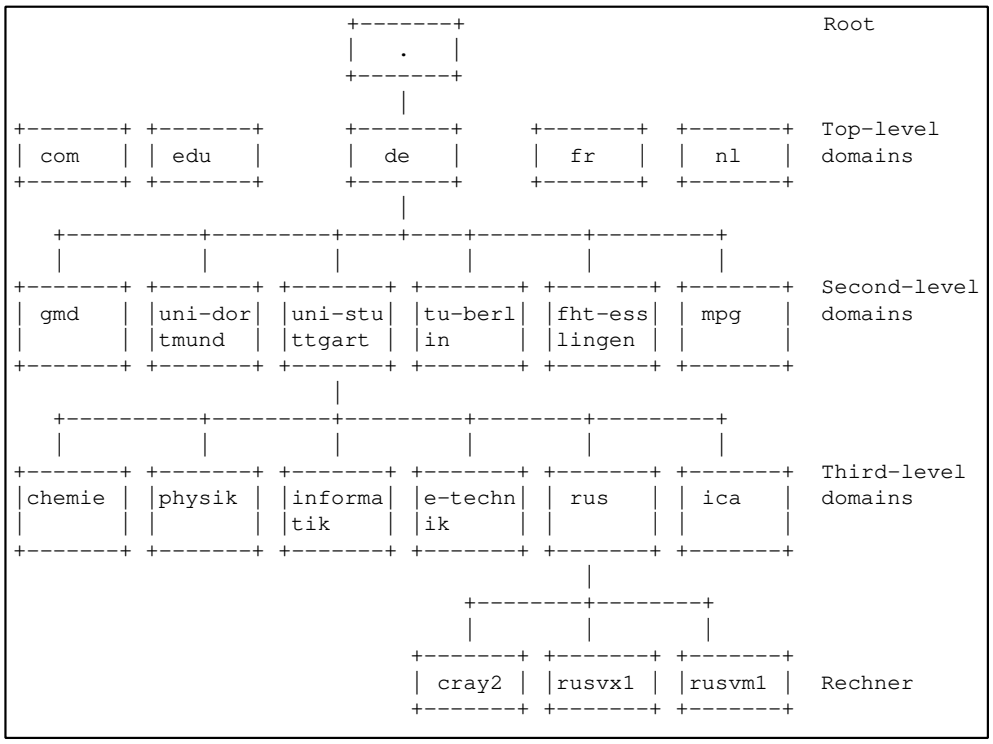

Abbildung 6: Auszug aus dem Domainnamensraum.

Die Verwaltung dieser Informationen wird von dem verteilten Directory-Dienst (Nameserver) unterstützt. Dieser Dienst hat die Funktionalität für die Zuordnung eines Namens zu einer Adresse. Die Daten können den Benutzern und Anwendungen über eine "Resolver"-Software zur Verfügung gestellt werden.
Der Nameserver wartet auf einem sog. "well-known" TCP Port auf Anfragen von anderen Nameservern und Resolvern. Über das Nameserver Protokoll werden die Informationen ausgetauscht.

Im DNS werden die Informationen, wie Zuordnung zwischen Rechnername und Internetnummer, sowie die Nameserver anderer Domains in der "Zonendatei" des Nameservers verwaltet.

Ein Eintrag in der Zonendatei eines Nameservers sieht folgendermaßen aus:

```
ifi.informatik.uni-stuttgart.de  IN  A  129.69.211.1
```

Hier wird dem Rechner `ifi` die Internetnummer 129.69.211.1 zugeordnet. Da der Rechnername und die Internetnummer in der Regel nur bei der Installation der Betriebssoftware in die Zonendatei eingetragen werden muß, läßt sich dies einfach über eine normale Text-Datei im Dateisystem des Nameserver realisieren. Es genügt, mittels eines Editors diese Zonendatei zu bearbeiten, um neue Einträge hinzufügen oder zu ändern (es müssen auch keine Vorkehrungen für gleichzeitiges Bearbeiten getroffen werden, da hier nur dem Systemadministrator erlaubt ist, in diese Datei zu schreiben).
Die Domain welcher der Rechner `ifi` angehört, folgt nach dem Rechnernamen. Im obigen Beispieleintrag ist dies die Domain `informatik`. Die Domain kennzeichnet eine Verwaltungseinheit, die jeweils einem anderen Nameserver und Administrator zugeordnet werden kann. Jede Domain gehört in der Regel einer anderen Verwaltungseinheit an, z.B. Institut oder eine bestimmte Firma. Ein Rechner kann auch mehreren Domains angehören, die durch einen Punkt getrennt werden (z.B. `uni-stuttgart` und `de`). Dies ergibt eine Baumstruktur, in der sich die Rechner bequem verwalten lassen. Die Aneinanderreihung der Domains ergibt dann einen Pfad, nachdem die Internetnummer eines Rechner gesucht wird. Im DNS wird hauptsächlich die Internetnummer über den Rechnernamen und seinen Domains gesucht.

Die Domain `informatik` ist auf einem Rechner im Institut für Informatik gespeichert. Die Suche nach der Internetnummer eines Rechners der Domain `informatik` erfolgt somit sehr schnell, was ja auch der größte Teil der "Namensumsetzung" im Institut ausmacht. Außerdem lassen sich neue Rechnernamen oder Änderungen *innerhalb* dieser Domain sehr leicht durchführen. Es muß kein anderer Nameserver bei der Aufnahme eines neuen Rechnernamen in diese Domain informiert werden. Jede Domain hat das Recht, weitere Unterteilungen in neue Domains vorzunehmen (Subdomains).

Aufgrund der Verknüpfung von Domains ergibt sich die Baumhierarchie. Diese Baumhierachie läßt sich aber nur von "Oben nach Unten" durchsuchen, da es keine Referenz auf den Nameserver einer Domain gibt, die "oberhalb" liegt. Man muß deshalb, bei einer Suche nach einem Rechner der sich in einer Domain befindet die oberhalb der eigenen liegt, immer bei der Wurzel des gesamten Baumes starten. Will man die Internetnummer eines Rechners von einer anderen Universität (z.B. `peanuts.informatik.uni-muenchen.de`) suchen, so muß man den sog. Root-Server der die oberste Domain verwaltet und *alle* darunterliegenden Domains kennt, kontaktieren. In Deutschland existiert dazu ein Root-Server, der die Internertadressen aller Nameserver unterhalb der Domain "`de`" kennt (`uni-stuttgart.de`, `uni-muenchen.de`, ...). Die Client-Software für das DNS bekommt vom Root-Server den Nameserver der nächsten Subdomain (`uni-muenchen`) und kann so durch weitere Anfragen an Nameserver die gesamte Hierarchie in dem sich der Rechner befindet durchlaufen, bis ein Nameserver die Internetnummer des gesuchten Rechners zurückliefert (der Nameserver für die `uni-muenchen` Domain weiß, wie die Internetnummer für den Nameserver der Domain `informatik` heißt und dieser Nameserver muß einen Eintrag für den Rechner `peanuts` haben).

3.4 Bewertung im Hinblick auf die Anforderungen

3.4.1 DCE

Das DCE bietet das Konzept des RPC auf allen Ebenen (lokal und entfernt) an. Dazu wird aber eine eigene Implementierung des RPC vorgenommen. Der RPC muß dabei auf jeden Rechner portiert werden (der verwendete RPC-Mechanismus ist aus dem "Network Computing System" von Digital). Bei der Server-Programmierung wird die Schnittstelle für den Client über den "Interface Definition Language - Compiler" [10] generiert. Bei der Kompilierung des Clients muß diese Schnittstelle mit eingebunden werden.

In APRICOTS hingegen, läßt sich ein Dienst im RM_Call allgemein über seinen Namen und zugehörige Klassen spezifizieren. Weiterhin kann der RM_Call in APRICOTS asynchron ablaufen. Im DCE müßte man sich der Threads bedienen, um während eines RPC weiterarbeiten zu können.

Der Nameservice im DCE ist sehr mächtig. Im lokalen Fall (Cell-Ebene) erfolgt die Umsetzung des Dienstes auf eine physikalische Adresse mittels des CDS. Für die Kommunikation zwischen dem Client und dem Server werden die Ports dynamisch zugeordnet. Der Server muß den Kommunikationsendpunkt nicht selbst verwalten. Die Verwaltung wird jedoch auf den rpcd-Prozeß ausgelagert.

In APRICOTS muß die Mailbox-ID und der Rechnernamen vom Nameservice verwaltet werden. Wegen der Eindeutigkeit der Mailbox-ID auf Rechnerebene kann es jedoch zu keinen Inkonsistenzen bezüglich doppelter IDs kommen.
Für den entfernten Fall wird ein Briefträger beauftragt, Nachrichten in die Mailbox einer Komponente zu transportierten.

Über den X.500 Standard lassen sich im DCE sehr große Strukturen modellieren. Es wird die Replikation der Datenbank unterstützt. Es existieren Tools für die Verwaltung und Konfiguration der Ressourcen.
Mittels X.500 können beliebige Ressourcen verwaltet werden, wodurch es unabhängig vom DCE eingesetzt werden kann. Beispielsweise kann mit X.500 auch die Zuordnung von Internet-Rechnernamen zu den IP-Adressen vorgenommen werden (bisher ist das Domain Name System dafür im Einsatz). Wegen die Einbeziehung von X.500 im DCE, mußte kein eigenes Konzept erstellt werden und bietet einen mächtigen Mechanismus für die Ressourcenverwaltung.
Problematisch ist jedoch die Verbreitung von X.500. Zur Zeit existieren wenig vollständige Implementationen. Dies wird sich aber in naher Zukunft wohl ändern, da X.500 zum ISO-Standard erhoben ist.

3.4.2 Camelot und Avalon

Camelot bietet wie das DCE einen transparenten RPC an. Es wird nicht zwischen dem lokalen und dem entfernten Aufruf unterschieden. Das Kommunikationssystem von Camelot basiert auf dem Mach-Kernel (in [11] wird zwar die Portierbarkeit des Systems angesprochen, doch müßte man wegen der Interprozeßkommunikation den gesamten Kernel auf ein anderes System übertragen). Der Transport von Nachrichten über die Ports und Messages funktioniert zwischen Threads, Tasks und über das Netzwerk. Man muß keine Sonderfälle wie prozeßlokale, rechnerlokale oder entfernte Aufrufe betrachten. Dadurch vereinfacht sich Kommunikationssystem.

Der Nameservice wird durch eine Avalon-Klasse in C++ übernommen. Er ist nicht in Camelot integriert. Über die Attributsliste können Server beliebig klassifiziert werden. Für die Datenhaltung (wie auch für die Zustände der Server und des Transaktionslog) wird ein "wiederherstellbarer" Speicher (recoverable storage) implementiert. Er läßt sich so im Falle eines Systemabsturzes ohne Verluste wieder aufsetzen.

3.4.3 Domain Name System

Das DNS wurde vor allem für relativ statische Informationseinheiten entwickelt: Die Zuordnung eines Rechnernamens zu einer Internetnummer wird in der Regel nur bei Neuinstallation der Betriebssoftware vorgenommen. Danach wird sich diese Information nur selten ändern. Es ist deshalb nicht notwendig, besondere Speicherungsmethoden für diese Informationen vorzunehmen; es genügt eine einfache Textdatei.
Die Suche nach einer Internetnummer erfolgt immer von oben nach unten im Baum. Es kann je nach Geschwindigkeit der Netzwerkleitungen sehr lange dauern, bis alle Nameserver der im Rechnernamen enthaltenen Domains kontaktiert wurden. Dies ist aufgrund der Statik der Information durchaus akzeptabel, da die gefundene Information in einem Cache gespeichert wird und nur "langsam" veraltet.

In APRICOTS dagegen, können sich die physikalischen Adressen der Dienste im Laufe der Zeit durch Migration und Neustart dynamisch ändern. Die Namensumsetzung sollte deshalb relativ schnell ablaufen. Es wird nicht vorteilhaft sein, eine Hierarchie wie im DNS aufzubauen, sondern bestimmte Gruppierungen eher vollständig zu vermaschen.

4 Architektur für die Kommunikation

Das folgende Kapitel beschreibt den Aufbau des Nameservice und des Briefträgers von APRICOTS. Die Bewertung der verteilten Systeme aus Kapitel 3 und Anforderungen an das Kommunikationssystem aus Kapitel 2 bilden die Grundlage für die Architektur der Kommunikation in APRICOTS. Es wurde versucht, bestimmte Mechanismen und Funktionalität, aus den dort vorgestellten Konzepten einzubringen.

4.1 Inselgruppen

Eine Inselgruppe ist die Zusammenfassung von einzelnen Knotenrechnern, die über das Netzwerk miteinander verbunden sind. Die Inselgruppe bildet eine logische Einheit für die Teilnahme an einem APRICOTS-Verbund. Über eine Inselgruppe kann eine physikalische Struktur (z.B. Abteilung, Firma) oder eine logische Struktur (z.B. bestimmte Rechnerknoten einer Firma an verschiedenen Orten) modelliert werden.

Jede Inselgruppe hat Zugriff auf einen gemeinsamen, stabilen Datenbereich, der Informationen über die angebotenen Dienste und die zugehörigen physikalischen Adressen speichert. Die Komponenten tragen bei ihrem Start die von ihnen angebotenen Dienste ein. Jeder Knoten hat dann die Möglichkeit, nach Diensten in seiner Inselgruppe zu suchen und in diese Anspruch zu nehmen.

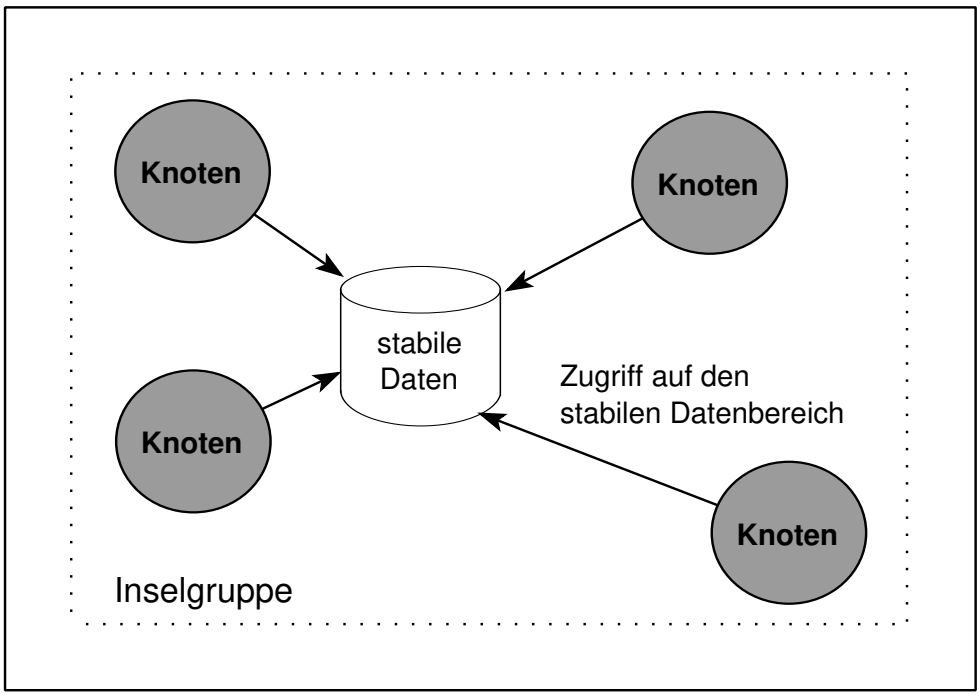

Abbildung 7: Aufbau einer Inselgruppe.

Das Zusammenfassen der Knoten zu einer Inselgruppe und der direkte Zugriff auf einen stabilen Datenbereich haben folgende Vorteile:

- Die Zuordnung zwischen den Diensten und ihrer physikalischen Adresse können sich im laufenden System mit der Zeit ändern: Komponenten werden auf einen anderen Rechner migriert, d.h. es ändert sich die physikalische Adresse oder es werden neue Komponenten im System gestartet. Dieses Verhalten erfordert einen kurzen Kommunikationspfad zwischen den Komponenten und dem stabilen Datenbereich, der die Informationen speichert. Über den direkten Zugriff auf den stabilen Datenbereich werden die Systemressourcen (Rechner, Netzwerk, etc.) möglichst wenig belastet.

- Die Änderung von Informationen wird für alle Komponenten der Inselgruppe *sofort* sichtbar. Eine Lastbalancierung kann so direkt auf bestimmte Komponenten Einfluß nehmen. Zum Beispiel können Anforderungen von Dienstaktivierungen einer überlasteten Komponente, auf eine relativ "freie" Komponente die denselben Dienst anbietet, übertragen werden.

- Eine Inselgruppe macht eine Konfiguration überschaubar. Es lassen sich einzelne Administratoren für die Verwaltung und Pflege der Inselgruppen zuordnen.

- Die Fehlertoleranz des Systems wird durch den stabilen Datenbereich gewährleistet. Die Informationen gehen bei Ausfall eines Rechners nicht verloren.

- In den stabilen Datenbereich lassen sich weitere Informationen über den Zustand des Verbundes, wie Anzahl der Fehlverbindungen, inaktive Komponenten, etc. eintragen und der Lastbalancierung zuführen.

Der stabile Datenbereich kann über eine Datenbank oder ein stabiles Filesystem[5] realisiert werden. Dieser Datenbereich ist in APRICOTS zusätzlich für die Speicherung weiterer Informationen über den ConTract-Ablauf notwendig.

Über eine geeignete Schnittstelle für den Zugriff auf den stabilen Datenbereich lassen sich verschiedene Implementierungen dafür einbinden.

[5] vergl. "Recoverable Storage" in [11], Kapitel 8

4.2 Der Nameserver

Die einzelnen Inselgruppen lassen sich mittels eines "Nameservers" untereinander verbinden. Über den Nameserver besteht die Möglichkeit, die Dienste einer Komponente auf einer anderen Inselgruppe zu nutzen. Es lassen sich so sehr viel größere Strukturen modellieren. Beispielsweise kann bei einer kombinierten Flug- und Hotelbuchung, je eine Inselgruppe für die Fluggesellschaft und für die Hotelagentur existieren. Die Buchung des Hotels kann über die eigene Inselgruppe erfolgen; für die Flugbuchung wird ein Dienst in der Inselgruppe der Fluggesellschaft aktiviert.

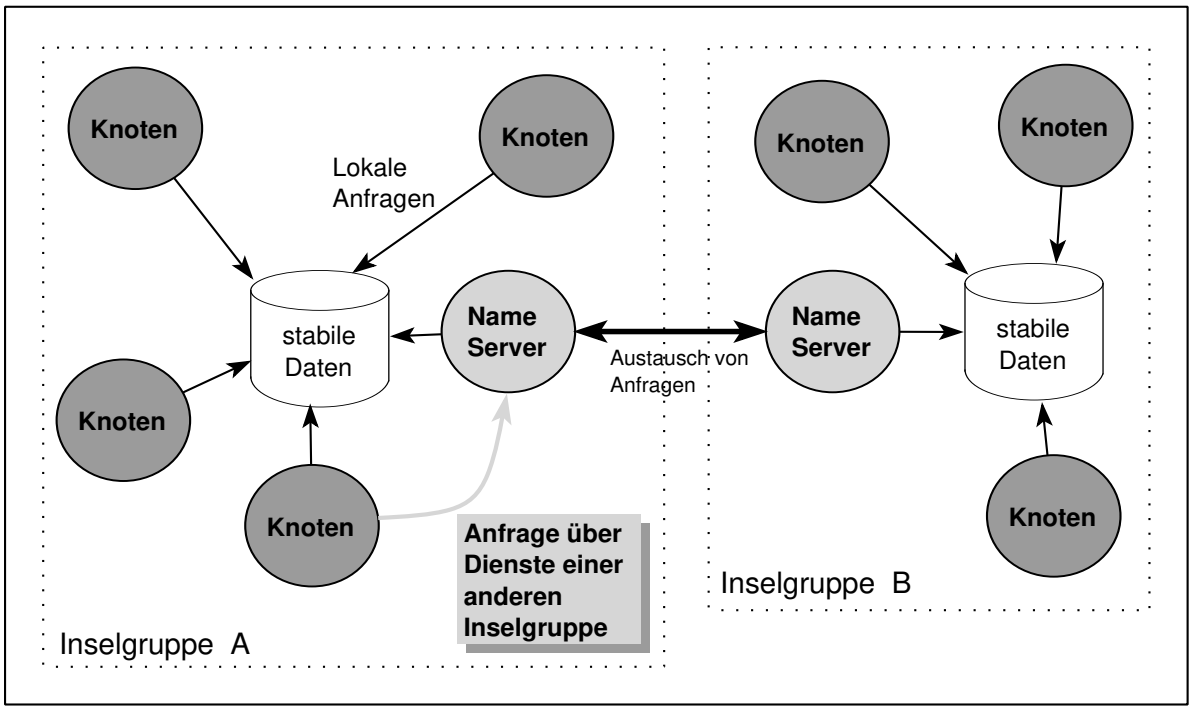

Abbildung 8: Verbindung der Inselgruppen über den Nameserver.

Findet ein Knoten einen gewünschten Dienst nicht im stabilen Datenbereich, so kann er den Nameserver seiner Inselgruppe beauftragen, diesen Dienst auf einer anderen Inselgruppe ausfindig zu machen. Die Nameserver haben dazu Informationen über die Nameserver anderer Inselgruppen. Sie bauen eine Verbindung zu den ihnen bekannten Nameservern auf und beauftragen diese, auf dem stabilen Datenbreich ihrer Inselgruppe nach einem Dienst zu suchen. Ist ein Dienst mit den spezifizierten Eigenschaften gefunden worden, so wird die physikalische Adresse über die Nameserver an den anfragenden Knoten übermittelt. Der Knoten ist damit in der Lage, Nachrichten an entfernten Knoten zu senden.

Die Nameserver haben in dem stabilen Datenbereich der Inselgruppe eine Liste mit den Adressen der Nameserver, die mit ihnen verbunden sind. Die Anzahl der verbundenen Inselgruppen ist damit von der Anzahl der Nameserver abhängig, die in dieser Liste gespeichert sind.

Das Konzept der Nameserver für die Verbindung zwischen den Insel basiert auf folgenden Überlegungen:

- Die Nameserver ermöglichen die Suche nach Diensten auf anderen Inselgruppen und vergrößern so den "Namensraum".
  Dieses Konzept ist den Global Directory Servern (GDS) vom DCE ähnlich. Auch dort lassen sich kleinere Strukturen (Cells) über den GDS zu größeren Strukturen zusammenfassen.

- Durch das Zusammenfassen der Knoten zu Inselgruppen und die Verbindung der Inselgruppen über die Nameserver lassen sich reale Anwendungssysteme in APRICOTS modellieren. In der Regel werden die meisten Dienstanfragen einer Anwendung auf einer begrenzten Anzahl von Knoten stattfinden und nur gelegentlich auf Dienste in anderen Inselgruppen zugreifen.
  Die lokale Suche nach einem Dienst ist somit immer die schnellste Möglichkeit der Namensumsetzung. Eine Dienstsuche auf einer anderen Inselgruppe kann je nach Netzwerktopologie und vorhandene Systemressourcen dagegen länger dauern. Der lokale Fall wird immer bevorzugt.

- Die Aufallsicherheit des Nameservers läßt sich durch redundante Nameserver erhöhen. Dazu können in einer Inselgruppe mehrere Nameserver gestartet werden. Fällt ein Nameserver aus, so kann einer der anderen Nameserver dessen Funktion übernehmen. Es ist aber immer nur ein Nameserver zu einem bestimmten Zeitpunkt aktiv.

- Über die Inselgruppe kann ein einfacher Zugriffsschutz auf die Dienste realisiert werden: die Nameserver können z.B. nur Anfragen von anderen Nameservern entgegennehmen, auf die auch sie Zugriff haben.

- Die Nameserver können neben den Anfragen nach Diensten, auch ihre Liste mit den Nameservern der verbundenen Inselgruppen transferieren. Der Namensraum eines Nameservers läßt sich dadurch vergrößern. Es kann so eine N:N Verbindung zwischen den Inselgruppen *dynamisch* vorgenommen werden.
  Beim Domain Name System (DNS) ist z.B. nur ein Baum mit den Domainnamen vorhanden, der von oben nach unten abgefragt werden muß. Je nach Stufenzahl des Baumes müssen wieder andere Nameserver kontaktiert werden. Dies kann u.U. relativ viel Zeit beanspruchen, falls die Netzwerkverbindung ausgelastet ist oder langsame Netzwerkleitungen benutzt werden müssen (Beispielsweise ausgelastete Überseeleitungen)
  Der APRICOTS Nameserver erlaubt dagegen die direkte Suche nach einem Dienst, ohne über einen zusätzlich dazwischengeschalteten Nameserver gehen zu müssen.

Der Zugriff auf die Informationen über lokale Dienste und Nameserver erfolgt immer auf den stabilen Datenbereich. Die Nameserver müssen deshalb selbst keine Informationen speichern. Fällt der aktuelle Nameserver aus, so kann ein anderer Nameserver der Inselgruppe ohne Probleme wiederaufsetzen.
Es lassen sich so die Schwierigkeiten, wie sie bei redundant gehaltenen Daten der Fall sind,

vermeiden. Zum Beispiel werden im DCE Änderungen nach einer gewissen Zeit oder aber sofort auf die anderen "Backup"-Nameserver verteilt. Dies belastet die Systemressourcen bei umfangreichen oder häufigen Änderungsanforderungen.

Weiterhin ermöglich dieses Konzept ein schnelles reagieren auf Netzwerkfehler und Ausfälle von Rechnern: wenn dies in dem zentralen Datenbereich der Inselgruppe gespeichert wird, so haben alle angeschlossenen Komponenten zur gleichen Zeit die Information über einen bestimmten Zustand der Rechnerknoten. Beispielsweise können Fehler (Verbindungsabbrüche, Zeitüberschreitungen bei der Dienstsuche, …) mit anderen Nameservern dort vermerkt werden. Die Lastbalancierung kann mit diesem Wissen bestimmte Verbindungen streichen oder umlenken.

4.3 Briefträger

Der Briefträger ist für den Nachrichtenaustausch zwischen Komponenten, die sich auf entfernten Rechnern befinden zuständig und arbeitet nach der Client-Server Architektur. Der Briefträger ist dabei der Server und das Kommunikationssystem fungiert als Client.
Wird in einem der RM_call Aufrufe (synchron oder asynchron) festgestellt, daß der Dienst weder prozeßlokal zur Komponente, noch auf demselben Rechner zu finden ist, nimmt das Kommunikationssystem Kontakt zum Briefträger des entfernten Knoten auf und überträgt an ihn die Nachricht über eine Dienstaktivierung. Der Briefträger transportiert dann die Nachricht anhand der Mailbox-ID in die Warteschlange der Empfängerkomponente. War die Aktion erfolgreich, wird die Verbindung abgebaut.
Jeder Rechner der an APRICOTS teilnimmt, muß deshalb einen Briefträger-Prozeß für die Kommunikation mit anderen Knoten starten.

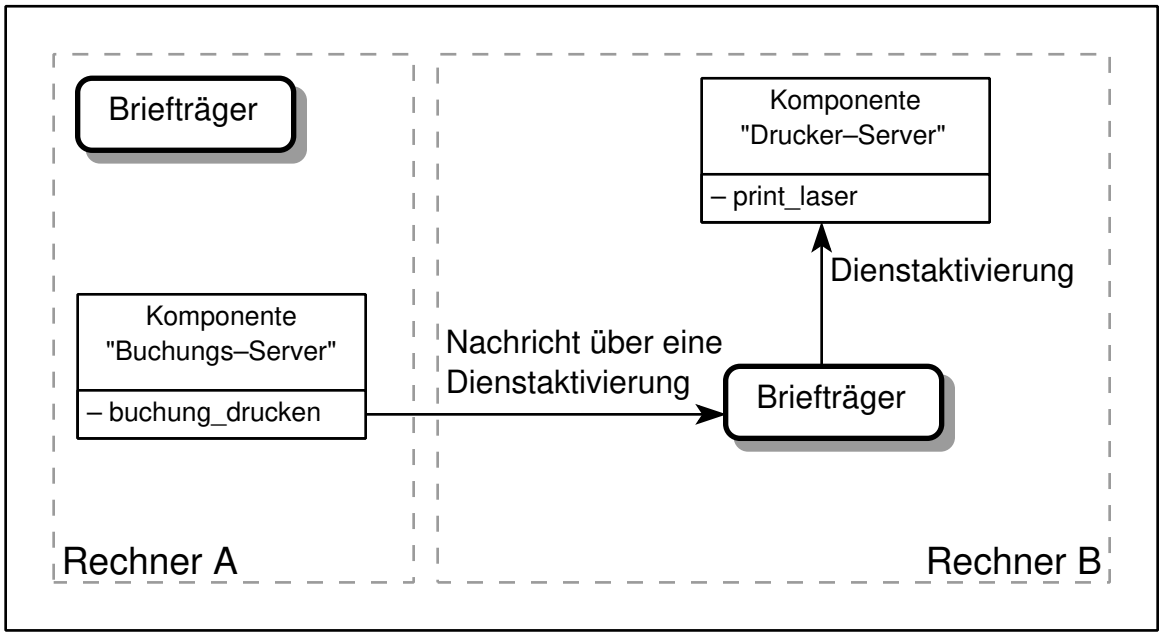

Abbildung 9: Schema der Kommunikation zwischen entfernten Rechnern.

In der obigen Abbildung führt der Dienst "buchung_drucken" auf Rechner A, einen RM_call für den Dienst "print_laser" aus. Über den Nameservice bekommt das Kommunikationssystem die physikalische Adresse, daß die Komponente "Drucker-Server" auf Rechner B den Dienst "print_laser" anbietet.
Das Kommunikationssystem nimmt Verbindung mit dem Briefträger auf Rechner B auf. Dann wird die Nachricht für die Dienstaktivierung und die Argumente für den Dienst "print_laser" zum Briefträger übertragen. Die Verbindung bleibt solange bestehen, bis der Briefträger auf Rechner B die Nachricht erfolgreich in die Mailbox der Zielkomponente abgelegt hat oder ein Fehler aufgetreten ist (Bsp.: Netzwerkfehler, Mailboxüberlauf, …). Danach wird die Verbindung zwischen dem Kommunikationssystem und dem Briefträger auf Rechner B abgebaut.

Die obige Architektur wurde aufgrund der nachfolgenden Anforderungen des Briefträgers entworfen:

- *Ausfallsicherheit (fail-safe)*
  Ein Absturz des Rechners auf dem sich der Briefträger befindet oder ein Netzwerkfehler der das ordnungsgemäße Weiterleiten von Nachrichten verhindert, darf den ConTract-Ablauf weder stören noch beeinflussen. Der Briefträger muß deshalb ohne internen Status arbeiten, der bei einem solchen Fehler verloren gehen würde. Fällt ein Briefträger aus, so ist zwar die Kommunikation mit diesem Rechner unterbrochen, aber er kann einfach durch Neustart wieder aktiviert werden. Andere Komponenten bekommen (ausgenommen von einer verzögerten Abarbeitung ihrer Nachrichten) nichts von dem Ausfall mit.
  Der Briefträger geht somit im Fehlerfall in einen, dem fail-safe entsprechenden, "sicheren" Zustand seiner Terminierung über.

- *Persistenz und Verklemmungsfreiheit*
  Nachrichten dürfern nicht verloren gehen oder dupliziert werden. Der Briefträger darf deshalb die Nachrichten nicht selbst puffern, sondern muß ankommende Nachrichten sofort in die Mailbox der adressierten Komponente anhängen und dem Absender das Ergebnis dieser Zustellung quittieren.
  Ist die Nachricht in der Mailbox der Komponente, so befindet sie sich in einem stabilen Datenspeicher und kann nicht verloren gehen.

- *Migration*
  Dienste können von der Lastbalancierung auf andere Rechnerknoten und Komponenten verlagert werden. Die in der Mailbox enthaltenen Nachrichten müssen dann storniert und zurückgeliefert werden, um sie in die Mailbox einer anderen Komponente anzuhängen.
  Der Briefträger muß eine Schnittstelle für das Überprüfen und Stornieren von Nachrichten in der Mailbox einer Komponente zur Verfügung stellen.

4.4 Rechnerübergreifende Kommunikation

Der Nameserver und der Briefträger arbeiten beide nach einer Client-Server Architektur. Für die Kommunikation zum Nameserver- und Briefträgerprozeß werden in das Kommunikationssystem "Client"-Funktionen eingebunden.

Die Realisierung der entfernten Kommunikation zum Briefträger und Nameserver basiert auf TCP/IP mit dem BSD4.3 socket-Interface.

Die TCP/IP (Transmission Control Protocol/Internet Protocol) Familie umfaßt auf der Transportschicht (OSI-Ebene 4), die Protokolle TCP und UDP (User Datagram Protocol).
TCP ist ein verbindungsorientiertes Protokoll, d.h. es übernimmt den Auf- und Abbau der Verbindung zwischen den Prozessen, liefert die Datenpakete in der richtigen Reihenfolge an und sorgt für einen zuverlässigen, bidirektionale Datenstom zwischen den Verbindungspartnern.
Das UDP-Protokoll dagegen, bietet einen verbindungslosen Paketdienst an. Der Anwendungsprozeß muß selbst für die korrekte Reihenfolge der Datenpakete und die Fehlerbehandlung (doppelte Pakete, timeouts, etc.) entsprechende Algorithmen und Prozeduren implementieren.
Der Anwendungsprozeß (OSI-Ebenen 5-7) wählt diese Protokolle aus und trifft spezifische Vorkehrungen für jedes Protokoll. Beide Protokolle sind für LAN (Local Area Network) und WAN (Wide Area Network) ausgelegt.

Ein Grund für die Verwendung der TCP/IP Protokollfamilie ist die weite Verbreitung dieses Standards. Es existieren fast für jeden Rechner und Betriebssystem Bibliotheken[6]. Dies erleichtert die Portierung auf andere Rechnerplattformen. Weiterhin wird die vorhandene Netzstruktur des IPVR mit der Internetadressierung direkt unterstützt, ohne das Umsetzungsmechansismen benutzt werden müssen (Bsp.: SNA nach TCP/IP).

Für die Implementierung der entfernten Kommunikation wurde wegen der Zuverlässigkeit und der einfachen Handhabung eines Datenstroms, TCP ausgewählt. UDP arbeitet zwar schneller als TCP, doch würde die Ausprogrammierung der notwendigen Zuverlässigkeit den Geschwindigkeitsvorteil zu TCP wieder einschränken[7].
Der Verbindungsaufbau und die Verbindungsentgegnahme ist mit TCP einfach zu programmieren. Die Verbindung wird als Datenstrom betrieben und es läßt sich deshalb mittels den Systemaufrufen read() und write() bequem auf die Daten zugreifen. Es muß z.B. kein aufwendiger Mechanismus für das Einsammeln und Überwachen von Paketen wie bei UDP implementiert werden.

---

[6] APRICOTS wird derzeit unter UNIX (Ultrix und SUN-OS) und VAX/VMS entwickelt. Beide Betriebssysteme haben Unterstützung für sockets und TCP/IP.

[7] siehe [5], Kapitel 12: "Trivial File Transfer Protocol" und [19] über den Vergleich verschiedener RPC Implementationen auf TCP und UDP Basis.

Weitere Möglichkeiten für die Kommunikation über ein Netzwerk sind z.B. SNA (Systems Network Architecture) von IBM, das OSI-Model und der SUN-RPC.
Da SNA über keine einheitliche Schnittstelle verfügt [5] und außerdem kaum am IPVR verbreitet ist (wenige RS/6000 und keine IBM-Mainframes), scheidet es als Kommunikationsprotokoll aus.
Im OSI-Modell sind die einzelnen Schichten noch im Entwicklungsstadium (siehe ISODE - ISO Developement Paket [20]), so daß eine Portierung des Briefträgers und Nameservers auf andere Rechner schwierig wäre
Auch der SUN-RPC Mechanismus kommt für die entfernte Kommunikation nicht in Frage, da auch hier die Portabilität eingeschränkt werden würde, weil dieser nicht auf jeder Plattform standardmäßig im Lieferumfang enthalten ist (der SUN-RPC unterscheidet sich zum Beispiel vom Apollo-RPC auf Apollo Workstations).

Für die rechnerübergreifende Kommunikation ist es notwendig, auf Verbindungsabbrüche und Zeitüberschreitungen (timeouts) zwischen dem Client und Server zu reagieren. Dabei dürfen der Server und der Client nicht einfach abstürzen, sondern müssen diese Fehler zurückliefern bzw. ignorieren können. Zu diesem Zweck wird die Verbindung mittels eines timeouts überwacht. Bei der Überschreitung dieser Zeitspanne wird die Verbindung zwischen den beiden Kommunikationspartnern abbgebrochen. Der Aufrufer der entsprechenden Kommunikationsroutine erhält einen Fehler zurückgeliefert. Es wird nicht versucht, die Verbindung erneut aufzubauen (z.B. über mehrmaliges, erneutes Versuchen). Vielmehr soll die Lastbalancierung einen neuen Kommunikationsversuch starten oder eine andere Komponente auswählen, die sich auf einer "sicheren" Verbindungsmöglichkeit befindet.

Die verwendeten Kommunikationsfunktionen für den Nachrichtentransport über das Netzwerk sollen möglichst auch von anderen Teilen APRICOTS genutzt werden können (Briefträger, Nameservice, Lastbalancierung, ...). Es bietet sich deshalb an, ein allgemeines Modul für den Datentransport über TCP/IP mit Fehlerbehandlungsmöglichkeiten zu implementieren. Dies vereinfacht weiterhin die Portierung auf andere Systeme.

5 Realisierung

Das folgende Kapitel beschreibt die Realisierung des Nameservice und des Briefträgers. Im ersten Unterkapitel wird zuvor auf die Implementierung der verwendeten Transportmechanismen über das Netzwerk eingegangen.
Die Implementierungen wurden unter Ultrix, SunOS und VMS vorgenommen. Es wurde versucht, sich möglichst nahe an den POSIX Standard zu halten. Die Kommunikation über TCP/IP und Sockets ist jedoch noch nicht in den POSIX Standard aufgenommen.

5.1 Netzwerkkommunikation

5.1.1 Realisierung des Client und Server

Um den Briefträger- oder Nameserverprozeß eindeutig im Netz zu identifizieren, ist es notwendig, den Server-Prozeß auf einem sog. *well-known* Port zu starteten. Über diesen Port kann der Client die Verbindung mit jedem beliebigen Rechner, auf dem ein Server läuft, aufnehmen. Es muß darauf geachtet werden, daß der *well-known* Port nicht mit bereits belegten Ports in Konflikt gerät: z.B. sind Ports 1 bis 1023 für Systemdienste reserviert. Es wäre erstrebenswert, für den Briefträger und Nameserver sich einen bestimmten Port registrieren zu lassen.

Der Ablauf einer Client-Server Verbindung mit dem Socket-Interface läßt sich mit wenigen Aufrufen implementieren nach [5]:

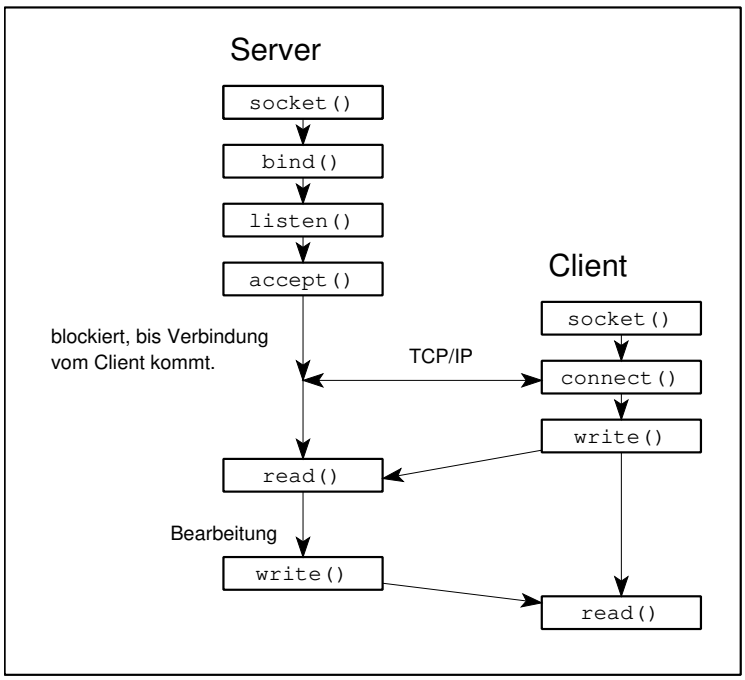

Abbildung 10: Socketfunktionen für eine TCP/IP-Verbindung.

Der Server wählt über den `socket`-Aufruf das Protokoll (TCP oder UDP) und die Adreßfamilie (UNIX Interprozeßkommunikation auf demselben Rechner oder Internet für den Netzwerktransport) aus.
Mittels `bind` wird dem Socket der *well-known* Port für die Kommunikation mittgeteilt. Auf diesem Port empfängt der Server die Verbindungen von den Clients.
Der `listen`-Aufruf spezifiert die Anzahl der eingehenden Verbindungsanforderungen, die gepuffert werden können (in der Regel 5 Anforderungen).
Der `accept`-Aufruf blockiert den Server, bis ein Client mittels `connect` den Server anspricht. Wurde die Verbindung mit dem Client aufgebaut, so liefert `accept` einen neuen socket-Deskriptor zurück. Über diesen neuen socket-Deskriptor wird eine aktuelle Verbindung gekennzeichnet. Der alte socket-Deskriptor dient weiterhin für eingehende Verbindungsanforderungen.

Der Client-Prozeß gibt im `connect`-Aufruf die Internetadresse und den Port an, auf dem der Server zu erreichen ist. Kommt dieser Aufruf ohne Fehler zurück, so steht die Verbindung zwischen Client und Server.
Danach wird über die `read`- und `write`-Systemaufrufe der Datentransport durchgeführt.

5.1.2 Unterprogramme für die Netzwerkkommunikation

Die Datenübertragung mittels TCP/IP ist in Unterprogramme organisiert, die sich wiederverwenden lassen. Die folgenden Prozeduren kapseln den Zugriff auf den Datenstrom, wie er im obigen Abschnitt beschrieben ist. Die Kommunikation zwischen Client und Server basiert auf dem Senden und Empfangen von Nachrichten mit fester Größe.
Für den Client und Server existieren die Unterprogramme `receive_message`, für den Empfang einer Nachricht und `send_message`, für das Senden einer Nachricht. Beide Unterprogramme arbeiten auf einer bereits "stehenden" Verbindung über den socket-Deskriptor. Weitere Parameter der `send_message` Prozedur sind ein Zeiger auf den Speicherblock in dem sich die Nachricht befindet und die Länge der Nachricht. Die `receive_message` Prozedur blockiert, bis eine Nachricht eingegangen ist. War die Datenübertragung erfolgreich, so wird ein Zeiger und die Größe eines allozierten Speicherbereichs, in dem die Nachricht abgelegt ist, zurückgegeben.
In den Prozeduren `receive_message` und `send_message` wird die Fehlerbehandlung bei einem Leitungsabbruch und die Zeitüberschreitung einer Verbindung behandelt (siehe folgenden Abschnitt).

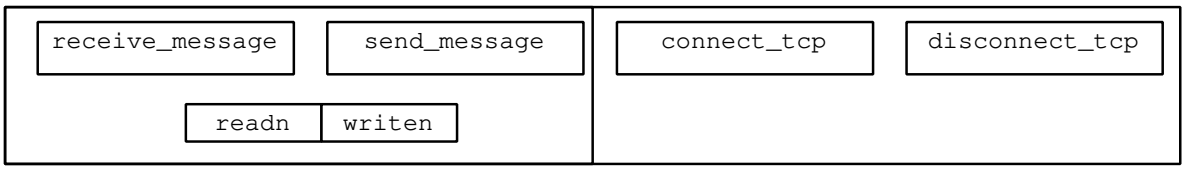

Abbildung 11: Unterprogramme für die Kommunikation über das Netzwerk.

Die `receive_message` und `send_message` Prozeduren benutzen die Unterprogramme `readn` und `writen` für die direkte Kommunikation mit dem Datenstrom. Dies ist notwendig, da die Größe der Sende- und Empfangspuffer des TCP/IP Moduls jeweils eine Konstante ist. Diese Puffer sind in der Regel auf 4096 Bytes ausgelegt (können aber mittels `setsockopt` in bestimmten Grenzen geändert werden). Ein `read`- und `write`-Systemaufruf liefert deshalb immer nur Blöcke in dieser bestimmten Puffergröße zurück. Will man eine Nachricht die größer als die Puffergröße ist über das Netz transportieren, so muß deshalb solange `read` oder `write` aufgerufen werden, bis der gesamte Datenblock übertragen wurde.
Die Prozedur `connect_tcp` initialisiert einen socket-Deskriptor, die `disconnect_tcp` Prozedur gibt ihn wieder frei.

5.1.3 Zeitüberschreitung und Fehlerbehandlung

Für die Ausfallsicherheit der Server-Prozesse und des Clients, der den Kontakt aufnimmt, müssen Netzwerkfehler und spontane Verbindungsabbrüche abgefangen werden. Außerdem darf die Datenübetragung nur eine bestimmte Zeit in Anspruch nehmen und den Server nicht komplett blockieren, sonst kann es zu Verklemmungen im Kontrollfluß kommen.

Der bevorzugte Mechanismus unter UNIX bzw. der "Standard C Bibliothek" um bestimmte interne oder externe Ereignisse dem Prozeß mitzuteilen, sind die "software interrupts" (signals). Signals werden asynchron erzeugt, d.h. ein Prozeß weiß nicht vorher, zu welcher Zeit der Interrupt kommt. Ein Prozeß kann auf ein bestimmtes signal Einfluß nehmen, in dem er eine Funktion angibt, die auf dieses signal reagiert und bestimmte Aktionen durchführt ("signal handler").

Die Netzwerkfunktion `send_message` und `receive_message` fangen in ihren signal handler die signals `SIGPIPE` und `SIGALRM` ab. Das signal `SIGPIPE` wird bei einem Verbindungsabbruch auf dem socket aktiviert; `SIGALRM` wird von dem `alarm`-Systemaufruf generiert. Das signal `SIGALRM` wird nach Ablauf der im Argument des `alarm`-Aufrufes bestimmten Zeitspanne erzeugt (derzeit 60 Sekunden).

Die Signal-Behandlung ist unter UNIX nicht einheitlich definiert. Je nach UNIX-Derivat werden bestimmte Konventionen oder Annahmen über die Behandlung eines signals getroffen. So können sog. race-conditions bei der Signal-Behandlung auftreten (Bsp.: ein weiteres signal tritt in einem signal handler auf). Außerdem werden z.B. in BSD4.3, bestimmte Systemaufrufe, wie `read` und `write`, nach einer Unterbrechung durch ein signal *neu* gestartet. Man befindet sich nach der Abarbeitung der signal handler - Funktion wieder im Systemaufruf. Es ist deshalb notwendig, mittels eines `long_jmp()`[8] Aufrufs aus dem signal handler auf eine vorher festgelegte Stelle im Programm zu springen, um den Neustart eines Systemaufrufs zu umgehen. Dort läßt sich dann das signal weitervearbeiten.

[8] **longjmp** stellt den Zustand her, der beim letzten Aufruf von **setjmp** gespeichert wurde.
Die Ausführung geht so weiter, als ob die Funktion **setjmp** gerade ausgeführt wurde und einen von Null verschiedenen Rückgabewert geliefert hat. Es lassen sich so globale Sprünge realisieren.

Die "sichere" Signal-Behandlung im Hinblick auf race-conditions ist erst unter POSIX eindeutig definiert.

5.2 Der Briefträger

Der Briefträger läuft als eigenständiger Prozeß auf einem Rechnerknoten. Es wird jeweils eine Verbindung nach der anderen abgearbeitet, der Server arbeitet also iterativ. Dieser Ansatz wurde gewählt, um eine Nachricht so schnell wie möglich zu verarbeiten, d.h. in die Mailbox der adressierten Komponente anzuhängen und danach die Verbindung abzubauen. Die andere Alternative wäre, für jede Verbindungsanforderung einen eigenen Prozeß zu starten (mittels `fork`). Dies ist bei einem "langsamen" Netz der günstigerer Ansatz, da hier die Übertragungsrate einer Nachricht sehr gering ist und damit die relativ langsame Prozeßerzeugung unter UNIX nicht ins Gewicht fällt. Geht man aber von einer Konfiguration der Rechner in einem lokalen, ethernetbasierten Netz aus, auf dem man sehr hohe Übertragungsraten hat, verbraucht die Prozeßerzeugung zu viele Systemressourcen.
Bei der iterativen Abarbeitung kann es vorkommen, daß der Server keine Verbindungsanforderungen mehr zwischenspeichern kann (maximal 5 Verbindungsanforderungen), solange er mit einer anderen Übetragung beschäftigt ist. Dies ist aber dann ein Indiz für den Client, daß der Server relativ hoch ausgelastet ist. Er kann diese Information der Lastbalancierung zur Verfügung stellen und einen weniger ausgelasteten Rechnerknoten kontaktieren.

Die Abbildung 12 zeigt den Aufbau des Briefträgers: Im Kommunkationssystem (CS - Communication System) wird der Client-Teil des Briefträgers eingebunden. Der Aufruf eines RM_Call führt zuerst eine Berechtigungsprüfung durch, die verhindern soll, daß die ConTract-Verarbeitung von nicht authorisierten Komponenten beeinflußt oder gestört wird. Danach erfolgt die Bestimmung der physikalischen Adresse der Komponente, die den Dienst anbietet. Als nächstes wird die Nachricht in das APRICOTS interne Format kodiert. Dies ist bei heterogenen Netzen mit unterschiedlichen Rechnerarchitekturen notwendig.
Auf der Ebene der Basiskommunikation wird anhand des Rechnernamens, den der Nameservice liefert bestimmt, ob die Dienstaktivierung prozesslokal, rechnerlokal oder auf einem entfernten Rechner stattfindet.
Ist der Rechnernamen der Komponente die den Dienst anbietet verschieden vom Rechnernamen der aufrufenden Komponente, so erfolgt der Nachrichtentransport über den Briefträger.

Der Briefträgerprozeß überträgt die Nachricht in die Mailbox der adressierten Komponente. Die Ziel-Komponente arbeitet in einer Bearbeitungsschleife. Dort werden die Nachrichten der Reihe nach ausgetragen und der in den Nachrichten spezifizierte Dienst aktiviert.

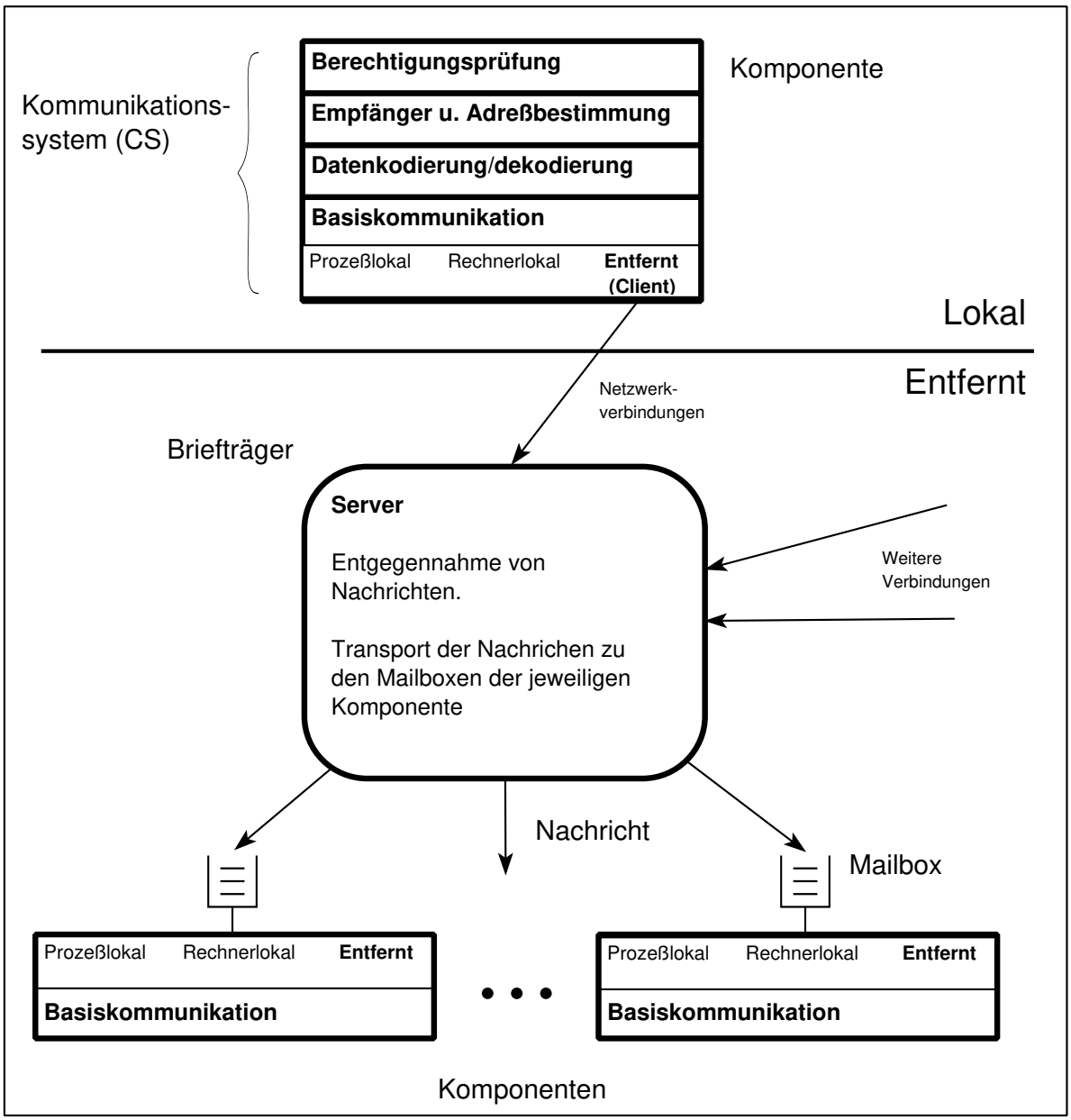

Abbildung 12: Aufbau des Briefträgers.

5.2.1 Funktion und Nachrichtenformat des Briefträgers

Die Nachrichten zwischen dem Client und dem Briefträgerprozeß setzen sich aus einem Funktionscode und einem Datenblock zusammen. Der Funktionscode besteht aus einem Zeichen, das die gewünschte Aktion auf den Datenblock angibt.
Der Datenblock ist die vom Client kodierte `CS_TRANSMISSION_UNIT` [2]. Sie enthält, neben Transaktionsinformationen, auch die Empfänger- bzw. Absenderadresse und den eigentlichen Nachrichtentext für die Dienstaktivierung in der Komponente. Der Server hat damit alle nötigen Informationen, um die Nachricht in der Mailbox abzulegen. Vom Briefträger werden folgende Funktion implementiert:

SEND Anhängen des Datenblockes (`CS_TRANSMISSION_UNIT`) in die Mailbox einer Komponente. Die Mailbox-ID befindet sich kodiert im Datenblock.

CANCEL Falls sich der Datenblock noch in der Warteschlange der Komponente befindet, wird er ausgetragen.

CHECK Test ob Datenblock noch in der Warteschlange der Komponente steht.

Die Struktur `CS_TRANSMISSION_UNIT` wird zuvor vom Client in das APRICOTS interne Nachrichtenformat gebracht. Dies ist notwendig, da z.B. integer-Werte, abhängig von der Rechnerarchitektur (big-/little-endian) und vom Compiler, unterschiedlich im Speicher abgelegt werden. Das Kommunikationssystem sieht vor, die einzelnen Einträge der Struktur in reine ASCII-Werte umzuwandeln und mit dem Trenner "\255" zu versehen.
Ein andere Alternative wäre, das XDR-Format von SUN zu benutzen. Die Routinen für die Konvertierung müßten dann aber zusätzlich portiert werden, falls APRICOTS auf einem Rechner laufen soll, der den SUN-RPC bzw. das XDR-Format nicht unterstützt (z.B. Apollo).

5.2.2 Protokoll der Datenübetragung

Die Kommunikation über das Netzwerk wird, wie in den vorherigen Abschnitten beschrieben, mittels den Prozeduren `send_message` und `receive_message` realisiert. Sie führen den Transport eines Datenblockes zwischen dem Client und dem Server durch. Nach Aufbau der Verbindung wird die Größe des Datenblocks übermittelt. Der Empfänger alloziert einen Speicherbereich dieser Größe und sendet eine Bestätigung. Nach Empfang dieser Bestätigung beginnt die Übertragung des Datenblockes. Der Erfolg wird durch eine weitere Bestätigung signalisiert.

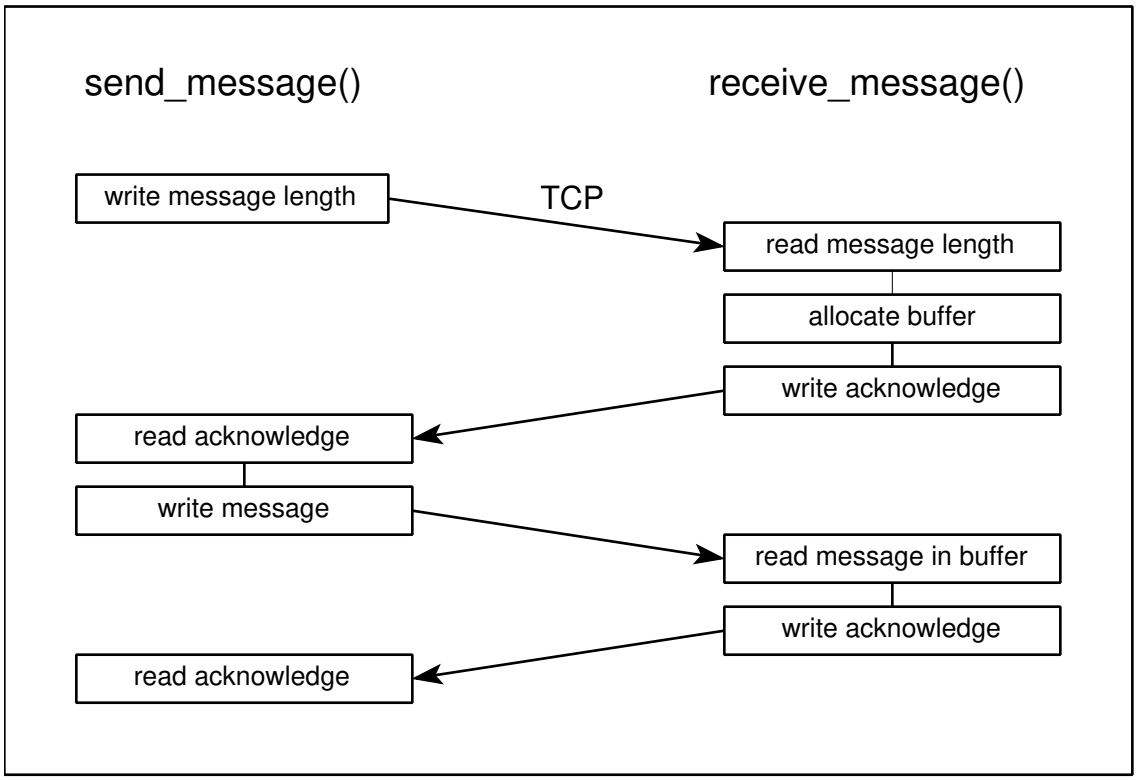

Abbildung 13: Ablauf des Protokolls zwischen send_message() und receive_message().

Die Zeitdauer der gesamten Übetragung wird überwacht. Falls die Schranke für die maximale Dauer einer Verbindung überschritten wird, bricht der `send_message`- oder `receive_message`- Aufruf die Verbindung ab und kehrt mit einem timeout-Fehler zurück. Dies kann von der Lastbalancierung verwendet werden, um von langsamen oder unsicheren Rechnerknoten auf schnellere und sichere überzugehen.

5.2.3 Bearbeitung einer Nachricht durch den Server

Der Briefträger-Server erhält nach dem Verbindungsaufbau den Funktionscode und die kodierte `CS_TRANSMISSION_UNIT` Struktur. Die lokale Kommunikation des Servers mit der Mailbox einer Komponente, um die Datenstruktur abzulegen, wird durch folgende Schnittstelle unterstützt:

`cs_connect_mailbox(mbx_id, mbx_handle)`
Zugriff auf die Mailbox `mbx_id` und Zurückliefern von `mbx_handle`. Die Mailbox wird dann über `mbx_handle` in den weiteren Funktionen der Schnittstelle angesprochen.

`cs_disconnect_mailbox(mbx_id, mbx_handle)`
Beenden der Verbindung zur Mailbox, die über `mbx_handle` angesprochen wird.

`cs _write_message_into_mailbox(mbx_id, mbx_handle, msg_handle, msg_len, msg)`
Eintragen der in `msg_handle` spezifizierten Nachricht in die durch `mbx_handle` angegebene Mailbox.

`cs_inquire_message_status(mbx_id, mbx_handle, msg_handle)`
Überprüfen, ob sich die durch `msg_handle` spezifizierte Nachricht in der Mailbox befindet.

`cs_delete_message_from_mailbox(mbx_id, mbx_handle, msg_handle, pto_msg_len, pto_msg)`
Löschen der Nachricht in `msg_handle` aus der Mailbox.

Die Mailboxfunktionen implementieren einen stabilen Speicher mit FIFO Organisation. Durch die iterative Arbeitsweise des Brieftägers ist eine Verklemmung oder Duplizierung von Nachrichten ausgeschlossen. Er befindet sich solange in Kontakt mit dem Client, bis die Nachricht abgeliefert wurde oder ein Fehler mit der Mailboxkommunikation auftrat.

Die Bearbeitung einer Verbindung zum Server und der Zugriff auf die Mailbox einer Komponente, ist hier im Pseudocode dargestellt:

```
do {
      Warte auf Verbindung mit einem Client

      receive_message(&function_code);
      receive_message(&data);
      "mailboxid" und "message" aus data bestimmen

      switch(function_code) {
         case SEND:
               cs_connect_mailbox(mailboxid,…);
```

```
                cs_write_message_into_mailbox(mailboxid, message, …);
                cs_disconnect_mailbox(mailboxid, …);
                Zurückliefern ob Erfolg/Fehler
                Verbindung zum Client abbauen
                break;
          case CHECK:
                cs_connect_mailbox(mailboxid,…);
                cs_inquire_message_status(mailboxid, message, …);
                cs_disconnect_mailbox(mailboxid, …);
                Zurückliefern ob Erfolg/Fehler
                Verbindung zum Client abbauen
                break;
          case CANCEL:
                cs_connect_mailbox(mailboxid,…);
                cs_delete_message_from_mailbox(mailboxid, message, …);
                cs_disconnect_mailbox(mailboxid, …);
                Zurückliefern ob Erfolg/Fehler
                Verbindung zum Client abbauen
                break;
     }
}
```

5.2.4 Schnittstelle zum Briefträger

Für die Kommunikation mit dem Briefträger steht dem Kommunikationssystem der Aufruf `remote_message` zur Verfügung. Dieser Aufruf implementiert eine allgemeine Schnittstelle für eine Nachrichtenübertragung zum Briefträger:

```
remote_message(function, ip, message, length)
```

Über `ip` wird die Internetadresse des Rechners bestimmt, dem die Nachricht zugesand werden soll. Auf dem adressierten Rechner muß auf einem *well-known* TCP-Port der Briefträger-Server laufen. Die Nachricht wird in `message` abgelegt; die Länge der Nachricht steht in `length`. Es ist dabei zu beachten, daß `message` eine ASCII-Kodierung der internen `CS_TRANSMISSION_UNIT` Struktur ist. Der `function` Parameter spezifiziert die Aktion, die der Briefträger auf diese Nachricht anwenden soll (`SEND`, `CANCEL` oder `CHECK`).
Die `remote_message`-Funktion liefert einen Fehler bei der Zeitüberschreitung (timeout) und bei einem Verbindungsabbruch zurück bzw. zeigt den Erfolg der Nachrichtenübertragung an.

Zu beachten ist, daß der `remote_message`-Aufruf blockierend abläuft. D.h. die Abarbeitung der Komponente wird solange angehalten, bis die Nachricht erfolgreich in die Mailbox der addressierten Komponente steht. Dies ist notwendig, um doppelte oder verlorene Nachrichten zu vermeiden. Sobald die Nachricht in der Mailbox einer Komponente abgelegt ist, ist sie persistent.

Die `remote_message`-Funktion wird in das Kommunikationssystem eingebunden. Die beiden Funktionen

```
cs_return_message(CS_TRANSMISSION_UNIT *message)
```
und
```
cs_transmit_message(CS_TRANSMISSION_UNIT *message)
```
des Kommunikationssystem übertragen eine Nachricht in eine Mailbox auf dem lokalen Rechner. Für den entfernten Fall muß dort der `remote_message`-Aufruf eingebunden werden.

5.3 Der Nameservice

Der Nameservice in APRICOTS wird mittels eines stabilen Datenspeichers implementiert. Für diesen stabilen Datenspeicher wurde in der jetztigen Implementierung eine Datenbank unter Ingres verwendet.
In dieser Datenbank befinden sich eine Tabelle für die Speicherung der Dienste und eine Tabelle für die der Insel bekannten Nameserver.
Der Nameserver ist nach dem Client - Server Prinzip aufgebaut. Der Client kann über das Nameserverprotokoll Anfragen auf andere Inselgruppen durchführen oder Informationen von entfernten Nameservern in die lokale Datenbank transferieren lassen, was eine Vergrößerung des Namensraumes bewirkt.

5.3.1 Datenstruktur und Schnittstelle zur Datenbank

In der Datenbank "apricots_cs" werden die Tabellen "local_services" für die Speicherung der Dienste der Inselgruppe und "global_namserver" für die der Inselgruppe bekannten Nameserver gehalten.

Aufbau der Tabelle "local_services":

Service_Name	Name des angebotenen Dienstes.
RM_Class_Name	Die Klasse, zu dem der Dienst angehört (dient zur näheren Spezifizierung eines Dienstes).
Log_Comp_Name	Komponentenname des laufenden Prozesses.
Process_ID	Nummer des laufenden Prozesses.
Hostname	Rechnername, auf dem der Dienst angeboten wird. Muß eindeutig sein (am besten vollständige Domain und Rechnernamen).
State_of_Component	Zustand der Komponente: verfügbar (up) oder nicht verfügbar (down).
State_of_Connection	Zustand der Verbindung zur Mailbox: noch nicht etabliert, etabliert und verfuegbar, gestoert, vom Partner aufgeloest worden.
Mailbox_ID	Auf einem Rechner eindeutige Identifikationsnummer der Mailbox.
Config_Reposit_Entry	Eintrag im globalen Configuration Repository.

Die Werte dieser Datenbankattribute werden für die interne Verarbeitung durch das Kommunikationssystem in der Struktur CS_ADDRESS_PARAM gespeichert und zu einer Liste verkettet.
Die Schnittstelle definiert Funktionen für das Anhängen und Änderen der Nameservice-Einträge in die Datenbank "local_services" (die Schnittstelle ist in der aktuellen Version mittels Embedded SQL unter INGRES implementiert):

`cs_name_service_insert_comp(address_list)`
Trägt die Dienste die in `address_list` spezifiziert sind, in die Datenbank ein. Der Parameter `address_list` ist eine verkettete Struktur, die die Einträge der Tabelle "local_services" enthält.

`cs_name_service_delete_comp(component_name)`
Löschen aller Dienste der Komponente `component_name` in der Datenbank.

`cs_name_service_update_component(address_entry, op)`
Ändern eines Eintrages in der Datenbank. Der Parameter `op` gibt an, welches Attribut in der Tabelle geändert wird (`Hostname`, `Process_ID`, …).

`cs_name_service_get_address(address)`
Liefert die `Mailbox_ID` und `State_of_Component` für die in `address` spezifizierte Komponente zurück.

`cs_name_service_lookup(service_name, rm_class_name, address_list)`
Suche nach einer Komponente, die den Service `service_name` in der Klasse `rm_class_name` implementiert. Rückgabe aller gefundenen Komponenten in der verketteten Liste `address_list`.

Die Tabelle "global_nameserver" für die der Inselgruppe bekannten Nameserver hat folgende Attribute:

Hostname	Rechnername des Nameservers (Primärschlüssel).
Islename	Name der Insel, zu welcher der Rechner gehört.
Preference	Dieses Attribut gibt eine Reihenfolge an, nach der die Nameserver einer Inselgruppe abgearbeitet werden (es können pro Inselgruppe mehrere Nameserver-Prozesse gestartet werden).
Nameserver_State	Status des Nameservers: verfügbar (up) oder nicht verfügbar (down).
Failures	Anzahl der fehlerhaften Verbindungen zum Nameserver. Beim überschreiten einer gewissen Fehleranzahl kann der Nameserver auf "down" gesetzt werden und ein anderer Nameserver-Prozeß kann dessen Funktion übernehmen.

Zugehörige Schnittstelle:

`cs_ns_db_add(nslist)`
Die Verkettete Liste `nslist`, mit Einträgen der obigen Attribute, wird an die Nameservertabelle "global_nameserver" angefügt.

`cs_ns_db_delete(hostname)`
Löschen des Eintrags, der `hostname` enthält.

`cs_ns_db_update(hostname, IsleName, state, failures)`
Aktualisieren der Attribute `IsleName`, `state` und `failures`, die über `hostname` qualifiziert werden.

`cs_ns_db_get(nslist, all)`
Liefert in Liste `nslist`, alle vorhandenen Einträge der Tabelle (ohne Einträge der eigenen Insel). Wenn das Flag `all` nicht gesetzt ist, werden alle Einträge, bei denen das Attribut `state` den Wert "`UP`" hat zurückgeliefert. Sonst wird der Wert des Attributs `state` ignoriert.

`cs_ns_db_nameserver(nslist, IsleName, all)`
Liefert in Liste `nslist` alle Einträge, der Insel `IsleName`. Ist das Flag `all` gesetzt, so werden alle Nameserver der Insel zurückgeliefert, bei denen das Attribut `state` den Wert "`UP`" hat. Ist das Flag `all` nicht gesetzt, so wird nur der Nameserver mit dem kleinsten `preference` Wert zurückgeliefert.

`cs_ns_db_inc_failures(hostname)`
Erhöht beim Eintrag von `hostname`, den Wert des Attributs `failures` um 1.

Die folgende Abbildung soll die Schnittstelle auf die beiden Tabellen der Datenbank noch einmal verdeutlichen.

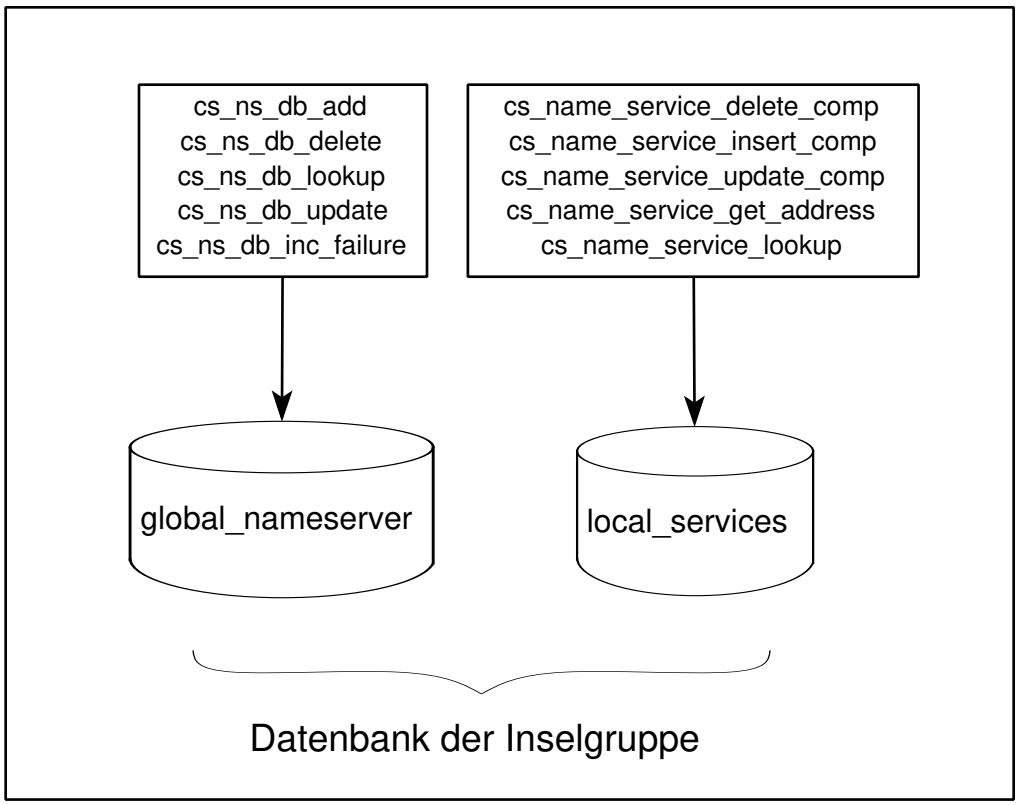

Abbildung 14: Schnittstelle zur Nameservice-Datenbank.

Beispielkonfiguration einer "global_nameserver" Tabelle mit den Attributen Hostname, IsleName, Preference, State U=UP, D=DOWN und Failures:

Insel A

```
Host1 InselA 1 U 0
Host2 InselA 2 U 1
Host3 InselB 1 D 100
Host4 InselB 2 U 1
Host5 InselC 1 U 0
```

Insel B

```
Host3 InselB 1 D 200
Host4 InselB 2 U 1
Host1 InselA 1 U 0
Host2 InselA 2 U 1
Host6 InselD 1 U 0
```

`Host1` und `Host2` sind hier die Nameserver der Inselgruppe A, `Host3` und `Host4` die der Insegruppel B. Beide Inselgruppen kennen die Nameserver von anderen Inselgruppen und können über diese Nameserver dort nach Diensten suchen. Die Inselgruppen lassen sich so zu einer N:N - Beziehung untereinander vernetzen (siehe Abbildung 15). Der Status bei `Host3` zeigt an, daß dieser zu viele Fehlverbindungen gehabt hatte. Deshalb wurde er als nicht verfügbar (`DOWN`) gekennzeichnet. Der aktuelle Nameserver für die Inselgruppe B ist nun der `Host4`.

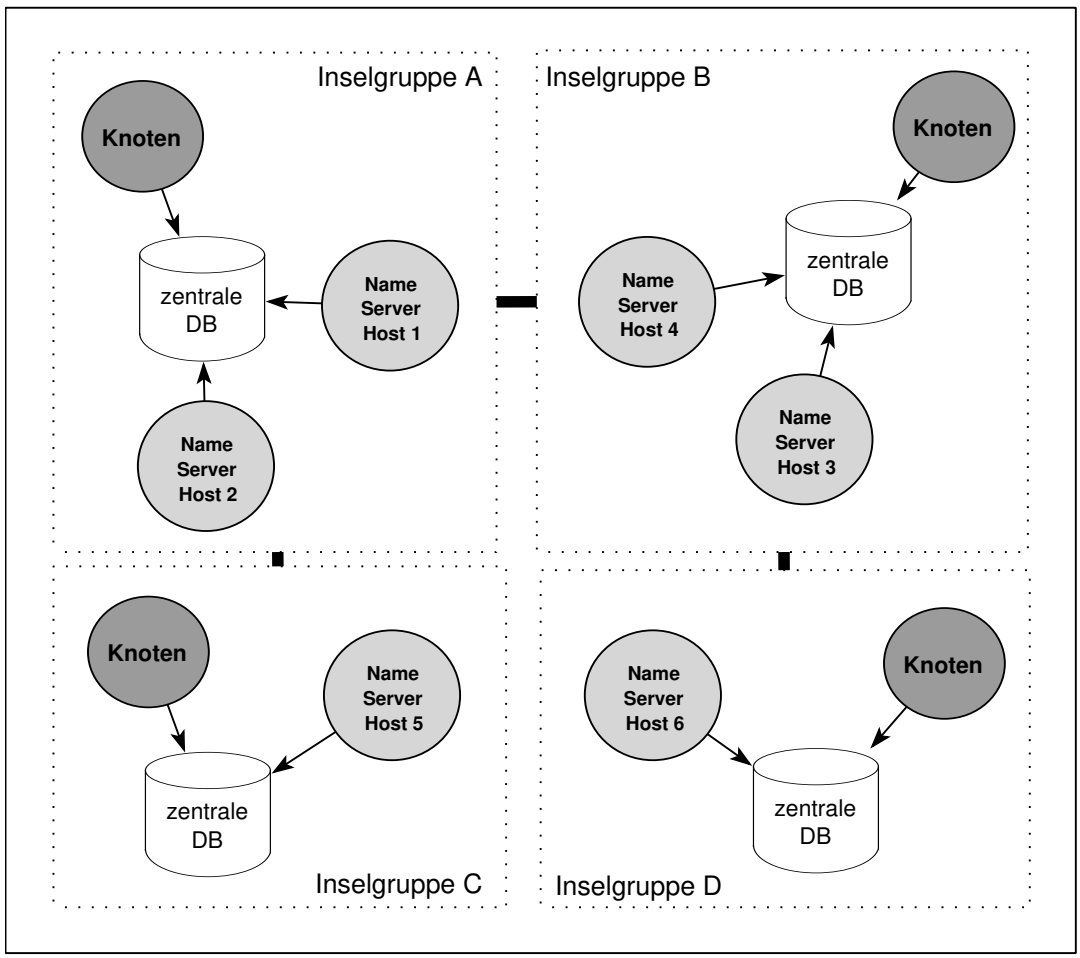

Abbildung 15: Vernetzung von Inselgruppen. Nach obigem Beispiel kennt die Inselgruppe A, die Inselgruppe B und C. Inselgruppe B kennt Inselgruppe A und D.

5.3.2 Der Nameserver

Im Gegensatz zum Briefträger bearbeitet der Nameserver seine Aufträge parallel. Für jede eingehende Verbindung wird ein neuer Prozeß erzeugt. Der Nameserver wird dadurch nicht von einem einzigen Aufrufer blockiert und bleibt immer ansprechbar. Der Geschwindigkeitsverlust bei der Prozeßerzeugung kann hier vernachlässigt werden, da die physikalische Adresse in der Regel nur zu Beginn einer Verarbeitung benötigt wird. Danach befindet sich die Adresse im Cache [2] der jeweiligen Komponente und kann so schnell gefunden werden.

Der Nameserver bietet Funktionen für die Client-Anfragen und für den Datenaustausch mit anderen Nameserver an. Über ein gemeinsames Protokoll werden die Client- und Nameserver-Anfragen unterstützt.
Der Nameserver implementiert folgende Funktionen:

LOOKUP Ein Client will einen Dienst auf einer anderen Inselgruppe suchen. Der Nameserver liefert dem Client bei Erfolg alle Rechnernamen mit den entsprechenden Mailbox-IDs zurück. Mit dieser Information ist der Client in der Lage, Verbindung zu einem Rechner aufzunehmen.

REQUESTDB Der Client beauftragt den Nameserver seiner Inselgruppe, einen Transfer der "global_nameserver" Tabelle einer anderen Inselgruppe, zur eigenen Inselgruppe durchzuführen. Die empfangenen Daten werden in die "global_nameserver" Tabelle der eigenen Inselgruppe angehängt. Eine Inselgruppe kann dadurch ihren Namensraum für die Dienstsuche vergrößern.

QUERY Ein entfernter Nameserver sucht nach einem bestimmten Dienst in der Inselgruppe. Die QUERY Nachricht wird bei der Bearbeitung der LOOKUP Nachricht generiert. Der lokale Nameserver besorgt sich dann die Informationen aus der Tabelle "local_services" mittels der `cs_name_service_lookup` Funktion.

NSDB Ein entfernter Nameservice will alle Einträge der "global_nameserver" Tabelle der Inselgruppe. Diese Nachricht wird bei der Bearbeitung der REQUESTDB Nachricht erzeugt.

Die Nachrichten LOOKUP und REQUESTDB werden vom Client über die Funktionen `cs_remote_name_service_lookup` und `cs_remote_name_service_nsdb` an den Nameserver gesendet. Die Nachrichten REQUESTDB und QUERY werden zwischen zwei Nameservern ausgetauscht.

Bearbeitungsschleife des Nameservers im Pseudocode:

```
do {
      Warte auf Verbindung mit einem Client
      Starte neuen Prozeß
         im Child:
            process_order() /* LOOKUP, QUERY, NSDB, REQUESTDB */
         im Parent: (warte auf weitere Verbindungen)
}

process_order()
{
      receive_message(&function_code);

      switch(function_code) {
         case LOOKUP:
                  /* Client sucht nach einem Dienst */
               break;
         case QUERY:
                  /* Entfernter Nameserver sucht nach Diensten */
               break;
         case REQUESTDB:
                  /* Client fordert die "global_nameserver" Datenbank */
                  /* eines Nameservers an */
               break;
         case NSDB:
                  /* Entfernter Nameserver will alle Einträge der */
                  /* lokalen "global_nameserver" Datenbank */
               break;

}
```

5.3.3 Protokoll des Nameservers

Das Protokoll vom Client zum Nameserver und zwischen den Nameservern, setzt sich aus dem Funktionscode (LOOKUP, QUERY, REQUESTDB, NSDB) und nachfolgenden Parametern (z.B. Dienst-, Klassenname) zusammen.

Die Antwort des Nameservers bei einer LOOKUP und QUERY Nachricht des Clients, sind Einträge der CS_ADDRESS_PARAM Struktur des Kommunikationssystems. Diese Einträge entsprechen einer Zeile der "local_services" Datenbank und werden jeweils einzeln verschickt. Um das Ende der Liste zu kennzeichnen wird ein EOT (End of Transmission) Code gesendet.
Zu beachten ist, daß die einzelnen Einträge als ASCII-Daten verschickt werden (z.B. wird der Integer Process_ID in seine ASCII-Darstellung umgewandelt), um auf verschiedenen Rechnerarchitekturen lauffähig zu sein.

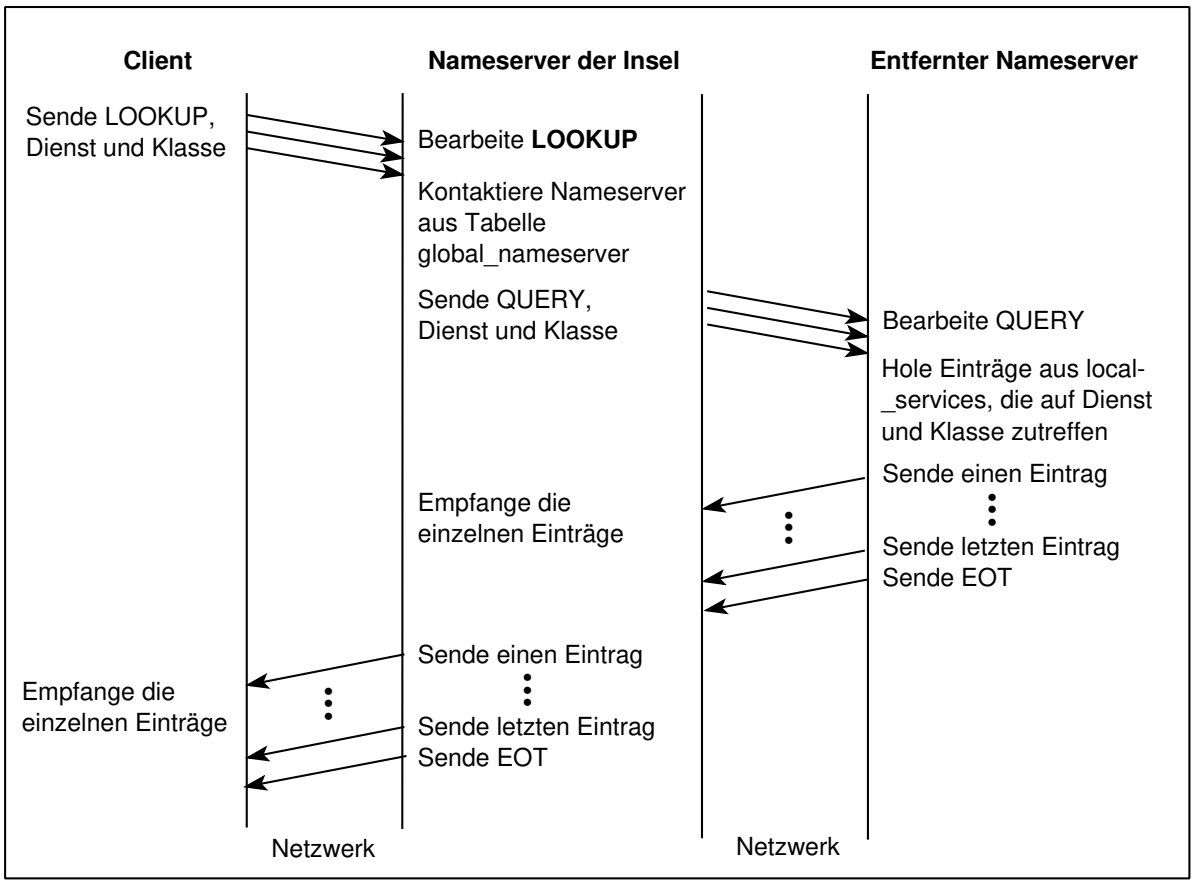

Abbildung 16: Protokoll bei einer LOOKUP Nachricht eines Clients.

Bei der REQUESTDB Nachricht wird als weiterer Parameter der Hostname eines entfernten Nameservers mitgegeben. Der lokale Nameserver fordert von diesem entfernten Nameserver, mittels einer NSDB Nachricht, alle Einträge aus dessen "global_nameserver" Datenbank. Die einzelnen Einträge werden kodiert und zurückgesendet. Sind keine Einträge mehr vorhanden, wird dies mittels des EOT Code angezeigt.

Die vom lokalen Nameserver der Inselgruppe empfangenen Daten werden direkt in die Datenbank der Inselgruppe abgelegt. Der Client bekommt eine Bestätigung, daß der Transfer zwischen den beiden Nameserver erfolgte.

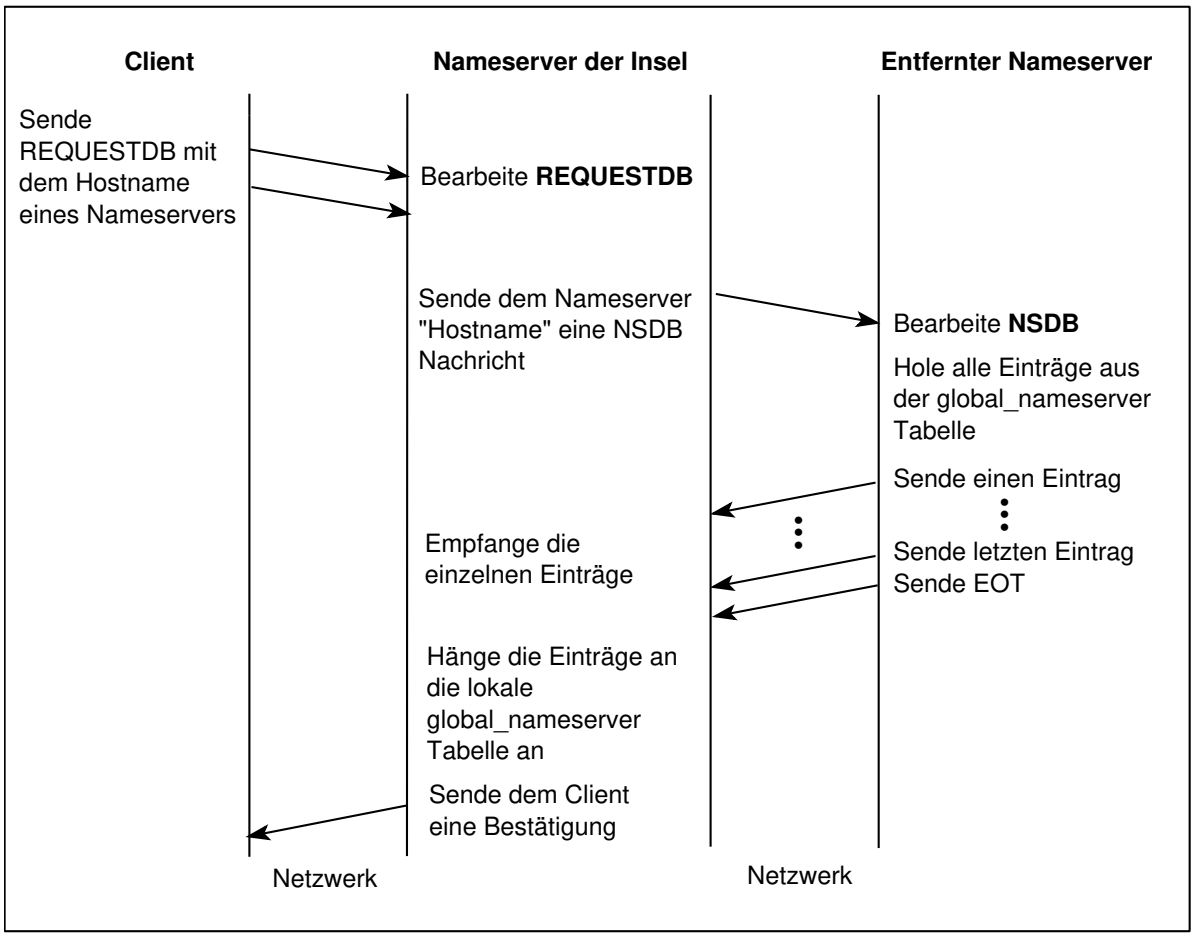

Abbildung 17: Protokoll einer REQUESTDB Nachricht.

5.3.4 Schnittstelle zum Nameserver

Der Client kommuniziert mit dem Nameserver über zwei blockierende Funktionen:

`cs_remote_name_service_lookup(service,rm_class,address_list)`
und `cs_remote_name_service_nsdb(hostname).`

Die `cs_remote_name_service_lookup` Funktion nimmt Verbindung mit einem verfügbaren Nameserver der Inselgruppe auf und sendet ihm eine LOOKUP Nachricht, gefolgt vom Dienst- und Klassennamen.
War der Aufruf erfolgreich, so befindet sich in dem Rückgabeparameter `address_list` eine Liste mit Einträgen von Diensten und zugehörigen physikalischen Adressen einer anderen Inselgruppe.

Mittels `cs_remote_name_service_nsdb` wird die "global_nameserver" Tabelle der Inselgruppe des mit `hostname` angegeben Nameservers angefordert. Es wird dem lokalen Nameserver dazu die Nachricht REQUESTDB gesendet. War der Transfer vom entfernten Nameserver erfolgreich, so trägt der lokale Nameserver die Daten in die "global_nameserver" Tabelle seiner Inselgruppe ein. Der Client bekommt eine Bestätigung.

Die `cs_remote_name_service_lookup` Funktion wird in das Kommunikationssystem eingebunden. Die Bestimmung eines Dienstes erfolgt in der Komponte dann dreistufig:

1. Suche des Dienstes im lokalen Cache der Komponente.
2. Suche des Dienstes auf der zentralen Datenbank der Inselgruppe ("local_services").
3. Suche des Dienstes auf einer anderen Inselgruppe (cs_remote_name_service_lookup).

Eine Lastbalancierung kann hier anhand von spezifischen Informationen, wie Netzwerk-, Rechnerauslastung, Fehlverbindungen, etc., die Suche nach einem Dienst steuern und den besten Kommunikationspfad auswählen.

5.3.5 Beispiel für eine Dienstsuche

Das folgende Beispiel soll den Datenfluß zwischen dem Client und den Nameservern bei der Dienstsuche auf einer anderen Inselgruppe verdeutlichen.

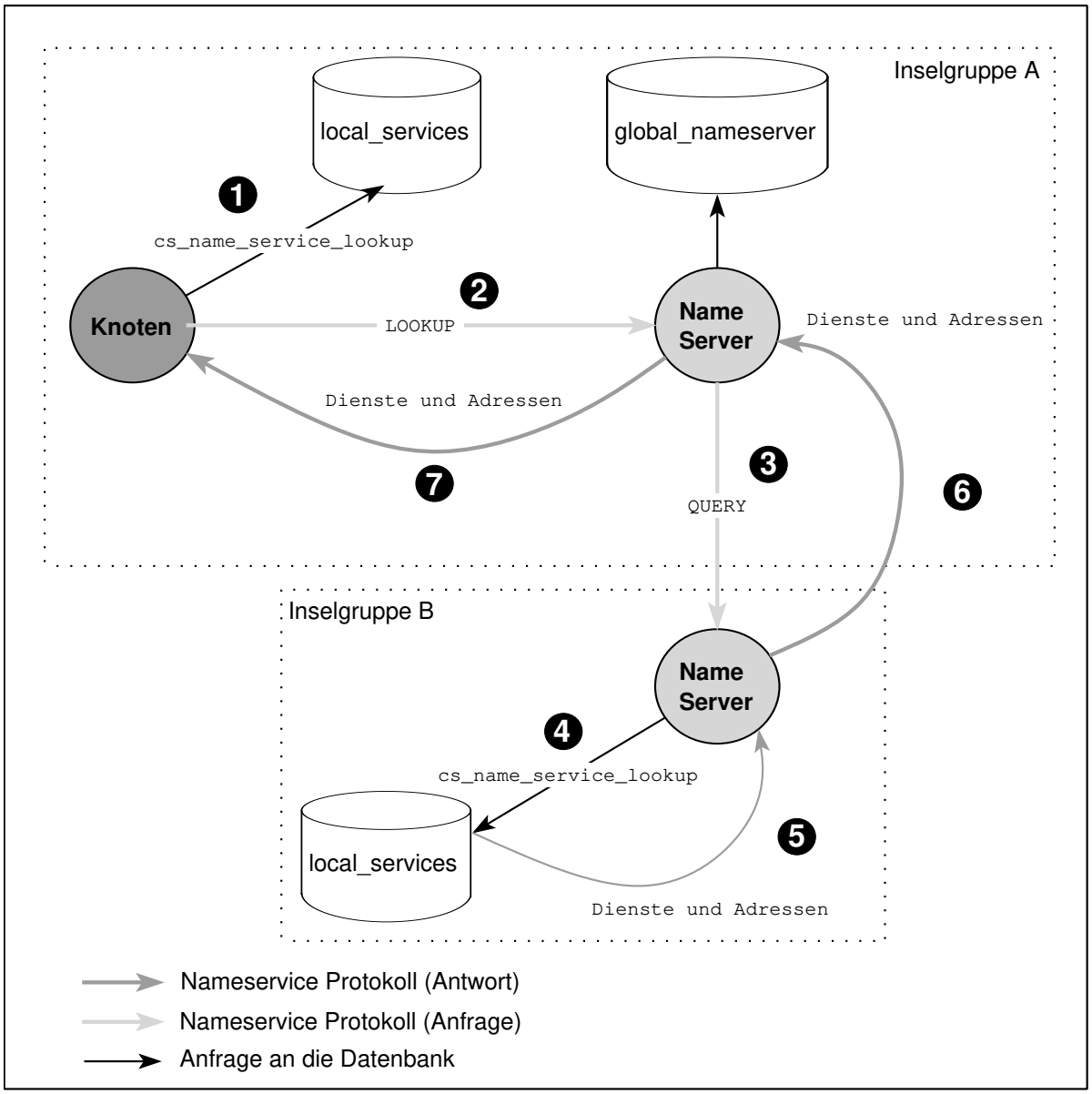

Abbildung 18: Ablauf einer Dienstsuche.

Beschreibung der Kommunikation:

(1) Das Kommunikationssystem der Komponente auf dem Rechnerknoten führt einen `cs_name_service_lookup()`-Aufruf auf die zentrale Datenbank aus. Der spezifzierte Dienst wurden nicht in der Tabelle "local_services" der Inselgruppe gefunden.

(2) Das Kommunikationssystem versucht nun, den Dienst auf einer anderen Inselgruppe zu lokalisieren. Dazu ruft es die Funktion `cs_remote_name_service_lookup()` auf. Dem lokalen Nameserver wird eine LOOKUP Nachricht, gefolgt vom Dienst- und Klassennamen gesendet.

(3) Der lokale Nameserver kontaktiert die Tabelle "global_nameservices" und erhält eine Liste mit Nameservern von anderen Inselgruppen. Er kontaktiert den ersten entfernten Nameserver aus der Liste und sendet ihm eine QUERY Nachricht. Wird die Verbindung aus irgendeinem Grund abgebrochen, so wird dies in "global_nameservices" vermerkt und der nächste Nameserver aus der Liste wird kontaktiert.

(4) Der entfernte Nameserver sucht mittels des lokalen `cs_name_service_lookup()`-Aufruf in seiner zentralen Datenbank nach dem Dienst.

(5) Sind die entsprechenden Einträge des spezifizierten Dienstes gefunden, so werden sie dem entfernten Nameserver zurückgeliefert.

(6) Der entfernte Nameserver sendet die Einträge zum Nameserver, der die QUERY Anfrage gestartet hat.

(7) Der lokale Nameserver transferiert die Einträge zum Client bzw. Kommunikationssystem.

5.4 Betrachtungen zur Portabilität

Der Briefträger und der Nameserver sind im Rahmen der Studienarbeit auf UNIX (DECStation, SUN) und VMS (VAX) implementiert worden. Durch Verwendung von TCP/IP und dem modularen Aufbau der Kommunikationsroutinen (send_message und receive_message) ließen sich beide Server und zugehörige Client Funktionen in einem Durchgang portieren.

Ein Problem trat jedoch bei der Portierung des Nameservers unter VAX/VMS auf. Aufgrund der Inkompatibilität der Standard VAX/C Bibliothek, die die socket-Kommunikation enthält und der POSIX Bibliothek mit dem fork-Aufruf mußte der Nameserver von einem paralleln zu einem iterativen Serverprozeß realisiert werden. Beide Bibliotheken implementieren nämlich eine eigene "filedescriptor table". Ein socket-Deskriptor der VAX/C Bibliothek wird deshalb, bei einem Aufruf der mit der POSIX Bibliothek gebunden wurde, überdeckt. Es war uns deshalb nicht möglich, einen parallelen Server über den POSIX fork-Aufruf zu implementieren. Der Server arbeitet deswegen unter VMS nur iterativ[9]. Eine Testverbindung zwischen einem Nameserver auf dem Ultrix-Pool zu einem Nameserver im VMS-Pool ließ sich jedoch ohne Probleme realisieren.

Die Anzahl der sockets ist unter den meisten Betriebssystemen eine konstante Größe. Es können deshalb nicht beliebig viele Anfragen von Clients auf die Server entgegen genommen werden. Die Anzahl der Verbindungen läßt sich bei manchen Betriebssystemen durch Ändern einer Kernel-Variable anpassen.
Ein weiteres Problem bei der TCP/IP Kommunikation über sockets ist die Wartezeit, bis ein socket wiederverwendet werden kann.
Werden nun sehr viele Nachrichten über den Briefträger transportiert ist es möglich, daß hier Verbindungen abgelehnt werden, obwohl der Server wenig aktuelle Verbindungen stehen hat und somit die Verbindungsanforderungen entgegen nehmen könnte.

Die well-known Ports des Briefträgers und des Nameserver dürfen nicht in Konflikt mit bereits allozierten Ports derselben Nummer stehen. Je nach Rechnersystem laufen verschiedene Netzwerkdienste, die alle einen bestimmten Port für sich registrieren. Beispielsweise belegt X-Windows die Portadressen ab 6000 für jeden gestarteten X-Server.
Bei der Konfiguration eines APRICOTS System sollte deshalb ein Port fest gewählt werden.

Die Geschwindigkeit für den Nachrichtentransport auf einem lokalen Netzwerk läßt sich evtl. durch Anpassen der Empfangs- und Sendepuffergröße[10] des TCP/IP-Moduls erhöhen. Dies vermindert den Fragmentierungsaufwand im Puffer. Eine geeignete Puffergröße läßt sich jedoch nur über einen Test ermitteln, da hier die durchschnittlichen Größe einer APRICOTS Nachricht ermittelt werden muß.
Bei großen APRICOTS Nachrichten kann sich hier durchaus eine betrachtliche Steigerung des Durchsatzes ergeben.

---

[9] Diese Problem ist eventuell in einer der nächsten POSIX Implementierungen unter VMS behoben.

[10] Die Puffergröße beträgt in der Regel 4096 Bytes.

6 Zusammenfassung und Ausblick

An dieser Stelle möchte ich abschließend die Konzepte des Briefträgers und des Namservice zusammenfassen und ihre Vor- und Nachteile bzw. Probleme aufzeigen.

6.1 Briefträger

Der Briefträgermechanismus ist nach dem Client-Server Prinzip aufgebaut. Der Transport einer Nachricht und die Angabe der Bearbeitungsfunktion laufen synchron ab, d.h. der Aufrufer von `remote_message` wird blockiert. Die Verbindung bleibt solange bestehen, bis die Daten in der Mailbox abgelegt sind oder ein Fehler auftritt. Der Briefträger hat keinen internen Status. Es kann so kein Deadlock oder eine Dateninkonsistenz auftreten.
Anhand des Rückgabewertes (Erfolg oder Fehler) lassen sich von der Lastbalancierung weitere Aktionen durchführen.

Durch die verwendeten Standards wie TCP/IP, sockets und signals sollte sich der Briefträger leicht auf andere Plattformen portieren lassen. Die Modularisierung der Kommunikation kann auch von anderen APRICOTS-Teilsysteme, wie dem Nameserver und der Lastbalancierung verwendet werden.

Der Briefträger-Server kann um weitere Funktionen leicht erweitert werden. So ist es z.B. möglich, über das Briefträger-Protokoll Informationen, wie Auslastung des Rechners oder der Komponenten, zu transportieren. Die Lastbalancierung kann den Briefträger für die Sammlung von System- und Auslastungsinformationen mitbenutzen.

Die Einschränkung, daß der Briefträger-Server nur iterativ arbeitet, kann mittels des `fork`-Aufrufs aufgehoben werden. Es können dabei jedoch folgende Probleme auftreten:

- bei schnellen Netzwerken macht sich der Overhead für die Prozeßerzeugung negativ bemerkbar.

- die Anzahl der sockets und damit die Anzahl der möglichen Verbindungen, sind vom Betriebssystem her begrenzt und können nicht immer konfiguriert werden. Außerdem sieht es die TCP/IP Implementation vor, einen socket nach Beenden der Verbindung eine bestimmte Zeit noch offen zu halten. Ein socket ist also nach dem `close`-Aufruf, nicht unmittelbar wieder zu verwenden.

En Nachteil des Briefträgers ist seine relativ geringe Ausfallsicherheit. Es werden keine Vorkehrungen im Falle eines Absturzes getroffen. Hier muß der Briefträger neu gestartet werden. Bei näherer Betrachtung zeigt sich jedoch, daß hier eigentlich keine besonderen Maßnahmen vorgenommen werden müssen. Es existieren nämlich zwei Arten von Ausfällen:

1. Der Briefträger-Prozeß stürzt ab. Die anderen Komponenten arbeiten jedoch weiter.
2. Der Rechner fällt aus oder das Netzwerk ist nicht mehr verfügbar.

Der zweite Fall ist, im Bezug auf die Ausfallsicherheit des Briefträgers, nicht problematisch: wenn der Rechner oder das Netzwerk ausfallen, so sind auch die darauf laufenden Komponenten nicht mehr vorhanden. Bei Neustart des Rechners werden die Komponenten und der Briefträger einfach wieder gestartet. Der Briefträger hat keinen internen Status, deshalb kann es auch keine Probleme mit doppelten oder verlorengegangenen Nachrichten geben.
Der erste Fall läßt sich im Prinzip nur mit redundanten Briefträger-Prozessen lösen. Bei TCP können jedoch nicht mehrere Server-Prozesse auf einen well-known Port zugreifen, um Verbindungen entgegen zunehmen. Also scheidet diese Möglichkeit aus. Der Prozeß wird dann in einem solchen Falle nochmal gestartet.
Die Ausfallsicherheit von APRICOTS und der ConTract-Verarbeitung beeinträchtigt dies jedoch nicht, so daß auch hier keine weiteren Vorkehrungen getroffen wurden.

Ein weiteres Problem bei der TCP/IP Kommunikation des Briefträgers ist der relativ langdauernde Verbindungsaufbau. Beim Senden einer Nachricht vom Client an den entfernten Briefträger wird jedesmal ein neuer socket initialisiert und das TCP/IP Modul tauscht Protokoll-Parameter mit dem entfernten Rechner aus.
Wenn eine Komponente nun viele Nachrichten an denselben Briefträger versendet, macht sich dieser ständige Verbindungsauf- und abbau bemerkbar.
Ein Lösung besteht darin, die Verbindung eine bestimmte Zeit offen zu halten und mittels einer Zugriffsstatistik auf Rechnernamen die Verbindungen zu optimieren.

6.2 Nameservice

Der Nameservice von APRICOTS baut auf zwei Konzepten für die Dienstsuche auf:

1. Zugriff auf die Dienste der Inselgruppe über die Tabelle "local_services".
2. Suche auf entfernten Inselgruppen über den Nameserver.

Jede Komponente hat über das Kommunikationssystem Zugriff auf die zentrale, stabile Datenbank der Inselgruppe. Die Komponente kann unmittelbar die lokalen Dienste über die Tabelle "local_services" erfragen, ohne einen Nameserver kontaktieren zu müssen. Dies beschleunigt die Suche und vermindert die Netzwerkkommunikation.
Die Suche nach Diensten auf entfernten Inselgruppen erfolgt über die Nameserver. Sie haben in der Tabelle "global_nameserver" die Adresse von Nameservern anderer Inselgruppen. Die Dienstsuche läßt sich so auf andere Inselgruppen ausdehnen. Jeder Nameserver hat weiterhin die Möglichkeit, die Tabelle "global_nameserver" einer anderen Inselgruppe zu erfragen und in seine lokale Tabelle anzuhängen. Die Inselstruktur ist also dynamisch erweiterbar.

Die Ausfallsicherheit der Nameserver läßt sich mittels redundanter Nameserverprozesse erhöhen. Sie können auf verschiedenen Rechnern gestartet werden. Die Reihenfolge der Nameserver, die im Falle eines Ausfalls zum Einsatz kommen, ist in der Tabelle "global_nameserver" gespeichert. Man kann damit gezielt Nameserver, den speziellen Eigenschaften eines Rechners entsprechend, verteilen.

Existiert auf einer Rechnerarchitektur kein fork-Aufruf, wie es z.B. bei der TCP/IP Implementation von ftp-Software unter DOS der Fall ist, so kann der Nameserver auch iterativ arbeiten.

Die Konfiguration des Nameservice erfolgt über die Werkzeuge, die bei den Datenbanken mitgeliefert werden (z.B. bei Ingres: ingmenu) oder lassen sich mit Hilfe von SQL-Scripts einrichten.

7 Literatur

[1] Friedemann Schwenkreis. APRICOTS - die prototypische Implementierung eines ConTract-Systems (Kontrollflußsteuerung und Kommunikationssystem).Technischer Bericht, IPVR Universität Stuttgart 1993.

[2] Dieter Laube. Transaktionsorientierte Kommunikationsmechanismen im verteilten Transaktionssystem APRICOTS. Diplomarbeit 865, IPVR Universität Stuttgart 1992.

[3] A. Reuter, F. Schwenkreis, H. Wächter. Zuverlässige Abwicklung großer verteilter Anwendungen mit ConTracts - Architektur einer Prototypimplementierung. Erschienen in Bayer/Härder/Lockemann:"Objektbanken für Experten". Springer-Verlag, 1992.

[4] H. Burkert. Systemarchitektur und Betriebsmodell für das verteilte Transaktionssystem APRICOT. Diplomarbeit 812, IPVR Universität Stuttgart, August 1991.

[5] W. Richard Stevens. UNIX Network Programming. Prentice-Hall, 1990.

[6] Leffler/McKusick/Karels/Quaterman. Das 4.3 BSD UNIX Betriebssystem. Addison-Wesley, 1990.

[7] Comer/Stevens. Internetworking with TCP/IP Volumes I & II. Prentice-Hall Internationall Editions, 1991.

[8] Kochan/Wood. UNIX Networking. Hayden Books, 1989.

[9] Quarterman/Wilhelm. The Open Standards Puzzle. Addison-Wesley, 1993.

[10] Introduction to OSF DCE. Prentice-Hall.

[11] Eppinger/Mummert/Spector. Camelot and Avalon (A Distributed Transaction Facility). Morgan Kaufmann Publishers, Inc., 1991.

[12] RFC 1032–1035. Das Domain Name System (DNS). In elektronischer Form erhältlich vom "Internet Network Information Center" (Internetadresse: NIC.DDN.MIL).

[13] ISO 9594. Beschreibung des X.500 Standards. ISO Standards befinden sich auf diversen FTP-Servern oder sind von folgender Adresse erhältlich: Omnicon Inc. 115 Park Street, SE. Vienna, VA 22180-4607.(800) 666-4266.

[14] Workshop on the future trends of Distributed Computing Systems in the 90s. IEEE Proceedings, 1988. Twiddle/Sloman:"Domain Based Configuration and Name Management for Distributed Systems".

[15] 6th International Conference on Distributed Computing Systems, 1986. Session 10A: "Distributed Name Management".

[16] Sape Mullender. Distributed Systems. ACM-Press, 1989.

[17] Peter Wahl. Interpretative Abwicklung von Prädikat-Transitionsnetzen. Studienarbeit 1021, IPVR Universität Stuttgart, 1991.

[18] Ioannis Masmanidis. Robuste Kontrollflußverwaltung für langlebige verteilte DB-Anwendungem. Diplomarbeit 885, IPVR Universität Stuttgart, 1992.

[19] BI. (Informationen für Nutzer des Rechenzentrums). Rechenzentrum Uni Stuttgart (RUS), Ausgabe 5/93: "DFN-RPC, SUN-RPC, DCE RPC und PVM im Vergleich".

[20] Das ISODE Paket befindet sich auf diversen FTP-Servern. Informationen lassen sich von folgender Adresse beziehen: NYSERNet Inc. Attn: M.T. Rose. 420 Whisman Court. Mountain View, CA 94043-2112. (415) 961-3380.

Anhang

Der folgende Anhang beschreibt die Schnittstellen zum Briefträger (Abschnitt "Briefträger") und Nameserver (Abschnitt "Nameserver"). Die angebotenen Funktionen werden vom Kommunikationssystem genutzt, um Nachrichten mit den entsprechenden Briefträger- und Nameserver-Prozessen auszutauschen.
Weiterhin werden die Datenstrukturen und die Schnittstelle des stabilen Datenbereich der Inselgruppe beschrieben. Dabei existieren Funktionen für die Tabelle "global_nameserver" (Abschnitt "Schnittstelle für den stabilen Datenbereich – Nameserver") und "local_services" (Abschnitt "Schnittstelle für den stabilen Datenbereich – Dienste").
Zum Schluß werden noch die Unterprogramme für die entfernte Kommunikation beschrieben (Abschnitt "Netzwerkkommunikation").

Briefträger

Name

`init_postman_client` – Initialisierung des Briefträger-Client

Aufruf

```
int init_postman_client();
```

Beschreibung

Initialisierung der Schnittstelle zum Briefträger-Prozeß. Die Funktion `init_postman_client` muß einmal aufgerufen werden, bevor über die `remote_message` Funktion auf den Briefträger-Server zugegriffen wird.

Die Funktion `init_postman_client` bereitet das Modul für die Netzwerkkommunikation vor (`init_sendmsg` und `init_recvmsg`).

Rückgabewerte

Keine.

Dateien

pm.h, cs.h

Weitere Informationen

```
remote_message (Briefträger),
init_sendmsg (Netzwerkkommunikation),
init_recvmsg (Netzwerkkommunikation)
```

Briefträger

Name

`remote_message` – Transport einer Nachricht zum Briefträger

Aufruf

```
long remote_message(char func, char *ip,
     unsigned char *amesg, long mesglen);
```

Beschreibung

Transport der Nachricht in `amesg` mit der Länge `mesglen` zum Briefträger-Server auf dem Rechner mit der Internetnummer `ip`. Der Serverprozeß befindet sich dabei auf dem well-known Port `SERV_TCP_PORT` aus der Include-Datei pm.h. Der Parameter `func` spezeifiziert die Aktion, die der Briefträger auf die Nachricht anwenden soll (`SEND`, `CANCEL`, `CHECK`).

Die Nachricht `amesg` ist eine kodierte `CS_TRANSMISSION_UNIT` im APRIOCTS Format (ASCII-Einträge mit Delimiter \255). Die Funktion `remote_message` läuft blockierend ab. Tritt eine Zeitüberschreitung oder ein Verbindungsabbruch auf, so wird dies über den Rückgabewert angezeigt.

Die Funktion `remote_message` benutzt die Unterfunktion für die Netzwerkkommunikation.

Rückgabewerte

`CS_PM_OK` – Übertragung war erfolgreich.
`CS_PM_ERROR` – Die Kommunikation wurde unterbrochen oder es trat eine Zeitüberschreitung auf.

Dateien

pm.h, cs.h

Weitere Informationen

`init_sendmsg, init_recvmsg (Netzwerkkommunikation),`
`send_message, receive_message (Netzwerkkommunikation),`
`connect_tcp, disconnect_tcp (Netzwerkkommunikation)`

Nameserver

Name

`cs_remote_name_service_lookup`
– Dienstsuche auf einer entfernten Inselgruppe

Aufruf

```
long cs_remote_name_service_lookup(
     CS_SERVICE_NAME service, CS_RM_CLASS_NAME rm_class,
     CS_ADDRESS_PARAM **address_list);
```

Beschreibung

Suche nach einem Dienst auf einer entfernten Inselgruppe. Die Funktion `cs_remote_name_service_lookup` nimmt Verbindung zum Nameserver der eigenen Inselgruppe auf und überträgt die Parameter `service` und `rm_class`, die den gewünschten Dienst spezifizieren. Über den Rückgabeparameter `address_list` wird eine verkettete Liste mit den physikalischen Adressen zurückgegeben. Wurden keine Dienste gefunden, so befindet sich in `address_list` ein NULL-Pointer. Diese Liste muß vom Aufrufer wieder freigegeben werden.

Ist der Parameter `rm_call` ein NULL-Wert, so werden alle Dienste, die durch `service` spezifiziert sind, gesucht.

Rückgabewerte

`CS_NS_ERROR` – Verbindungsfehler oder Zeitüberschreitung.
`CS_NS_OK` – Übertragung war erfolgreich.

Dateien

ns.h, cs.h, nsprot.h

Weitere Informationen

`Netzwerkkommunikation, Schnittstelle für den stabilen Datenbereich der Insel.`

Nameserver

Name

`cs_remote_name_service_nsdb`
– Erweitern des Namensraumes

Aufruf

```
long cs_remote_name_service_nsdb(char *hostname);
```

Beschreibung

Erweitern des lokalen stabilen Datenbereiches "global_nameserver" um die Einträge einer Inselgruppe, für die der Rechner `hostname` Nameserver ist.

War der Aufruf erfolgreich, so befinden sich die Einträge der entfernten Inselgruppe im lokalen Datenbereich.

Rückgabewerte

`CS_NS_ERROR` – Verbindungsfehler oder Zeitüberschreitung.
`CS_NS_OK` – Übertragung war erfolgreich.

Dateien

ns.h, cs.h, nsprot.h

Weitere Informationen

`Netzwerkkommunikation, Schnittstelle für den stabilen Datenbereich der Insel.`

Schnittstelle für den stabilen Datenbereich

Die folgenden Funktionen implementieren die Schnittstelle zum stabilen Datenbereich der Inselgruppe.

Der erste Teil (**Nameserver**) beschreibt die Zugriffsfunktionen für die bekannten Nameserver der Inselgruppe. Im Rahmen der Studienarbeit wurde der stabilen Datenbreich mittels einer Ingres und RDB Datenbank über Embedded SQL realisiert. Die Tabelle "global_nameserver" enthält folgende Attribute:

Hostname	Rechnername des Nameservers.
Islename	Name der Inselgruppe, zu welcher der Rechner "Hostname" angehört.
Preference	Reihenfolge, nach der die Nameserver einer Inselgruppe abgearbeitet werden.
Nameserver_State	Status des Nameservers: verfügbar (up) oder nicht verfügbar (down).
Failures	Anzahl der fehlerhaften Verbindungen zum Nameserver.

Der zweite Teil (**Dienste**) beschreibt die Zugriffsfunktionen für die Speicherung der angebotenen Dienste einer Inselgruppe. Diese Funktionen wurden wie oben unter INGRES und RDB über Embedded SQL realisiert. Die Tabelle "local_services" enthält folgende Attribute:

Service_Name	Name des angebotenen Dienstes.
RM_Class_Name	Die Klasse, zu dem der Dienst angehört.
Log_Comp_Name	Komponentenname des laufenden Prozesses.
Process_ID	Nummer des laufenden Prozesses.
Hostname	Rechnername, auf dem der Dienst angeboten wird.
State_of_Component	Zustand der Komponente: verfügbar (up) oder nicht verfügbar (down).
State_of_Connection	Zustand der Verbindung zur Mailbox: noch nicht etabliert, etabliert und verfuegbar, gestoert, vom Partner aufgeloest worden.
Mailbox_ID	Auf einem Rechner eindeutige Identifikationsnummer der Mailbox.
Config_Reposit_Entry	Eintrag im globalen Configuration Repository.

Schnittstelle für den stabilen Datenbereich – Nameserver

Name

`cs_ns_db_add` – Hinzufügen von Nameservereinträgen

Aufruf

```
long cs_ns_db_add(struct NameserverList *nslist);
```

Beschreibung

Die verkettete Liste `nslist` wird an die Nameservertabelle der Insel angefügt. Doppelte Einträge werden ignoriert. Bei Differenzen (andere Inselgruppenzugehoerigkeit eines Rechners, etc) hat die Liste `nslist` Vorrang gegenüber den Daten in der Tabelle.

Die Struktur `NameserverList` bildet eine verkettet Liste und hat folgenden Aufbau (nsdbiface.h):

```
struct NameserverList {
     char   hostname[CS_LOGICAL_NAME_LEN];
     char   IsleName[CS_LOGICAL_NAME_LEN];
     int    pref;
     char   state;     /* U = up; D = down */
     int    failures;
     struct NameserverList   *next;
};
```

Das Ende der Liste wird mittels eines NULL-Pointers im Eintrag `next` gekennzeichnet.

Rückgabewerte

`CS_NS_ERROR` – Fehler mit der Datenbankkommunikation.
`CS_NS_OK` – Daten wurden erfolgreich in die Tabelle eingetragen.

Dateien

nsdbiface.h definiert die Schnittstelle und zugehörige Datenstrukturen.

Weitere Informationen

Schnittstelle für den stabilen Datenbereich – Nameserver

Name

`cs_ns_db_delete` – Löschen eines Eintrages

Aufruf

```
long cs_ns_db_delete(char *hostname);
```

Beschreibung

Löschen des Eintrages, der `hostname` enthält.

Rückgabewerte

`CS_NS_ERROR` – Fehler mit der Datenbankkommunikation.
`CS_NS_OK` – Der Eintrag wurde erfolgreich gelöscht.

Dateien

nsdbiface.h definiert die Schnittstelle und zugehörige Datenstrukturen.

Weitere Informationen

Schnittstelle für den stabilen Datenbereich – Nameserver

Name

`cs_ns_db_update` – Aktualisieren eines Eintrages

Aufruf

```
long cs_ns_db_update(char *hostname, char *islename,
     char state, int pref ,int failures);
```

Beschreibung

Aktualisieren der Attribute `islename`, `state`, `pref` und `failures`, die über `hostname` qualifiziert werden.
Es wird eine komplette Änderung der angegeben Werte durchgeführt.

Rückgabewerte

`CS_NS_ERROR` – Fehler mit der Datenbankkommunikation.
`CS_NS_OK` – Der Eintrag wurde erfolgreich aktualisiert.

Dateien

nsdbiface.h definiert die Schnittstelle und zugehörige Datenstrukturen.

Weitere Informationen

Schnittstelle für den stabilen Datenbereich – Nameserver

Name

`cs_ns_db_get` – Allgemeine Datenbankanfrage

Aufruf

```
long cs_ns_db_get(struct NameserverList **nslist,
     char all);
```

Beschreibung

Es werden alle aktuellen Einträge der Tabelle "global_nameserver" in `nslist` zurückgeliefert. Der Parameter `nslist` bildet eine verkettete Liste, die von `cs_ns_db_get` aufgebaut wird. Ist der Parameter `nslist` ein NULL-Pointer, so wurden keine Einträge gefunden. Die allozierte Liste kann mittels `cs_ns_db_free_nslist` wieder freigegeben werden.

Über das Flag `all` lassen sich die Einträge einschränken:
`all` == 1: Zurückliefern aller Einträge.
`all` == 0: Zurückliefern von Einträgen, bei denen das Attribute `state` den Wert `U` (up) hat.

Rückgabewerte

`CS_NS_ERROR` – Fehler mit der Datenbankkommunikation.
`CS_NS_OK` – Der Eintrag wurde erfolgreich aktualisiert.

Dateien

nsdbiface.h definiert die Schnittstelle und zugehörige Datenstrukturen.

Weitere Informationen

`cs_ns_db_free_nslist (stabiler Datenbereich – Nameserver), struct NameserverList (cs_ns_db_add)`

Schnittstelle für den stabilen Datenbereich – Nameserver

Name

`cs_ns_db_nameserver` – Spezielle Datenbankanfrage

Aufruf

```
long cs_ns_db_nameserver(struct NameserverList **nslist,
     char *islename, char all);
```

Beschreibung

Es werden alle aktuellen Einträge der Inselgruppe `islename` der Tabelle "global_nameserver" in `nslist` zurückgeliefert, . Der Parameter `nslist` bildet eine verkettete Liste, die von `cs_ns_db_nameserver` aufgebaut wird. Ist der Parameter `nslist` ein NULL-Pointer, so wurden keine Einträge gefunden. Die allozierte Liste kann mittels `cs_ns_db_free_nslist` wieder freigegeben werden.

Über das Flag `all` lassen sich die Einträge weiter einschränken:
`all` == 1: Zurückliefern aller Einträge der Inselgruppe `islename`.
`all` == 0: Zurückliefern von Einträgen, bei denen das Attribut `preference` den kleinsten Wert *und* das Attribute `state` den Wert `U` (up) hat.

Rückgabewerte

`CS_NS_ERROR` – Fehler mit der Datenbankkommunikation.
`CS_NS_OK` – Der Eintrag wurde erfolgreich aktualisiert.

Dateien

nsdbiface.h definiert die Schnittstelle und zugehörige Datenstrukturen.

Weitere Informationen

`cs_ns_db_free_nslist (stabiler Datenbereich – Nameserver), struct NameserverList (cs_ns_db_add)`

Schnittstelle für den stabilen Datenbereich – Nameserver

Name

`cs_ns_db_inc` – Erhöhen des Wertes beim Attribut failures.
`cs_ns_db_free_nslist` – Freigeben der verketteten Liste

Aufruf

```
long cs_ns_db_inc_failures(char *hostname);
long cs_ns_db_free_nslist(struct NameserverList *nslist);
```

Beschreibung

`cs_ns_db_inc_failures`
Erhöhen des Wertes um eins beim Attribut `failures`. Wird bei einer Fehlverbindung mit dem entsprechenden Nameserver `hostname` benutzt.

`cs_ns_db_free_nslist`
Freigeben der verketteten Liste `nslist`.

Rückgabewerte

`cs_ns_db_inc_failures`
`CS_NS_ERROR` – Fehler mit der Datenbankkommunikation.
`CS_NS_OK` – Der Eintrag wurde erfolgreich aktualisiert.

`cs_ns_db_free_nslist`
`CS_NS_OK` – wird immer zurückgegeben.

Dateien

nsdbiface.h definiert die Schnittstelle und zugehörige Datenstrukturen.

Weitere Informationen

Schnittstelle für den stabilen Datenbereich – Dienste

Name

`cs_name_service_delete_comp` – Löschen aller Dienste einer Komponente

Aufruf

```
CS_RETURNCODE
cs_name_service_delete_comp(
     CS_LOGICAL_NAME component_name);
```

Beschreibung

Löscht alle Dienste der Komponente `component_name` aus der Tabelle "local_services".

Rückgabewerte

`CS_RC_NAME_SERVICE_ERROR` – Fehler mit der Datenbankkommunikation.
`CS_RC_SUCCESS` – Der Eintrag wurde erfolgreich gelöscht.

Dateien

cs.h

Weitere Informationen

Schnittstelle für den stabilen Datenbereich – Dienste

Name

`cs_name_service_insert_comp`
– Anfügen von Diensten an eine Komponente

Aufruf

```
CS_RETURNCODE
cs_name_service_insert_comp(CS_ADDRESS_PARAM
     *address_list);
```

Beschreibung

Fügt die in der verketteten Liste `address_list` enthaltenen Einträge in die Datenbank "local_services". Der Komponentennamen befindet sich jeweils im Eintrag selbst.

Rückgabewerte

`CS_RC_NAME_SERVICE_ERROR` – Fehler mit der Datenbankkommunikation.
`CS_RC_SUCCESS` – Die Liste wurde erfolgreich angehängt.

Dateien

cs.h

Weitere Informationen

Schnittstelle für den stabilen Datenbereich – Dienste

Name

`cs_name_service_update_comp` – Aktualisierung einer Komponente

Aufruf

```
CS_RETURNCODE
cs_name_service_update_comp(CS_ADDRESS_PARAM
     *address_entry, long op);
```

Beschreibung

Aktualsieren einer Komponente. Die Selektor `op` bestimmt, welches Attribut geändert wird (`CS_UPDATE_ADDRESS`, `CS_UPDATE_PID`, `CS_UPDATE_HOST`, `CS_UPDATE_COMPONENT_STATE`). Der Parameter `address_entry` zeigt auf einen `CS_ADDRESS_PARAM` Entrag.

Rückgabewerte

`CS_RC_NAME_SERVICE_ERROR` – Fehler mit der Datenbankkommunikation.
`CS_RC_SUCCESS` – Die Änderung wurde erfolgreich vorgenommen.

Dateien

cs.h

Weitere Informationen

Schnittstelle für den stabilen Datenbereich – Dienste

Name

`cs_name_service_get_address` – physikalische Adresse einer Komponente

Aufruf

```
CS_RETURNCODE
cs_name_service_get_address(CS_ADDRESS_PARAM *address);
```

Beschreibung

Liefert die physikalische Adresse (`mbx_address`) und den Status (`componente_state`) der im Parameter `address` spezifzierten Komponente zurück.

Rückgabewerte

`CS_RC_NAME_SERVICE_ERROR` – Fehler mit der Datenbankkommunikation.
`CS_RC_SUCCESS` – Die Informationen wurden gefunden und sind gültig.

Dateien

cs.h

Weitere Informationen

Schnittstelle für den stabilen Datenbereich – Dienste

Name

`cs_name_service_lookup` – Suche nach Diensten

Aufruf

```
CS_RETURNCODE
cs_name_service_lookup(CS_SERVICE_NAME service,
     CS_RM_CLASS_NAME rm_class,
     CS_ADDRESS_PARAM **address_list);
```

Beschreibung

Suche nach dem Dienst `service` in der Tabelle "local_services". Der Parameter `rm_class` kann optional die Suche weiter einschränken. In `address_list` wird eine verkettete Liste aufgebaut, die explizit weder freigegeben werden muß.

Rückgabewerte

`CS_RC_NAME_SERVICE_ERROR`
– Fehler mit der Datenbankkommunikation oder keine Einträge gefunden.
`CS_RC_SUCCESS`
– Es wurden gültige Einräge in address_list zurückgeliefert.

Dateien

cs.h

Weitere Informationen

Netzwerkkommunikation

Name

`connect_tcp` – Verbindungsaufbau zu einem Server-Prozeß über TCP

Aufruf

```
int connect_tcp(char *remote_ip_address);
```

Beschreibung

Die Funktion `connect_tcp` alloziert einen Socket und nimmt Verbindung zum Rechner mit `remote_ip_address` auf. Der well-known Port ist über die Konstante `SERV_TCP_PORT` spezifiziert.
Bei Verbindungserfolg wird ein gültiger Socket-Deskriptor zurückgegeben.

Rückgabewerte

Gültiger Socket-Deskriptor oder im Fehlerfall -1.

Dateien

msgconf.h

Weitere Informationen

Name

`disonnect_tcp` – Verbindungsabbau zu einem Server-Prozeß

Aufruf

```
int disconnect_tcp(int sockfd);
```

Beschreibung

Schließen der über sockfd angegebenen TCP/IP Verbindung.

Rückgabewerte

Bei Erfolg 0, im Fehlerfall -1.

Dateien

msgconf.h

Weitere Informationen

Netzwerkkommunikation

Name

`send_message` – Transport einer Nachricht über TCP

Aufruf

```
long send_message(int sockfd, char unsigned *buffer,
     long len);
```

Beschreibung

Übertragen der in `buffer` angegebenen Nachricht auf dem Datenstrom `sockfd`. In `len` wird die Größe von `buffer` spezifiziert.
Die Funktion `send_message` erwartet im Zeitraum TIMEOUTSEND eine erfolgreiche Nachrichtenübertragung.

Rückgabewerte

Bei Erfolg wird die Anzahl der übertragenen Bytes zurückgegeben. Im Fehlerfall ein Wert <= 0.

Dateien

msgconf.h

Weitere Informationen

`receive_message (Netzwerkkommunikation)`

Netzwerkkommunikation

Name

`receive_message` – Empfang einer Nachricht

Aufruf

```
long receive_message(int sockfd, char **buffer);
```

Beschreibung

Emfang einer Nachricht über den Datenstrom `sockfd`. In `buffer` wird ein von `receive_message` allozierter Speicherbereich abgelegt. Die Größe des Speicherbereichs in `buffer` wird bei Erfolg zurückgegeben.

Der allozierte Speicherbereich in `buffer` muß explizit mit `free` wieder freigegeben werden.

Rückgabewerte

Bei Erfolg wird die Anzahl der empfangenen Bytes zurückgegeben. Im Fehlerfall ein Wert <= 0.

Dateien

msgconf.h

Weitere Informationen

`free()`
`send_message (Netzwerkkommunikation)`

Erklärung

Hiermit versichere ich, diese Arbeit selbständig verfaßt und nur die angegeben Quellen benutzt zu haben.

Michael Häuptle