

**UNIVERSITÄT  
BREMEN**



Entwurf und Implementierung einer  
Protokollagnostische erweiterbaren  
Mbus-Architektur zur Verteilung von  
IP-Telefonie- Routing-Informationen

Victor Wundersee<sup>1</sup>  
Fachbereich 3 Informatik  
Universität Bremen

04. Mai 2002

<sup>1</sup>vic@tzi.org



# Vorwort

Diese Arbeit befaßt sich mit der Auflösung von Adressen in der IP-Telefonie, einer noch recht jungen Technik, die das Potential hat, die normale Telefonie abzulösen. Dabei sollen schon bestehende Internet-Strukturen beibehalten werden, um die Akzeptanz zu erhöhen und die Ausbreitung der IP-Telefonie zu beschleunigen.

Zu Beginn der Arbeit war mir zwar *IP-Telefonie* ein Begriff, aber die genauen Begriffe und Komponenten bedurften zuerst einer eingehenden Einarbeitung. Dadurch wurde mir dann Bewußt, welche Vorteile diese Techniken in der Zukunft ermöglichen werden. Da die für diese Arbeit notwendigen Protokolle noch in der Entwicklung waren, kam es in der Implementierungsphase häufiger vor, daß Schnittstellen geändert oder angepaßt werden mußten. Letztendlich hat dies aber nur dazu beigetragen die Materie besser zu verstehen.

Dank der Unterstützung meines Betreuers Stefan Prella ist als Ergebnis der Arbeit das Programm **IP-Exchange** entstanden. Es tauscht Daten über die Erreichbarkeit von Telefon-Endstellen mit anderen Programmen aus und gibt bei Bedarf die für die Initialisierung eines Anrufes benötigten Informationen zurück. Es übernimmt damit innerhalb der IP-Telefonie die bisher nicht realisierte Funktion einer Vermittlungsstelle.

Victor Wundersee,  
Bremen, Mai 2002



# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>1</b>
1.1	Telefonie . . . . .	1
1.1.1	Das Telefon . . . . .	1
1.1.2	Geschichte der Vermittlung . . . . .	2
1.2	Das Internet . . . . .	3
1.2.1	Netzkopplung . . . . .	4
1.2.2	Datentransport . . . . .	5
1.3	IP-Telefonie . . . . .	6
1.4	Das Telefonielabor an der Universität Bremen . . . . .	9
1.5	Gliederung . . . . .	10
<b>2</b>	<b>Verwendete Protokolle</b>	<b>11</b>
2.1	Border-Gateway-Protokoll . . . . .	12
2.2	Telephony Routing over IP . . . . .	13
2.2.1	OPEN . . . . .	15
2.2.2	UPDATE . . . . .	16
2.2.3	KEEPALIVE . . . . .	17
2.2.4	NOTIFICATION . . . . .	17
2.2.5	Route Aggregation . . . . .	18
2.2.6	Zusammenfassung . . . . .	18
2.3	Verbreitung von E.164-Nummern über DNS (ENUM) . . . . .	19
2.4	Session Initiation Protocol . . . . .	20
2.5	ITU-T-Standard H.323 . . . . .	21
2.6	ITU-T-Standard H.225.0 Annex G . . . . .	24
2.6.1	Protokollablauf . . . . .	26
2.6.2	Auffinden von BorderElementen . . . . .	26
2.6.3	Adreßeinträge . . . . .	27
2.6.4	Annex G-Nachrichten . . . . .	28
2.6.5	Zusammenfassung . . . . .	31

2.6.5.1	Vergleich TRIP und Annex G . . . . .	31
2.7	IETF-Drafts zum Mbus . . . . .	32
2.7.1	Nachrichtenformat . . . . .	33
2.7.2	Adressierung . . . . .	33
2.7.3	Kommando-Syntax . . . . .	34
2.7.4	Parameter-Richtlinien . . . . .	35
2.7.5	Verwendung des Mbus . . . . .	35
2.7.6	Sicherheit . . . . .	35
2.8	Resümee . . . . .	36
<b>3</b>	<b>Vermittlung von Telefonaten</b>	<b>37</b>
3.1	Vermittlung in der Telefonie . . . . .	38
3.2	Vermittlung in der IP-Telefonie . . . . .	38
3.2.1	Adressenzuordnung mittels einer Datei . . . . .	40
3.2.2	Adressenzuordnung mittels TRIP . . . . .	41
3.2.3	Adressenzuordnung mittels H.323 . . . . .	42
3.2.4	Adressenzuordnung mittels Annex G . . . . .	42
3.2.5	Adressenzuordnung mittels DNS und SIP . . . . .	43
3.3	Schlußfolgerung . . . . .	43
<b>4</b>	<b>Gesamtentwurf von IP-Exchange</b>	<b>45</b>
4.1	Anforderungen an IP-Exchange . . . . .	46
4.1.1	Eintragen und Löschen von Telefonnummern . . . . .	46
4.1.2	Suche nach Telefonnummern . . . . .	47
4.1.3	Konfiguration . . . . .	48
4.1.4	Status-Informationen . . . . .	49
4.2	Der Aufbau einer gemeinsamen Schnittstelle . . . . .	49
4.3	Bereitgestellte Schnittstellen . . . . .	50
4.3.1	Konfiguration der Vermittlungsmodule . . . . .	51
4.3.2	Eintragen von Adressen . . . . .	51
4.3.3	Löschen von Adressen . . . . .	53
4.3.4	Anfrage von Adreßinformationen . . . . .	53
4.3.5	Statusabfrage des Vermittlungsmoduls . . . . .	54
4.3.6	Datenbank löschen . . . . .	54
4.4	Aufbau eines Vermittlungsmoduls . . . . .	55
4.5	Implementierung . . . . .	57
<b>5</b>	<b>Externe Schnittstellen von IP-Exchange</b>	<b>59</b>
5.1	Mbus-Nachrichten . . . . .	59
5.1.1	Hinzufügen und Löschen von Adressen . . . . .	61
5.1.2	Bestätigen von Adressen . . . . .	65

5.1.3	Suchen von Adressen . . . . .	65
5.1.4	Konfigurieren von Vermittlungsmodulen . . . . .	68
5.1.5	Übermitteln von Statusinformationen . . . . .	69
5.1.6	Beenden von IP-Exchange . . . . .	70
5.2	Schlußfolgerung . . . . .	71
<b>6</b>	<b>Der TRIP-Location Server</b>	<b>73</b>
6.1	Kommunikation mit IP-Exchange . . . . .	73
6.2	Datenklassen . . . . .	74
6.3	Programmablauf . . . . .	74
6.4	Das Programm ltest . . . . .	75
<b>7</b>	<b>Verwendung der IP-Exchange</b>	<b>81</b>
7.1	Systemanforderungen . . . . .	81
7.2	Interaktion mit dem Uni-Gatekeeper . . . . .	82
7.3	Installation . . . . .	82
7.4	Konfiguration . . . . .	82
7.5	Fehlerhandhabung . . . . .	83
<b>8</b>	<b>Zusammenfassung und Ausblick</b>	<b>85</b>
<b>A</b>	<b>Konfigurationsdateien</b>	<b>87</b>
A.1	Konfiguration der IP-Exchange . . . . .	87
A.2	Konfiguration des TRIP-Location Servers . . . . .	88
A.3	Mögliche Statusinformationen . . . . .	89
A.4	Mbus-Konfiguration . . . . .	89
<b>B</b>	<b>Verwendete Mbus-Nachrichten</b>	<b>91</b>
B.1	telephony.routing.add . . . . .	91
B.2	telephony.routing.add.return . . . . .	91
B.3	telephony.routing.remove . . . . .	91
B.4	telephony.routing.remove.return . . . . .	92
B.5	telephony.routing.query . . . . .	92
B.6	telephony.routing.query.return . . . . .	92
B.7	telephony.routing.configure . . . . .	92
B.8	telephony.routing.status . . . . .	92
B.9	telephony.routing.status.return . . . . .	92
B.10	telephony.routing.commit . . . . .	93
B.11	telephony.routing.commit.return . . . . .	93
B.12	mbus.quit . . . . .	93

B.13 mbus.byee . . . . . 93

# Abbildungsverzeichnis

1.1	Vermittlungsschrank . . . . .	1
1.2	Vermittlungszentrale anno 1881 . . . . .	2
1.3	Netztopologie - Routing . . . . .	4
1.4	Architektur in der IP-Telefonie . . . . .	7
2.1	Beispiel für mögliche Location Server-Verbindungen . . . . .	15
2.2	Ablauf des Verbindungsaufbaues . . . . .	16
2.3	Format der OPEN-Nachricht . . . . .	16
2.4	Möglicher Aufbau eines H.323-Anrufs . . . . .	22
2.5	vollvermaschte Administrative Bereiche . . . . .	24
2.6	Administrative Bereiche, sternförmig Angeordnet . . . . .	25
2.7	Administrative Bereiche, hierarchisch Angeordnet . . . . .	25
3.1	Handvermittlung um 1920 . . . . .	37
3.2	Mögliche Struktur der Telefonvermittlung . . . . .	39
3.3	Telefonvermittlung übers Netz . . . . .	41
4.1	Aufbau von IP-Exchange . . . . .	46
5.1	Mbus-Anbindung von IP-Exchange . . . . .	60
6.1	ITAD-Topologie mit Ring . . . . .	76
6.2	ITAD-Topologie mit abgetrennter ITAD . . . . .	77
6.3	ITAD-Topologie mit abgetrennter ITAD(2) . . . . .	78
6.4	ITAD-Topologie in zwei Teile getrennt . . . . .	79



# Tabellenverzeichnis

2.1	Das Format eines Deskriptors . . . . .	28
4.1	Schnittstelle zur Modul-Konfiguration . . . . .	51
4.2	Schnittstelle zum Eintragen von Adressen . . . . .	52
4.3	Rückgabewerte und ihre Bedeutung . . . . .	52
4.4	Mögliche Werte für Protokoll- und Familie-Felder . . . . .	53
4.5	Schnittstelle zum Löschen von Adressen . . . . .	54
4.8	Schnittstelle zur Löschung der Datenbank . . . . .	54
4.6	Schnittstelle zur Adreß-Suche . . . . .	55
4.7	Schnittstelle zur Statusabfrage . . . . .	55
4.9	Programm-Skelett für ein BorderElement . . . . .	57
4.10	Programm-Skelett für ein BorderElement (Fortsetzung) . . . . .	57
5.1	Hinzufügen und Löschen von Adressen - Nachrichtenformate . . . . .	61
5.2	Hinzufügen und Löschen von Adressen - Nachrichtenformat der Rückantwort . . . . .	63
5.3	Bestätigen von Adressen . . . . .	65
5.4	Bestätigen von Adressen . . . . .	65
5.5	Suchen von Adressen - Nachrichtenformat des Kommandos . . . . .	66
5.6	Suchen von Adressen - Nachrichtenformat der Rückantwort . . . . .	66
5.7	Konfiguration von Vermittlungsmodulen - Nachrichtenformat des Kommandos . . . . .	68
5.8	Übermitteln von Statusinformationen - Nachrichtenformat des Kommandos . . . . .	69
5.9	Übermitteln von Statusinformationen - Nachrichtenformat der Rückantwort . . . . .	69
5.10	Beenden von IP-Exchange . . . . .	70
5.11	Ende-Nachricht von IP-Exchange . . . . .	71
B.1	telephony.routing.add . . . . .	91
B.2	telephony.routing.add.return . . . . .	91
B.3	telephony.routing.remove . . . . .	91

B.4	telephony.routing.remove.return . . . . .	92
B.5	telephony.routing.query . . . . .	92
B.6	telephony.routing.query.return . . . . .	92
B.7	telephony.routing.configure . . . . .	92
B.8	telephony.routing.status . . . . .	92
B.9	telephony.routing.status.return . . . . .	92
B.10	telephony.routing.commit . . . . .	93
B.11	telephony.routing.commit.return . . . . .	93
B.12	mbus.quit . . . . .	93
B.13	mbus.bye . . . . .	93

# Kapitel 1

## Einleitung

### 1.1 Telefonie

Seit der Erfindung des Telefons wurde die Kommunikation über große Entfernungen ein wichtiger Bestandteil des (Geschäfts-) Lebens. Im Laufe der Zeit nahm das Gesprächsvolumen immer mehr zu, während die Anzahl der nutzbaren Leitungen - bzw. Funk-Frequenzen - ab einem bestimmten, noch nicht erreichten, Punkt an stagnieren muß (Funk), bzw. hohe Kosten verursacht (Leitungen). Eine bessere Ausnutzung der bestehenden Kapazitäten erscheint als einzige sinnvolle Lösung. In diesem Zusammenhang bietet sich die Nutzung bestehender Internet-Techniken und Infrastrukturen an, da nur wenige Anpassungen notwendig sind, um sie nutzen zu können.

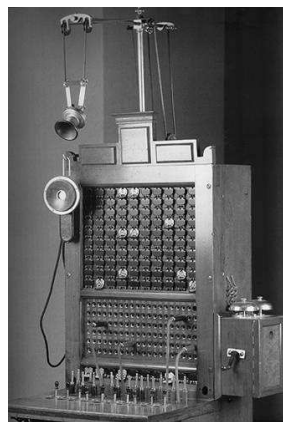


Abbildung 1.1: Vermittlungsschrank mit 100 Anschlüssen [23]

#### 1.1.1 Das Telefon

Das Telefon wurde Mitte des 19. Jahrhunderts erfunden. Die ersten Versionen hatten noch mit Problemen zu kämpfen und konnten sich nicht durchsetzen; dies gelang erst dem Telefon, das Alexander Graham Bell 1876 hat patentieren lassen. Seine Erfindung erreichte eine bessere Sprachqualität und brachte

dem Telefon den Durchbruch, und als Bell 1922 verstarb, waren in den USA bereits 14.374.000 Telefone in Betrieb. In Deutschland wurden 1877 die ersten Versuche mit Bell-Telefonen durchgeführt. Danach wurden ungefähr 200 Telefone pro Tag von der Firma Siemens hergestellt und 1897 gab es allein in Deutschland insgesamt 144.000 Fernsprecher an 529 Orten[1].

### 1.1.2 Geschichte der Vermittlung

Während sich die technischen Details bei den Fernsprechern im Laufe der Zeit nur unwesentlich änderten, waren die Änderungen in den Vermittlungsämtern deutlicher zu erkennen.

Mangels anderer technischer Möglichkeiten wurden Gespräche zunächst mittels Handvermittlung hergestellt. Dabei rief der Anrufer die Vermittlungsstelle, dem Ort an dem die Verbindung vom Anrufer zum Angerufenen hergestellt wird, an und nannte seinen Gesprächswunsch. Die Vermittlungsstelle rief daraufhin den gewünschten Teilnehmer an und verband die beiden Anschlüsse miteinander. Falls der Gesprächspartner in einem anderen Ort wohnte — und es eine Leitung dorthin gab — mußte die dortige Vermittlungsstelle ebenfalls eingeschaltet werden. Das Gesprächsende mußte der Vermittlungsstelle akustisch mitgeteilt werden.

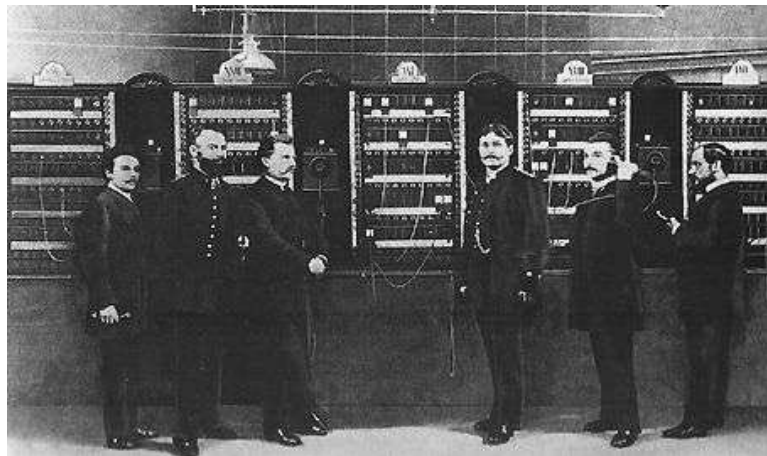


Abbildung 1.2: Vermittlungszentrale in Berlin um 1881 [23]

Mit steigender Teilnehmerzahl wurde dieses Verfahren immer unübersichtlicher. Die Schränke, zu denen die einzelnen Anschlüsse gelegt wurden, mußten

immer größer werden. Irgendwann reichte ein Schrank nicht mehr aus, und mehrere Schränke standen dann nebeneinander — und natürlich kam es dann vor, daß zwischen Schränken verbunden werden mußte (siehe Abbildung 1.2). Auch ließ diese Unübersichtlichkeit Platz für menschliche Bedienungsfehler, wie z.B. das Verbinden mit einem falschen Anschluß.

Aus diesen Gründen wurde nur wenige Jahre nach der Einführung der Handvermittlung eine erste Automatisierung des Verbindungsaufbaus eingeführt. Mit der neu eingeführten Wählscheibe konnten von einem Telefonapparat direkt andere Apparate angesprochen werden. Die Wählscheibe lieferte die notwendigen Impulse (Impulswahlverfahren) für den „Hubdrehwähler“ . Mit dem Hubdrehwähler konnten 100 Anschlüsse angewählt werden, bei mehr als 100 Anschlüssen mußten mehrere Hubdrehwähler hintereinander geschaltet werden.

Mit der Einführung von Transistoren und integrierten Schaltungen wurden die Hubdrehwähler langsam durch die elektronische Vermittlung abgelöst. Mit den rechnergesteuerten Wählsystemen und dem Tonwahlverfahren konnte eine größere Vermittlungsgeschwindigkeit erreicht werden als bisher.

## 1.2 Das Internet

Während der Verbreitung der Computer in der zweiten Hälfte des letzten Jahrhunderts entstanden Bemühungen, diese Rechner zum Datenaustausch über große Entfernungen zu verbinden. Im Laufe dieser Forschungen stellte sich heraus, daß die bis dahin verwendeten leitungsvermittelten Datenübertragungen Nachteile hatten. Bei leitungsvermittelten Datenübertragungen belegt jede Verbindung einen eigenen Kanal und verursacht entsprechende Kosten. Dabei ist es unwesentlich, ob der vorhandene Datenstrom den Kanal vollständig auslastet, oder sie kaum benutzt. Als günstiger wurden Verfahren befunden, bei denen Datenströme über einen gemeinsamen Kanal versendet werden können (Multiplexing), ohne dabei Kapazitäten zu reservieren, die sie nicht auslasten. Die Datenströme müssen zusätzlich in Pakete unterteilt werden, die dann einzeln verschickt werden. Dabei kann es dann aber vorkommen, daß die Pakete in einer anderen Reihenfolge beim Empfänger ankommen als der Sender sie verschickt hat oder verschwunden bleiben. Der Empfänger muß deshalb die Pakete selber sortieren und dem Sender fehlende Pakete mitteilen.

### 1.2.1 Netzkopplung

Das Internet war und ist ein Gebilde aus unterschiedlichen Netzen, in dem jedes Netz mit einem oder mehreren „Nachbarnetz(en)“ verbunden ist. Durch die Kopplung unterschiedlicher Netze mußte das Problem der Wegewahl zwischen den einzelnen Netzen gelöst werden. Der Weg vom Sender zum Empfänger kann dadurch viele Zwischenschritte erfordern. Auch muß jedes Netz entscheiden, welches der Nachbarnetze zum Empfänger führt bzw. den kürzesten oder billigsten Weg dorthin bereitstellt. Durch die Größe des Internets ist die Anzahl der möglichen Wege sehr groß geworden. Aus diesem Grund mußten Methoden gefunden werden, um eine Wegewahl (Routing) einfach und effizient zu gestalten. Spezielle Netzknoten — Router — suchen aus ihrem Datenbestand einen (möglichst optimalen) Weg zum Empfänger.

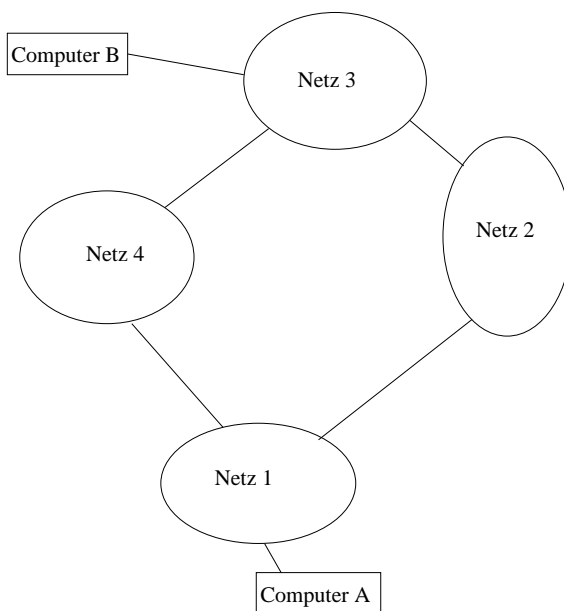


Abbildung 1.3: Mögliche Netztopologie

Wie in Abbildung 1.3 gezeigt wird, können mehrere Wege von Computer A zu Computer B führen (Netz 1—Netz 2—Netz 3 oder aber Netz 1—Netz 4—Netz 3). Jeder *Router* auf der Strecke kann entscheiden, welcher Router der nächste auf dem Weg zum Ziel ist. Wenn einer der Router ausfällt und der entsprechende Weg nicht mehr zur Verfügung steht, wird vom letzten ansprechenden Router ein anderer Weg ausgewählt.

### 1.2.2 Datentransport

Die Internet-Architektur setzt auf bereits bestehende (Teil-)Netze auf. Diese waren am Anfang lokal begrenzt und übersichtlich, aber oft wurden unterschiedliche Lösungen für Zugangsverfahren, Netztopologie oder Protokolle benutzt. Um diese Netze zu verknüpfen, mußten Möglichkeiten gefunden werden, die eine netzübergreifende Adressierung und Protokollumsetzung ermöglichten. Als Lösung wurde **IP** (Internet Protokoll) erdacht, das auch heute noch die Grundlage der Datenübertragung im Internet darstellt. Eine IP-Adresse besteht aus 32 Bits und kann weltweit eindeutig zugeordnet werden.

IP ist für die verbindungslose Datenübertragung ausgelegt und sieht nur die Versendung von Paketen (**Datagrammen**) vor. Eine Sequenzerhaltung ist nicht vorgesehen, d.h. die Reihenfolge der Pakete kann sich auf dem Weg zum Empfänger verändern, da die einzelnen Pakete unterschiedliche Wege zum Ziel nehmen und später versendete andere überholen können.

Sofern die Reihenfolge der Pakete wichtig ist und erhalten werden soll bietet sich **TCP** (Transmission Control Protocol) als Protokoll an. Hierbei wird zuerst zwischen Sender und Empfänger eine verbindungsorientierte Kommunikation initiiert. Der Empfänger bestätigt, welche Pakete er schon erhalten hat, damit kann der Sender nötigenfalls Pakete nochmals versenden. Der Empfänger kann auch zusätzlich angeben, wie viele Pakete er momentan empfangen bzw. verarbeiten kann, dies entspricht einer Flußkontrolle für die Pakete.

Bei mit **UDP** (User Datagram Protokoll), einem verbindungslosen Protokoll, gesendete Daten gibt es, im Gegensatz zu *TCP*, keine Garantie, daß sie beim Empfänger ankommen, da eine Benachrichtigung über den Empfang für den Sender nicht vorgesehen ist und eine Verbindung von Sender und Empfänger nicht existiert.

Als weitere Möglichkeit der Adressierung steht bei IP *Multicasting* zur Verfügung. Dabei werden IP-Pakete an einzelne Gruppen oder in das ganze Internet versendet. Jeder Empfänger teilt „seinem“ Router mit, für welche Gruppen er Pakete empfangen will. Der Sender wiederum schreibt in das Adreßfeld des Pakets an welche Gruppe er es verschicken will. Damit weiß der Sender zwar, welche Gruppe seine Pakete empfängt, aber er weiß nicht, wer alles zu den Gruppen gehört.

### 1.3 IP-Telefonie

IP-Telefonie benutzt die vorhandenen IP-Netze als Infrastruktur für die Übermittlung von Daten (z.B. Gespräche oder Video-Konferenzen). Während in der normalen Telefonie jeder Anruf einen Kanal belegt, wobei aus Kostengründen weniger Kanäle als Telefonapparate vorhanden sind, können in der IP-Telefonie durch die Verwendung von IP Daten in Pakete aufgeteilt und unter Benutzung von *Multiplexing* die Anzahl der benötigten Kanäle (und Kosten) reduziert werden.

Zusätzlich kann auf besondere Vermittlungstechnik verzichtet werden. Programme — im folgenden **Telefonie-Server** gekannt, die spezielle Protokolle wie z.B. **H.323**[10] oder **SIP**[11] (Session Initiation Protocol) benutzen, übernehmen die Funktion der komplexen und teuren Telefonvermittlung und bieten darüber hinaus weitere Funktionalitäten an.

Die Protokolle **H.323** und **SIP** erläutern, wie Sitzungen (z.B. Anrufe oder Videokonferenzen) zwischen Benutzern initialisiert und durchgeführt werden können.

Selbst wenn keiner der Teilnehmer direkten Zugang zu IP-Telefonen hat, können durch die Benutzung von Gateways (s.u.) Kosten gespart werden. Der Anrufer oder, falls es der Telefon-Provider anbietet, der *Telefonie-Server* entscheidet welches die kostengünstigste Variante des Telefongespräches ist, wählt ein Gateway in der Nähe aus, und dieses leitet das Gespräch über das Internet zu einem Gateway in der Nähe des Empfängers weiter. Es entstehen dann Kosten für 2 Ortsgespräche, was meistens billiger ist als ein Fern<sup>1</sup>- oder Auslandsgespräch.

Trotzdem wird es Zeit brauchen, bis die IP-Telefonie auf breiter Basis eingesetzt werden wird. Neben den bestehenden (und teilweise Begründeten) Vorurteilen, wie z.B. schlechtere Sprachqualität gegenüber normalen Telefonen, erzeugt die Neuanschaffung einer IP-Telefonanlage höhere Kosten bei der Anschaffung[26]. Zusätzlich werden bestehende Haustelesonanlagen kaum ersetzt, und wenn dann nur Schrittweise[25], um den Nutzen für das Gesamtsystem abschätzen zu können.

Deswegen werden — nach Einschätzung des Autors — IP-Telefone wohl primär dort eingesetzt, wo kurzfristig Kosten reduziert oder zu großen Teilen abgeschrieben werden können, z.B. bei Firmen mit großen Ausgaben für

---

<sup>1</sup>Außer in Deutschland, wo ein Ortsgespräch teilweise teurer ist als ein Ferngespräch.

Auslandsgespräche.

## Komponenten eines IP-Telefonsystems

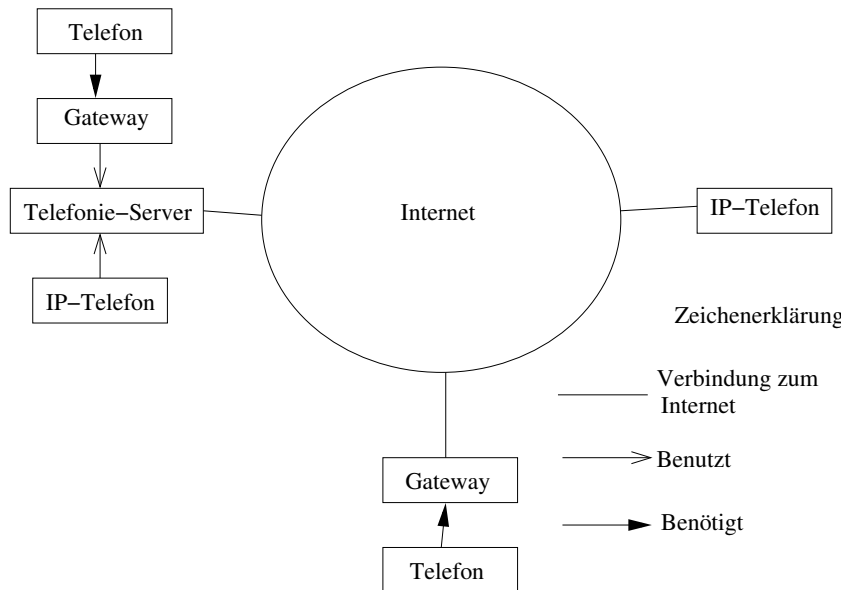


Abbildung 1.4: Mögliche Kombinationen von Komponenten in der IP-Telefonie

Ein IP-Telefonsystem besteht aus mehreren Komponenten, deren Zusammenarbeit in Abbildung 1.4 gezeigt wird.

Zum einen gibt es die *Telefon-Endpunkte*, die aus einem IP-Telefon oder einem Computer mit entsprechender Hard- und Software bestehen - er sollte Audiodaten senden, empfangen und wiedergeben können - und das Äquivalent zu einem normalen Telefon sind. Ähnlich wie bei *ISDN* können bei IP-Telefonie zusätzliche Dienste neben dem eigentlichen Telefonieren zur Verfügung gestellt werden, wie z.B. Videokonferenzen mit vielen Teilnehmern. Die Telefon-Endpunkte können entweder direkt ans Internet angeschlossen sein oder einen Telefonie-Server benutzen.

Zum anderen gibt es **Telefonie-Server**, die alle für ein IP-Telefonat notwendigen Informationen bereitstellen bzw. den Gesprächspartner lokalisieren und anrufen.

Die schon erwähnten *Gateways* ermöglichen es normale Telefone über das Internet anzusprechen.

## Adreßauflösung in der IP-Telefonie

Zwei Endpunkte können direkt (d.h. ohne weitere Komponenten) miteinander in Verbindung treten. Dabei muß der Anrufer aber vorher in Erfahrung bringen, wo sich der Gesprächspartner gerade aufhält und auf welcher Adresse er empfangsbereit ist. Dadurch kann auf Adreßänderungen nur langsam reagiert werden und es können nur Personen (bzw. Telefon-Endpunkte) angerufen werden, deren IP-Adresse man kennt. Diese Adresse kann sich aber schnell ändern, wenn z.B. die Person mit einem Laptop arbeitet und bei einem Arbeitsplatzwechsel sich die Adresse ebenfalls ändert. Aus diesem Grund müssen Adressen und Namen von Personen zu IP-Adressen aufgelöst werden. Diese Aufgabe wird von einem *Location Service* übernommen.

Die **ITU-T** (International Telecommunication Union - Telecommunication Standardization Sector) hat u.a. für diesen Zweck im, für die IP-Telefonie maßgebenden, Standard H.323 einen zentralen Telefonie-Server — einen sogenannten *Gatekeeper* — definiert.

Ein einzelner, zentraler Telefonie-Server, der weltweit alle Namen zu Adressen auflösen könnte, ist bei heutigen Stand der Technik nicht realistisch. Er wäre durch die Anfragen ständig überlastet und zusätzlich ein guter Angriffspunkt für „Hackerangriffe“. Aus diesem Grund werden Telefon-Endpunkte **Administrativen Bereichen** zugeordnet. Die Abgrenzung der Bereiche erfolgt dann durch logische Grenzen, z.B. die Universität Bremen oder auch alle Kunden eines Internet-Providers fallen in eigene Administrative Bereiche. In jedem Bereich, für den ein eigener Telefonie-Server zuständig ist, können sich mehrere IP-Telefone oder Computer mit IP-Telefon-Software befinden.

Die Kommunikation zwischen einem IP-Endgerät und einem „normalen“ Telefon erfordert allerdings eine weitere Komponente - ein **Gateway**. *Gateways* ermöglichen zum einen Telefonanrufe zwischen IP-Telefonen und PSTN-Telefonen (Public Switched Telephone Network). Es ist aber auch möglich, über *Gateways* mit dem Umweg über das Telefonnetz von einem Intranet in ein anderes Intranet zu telefonieren oder über den Umweg über das Internet von einem Telefonnetz in ein anderes.

Falls Telefon-Endpunkte sich bei einem Telefonat in unterschiedlichen *Administrativen Bereichen* kann es notwendig werden, daß die eingesetzten

Telefonie-Server miteinander kommunizieren, um die für das Gespräch notwendigen Adreßdaten auszutauschen. Im **Annex G** des Protokolls H.323-H.225.0 (siehe Kapitel 2.6) wird erklärt, wie Daten zwischen sogenannten *BorderElements* ausgetauscht werden können. Das **TRIP**-Protokoll (Telephony Routing over IP) bietet eine ähnliche Funktionalität an: Telefon-Routing-Informationen werden zwischen den zuständigen *Location Servern* ausgetauscht. Aber wie Telefonie-Server intern an diese Informationen gelangen sollen, wird zur Zeit noch erforscht. Auf lokaler Ebene schließt diese Arbeit die Lücke zwischen den Telefonie-Servern und den von den Protokollen bereitgestellten *BorderElements* und *Location Servern*.

## 1.4 Das Telefonielabor an der Universität Bremen

Im Fachbereich 3 der Universität Bremen ist in den letzten Jahren ein Telefonie-Labor entstanden. Mit Hilfe von wissenschaftlichen Projekten wurden Lösungen für Video-Konferenzen, die über den **Message Bus** (siehe Kapitel 2.7) initialisiert werden, entwickelt. Auch wurden Programme für IP-Telefonie entwickelt, die **H.323** (siehe Kapitel 2.5) und **SIP** (siehe Kapitel 2.4) implementieren und verwenden.

Bei diesen Programmen wurde darauf geachtet, daß die lokale Kommunikation mit anderen Programmen oder Komponenten über den *Message Bus* abläuft. Damit sind die Programme unabhängig, können aber nötigenfalls eine existierende Kommunikationsstruktur benutzen.

Zu diesen Programmen gehören unter anderem ein H.323-Gatekeeper[24] und ein *SIP-Proxy* als Telefonie-Server, ein H.323-Gateway für die Einbindung von normalen Telefonen, ein Media-Processor um Medienströme weiterzuleiten und ggf. umzucodieren und eine H.323-Endstelle. Allerdings bietet keines der entwickelten Programme die Möglichkeit Telefon-Routing-Informationen dynamisch zu verwalten und bei Bedarf an andere Komponenten weiter zu leiten.

Diese Lücke wird mit dem Programm *IP-Exchange*, das als Teil dieser Arbeit entwickelt wurde, geschlossen. Durch die Kommunikation mit anderen Programmen (in diesem Fall dem *Gatekeeper*) über den *Mbus* fügt sich *IP-Exchange*, neben einer Datenbank für die Verwaltung von Benutzerdaten und

einem Modul für das *Call Processing*, nahtlos in die vorhandenen Strukturen ein.

## 1.5 Gliederung

In der folgenden Gliederung wird ein kurzer Überblick über den Inhalt der einzelnen Kapitel gegeben.

- **Kapitel 2** beschreibt die verschiedenen Protokolle, Entwürfe und Drafts der Standardisierungsgremien **IETF** und **ITU-T**, die für diese Arbeit von Bedeutung sind.
- **Kapitel 3** stellt den Stand der Vermittlungstechnik in der „normalen“ Telefonie und der IP-Telefonie vor.
- **Kapitel 4** beschreibt die Anforderungen für die Kommunikation zwischen Vermittlungsmodulen und IP-Exchange. Die internen Schnittstellen von IP-Exchange und der Entwurf neuer Vermittlungsmodule werden erläutert.
- In **Kapitel 5** werden die Anforderungen, der Aufbau - und die Motivation - der externen Schnittstellen gezeigt, die das Programm IP-Exchange zur Verfügung stellt.
- IP-Exchange ist ohne ein Vermittlungsmodul nicht funktionsfähig. In **Kapitel 6** wird die Implementierung des *TRIP-Location Servers* beschrieben.
- In **Kapitel 7** wird gezeigt, wie IP-Exchange installiert wird und welche Konfigurationsmöglichkeiten es gibt.
- **Kapitel 8** stellt mögliche Erweiterungen für IP-Exchange vor und gibt, neben einer Zusammenfassung der Arbeit, einen kurzen Ausblick in die Zukunft.
- Im **Anhang** werden die Strukturen der verwendeten Konfigurationsdateien erläutert. Auch werden die verwendeten **Mbus**-Kommandos nochmals aufgelistet.

# Kapitel 2

## Verwendete Protokolle

In diesem Kapitel werden die Standards und Empfehlungen vorgestellt, deren Verwendung für den Entwurf des Programms **IP-Exchange** oder die für das Verständnis dieser Arbeit notwendig sind. Die Protokolle *BGP*, *TRIP*, *H.323*, *H.225.0 Annex G*, sind für den Entwurf des internen Systems wichtig, während der *Message Bus* für die Kommunikation nach außen notwendig ist. *E.164 Numbers and DNS* wird der Vollständigkeit halber erwähnt, da es eine Alternative zu *TRIP* darstellt.

IP-Exchange stellt eine Möglichkeit für die Kommunikation zwischen einem Telefonie-Server und einem Modul, das ein Vermittlungsprotokoll<sup>1</sup> implementiert, bereit. Damit wird es dann möglich den, für die Verbreitung von Telefon-Routing-Informationen benutzten, Protokollen Daten von anderen Programmen zu liefern. Zunächst werden, der Übersicht halber, die Standardisierungsgremien beschrieben, die die für diese Arbeit relevanten Protokolle verabschiedet bzw. empfohlen haben.

### Internationale Standardisierungsgremien

Die Kommunikation zwischen zwei oder mehreren Partnern (seien es nun Menschen oder Computer) kann nur funktionieren, wenn sich alle an Vereinbarungen - Protokolle - halten<sup>2</sup>. Während Menschen für ihre Kommunikation Protokolle sehr flexibel vereinbaren und auch verändern können, stehen Computer, mangels Abstraktionsvermögens, vor kaum überbrückbaren Pro-

---

<sup>1</sup>Ein Protokoll, bei dem die für die IP-Telefonie notwendigen Zuordnungen einer Telefonnummer zu einer IP-Adresse ausgetauscht werden

<sup>2</sup>Menschen z.B. sollten eine Sprache benutzen, die der Partner auch versteht bzw. spricht.

blemen.

Erst wenn alle Beteiligten — wie z.B. Computer- oder Telefonhersteller — dieselben Protokolle benutzen, ist eine Kommunikation zwischen den Rechnern in aller Welt möglich. Internationale Standardisierungsgremien bieten jedem Hersteller die Möglichkeit, seinen Wunsch (bzw. Bedarf) in ein Protokoll einfließen zu lassen. Dadurch, daß diese Protokolle dann öffentlich zugänglich sind, kann sie jeder in seiner Implementierung verwenden und von fremder Software unabhängig sein.

Die bedeutensten Gremien sind die **ITU** (International Telecommunication Union) und die **IETF** (Internet Engineering Task Force). Alle für diese Arbeit relevanten Protokolle wurden von diesen beiden Gremien verabschiedet.

## 2.1 Border-Gateway-Protokoll (BGP)

Das **Border Gateway Protocol**[2] definiert ein Routing-Protokoll für die Kopplung autonomer Systeme im Internet. Hauptaufgabe des Protokolls ist es, Informationen über die Erreichbarkeit von Netzen zwischen *Autonomen Systemen*, die den *Administrativen Bereichen* entsprechen, auszutauschen. Neben dieser Erreichbarkeitsinformation wird auch eine Liste verbreitet, welchen Weg durch die *Administrativen Bereiche* die Information genommen hat. Ein mögliches Zusammenfassen der Informationen (oder auch der Wegeliste) reduziert das Datenaufkommen.

*BGP* benutzt *TCP* als Übertragungsprotokoll, denn es ist zuverlässig, d.h. wenn ein Übertragungsfehler auftritt, wird er von *TCP* korrigiert und *BGP* muß sich darum nicht kümmern. Auch werden (im Prinzip) ausstehende Daten noch ausgeliefert, bevor eine Verbindung getrennt wird.

*BGP*-Knoten - dies können Router oder Computer, die mit Routern kommunizieren, sein - tauschen vier Arten von Nachrichten aus:

- Verbindungsaufbau (OPEN)
- Informationsaustausch (UPDATE)
- Verbindungsabbruch (NOTIFICATION)
- Verbindung aufrechterhalten (KEEPALIVE)

Die erhaltenen Informationen werden dann in Datenbasen gespeichert, wobei nach internen - Daten die aus dem eigenen *Administrativen Bereich* stammen - und externen - der Rest - getrennt wird. Jede Datenbasis wird einzeln gepflegt, und die Daten, die an andere *BGP*-Knoten versendet werden, stammen aus allen Datenbasen. Aus den gesammelten Daten wird eine **Routing-Tabelle** erzeugt, die dann für den *Administrativen Bereich* angewendet wird. Sie wird erst dann verändert, wenn neue Informationen zur Verfügung stehen oder alte Informationen ergänzt oder gelöscht werden. Aus dieser *Routing-Tabelle* wird der beste mögliche Pfad errechnet, der dann von den lokalen Routern benutzt wird.

*BGP* stellt ein Routing-Protokoll für IP dar. Während die hier beschriebenen Methoden benutzt werden, um Netz-Routing-Informationen zu verbreiten, benutzt **TRIP** die selben Methoden auf Applikationsebene für die Verbreitung von Telefon-Routing-Informationen.

## 2.2 Telephony Routing over IP (TRIP)

Das Protokoll TRIP (Telephony Routing over IP)[3] stellt Möglichkeiten zur Verfügung, um die Erreichbarkeit von Telefonie-Endstellen (das können z.B. (IP-)Telefone oder Computer sein) bekannt zu machen. Diese Informationen werden von Programmen wie z.B. **Gatekeeper** oder **Gateways** zur Verfügung gestellt, und werden zwischen **Location Servern** (LS) ausgetauscht. Ein *LS* (oder auch mehrere) ist dabei Bestandteil eines **Administrativen Bereiches**, bei *TRIP* ITAD (IP Telephony Administrative Domain) genannt. Dies ist ein logischer Zusammenschluß wie z.B. eine Institution oder Gesellschaft, bei denen Endbenutzer angemeldet sind. Die ITAD-Nummern werden fürs Internet eindeutig von der IANA (Internet Assigned Numbers Authority) vergeben. So können unter anderem die Deutsche Telekom (als Telefonbetreiber) oder auch die Universität Bremen (als Institution) eine eigene ITAD-Nummer bekommen. *Administrative Bereiche* können unterschiedlich groß sein, d.h. unterschiedlich viele Endbenutzer haben.

Die ausgetauschten Informationen enthalten unter anderem Angaben, unter welcher IP-Adresse die jeweilige Telefon-Endstelle zu erreichen ist - bzw. bei welcher IP-Adresse der zuständige *Gatekeeper* oder das zuständige *Gateway* zu finden ist. Auch werden Angaben gemacht, welche Protokolle der *Gatekeeper* oder das *Gateway* verwenden (können). *TRIP* kennt im Moment vier Vermittlungsprotokolle (wobei neue ohne Probleme eingebunden werden

können):

- H.323-H.225.0-Q.931
- H.323-H.225.0-RAS
- SIP
- H.323-H.225.0 *Annex-G*

Die Telefonnummern der Endstellen können von *TRIP* in dezimaler („0-9“, z.B. „0421307010“), pentadezimaler („0-E“, z.B. „0A234BE4“) oder in E.164-Notation („0-9“, z.B. „49421414428“) dargestellt werden.

Allerdings wird in *TRIP* nicht gesagt, wie ein *Location Server* Informationen über die Endpunkte des eigenen **Administrativen Bereiches** erhalten soll und auch nicht, wie ein *Gatekeeper* oder *Gateway* des **Administrativen Bereiches** Informationen vom *Location Server* erhalten sollen.

*TRIP* enthält viele Elemente vom **Border Gateway Protokoll** (*BGP*, siehe 2.1) wie z.B. Kommunikationsaufbau oder ähnliche Datenformate und Informationsattribute. Einige wenige Elemente von anderen Protokollen (wie z.B. **Open Shortest Path First**[4]) werden ebenfalls benutzt - diese Anteile sind jedoch so gering, daß auf eine Beschreibung dieser Protokolle verzichtet werden kann. Ein wesentlicher Unterschied zu *BGP* ist, daß **Administrative Bereiche** beliebig miteinander verknüpft werden können, während bei *BGP* jeder **Administrative Bereich** (bzw. jedes *Autonome System*) mit allen anderen verbunden sein muß.

Während bei *BGP* die, durch die Verteilung der Routing-Informationen entstandene, Topologie direkt angibt, über welchen Weg IP-Pakete versendet werden, ist dies bei *TRIP* nicht eindeutig. Es ist bei *TRIP* zwar ersichtlich, welchen Weg die Telefon-Routing-Informationen genommen haben, dieser wird aber nicht zwangsläufig für die Initialisierung des Telefongesprächs (bzw. für das Telefonat selber) benutzt. Die Topologie bei *TRIP* sagt vielmehr aus, welche der *Administrativen Bereiche* eine Zusammenarbeit vereinbart haben und ist auf einer anderen Ebene angelegt als bei *BGP*.

In Abb. 2.1 wird dargestellt, wie *Location Server* unterschiedlicher *Administrativer Bereiche* miteinander verbunden sein können. Denkbar sind auch ein Ring, bei denen jeder **Administrative Bereich** mit zwei anderen verbunden ist, oder auch ein Stern, bei dem ein ausgezeichnete **Administrativer**

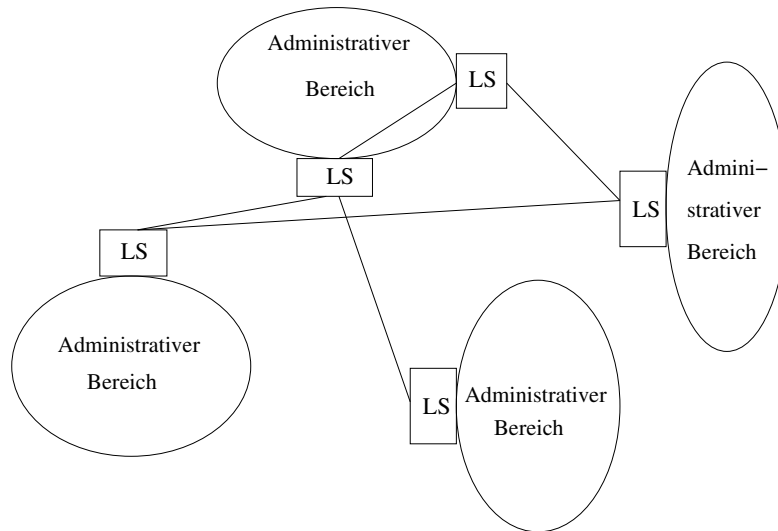


Abbildung 2.1: Beispiel für mögliche Location Server-Verbindungen

**Bereich** mit allen anderen verbunden ist, diese aber nicht untereinander.

Im folgenden werden die von *TRIP* verwendeten Nachrichtenformate vorgestellt.

### 2.2.1 OPEN

Mit einer **OPEN**-Nachricht (siehe Abbildung 2.2) zeigt ein *Location Server* einem anderen *LS*, daß er eine Kommunikationsverbindung herstellen will. In der Nachricht (siehe Abbildung 2.3) gibt der *LS* unter anderem an, welche Version des *TRIP*-Protokolls er versteht, zu welchem **Administrativen Bereich** er gehört oder auch einen Bezeichner, der ihn innerhalb seines *Administrativen Bereiches* eindeutig identifiziert. Die Angabe von optionalen Parametern ermöglicht es, die Kommunikation zwischen zwei *Location Servern* auf Routing-Information bestimmter Protokolle einzugrenzen, bzw. mitzuteilen, ob der *LS* Daten nur empfangen, nur senden oder beides will.

Falls der Partner-*LS* ebenfalls eine Kommunikation wünscht, schickt er als Antwort eine **OPEN**-Nachricht, in der seine Parameter versendet werden. Wenn ein oder mehrere Parameter für ihn nicht akzeptabel sind, wird er eine **NOTIFICATION**-Nachricht zurücksenden, um die Gründe der Ablehnung zu erläutern.

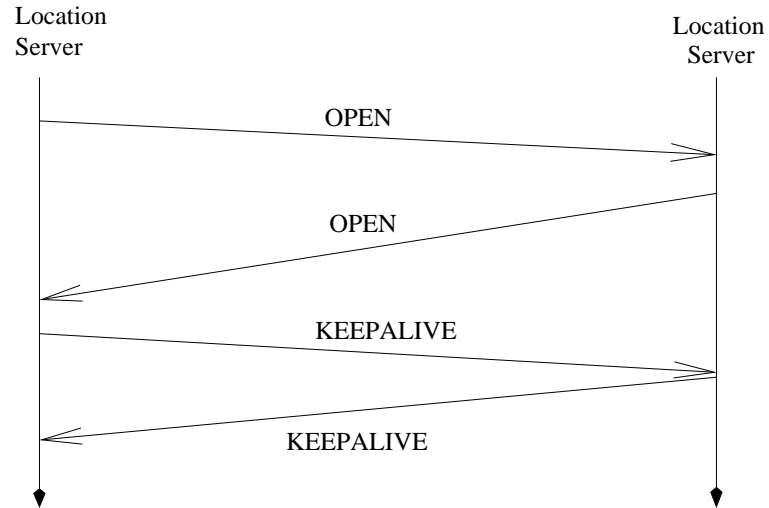


Abbildung 2.2: Ablauf des Verbindungsaufbaues

Version	Reserviert	Hold Time
ITAD		
TRIP Identifier		
Length of Optional Parameters	Optional Parameters	
Route-Types supported Send/Receive		

Abbildung 2.3: Format der OPEN-Nachricht

### 2.2.2 UPDATE

Die **UPDATE**-Nachricht wird vom *LS* benutzt, um den Verbindungspartnern mitzuteilen, welche Telefonnummern bekannt sind bzw. welche Änderungen sich in der Datenbasis seit der letzten **UPDATE**-Nachricht ergeben haben. Dabei können dann mehrere Telefonnummern in einer Nachricht zusammengefaßt werden, wenn sie vom selben **Administrativen Bereich** stammen und den selben Weg bis zum *LS* genommen haben. Die Nachricht kann die zu löschenden oder neu einzutragenden Routing-Informationen enthalten. Zusätzlich muß dann aufgeführt werden, welchen Weg diese Informa-

tionen gegangen sind (d.h. durch welche *Administrativen Bereiche* sie weitergereicht wurden), bei welchen *Administrativen Bereichen* sie verändert wurden und welcher Server weitere Anfragen zu den Telefonnummern beantworten kann. Zusätzlich — oder als Alternative — können *UPDATE*-Nachrichten an interne *LS* noch eine Liste aller Verbindungen zu anderen internen *Location Servern* enthalten.

*TRIP* ist erweiterbar, so daß neue Formen von Informationen leicht hinzugefügt werden können - diese aber die Funktion älterer Implementierungen nicht behindern.

**TRIP** unterscheidet beim Versenden von **UPDATE**-Nachrichten zwischen 'lokalen' und 'externen' Kommunikationspartnern. Lokale Kommunikationspartner sind im selben **Administrativen Bereich** gemeldet wie der *LS*, und erhalten **UPDATE**-Nachrichten sofort, sobald der *LS* sie erhält. Dabei werden die Nachrichten nicht bearbeitet (wie z.B. in Abschnitt 2.2.5). Externe Kommunikationspartner sind in anderen **Administrativen Bereichen** gemeldet und erhalten **UPDATE**-Nachrichten erst, nachdem der *LS* sie in seiner Datenbank eingetragen und evtl. bearbeitet (z.B. Routen-Aggregation) hat. Zusätzlich, um das Volumen der versendeten Daten zu reduzieren, läßt jeder *LS* ein vordefiniertes Zeitintervall verstreichen, bevor er mit einer *KEEPALIVE*-Nachricht das Bestehen der Verbindung verlängert.

### 2.2.3 KEEPALIVE

Die **KEEPALIVE**-Nachricht wird vom *LS* versendet, um Kommunikationspartnern mitzuteilen, daß die Verbindung noch besteht und auch weiterhin bestehen soll. Die **KEEPALIVE**-Nachricht muß spätestens am Ende der, in der **OPEN**-Nachricht vereinbarten, **Hold Time** beim Partner eingetroffen sein, sonst wird die Verbindung unterbrochen. Bei dieser Nachricht müssen keine Daten verschickt werden, allein der Nachrichtentyp reicht als Information aus.

### 2.2.4 NOTIFICATION

Die **NOTIFICATION**-Nachricht stellt einen Mechanismus bereit, mit dem ein *LS* einem Partner mitteilen kann, daß ein Fehler in der gemeinsamen Kommunikation entdeckt wurde. Mögliche Fehler sind unbekannte oder inkompatible Parameter in **OPEN** oder **UPDATE**-Nachrichten oder das Ablaufende des **KEEPALIVE**-Timers. Auch ein expliziter Abbauwunsch kann

so übertragen werden. Hauptbestandteil der Nachricht ist der Grund aus dem die Verbindung abgebrochen wird. Falls möglich wird der unbekannte oder fehlerhafte Teil der **OPEN**- bzw. **UPDATE**-Nachricht mitgeschickt.

### 2.2.5 Route Aggregation

Unter **Route Aggregation** versteht man das Zusammenfassen von bekannten Telefonnummern zu neuen, kürzeren Telefonnummern. Es ist eine wichtige Funktion, mit der bei *TRIP* die Datenmenge reduziert werden soll.

Dabei wird bei bestimmten Bedingungen eine neue Telefonnummer erzeugt, die eine (oder mehrere) Endstelle(n) kürzer ist als die Ursprungsnummern. Dazu müssen dem *Location Server* die entsprechenden, je nach Adreßfamilie unterschiedlich viele, Telefonnummern bekannt sein. Während der Generierung der neuen Telefonnummer werden die vorhandenen Informationen, die der *LS* durch **UPDATE**-Nachrichten über die Nummern erhalten hat, zusammengefaßt und wenn möglich gekürzt.

Wenn ein *LS* z.B. alle Telefonnummern von 31200 bis 31299 kennt, reicht es aus, wenn er die Telefonnummer 312 verbreitet - die letzten beiden Stellen werden weggelassen. Dadurch muß nur noch eine Nummer gesendet werden und nicht 100. Wenn der *LS* noch größere Nummernbereiche abdeckt (z.B. einen ganzen Ort wie Berlin) können noch mehr Datentransfers eingespart werden. Da jede Telefonnummer der Welt eindeutig ist (durch Landes- und Ortsvorwahl) besteht auch nicht die Gefahr von Verwechslungen.

### 2.2.6 Zusammenfassung

*TRIP* ermöglicht es, Telefon-Routing-Informationen zu verbreiten. Dabei ist es nicht auf ein Signalisierungsprotokoll begrenzt, sondern ist in der Lage, Routing-Informationen beliebiger Signalisierungsprotokolle zu versenden die sich alle ergänzen können — falls eine Adresse über mehrere Protokolle angesprochen werden kann, hat der Anrufer die Wahl, welches er benutzen will. Allerdings werden keine zusätzlichen Informationen (wie z.B. Kosten) übergeben, und es gibt keine Mechanismen, um in der Datenbasis nicht vorhandene Informationen zu bekommen.

## 2.3 Verbreitung von E.164-Nummern über DNS (ENUM)

In [20] und [21] werden Methoden beschrieben, wie durch Transformation von *E.164*-Nummern in DNS-Namen (**ENUM**) die Möglichkeit gegeben wird, dynamisch — über das DNS-System — das Vorhandensein einer Telefonnummer, und der für diese Nummer möglichen bzw. zuständigen Dienste, zu überprüfen

Dabei werden von einer *E.164*-Telefonnummer die Ziffern genommen und in umgekehrter Reihenfolge und als Adresse aufgeschrieben, z.B. wird aus „4212184711“ „1.1.7.4.8.1.2.1.2.4.e164.arpa“ erzeugt. Die Deklaration der Telefonnummer als Teile der „e164.arpa“-Domain bietet sich an: es ist ein funktionierendes System, das weltweit arbeitet und kurze Antwortzeiten hat. Der Ausfall eines *Name-Servers* bedingt nicht das Verschwinden der Informationen.

Bei der Initialisierung eines IP-Telefonats kann dann nach dem Wählen einer *E.164*-Telefonnummer der eigene Telefonie-Server die Nummer umwandeln, und mit einer DNS-Anfrage in Erfahrung bringen, welcher Telefonie-Server für die gewählte Telefonnummer zuständig ist. Falls die Anfrage kein Resultat liefert ist die Telefonnummer nicht erreichbar, ansonsten kann die Initialisierung fortgesetzt werden. Dabei kann die DNS-Anfrage zusätzliche Informationen wie z.B. EMail-Adressen oder Adressen von Fax-Apparaten liefern. Es wird also möglich unterschiedliche Medien unter einer einzigen Telefonnummer anzusprechen.

Sowohl *TRIP* wie auch *ENUM* bieten Möglichkeiten Telefonnummern im Internet bekannt zu machen. Bei *TRIP* werden Telefon-Routing-Informationen zwischen *Administrativen Bereichen* ausgetauscht. Dabei können die Betreiber der einzelnen *Administrativen Bereiche* entscheiden, mit wem Informationen getauscht werden sollen. Jede einzelne Information enthält Angaben, über welches Protokoll und unter welcher Adresse eine Telefonnummer zu erreichen ist. Bei *ENUM* werden aus Telefonnummern Einträge einer (bestimmten) Domain erzeugt. Wenn diese Einträge aufgelöst werden enthält das Ergebnis eine Liste von zuständigen Servern der einzelnen Medien.

Während bei *ENUM* Änderungen in den Einträgen nur langsam weitergegeben werden, sind sie bei *TRIP* innerhalb kurzer Zeit überall bekannt.

Welches — wenn überhaupt — Protokoll sich durchsetzen wird ist nicht ab-

zusehen, denn beide Protokolle bieten die selbe Funktionalität mit teilweise unterschiedlichen Diensten an, die nicht in Konkurrenz zu einander stehen.

## 2.4 Session Initiation Protocol

**SIP**[11] ist ein Protokoll, um Sessions (Sitzungen) zwischen zwei oder mehr Kommunikationspartnern zu erzeugen, modifizieren und zu beenden. Die Sitzungen können (unter anderem) Konferenzen oder Telefonate übers Internet sein.

Folgende Komponenten werden innerhalb eines SIP-Systems benutzt:

- **Endpunkt**

Ein Endpunkt, normalerweise ein Computer mit entsprechender Software oder ein Telefon, kann durch *SIP* Gespräche initialisieren oder von anderen Endpunkten angesprochen werden. Jeder Endpunkt kann durch eine oder mehrere Adressen angesprochen werden. Diese Adressen werden als *SIP-URLs* dargestellt, die einer URL-Adresse („user@host“) weitestgehend entspricht. Dabei kann der „user“ eine Person oder auch eine Telefonnummer sein, während der „host“ eine IP-Adresse oder ein Domain-Name sein kann.

- **Registrar**

Ein Registrar-Server akzeptiert Anmeldungen von Endpunkten und verwaltet diese für *Proxy-* und *Redirect-Server*. Ohne die Angaben des Registrars können keine Adreßanfragen aufgelöst werden, deswegen ist er — obwohl ein eigenständiger Dienst — meist bestandteil von *Proxy* und *Redirect-Servern*.

- **Redirect-Server**

Ein Redirect-Server akzeptiert Adreßanfragen und bildet die angefragte *SIP*-Adresse, wenn nötig, auf andere Adressen ab, die er dann zurückgibt. Der Anrufer muß dann den Gesprächspartner bei den neu erhaltenen Adressen suchen. Anrufe können vom Redirect-Server nicht entgegengenommen werden.

- **Proxy-Server**

Der Proxy-Server verwaltet die einzelnen Endpunkte in einem *SIP*-System. Dieser übernimmt dann Wegewahl und Adreßauflösung, was der Funktionalität des Telefonie-Servers entspricht.

*SIP* ist Text-basiert, wie z.B. *HTML*, und die versendeten Daten sind von Menschen lesbar, allerdings werden dadurch die versendeten Nachrichten größer als bei Binär-basierten Protokollen (wie z.B. *H.323*).

Wenn eine Endstelle einen Anruf tätigt, versucht sie den einen *SIP*-Server anzusprechen, der unter dem „host“-Teil der Adresse zu finden ist, dabei wird „sip.host“ als Adresse für den Server empfohlen. Wenn unter dieser Adresse kein *SIP*-Server gefunden werden kann, ist der Anruf nicht möglich. Nachdem der Server gefunden wurde, schickt der Anrufer eine oder mehrere Nachrichten an ihn und erhält eine oder mehrere Antworten darauf. Mögliche Nachrichten sind z.B. das Anmelden der eigenen Adresse(n) (REGISTER) oder auch Einladungen für ein Gespräch (INVITE), die dann alle notwendigen Informationen für den Aufbau des Gespräches enthalten.

Da *SIP* davon ausgeht, daß eine Person mobil ist und sich an verschiedenen Arbeitsplätzen aufhalten kann, ist es notwendig die Person genau so lokalisieren. Dafür können Dienste wie *finger*[13] oder Protokolle wie *TRIP* genutzt werden. Dabei kann es dann vorkommen, daß die Person bei mehreren Stellen gleichzeitig angemeldet ist. Der *SIP*-Server erhält dann eine Liste der möglichen Telefon-Endstellen und kann sie der reihe nach oder auch parallel Ausprobieren.

Sobald der gewünschte Kommunikationspartner gefunden wurde, schickt der *SIP*-Server diesem nun die Einladung mit einer Beschreibung der gewünschten Session, in der die erlaubten Medien und Formate aufgeführt sind. Falls der Gesprächspartner der Session zustimmt, werden seine erlaubten Medien und Formate zurückgeschickt. Nach dem Gespräch (oder der Konferenz) wird eine Ende-Nachricht gesendet, um dem *SIP*-Server das Gesprächsende anzuzeigen.

## 2.5 ITU-T-Standard H.323

Das Protokoll *H.323*[10] erläutert wie (Multimedia-) Kommunikation über IP-Netze, unter anderem Audio- und Video-Kommunikation, abgehandelt werden soll. Dabei stellt aber dieses Protokoll nur den Rahmen für weitere Protokolle (wie z.B. *H.225.0*[12], *H.245*[14], *H.261*[17], *G.711*[16] oder auch *T.120*[15]) und erläutert wie diese miteinander zusammenarbeiten sollen. Dabei ist es nicht notwendig, daß immer alle Protokolle vorhanden sind. Zur Benutzung eines IP-Telefones würden Protokolle ausreichen, die für die Rea-

lisierung des Anrufes und die Kodierung der Gesprächsdaten notwendig sind: H.225.0 und Q.931 (Anrufrealisierung) sowie G.711 (Audio-Daten).

### H.323-Komponenten

Im Protokoll werden auch die einzelnen Komponenten eines H.323-Systems beschrieben und wie deren Zusammenarbeit aussehen soll. Neben dem **Gatekeeper**, der die Funktion des Telefonie-Servers übernimmt, der dann alle Geräte innerhalb seines *Administrativen Bereichs* verwaltet, gehören dazu noch die Telefon-Endpunkte. Ein **Gateway** ermöglicht es, Verbindungen zwischen verschiedenen Netzen (z.B. IP und ISDN) oder Signalisierungsprotokollen (wie *SIP*) aufzubauen. Abbildung 1.4 auf Seite 7 zeigt die möglichen Kombinationen von Komponenten.

Für Konferenzen werde zusätzliche, optionale Komponenten benötigt. Der **Multipoint Controller** steuert die Konferenz und handelt mit allen Teilnehmern die verwendeten Protokolle aus. Ein **Multipoint Processor** ist, sofern er vorhanden ist, dafür zuständig eingehende Medienströme zu mischen und diese an die Teilnehmer der Konferenz weiterzuleiten. Bei Bedarf wandelt er sie zusätzlich von einem Protokoll in ein anderes um.

### Ablauf von Telefonaten bei H.323

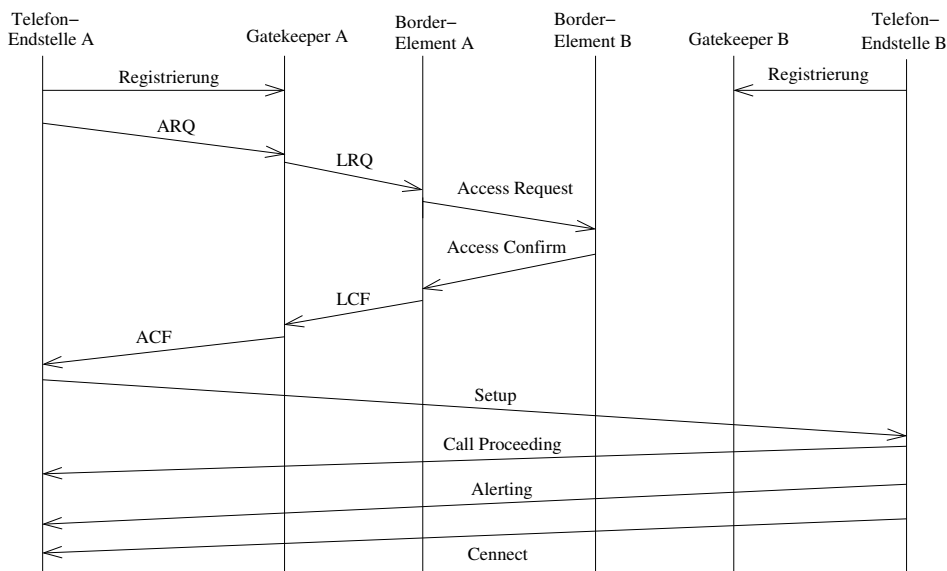


Abbildung 2.4: Möglicher Aufbau eines H.323-Anrufs

Ein Anruf unter *H.323* ist in mehrere Phasen gegliedert. Abbildung 2.4 zeigt einem möglichen Anruf, bei dem sich die Telefon-Endpunkte in unterschiedlichen *Administrativen Bereichen* befinden. Im Normalfall melden — bzw. suchen — sich Telefon-Endstellen bei Betriebsbeginn beim zuständigen Gatekeeper an. Bei einem Gesprächswunsch von Endstelle A zu Endstelle B beantragt die Endstelle beim Gatekeeper A die Berechtigung für den Anruf (ARQ).

In dieser Anfrage steht dann, wer angerufen werden soll und welche Bandbreite für den Anruf notwendig ist. Die Adresse des Gesprächspartners kann dabei unterschiedliche Formate haben (wie z.B. Telefonnummer oder URL-Adresse) und muß vom Gatekeeper aufgelöst werden. Falls der Gatekeeper — wie in diesem Fall — die Adresse nicht selber auflösen kann, muß er selber die Adresse auflösen lassen, z.B. durch ein *BorderElement*, das in Kapitel 2.6 vorgestellt wird (LRQ). Das *BorderElement* fragt dann selber bei anderen *BorderElementen* nach, ob sie die Adresse auflösen können (Access Request) und erhält so IP-Adresse von Telefon-Endstelle B (Access Confirm). Das *BorderElement* gibt diese Information an den Gatekeeper zurück (LCF) und dieser wiederum gibt dann die Berechtigung für diesen Anruf und teilt Telefon-Endstelle A die IP-Adresse von B mit (ACF).

Sobald die Telefon-Endstelle A die Adresse des Gesprächspartners vom Gatekeeper erfahren hat wird versucht, dessen Telefon-Endstelle (Setup) zu erreichen. Diese sendet eine Nachricht (Call Proceeding) zurück, um anzuzeigen, das der Aufbauwunsch angekommen ist und das Gespräch aufgebaut wird, sobald der Gesprächspartner bereit ist. Nach einem Signal (Alerting) wird dann das Gespräch aufgebaut (Connect). Dann wird ausgehandelt, wie die Gesprächsdaten kodiert werden und dann das Gespräch gestartet. Es gibt verschiedene Alternativen, wie Gespräche ablaufen können, z.B. können die jeweiligen Gatekeeper entscheiden, daß nur sie die Gesprächsdaten an die Telefon-Endstellen weiterleiten dürfen.

**SIP** und **H.323** stellen, obwohl von verschiedenen Gremien beschlossen, eine ähnliche Funktionalität zur Verfügung. Dabei schließen sie sich aber nicht gegenseitig aus. Durch die Benutzung eines *Gateways* oder einer Telefon-Endstelle, die beide Protokolle unterstützt, ist eine Kommunikation zwischen beiden möglich.

## 2.6 ITU-T-Standard H.225.0 Annex G

*Annex G*[5] ist ein Anhang des Protokolls H.225.0[12]. Während im Protokoll erläutert wird, wie Audio-, Video-, Anwendungs- und Steuerinformationen verwendet und kodiert werden, um zwischen Telefon-Endpunkten übertragen zu werden, ist die Aufgabe des Anhangs Methoden zu beschreiben, wie Adressen aufgelöst werden sollen, wie Zugriffskontrolle ermöglicht werden kann und wie Berichte über die Benutzung zwischen *Administrativen Bereichen* ausgetauscht werden können.

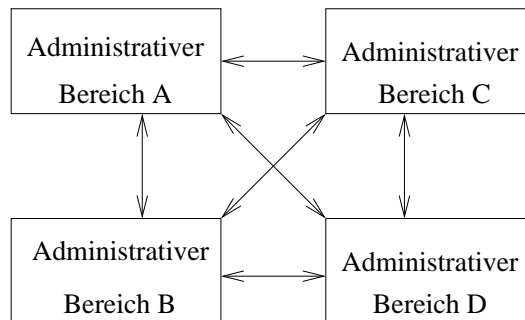


Abbildung 2.5: Administrative Bereiche, vollvermascht

**Administrative Bereiche** sind Teile von einem H.323-Netz und benutzen für die Kommunikation mit anderen *Administrativen Bereichen* **BorderElemente**, wobei nicht verlangt wird, daß die einzelnen *Administrativen Bereiche* ihre internen Strukturen offenlegen. Die *Administrativen Bereiche*, bzw. deren *BorderElemente*, können unterschiedlichen Modellen angeordnet werden. Neben einer vollvermaschten Struktur, bei der jeder *Administrative Bereich* von allen angeschlossenen Bereichen Informationen bekommt (siehe Abbildung 2.5) gibt es noch hierarchische (Abbildung 2.7) und sternförmige Modelle (Abbildung 2.6). Des weiteren kann ein **Clearing House** (siehe Kapitel 2.6.2) die Funktion einer zentralen Adreßauflösung übernehmen und eine Sternstruktur aufweist.

Im allgemeinen speichert ein *BorderElement* Informationen zu den Adressen, dies können IP-Adressen, Telefonnummern oder auch URL-Adressen (z.B. ‚prelle@tzi.org‘) sein, und deren Erreichbarkeit innerhalb seines *Administrativen Bereiches*, dies sind Informationen darüber, ob der Anruf sofort getätigt werden kann oder beim zuständigen Gatekeeper beantragt werden muß. Dadurch werden die unterschiedliche Methoden, einen Anruf zu vermit-

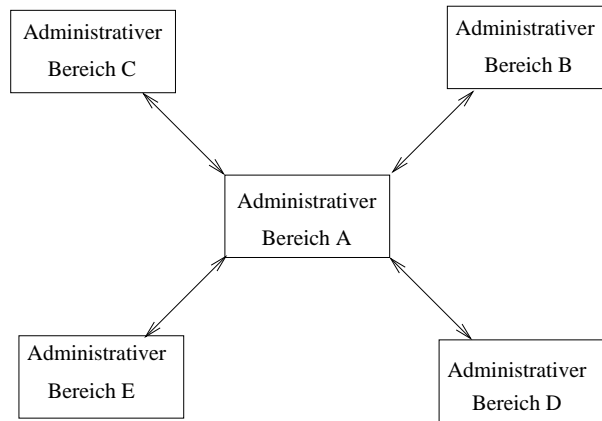


Abbildung 2.6: Administrative Bereiche, sternförmig Angeordnet

teln, wie Vermittlung durch einen *Gatekeeper* oder eine direkte Verbindung, unterstützt.

Diese Informationen werden mit anderen *BorderElementen* ausgetauscht, wobei jedes steuern kann, welche der eigenen Informationen bekannt gegeben werden. Zusätzlich verfügbare Informationen innerhalb der *Administrativen Bereiche* ermöglichen die Bestimmung der am besten geeigneten Route für den Anruf.

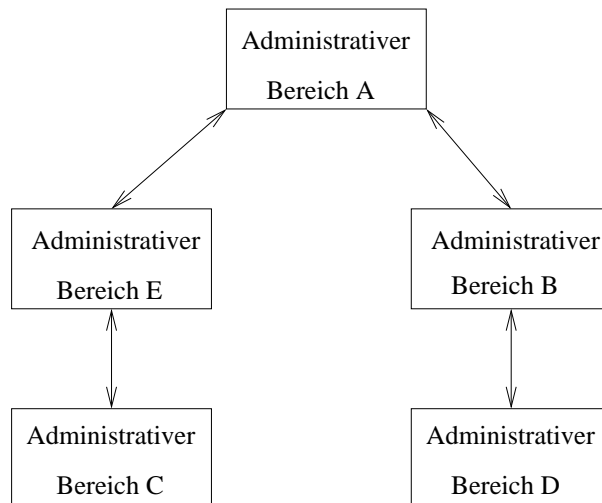


Abbildung 2.7: Administrative Bereiche, hierarchisch Angeordnet

Obwohl *Annex G* davon ausgeht, daß die *BorderElemente* mit jeder anderen H.323-Komponente (wie z.B. Gatekeeper oder Gateway) zusammenarbeiten können, ist die Kommunikation mit diesen Komponenten nicht Bestandteil des Anhangs. Hier wird nur die Kommunikation zwischen den *BorderElementen* beschrieben.

Im folgenden wird beschrieben, wie die Kommunikation zwischen *BorderElementen* abläuft, wie andere *BorderElemente* gefunden werden können, wie Adreßeinträge und Deskriptoren aufgebaut sind, und welche Nachrichten von den *BorderElementen* akzeptiert werden.

### 2.6.1 Protokollablauf

Im Normalfall versucht ein **BorderElement** (*BE*) eine Verbindung zu anderen *BE* aufzunehmen (*Service Request*). Sobald diese zustande gekommen ist, kann der **BE** bei Bedarf (z.B. einem Anruf) nach Adreßinformationen fragen und erhält im Gegenzug einen Deskriptor, der die gewünschte Information enthält. Falls sich ein Adreßeintrag ändert, wird der **BE** vom Kommunikationspartner benachrichtigt, und kann den entsprechenden Deskriptor direkt anfordern. Ein *BE* kann auch schon vorsorglich Deskriptoren bei anderen *BE* anfordern und sie bis zum Bedarfsfall zwischenspeichern.

### 2.6.2 Auffinden von BorderElementen

Damit *BorderElemente* unterschiedlicher *Administrativer Bereiche* Deskriptoren mit Adreßeinträgen untereinander austauschen können, müssen sie in Erfahrung bringen können, wo die Deskriptoren der anderen *Administrativer Bereiche* zu finden sind. Dafür gibt es folgende Möglichkeiten:

- **Kommunikation zwischen BorderElementen**

Bei dieser Kommunikation wird zwischen den *Administrativen Bereichen* Verbindungen vereinbart. Zwischen den beteiligten *BE* können dann Deskriptoren ausgetauscht werden.

- **Kommunikation über ein Clearing House**

Diese Kommunikation stellt eine Alternative zur Kommunikation zwischen *BE* dar. Jeder *BE* schickt seine Deskriptoren an ein *Clearing House* und dieses baut ein globales Verzeichnis auf — allerdings ist nicht abzusehen, wie viele Ressourcen dies benötigt.

- **URL-Adressen**

Für URL-Adressen gibt es auch eine dynamische Variante durch das DNS-System: wenn die Adresse `rupp@tzi.org` angerufen werden soll und die IP-Adresse `_h2250-annex-g.udp.tzi.org` existiert, kann der Anruf zum Gesprächspartner durchgeführt werden.

## Adreßauflösung

Wenn eine Adresse innerhalb des eigenen Administrativen Bereiches angefragt wird, kann das *BorderElement* die vorhandenen Adreßeinträge durchsuchen. Je nachdem, welche Informationen im Adreßeintrag stehen, kann ein Anruf sofort getätigt werden oder muß bei der zuständigen Stelle zuerst angefordert werden, z.B. da deren Kapazitäten stark begrenzt sind. Dabei werden exaktere Einträge, mit denen der Anruf sofort getätigt werden kann, bevorzugt. Wenn mehrere Adreßeinträge als gleichwertig erkannt werden, werden sie alle in die Rückantwort eingefügt. Wenn keine sofortige Verbindung erlaubt ist, muß das *BorderElement* eine Verbindung beim zuständigen *Gatekeeper* beantragen (**Access Request**) und dann an den Anfrager weiterleiten.

Wenn eine Adresse aus einem anderen Administrativen Bereich angefragt wird, werden - analog zur lokalen Adreßsuche - die vorhandenen Adreßeinträge durchsucht. Wird eine Adresse mit sofortigem Verbindungsaufbau gefunden, kann sie an den Anfrager zurückgegeben werden. Wird aber nur ein Eintrag gefunden, bei dem eine Verbindung beantragt werden muß, kann das *BorderElement* entweder die Verbindung beantragen oder den Adreßeintrag an den Anfrager zurücksenden.

### 2.6.3 Adreßeinträge

Adreßeinträge werden in *Annex G* in sogenannten **Address Templates** definiert. Dort sind neben der Adresse und deren Aliase, z.B. vom Telefon-Endgerät gelieferte E.164-Alias-Nummern, auch Kosteninformationen aufgeführt. Adreßeinträge werden in **Deskriptoren** zusammengefaßt. Änderungen eines Adreßeintrags — wie z.B. Änderung der Preisinformationen — innerhalb eines *Deskriptors* erfordern dann eine Änderung dieses *Deskriptors*. Ein Administrativer Bereich verbreitet dann die Adreßeinträge, die er auflösen kann.

Tabelle 2.1: Das Format eines Deskriptors

descriptorInfo	descriptorID	Ein eindeutiger Bezeichner, der es ermöglicht, diesen Deskriptor aus einer Menge ähnlicher Deskriptoren zu erkennen
	lastChanged	Datum und Zeit, zu der der Deskriptor zuletzt geändert wurde
templates	Eine Menge von Einträgen, die definieren, welche Adressen dieser Deskriptor auflösen kann	
gatekeeperID	Beschreiber, der den Besitzer des Deskriptors identifiziert (z.B. der Gatekeeper, der die Nachricht erzeugt hat)	

Ein Adreßeintrag ist wie folgt aufgebaut:

- **Pattern** Eine Liste von Adressen im Telefonnummern- oder URL-Format
- **RouteInfo** Eine Liste von Routing-Informationen für diesen Adreßeintrag
- **TimeToLive** Anzahl von Sekunden, für die dieser Eintrag gültig ist.

Ein *BorderElement* erhält Adreßeinträge durch statische Konfiguration, als Teil von Deskriptoren anderer *BorderElemente* oder als Antwort auf eine gezielte Anfrage.

#### 2.6.4 Annex G-Nachrichten

Es gibt eine Vielzahl von *Annex G*-Nachrichten, die verschiedene Reaktionen beim Empfänger auslösen können. Die Anforderung eines Deskriptors (Descriptor Request) kann als Antwort den gewünschten Deskriptor enthalten (Descriptor Confirmation) oder die Anforderung kann abgelehnt werden (Descriptor Rejection). Im folgenden werden die einzelnen Nachrichten und die möglichen Antworten kurz beschrieben.

- **Service Request:** Ein *BorderElement* beantragt mit dieser Nachricht eine einseitige Verbindung, d.h. daß bei dieser Verbindung nur er Anfragen (Requests) bei einem anderen *BE* stellt. Teil der Nachricht sind die

zu verwendenden Sicherheitsmechanismen. Mit **Service Confirmation** bestätigt der Empfänger, daß die Verbindung aufgebaut ist. Wenn schon vorher eine Verbindung zwischen diesen beiden *BE* bestanden hat, gelten nun die Parameter der aktuellen Anforderung. Mit **Service Rejection** zeigt der Empfänger an, daß die aktuelle Anforderung abgelehnt worden ist, und nennt den Grund hierfür. Falls vorher schon eine Verbindung bestanden hat, wird diese fortgeführt.

- **Service Release:** Jeder der beiden beteiligten *BE* kann die Verbindung mit dieser Nachricht beenden.
- **Descriptor ID Request:** Diese Nachricht fordert eine Liste der vorhandenen Deskriptoren beim Empfänger an. Mit **Descriptor ID Confirmation** sendet der Empfänger die Liste der vorhandenen Deskriptoren, mit **Descriptor ID Rejection** schickt er die Gründe, warum die Liste nicht versendet wurde.
- **Descriptor Request:** Ein *BorderElement* beantragt mit dieser Nachricht beim Empfänger einen (oder mehrere) Deskriptoren, die er über *Descriptor ID Request* erfahren hat. Mit **Descriptor Confirmation** sendet dieser die gewünschten Deskriptoren zurück, mit **Descriptor Rejection** wird der Grund genannt, warum der (die) Deskriptor(en) nicht versendet wird.
- **Descriptor Update:** Diese Nachricht zeigt an, daß sich Adreßinformation in einem Deskriptor (oder mehreren) geändert haben. Der Empfänger sollte dem Sender den Erhalt der Nachricht mit **Descriptor Update Acknowledgement** bestätigen und bei Bedarf den Deskriptor dann neu anfordern.
- **Access Request:** Mit dieser Nachricht wird beim Empfänger die Adreßauflösung für eine bestimmte Adresse beantragt. Die gewünschten Adreßinformationen werden mit **Access Confirmation** geschickt. Wenn die Adresse nicht aufgelöst werden konnte, werden die Gründe mit **Access Rejection** gesendet. Falls die Adreßauflösung länger dauert als für den Normalfall definiert, kann mit **Request in Process** angezeigt werden, daß die Antwort später kommt.
- **Non Standard Request:** Mit dieser Nachricht können Dienste beantragt werden, die nicht in *Annex G* definiert werden. Mit **Non Standard Confirmation** werden sie bestätigt bzw. ausgeführt und mit **Non Standard Rejection** abgelehnt.

- **Unknown Message Respond:** Falls einer der *BE* eine ihm unbekannte Nachricht erhalten hat, sollte er den Vorfall mit dieser Nachricht melden. Bekannte Nachrichten mit falschen Parametern sollten mit den entsprechenden Nachrichten beantwortet werden.
- **Usage Request:** Mit dieser Nachricht beantragt der Sender *Usage Indication*-Nachrichten, die einen bestimmten Anruf betreffen. Mit **Usage Confirmation** teilt der Empfänger mit, daß er zustimmt und entsprechende Nachrichten versenden wird. Mit **Usage Rejection** wird angezeigt, daß der Antrag abgelehnt wird.
- **Usage Indication:** Diese Nachricht enthält Informationen zu einem bestimmten Anruf. Mit *Usage Indication Confirmation* zeigt der Empfänger, daß er die *Usage Indication* akzeptiert hat, mit **Usage Indication Rejection**, daß er die *Usage Indication* ignorieren wird.
- **Validation Request:** Diese Nachricht kann von einem *BorderElement* verschickt werden, das einen Anruf beenden will und bei einem anderen *BE* überprüfen will, ob der Ursprung des Anrufes gültig war, d.h. der Anrufer der ist, der er vorgibt zu sein. Der Empfänger kann mit **Validation Confirmation** die Gültigkeit des Anrufes bestätigen - dann kann der Anruf beendet werden, oder mit **Validation Rejection** anzeigen, daß der Anruf nicht gültig ist.

Im folgenden werden die Informationsfelder aufgeführt, die Bestandteil aller Nachrichten sind.

- **sequenceNumber** Jede Anfrage hat eine eindeutige Nummer. Wenn auf die Anfrage geantwortet wird, sollte diese Nummer mit angegeben werden.
- **replyAddress** An diese Adresse sollte die Antwort auf eine Anfrage geschickt werden.
- **version** Die Protokollversion des Absenders der Nachricht.
- **hopCount** Die Anzahl der *BorderElements*, über die diese Nachricht weitergeleitet werden darf. Wenn die maximale Zahl erreicht ist, schickt das letzte *BorderElement* die bei ihm vorhandenen Informationen zurück. Sollten keine Informationen vorhanden sein, sollte die Nachricht abgelehnt werden. Wenn der **HopCount** einer Anfrage abgelaufen ist, kann das *BorderElement* die Verbindungsanfrage-Nachricht nicht an

andere *BorderElemente* weiterleiten. Es kann dann aber alle zutreffenden Adreßeinträge zurücksenden. Falls keine Einträge vorhanden sind, muß der Anfrager über den Grund der Ablehnung in Kenntnis gesetzt werden.

- **integrityCheckValue** Eine Checksumme über die verschlüsselte Nachricht
- **tokens** Daten, die für die gewünschte Operation notwendig werden können.
- **cryptoTokens** Verschlüsselte Tokens
- **nonStandard** Nicht standardisierte Informationen

Der Teil der Nachrichten, der zwischen *BorderElementen* ausgetauscht wird betrifft IP-Exchange nicht. Adressen, die der Telefonie-Server verbreiten will werden über IP-Exchange an das lokale *BorderElement* weitergegeben, und IP-Exchange gibt dem Telefonie-Server die Informationen, die für den Aufbau eines Gesprächs nötig sind, wenn dieser bei IP-Exchange nachfragt.

## 2.6.5 Zusammenfassung

*Annex G* ermöglicht es Telefonie-Servern Telefon-Routing-Informationen zu Adressen zu bekommen, die sie nicht selber auflösen können. Dabei können neben den eigentlichen Informationen auch noch z.B. Tarifinformationen bekannt gemacht werden. Informationen, die nicht in der Datenbasis gefunden werden, können bei Bedarf gesucht werden. *Annex G* kann nur Routing-Informationen von Protokollen verbreiten, die in H.225.0 beschrieben werden.

### 2.6.5.1 Vergleich TRIP und Annex G

Mit *Annex G* können zu jeder Adresse mehr Informationen übermittelt werden als mit *TRIP*, auch müssen die einzelnen *BorderElemente* keine große Datenbasis aufbauen, da bei Bedarf nach notwendigen Informationen gesucht werden kann, während bei *TRIP* eine Suche nicht möglich ist. Allerdings müssen bei *Annex G* die Routing-Informationen von den Protokollen stammen die in H.225.0 beschrieben werden.

## 2.7 IETF-Drafts zum Mbus

In Kooperation mit dem *University College London* (UCL) wurde in der *AG Rechnernetze* eine Schnittstelle und ein Protokoll entwickelt, mit deren Hilfe ein besonderes Konzept für Konferenz-Software verwirklicht wird: jede Anwendung besteht aus mehreren Komponenten oder Modulen. Diese werden aber nicht zu einem Programm zusammengefügt, sondern laufen gleichzeitig als eigenständige Prozesse ab.

Für die Zusammenarbeit zwischen den Modulen und Komponenten muß eine geeignete Kommunikationsmöglichkeit gefunden werden, da Funktionsaufrufe bei eigenständigen Prozessen, die jeder ihren eigenen Adreßraum haben, nicht mehr möglich sind. Dafür wird auf der lokalen Ebene der **Message Bus** (Mbus) verwendet.

Der *Mbus* ist eine lokale, nachrichtenorientierte Infrastruktur für Kommunikation innerhalb einer Gruppe von Endstellen. Er stellt Funktionen für das Auffinden einzelner Endpunkte, Adressierungen nach dem Betreff und zuverlässigen Nachrichtentransport bereit. *Mbus* benutzt eine IP-Multicast-Gruppe für die Kommunikation zwischen den Endpunkten.

Die Dokumente [6], [7] und [8] beschreiben zum einen Aspekte der Adressierung, des Transportes und der Sicherheit und zum anderen Syntax und Semantik der gemeinsamen Nachrichten.

*MBus* wurde für die Kommunikation zwischen Endstellen entwickelt, aber es stellte sich im Laufe der Zeit heraus, daß der Mbus auch für andere Anwendungen, wie Infrastruktur-Komponenten nutzbar ist. Das Programm *IP-Exchange* z.B. verwendet für die Kommunikation mit Telefonie-Servern *Mbus*-Nachrichten. Als Einschränkung daraus ergibt sich, daß die Kommunikation von *IP-Exchange* mit Telefonie-Servern über den *Mbus* erfolgen muß, d.h. das jeder Telefonie-Server entweder selber eine *Mbus*-Kommunikation oder ein Modul dafür implementieren muß. Da *Mbus* für die lokale Kommunikation ausgelegt ist, müssen sich *IP-Exchange* und Telefonie-Server in einem solchen Umfeld befinden.

Im Folgenden wird erläutert, wie *Mbus*-Nachrichten aufgebaut werden.

### 2.7.1 Nachrichtenformat

Eine Mbus-Nachricht besteht aus einem Nachrichtenkopf und dem Inhalt der Nachricht. Im Kopf stehen Informationen, die anzeigen wie und wohin eine Nachricht gesendet werden soll. Alle Nachrichten des Mbus sollten in UTF-8 kodiert sein, so können ASCII-Nachrichten ohne Umkodierungen übertragen werden. Folgende Informationen stehen im Nachrichtenkopf.

- Ein **Message Digest**, der eine Prüfsumme der gesamten Nachricht enthält.
- Ein Protokollkennzeichner (**ProtocolID**).
- Eine fortlaufende **Laufnummer**. Jede Mbus-Applikation numeriert ihre Nachrichten in aufsteigender Reihenfolge bei 0 beginnend.
- Ein **Zeitfeld** enthält die verstrichenen Millisekunden seit *epoch*<sup>3</sup>
- Der **Nachrichtentyp**, der angibt, ob die Nachricht bestätigt werden muß
- Die **Senderadresse**
- Die **Empfängeradresse**
- Eine **Bestätigungsliste** von Laufnummern, die durch diese Nachricht bestätigt werden.

### 2.7.2 Adressierung

Jedes Mbus-Modul wird durch ein n-Tupel identifiziert, wobei jede Komponente des Tupels als *Schlüssel/ Wert*-Paar dargestellt wird. Mögliche Schlüssel sind **conf**, **media**, **module**, **app** und **instance**. Die möglichen Werte hängen vom verwendeten Schlüssel ab und können beispielsweise wie folgt aussehen: (media:control module:engine app:exchange id:17881-0@134.102.218.68).

Daten werden beim Mbus über lokales Multicast bzw. Unicast versendet. Aus diesem Grund muß jede Anwendung, die mittels des Mbus kommunizieren will, auf einer definierten Adresse horchen und eigene Daten entweder direkt an andere lokale Anwendungen (d.h. an andere Ports auf demselben Computer) verschicken oder Multicast-Pakete versenden. Unicast-Pakete werden

---

<sup>3</sup>1. Januar 1970

nur verwendet, wenn eine Nachricht an eine schon bekannte Anwendung am Mbus gesendet werden soll.

Jedes Mbus-Modul empfängt und beantwortet alle Nachrichten, bei denen die Empfängeradresse einer Teilmenge der Modul-Adresse entspricht. Das Modul mit der obigen Adresse würde z.B. auf Nachrichten mit folgenden Empfängeradressen reagieren: (`media:control app:exchange`), (`module:engine id:178810@134.102.218.68`), (`app:exchange`) oder ()

### 2.7.3 Kommando-Syntax

Nach dem Kopf der Nachricht folgen zeilenweise Kommandos mit folgender Syntax:

**kommando ( parameter parameter ...)**

Jedes Kommando folgt einer Konvention in der Namensgebung: die Benennung der Kommandos soll eine Hierarchie bilden, damit die Zusammengehörigkeit verwandter Kommandos deutlich wird. Der Punkt "." wird hierbei als Trennzeichen benutzt.

Neben allgemeinen Kommandos, die von allen Mbus-Applikationen verstanden werden sollen, kann jedes Modul eigene Kommandos implementieren, die für die Funktion des Moduls notwendig sind. Die folgende Aufstellung stellt die möglichen Kommandoklassen kurz vor:

- **Remote Commands** werden benutzt, um eine asynchrone Operation beim Zielsystem auszulösen. Der Kommandoname ist mit einer bestimmten Operation beim Empfänger verbunden. Die mitgeschickten Parameter können dann vom Empfänger interpretiert werden.
- **RPC-style Commands** erlauben es, eine Operation beim Zielsystem auszulösen, und sollten benutzt werden, wenn eine Rückantwort erwartet wird, die Parameter zurückliefert.
- **Transactions** sind den *Remote Commands* insofern ähnlich, als daß sie ebenfalls eine Operation beim Zielsystem auslösen. Allerdings sendet das Zielsystem eine Empfangsbestätigung zurück, und wartet dann darauf, daß der Sender das Ausführen der Aktion bestätigt oder abbricht.
- **Properties** erlauben es einem, Prozeßinformationen vom Empfänger zu erhalten. Nach dem Versenden des *Property*-Namens wird, ähnlich

wie bei RPC-Commands, eine Rückantwort geschickt, in der der Wert der gewünschten Property steht.

- **Event notifications** werden benutzt, falls eine Mbus-Entity regelmäßig Kommandos verschickt, um Zustandveränderungen anzuzeigen.
- **Contexts** erzeugen zwischen *Mbus*-Entities eine Beziehung, die über die Lebensdauer einzelner Kommandos hinaus bestehen bleibt. Dadurch können einzelne Kommandos (wie RPC's) durch den Kontext unterschieden werden.

### 2.7.4 Parameter-Richtlinien

Folgende Richtlinien sollten befolgt werden, wenn Parameter für Mbus-Kommandos spezifiziert werden:

- Jeder Parameter sollte einem der wohldefinierten Parameter-Typen entsprechen
- Homogene Listen (Listen, die nur aus einem Typ bestehen) sollten durch die Angabe des Typs spezifiziert werden (z.B. Liste von String)
- Bei heterogenen Listen (Listen, die aus mehreren Typen bestehen) sollten alle Elemente eine Typangabe beinhalten
- Wenn optionale Parameter erlaubt sind, sollten sie im letzten Parameter zusammengefaßt sein und eine Angabe der möglichen Typen beinhalten

### 2.7.5 Verwendung des Mbus

Wie schon erwähnt, wird der Mbus vor allem bei Konferenz-Systemen eingesetzt, und ein Großteil der Kommandos dienen der Anruf- und Mediensteuerung. Da aber eigene Kommandos definiert werden können, läßt sich der Message Bus auch für andere Anwendungen als flexibles Medium der Kommunikation verwenden.

### 2.7.6 Sicherheit

Da der Mbus lokales Multicast verwendet, ist es im Prinzip möglich, neue Anwendungen zu starten, die einen laufenden Mbus belauschen können.

Um das Abhören von Mbus-Nachrichten zwischen Anwendungen zu verhindern bzw. um Geheimhaltung zu ermöglichen werden die Nachrichten, die auf dem Mbus versendet werden, normalerweise verschlüsselt. Vertraulichkeit wird durch den Einsatz verschiedener Verschlüsselungsverfahren ermöglicht.

Da alle Programme, die von einem Anwender gestartet werden, einen gemeinsamen Schlüssel haben, kann mit Hilfe des **Message Digest** einer Nachricht kann dann überprüft werden, ob Absender oder Inhalt verändert wurden.

## 2.8 Resümee

IP-Exchange führt Kommunikation in zwei Richtungen. Die externe Kommunikation stellt die Verbindung zu einem Telefonie-Server her. Da Telefonie-Server eigenständige Programme sind, bietet sich dafür die Verwendung des *Message Bus* an. Der Kommunikationsablauf und die entworfenen *Mbus*-Nachrichten werden in Kapitel 5 beschrieben.

Die interne Kommunikation bindet die von IP-Exchange benötigten Module (bzw. Programmteile) ein, die Telefon-Routing-Informationen sammeln, die IP-Exchange später, bei Bedarf, an den Telefonie-Server weiterzugeben. Im Moment sind mögliche Protokolle für die Informationsgewinnung — mangels Alternativen — *TRIP* und *Annex G*. Die Beschreibung der für die Kommunikation definierten Schnittstellen ist in Kapitel 4 zu finden.

# Kapitel 3

## Vermittlung von Telefonaten

Wie schon in Kapitel 1.1.2 erwähnt, müssen Verbindungen zwischen Telefonen hergestellt werden, damit Gespräche geführt werden können. Diese Verbindungen sind flexibel gehalten, damit nicht jedes Telefon eine Leitung zu allen anderen benötigt. Dadurch muß in einer Vermittlungsstelle der Appa-

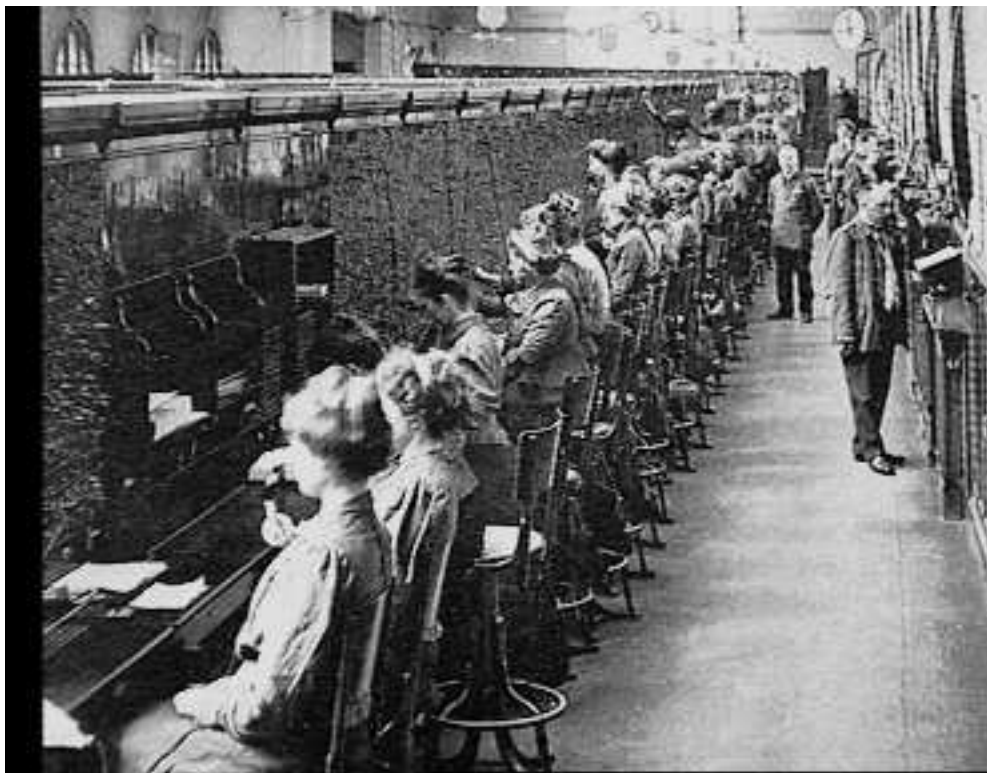


Abbildung 3.1: Die Fräulein vom Amt - um 1920[22]

rat des Gesprächspartners ermittelt und die Verbindung aktiviert werden. In diesem Kapitel wird kurz erläutert wie, das Auffinden des Gesprächspartners in der heute üblichen Telefonie und der IP-Telefonie vor sich geht.

### 3.1 Vermittlung in der Telefonie

Das (weltweite) Telefonnetz ist ein hierarchisches System mit dem Endgerät (Telefonapparat) auf der untersten Ebene[22]. Das Endgerät ist direkt mit der Ortvermittlungsstelle (OVS) verbunden - für große Orte sind mehrere miteinander vernetzte Ortsvermittlungsstellen notwendig. Die *OVS* sind auch mit Knotenvermittlungsstellen verbunden, die wiederum mit Haupt- und Zentralvermittlungsstellen kommunizieren (siehe Abb. 3.2). Anrufe in eine benachbarte *Ortsvermittlungsstelle* können dann ohne die Einschaltung der Zentralvermittlungsstelle abgehandelt werden.

Internationale Vermittlungsstellen — als oberste Ebene des Systems — stellen dann Verbindungen zu anderen Ländern her. Diese Hierarchie ist allerdings nicht zwingend, in Bereichen mit geringer Bevölkerungsdichte (und somit relativ wenigen Telefonen) kann eine Ebene von Vermittlungsstellen entfallen.

Welche Vermittlungsstellen bei einem Anruf benötigt bzw. eingesetzt werden, hängt von der gewählten Telefonnummer ab. Fängt die Telefonnummer nicht mit einer **0** an, ist der Anruf ein Ortsgespräch und die Ortsvermittlungsstelle — je nach Größe des Ortes können dies auch Knotenvermittlungsstellen sein — kann die gewünschte Verbindung herstellen. Ansonsten sagt die 3, 4 oder 5-stellige Vorwahl, welche Zentral- und Hauptvermittlungsstelle für den Anruf zuständig ist. Auslandsgespräche werden initiiert, wenn die gewählte Telefonnummer mit **00** anfängt - dann wird sofort die Internationale Vermittlungsstelle der eigenen Zentralvermittlungsstelle aufgerufen.

### 3.2 Vermittlung in der IP-Telefonie

Die IP-Telefonie hat prinzipiell die selben Aufgaben und Probleme wie die „normale“ Telefonie. Allerdings werden andere Lösungen dafür verwendet. Im Gegensatz zur üblichen Telefonie-Welt kann sich jede IP-Telefon-Endstelle — und mit der Verwendung von *Gateways* auch jedes normale Telefon — dy-

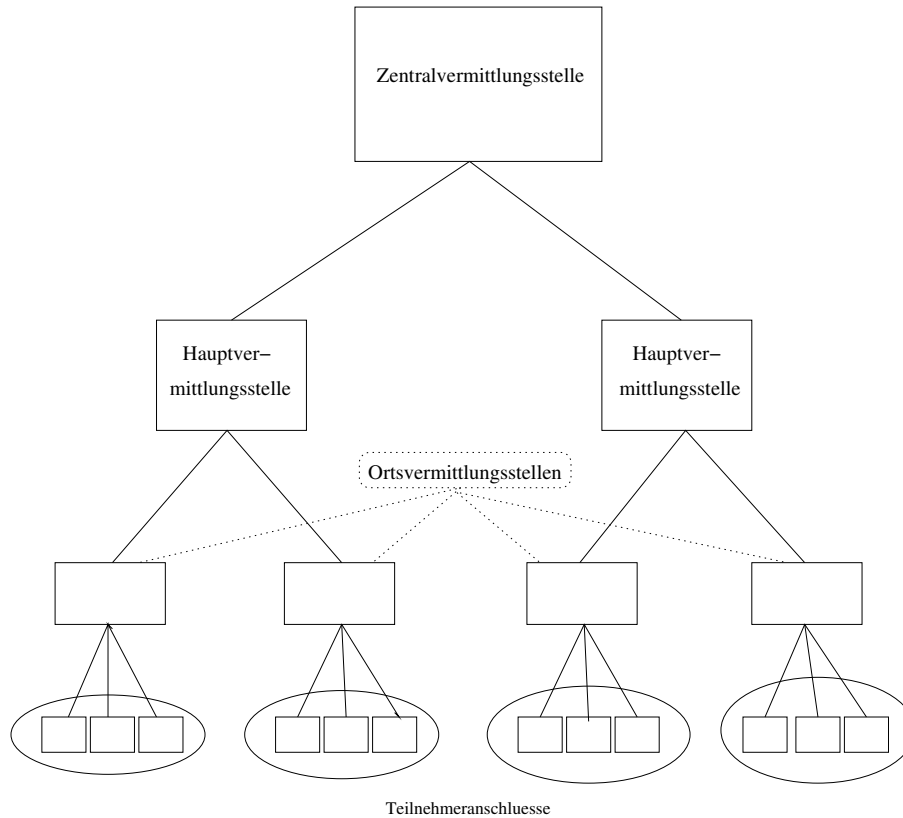


Abbildung 3.2: Mögliche Struktur der Telefonvermittlung

namisch bei einem Telefonie-Server im gleichen Administrativen Bereich an- und abmelden. Aus diesem Grund sind starre Verzeichnisse, wie z.B. Telefonbücher, nicht praktikabel.

Jeder Telefonie-Server kann für die bei ihm gemeldeten Telefon-Endstellen eine Zuordnung von Adresse (z.B. Telefonnummer oder URL-Adresse) zu einer IP-Adresse vornehmen. Nur wenn er diese Zuordnungen — wie auch immer — an andere Telefonie-Server weitergibt, können Telefon-Endstellen über ihn angerufen werden.

Bei einem Gesprächsaufbau muß der Ziel-Adresse ebenfalls eine IP-Adresse zugeordnet werden. Sobald die IP-Adresse des Gesprächspartners, bzw. des dafür zuständigen Telefonie-Servers, gefunden wurde, kann das Telefonat initialisiert werden. Für die Initialisierung können verschiedene Protokolle verwendet werden (z.B. *SIP* oder *H.225.0-RAS*). Jedes Protokoll bietet Möglich-

keiten an, um Parameter zu setzen oder zu verändern.

IP-Telefonie benutzt die bestehenden Netze des Internets und verwendet IP für den Transport der Gesprächsdaten und für die Initialisierung. Das Gespräch wird — wie bei IP üblich — in einzelne Pakete aufgeteilt und diese werden, unter Umständen auf unterschiedlichen Wegen, zum Gesprächspartner geleitet.

Das Zuordnen von Adressen zu IP-Adressen, bzw. die Verteilung dieser Information, ist eines der Hauptprobleme der IP-Telefonie. Die Protokolle, die für die IP-Telefonie eingesetzt werden, bieten teilweise eigene Lösungen an, weil es Parallelentwicklungen sind die miteinander konkurrieren oder unterschiedliche Funktionalitäten bieten.

In den folgenden Abschnitten werden die einzelnen Lösungsansätze der in Kapitel 2 vorgestellten Protokolle nochmals vorgestellt und gegeneinander abgewogen.

Anhand von Abb. 3.3 wird der Ablauf eines IP-Telefonates verdeutlicht. Telefon A hat sich bei seinem Telefonie-Server angemeldet. Sobald ein Telefonat geführt werden soll, signalisiert das Telefon dem Telefonie-Server den Verbindungswunsch und fordert die dafür notwendige Bandbreite an. Gleichzeitig wird dem Telefonie-Server noch die Adresse des Empfängers mitgeteilt. Der Telefonie-Server bringt die notwendige (IP-)Adresse - auf welchem Wege wird weiter unten erklärt - in Erfahrung. Sobald die IP-Adresse bekannt ist wird die Verbindung zum Empfänger (Telefon D) hergestellt. Dabei ist es unerheblich, welchen Weg übers IP-Netz die Datenpakete des Gespräches dann tatsächlich nehmen.

### 3.2.1 Adressenzuordnung mittels einer Datei

Bei dieser Methode werden die Zuordnungen von IP-Adressen zu Telefonnummern außerhalb des Telefonie-Servers permanent bereit gehalten (z.B. eine Datei auf der Festplatte). Wenn eine Änderung oder eine neue Zuordnung gemacht werden muß, muß die Datei von Hand geändert werden. Der Telefonie-Server kann bei Bedarf die Datei nach der notwendigen Zuordnung durchsuchen. Wenn aber keine Zuordnung vorhanden ist, kann die Telefonnummer nicht aufgelöst werden, da der Telefonie-Server keine weiteren Informationen von anderen Stellen erhalten kann. Auch können keine Zuordnungen zu Telefonen/Endstellen erzeugt werden, die ihre IP-Adresse dynamisch

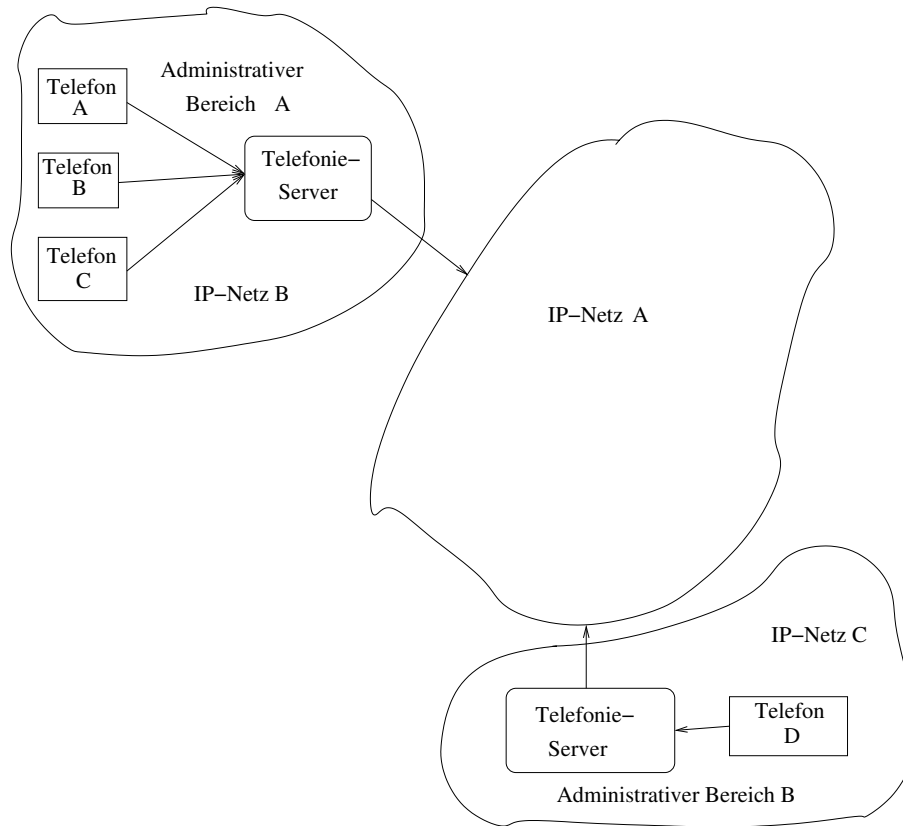


Abbildung 3.3: Telefonvermittlung übers Netz

(z.B. über *DHCP*) erhalten.

Dieser Ansatz ist für „starre“ Netze - z.B. ein Netz in einem Unternehmen - gerade noch vertretbar, da dort selten Änderungen in den Zuordnungen vorkommen. Wenn aber die Kommunikation weltweit erfolgen kann, ist es nicht möglich, alle notwendigen Zuordnungen innerhalb einer Datei zu speichern.

### 3.2.2 Adressenzuordnung mittels TRIP

Bei *TRIP* (siehe Kapitel 2.2) wird bei einem *Location Server* (*LS*) eine Zuordnungstabelle erzeugt. Jeder Telefonie-Server trägt seine Telefonnummern bei dem für ihn zuständigen *LS* ein. Dann tauschen miteinander verbundene *LS* diese Informationen untereinander aus. Das Ziel ist, daß alle in einem Netz verbundenen *LS* alle Informationen in der Zuordnungstabelle haben.

Telefonnummern, die nicht in der Tabelle vorhanden sind, können nicht aufgelöst werden.

Bei diesem Ansatz können Änderungen der Zuordnungen dynamisch angepaßt werden. Auch können neue Telefonnummern problemlos eingetragen werden. Allerdings muß die Liste der *Location Server* von Hand verwaltet werden - dies ist aber ein relativ geringer Aufwand, da hier die Verbindungen zu den *Location Servern* von anderen Administrativen Bereichen eingetragen werden. Bevor der *Location Server* voll einsatzfähig ist, muß er erst einmal Zuordnungsdaten sammeln. *TRIP* eignet sich gut dafür, viele ähnliche Adressen (wie z.B. Telefonnummern, die über ein *Gateway* bekannt gemacht werden) zu verwalten; es bietet spezielle Mechanismen für das Zusammenfassen solcher Adressen (siehe Kapitel 2.2.5).

### 3.2.3 Adressenzuordnung mittels H.323

Im H.323-Protokoll wird auf eine Zuordnungstabelle verzichtet (ein entsprechender Ansatz wurde erst bei *Annex G* vorgestellt). Dort wird davon ausgegangen, daß der **Gatekeeper** (der Telefonie-Server bei H.323) bei anderen angeschlossenen *Gatekeepern* nachfragt, ob sie die Telefonnummer auflösen können. Zusätzlich können *Gatekeeper*, die sich im selben Administrativen Bereich befinden, Daten untereinander austauschen.

Dieser Ansatz ist sehr dynamisch, das Auffinden einer Adresse kann aber — je nachdem wie lange nach der Adresse bei den *Gatekeepern* gesucht werden muß — lange Zeit in Anspruch nehmen. Allerdings würde mit zunehmender Nutzung der IP-Telefonie ein Teil der Bandbreite nur dafür eingesetzt werden, um Telefonnummern aufzulösen. Aus diesem Grund wurde der Annex G zum Protokoll H.323.H.225.0 erdacht.

### 3.2.4 Adressenzuordnung mittels Annex G

Einen ähnlichen Ansatz wie *TRIP* verfolgt *Annex G* (siehe Kapitel 2.6). Dabei werden die Zuordnungen zwischen Telefonnummern und IP-Adressen in einer Tabelle zur Verfügung gehalten. Diese Tabelle wird beim lokalen *BorderElement* geführt und besteht aus den Telefonnummern des eigenen Telefonie-Servers und den durch Adreßanfragen bei anderen *BE* in Erfahrung gebrachten Zuordnungen. Dabei gibt es viele Möglichkeiten, wie die *BE* miteinander verbunden werden können. Es sind einfache Verknüpfungen, Sammelpunkte

für mehrere *BE* oder auch ein *ClearingHouse*, das für sehr viele *BE* zuständig ist, möglich. Dabei wird aber immer auf eine vollständige Tabelle, in der allen möglichen Telefonnummern IP-Adressen zugeordnet sind, verzichtet. Wenn ein *BorderElement* eine Zuordnung nicht selber verzeichnet hat, kann es bei anderen *BE* nachfragen, ob sie dort bekannt ist. Erst wenn die gewünschte Telefonnummer nirgendwo aufgelöst werden kann, ist der Anruf nicht möglich.

Auch bei diesem Ansatz können die Zuordnungen dynamisch geändert und neue Telefone angemeldet werden. Auch hier müssen Verbindungen der einzelnen *BE* zwischen den einzelnen Zonen abgesprochen werden. Der Verbindungsaufbau kann länger dauern als bei *TRIP*, da die Zuordnungen erst noch erfragt werden müssen. Dafür sind Systeme, die *Annex G* verwenden, aber von Anfang an voll einsatzfähig.

### 3.2.5 Adressenzuordnung mittels DNS und SIP

Im H.323-Umfeld gibt es — wie schon erwähnt — neben Telefonnummern noch weitere Adreßarten. Die URL-Adresse ist im Format `user@domain` gehalten. Durch das DNS-System kann dann überprüft werden, ob für die Domain ein Server existiert, der die Adresse auflösen kann. Dies ist aber nicht auf URL-Adressen beschränkt, es ist auch möglich E-164-Nummern so zu kodieren, das über DNS überprüft werden kann, ob eine solche Adresse aufgelöst werden kann [20] [21] (siehe Kapitel 2.3).

Das **SIP**-Protokoll[10] benutzt ausschließlich eine Adreßzuordnung mittels DNS. Ein SIP-Telefon bzw. der zuständige Server fragt nach, welcher SIP-Server für die `domain` zuständig ist und erfährt von diesem, unter welcher IP-Adresse und welchem Port die gewünschte Telefon-Endstelle zu finden ist.

Dieser Ansatz ist sehr dynamisch - nirgendwo muß eine Liste bereitgehalten werden, in der steht, bei welchen Stellen man nach Zuordnungen für eine Telefonnummer nachfragen kann. Allerdings müssen dann in jeder Domain Server online sein.

## 3.3 Schlußfolgerung

Mit dieser Arbeit wird das Programm *IP-Exchange* vorgestellt, das in der Lage ist, jede der vorgestellten Adreßzuordnungen zu nutzen. Durch den modularen Aufbau von *IP-Exchange* ist es möglich, für jede Art der Adreßzuord-

nung ein Modul zu schreiben, das die notwendige Funktionalität bereitstellt. Ein Schwerpunkt dieser Arbeit ist das Anlegen einer Datenbasis für Adreßzuordnungen. Deswegen sind Arten von Adreßzuordnungen, die den Aufbau einer Datenbasis benötigen bzw. ermöglichen, in diesem Fall *TRIP* und *Annex G* interessant. Adreßzuordnungen, die in einer Datei bereitgehalten werden, und dynamische Adreßzuordnungen benötigen nur eine kleine — wenn überhaupt — Datenbasis und sind für diese Arbeit nicht weiter aussagekräftig.

Aus diesem Grund wird in den nächsten Kapiteln der Aufbau von IP-Exchange und die Kommunikation von IP-Exchange mit Telefonie-Servern anhand von *TRIP* und *Annex G* erläutert.

# Kapitel 4

## Gesamtentwurf von IP-Exchange

IP-Exchange erfüllt in der Internet-Telefonie die Funktion einer Vermittlungsstelle in der realen Telefonie. Während dort, bedingt durch den statischen Aufbau, eine Vermittlung von Gesprächen recht einfach ist (siehe Kapitel 3), ist dies in der Internet-Telefonie komplizierter. Jeder Adresse (ob Telefonnummer oder URL-Adresse) muß eine IP-Adresse zugeordnet werden, damit ein Anruf möglich ist. Da IP-Adressen auch dynamisch vergeben werden, ist ein Verzeichnis notwendig, bei dem die Zuordnungen von Adressen von Telefonie-Endpunkten zu IP-Adresse gemacht werden. Diese Aufgabe wird vom Programm IP-Exchange erfüllt. Es schließt die Kommunikationslücke zwischen den Telefonie-Servern (wie z.B. dem *H.323-Gatekeeper*) und den Vermittlungsprotokollen, die Adressen im Internet verbreiten (wie *TRIP* oder *Annex G*). IP-Exchange stellt einzelne Vermittlungsmodule zur Verfügung, die die Adressen in einer Datenbasis sammeln bzw. bei Bedarf nach den Adressen suchen.

In diesem Kapitel wird erläutert, unter welchen Gesichtspunkten die notwendigen Schnittstellen für die Kommunikation von IP-Exchange und den Vermittlungsmodulen entwickelt wurden. IP-Exchange wurde modular aufgebaut, so kann jedes Protokoll, das Telefonnummern im Internet verbreitet, als eigenständiges Modul implementiert werden — es ist dann ohne großen Aufwand möglich, daraus eigenständige Programme zu entwickeln. Dadurch ist es recht einfach möglich, neue Protokolle als Modul zu implementieren und an IP-Exchange anzubinden.

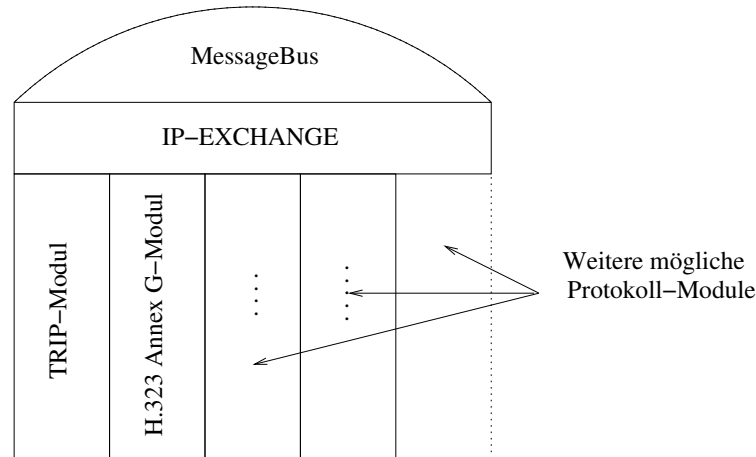


Abbildung 4.1: Aufbau von IP-Exchange

## 4.1 Anforderungen an IP-Exchange

Die Kommunikation zwischen dem Telefonie-Server und IP-Exchange erfolgt über den *Message Bus* mittels Nachrichten. Die für IP-Exchange wichtigen Nachrichten sind das Eintragen und Löschen von (eigenen) Telefonnummern, das Erfragen von Telefon-Routing-Informationen und die Konfiguration von IP-Exchange.

Da IP-Exchange erweiterbar sein soll und nicht nur auf ein einzelnes Vermittlungsprotokoll oder eine einzelne Art von Telefonie-Server abgestellt werden soll, müssen diese Nachrichten so aufgebaut werden, daß neue Vermittlungsprotokolle oder Änderungen in den bestehenden keine Neu-Implementierung von IP-Exchange bedingen. Die durch die *Mbus*-Nachrichten erhaltenen Informationen sollten alles beinhalten, was für den Aufbau einer Datenbasis notwendig ist. Aus den Antworten von IP-Exchange muß der Telefonie-Server alle gewünschten Informationen für den Verbindungsaufbau erhalten.

Die folgenden Abschnitte zeigen auf, worauf man bei dem Aufbau der einzelnen Nachrichten achten sollte, und gehen dabei auf *TRIP* und *Annex G* ein.

### 4.1.1 Eintragen und Löschen von Telefonnummern

*TRIP* kann Telefonnummern (Adressen die nur aus Ziffern bestehen) weiterleiten, die unterschiedliche Protokolle für den Anruf verwenden — namentlich

SIP, H.323-H.225.0-Q.931, H.323-H.225.0-RAS und H.323-H.225.0-Annex G. *TRIP* setzt dabei voraus, daß die Telefonnummern das E.164-, das dezimale oder das pentadezimale Zahlenformat haben (siehe auch Kapitel 2.2, Seite 13ff.). Für den Einsatz von *TRIP* ist es deswegen notwendig, daß für die Telefonnummer, die eingetragen oder gelöscht werden soll, explizit das jeweilige Protokoll und die Telefonnummer angegeben werden muß.

*Annex G* ist in der Lage Adressen, die aus Ziffern bestehen oder im URL-Format abgelegt sind, zu verbreiten. Jedes *BE* (siehe Kapitel 2.6.3) faßt seine Adressen in einen (oder mehrere) Deskriptor(en) zusammen. Eine Unterscheidung von verschiedenen Adreßfamilien findet nicht statt, es wird davon ausgegangen, daß der angeschlossene Telefonie-Server mit der Adresse umgehen kann. Jede Adresse kann noch weitere Informationen enthalten, wie z.B. Tarifinformationen und deren Gültigkeit.

Damit IP-Exchange protokollunabhängig und erweiterbar sein kann, ist es wichtig, daß Adressen in jeder Kodierung akzeptiert und an die zuständigen Protokoll-Module weitergeleitet werden. Adressen bestehen aus Zeichen- und Ziffernfolgen, und obwohl die meisten Telefonnummern aus dezimalen Ziffern bestehen, gibt es schon jetzt Länder, in denen die Nummern in anderen Zahlensystemen kodiert werden. Deswegen müssen am besten alle möglichen Zahlen- bzw. Zeichenfolgen erlaubt sein. Auch sollte die Möglichkeit gegeben sein, „Wildcards“ zu benutzen. IP-Exchange muß auch in der Lage sein, zusätzliche Informationen weiterleiten zu können, wenn ein Protokoll es vorsieht. Ob die Vermittlungsmodule diese Informationen verarbeiten ist dann deren Entscheidung.

Die Informationen, die zwischen dem Telefonie-Server, IP-Exchange und den Vermittlungsmodulen ausgetauscht werden, müssen darum die Adresse, das verwendete Protokoll, die benutzte Adreßfamilie und mögliche optionale Parameter enthalten.

### 4.1.2 Suche nach Telefonnummern

*TRIP* kann, wie schon erwähnt, Adressen verschiedener Protokolle weiterleiten. Deswegen ist es bei einer Anfrage des Telefonie-Servers möglich, daß das *TRIP*-Modul in der Antwort mehrere mögliche Protokollarten für eine Telefonnummer angibt. Es liegt am Telefonie-Server, ob - und wenn ja wie - er die möglichen Protokolle einschränken will.

Ähnlich wie bei *TRIP* kann der Telefonie-Server beim *Annex G*-Modul nach

einer Adresse fragen. Jedoch kann das Modul bei entsprechenden Partnern nach dieser Adresse suchen, was bei *TRIP* nicht möglich ist. Dafür werden dann vom Telefonie-Server Angaben benötigt, welche Partner angesprochen werden sollen, und wie lange nach der Adresse gesucht werden soll.

Die Suchfunktion soll dem Telefonie-Server den Ansprechpartner für die weitere Kommunikation liefern. Dies kann der Server sein, bei dem der gesuchte Telefonie-Endpunkt angemeldet ist, oder direkt die IP-Adresse des gesuchten Telefonie-Endpunkts. Die mögliche Antwort hängt dann vom Vermittlungsprotokoll ab, über das die Adresse aufgelöst wurde.

Dem Telefonie-Server sollte es möglich sein anzugeben, bei welchen Protokollen gesucht werden soll. Auch sollte die Antwort so formuliert werden können, daß, wenn die gesuchte Telefonnummer bei mehreren Protokollen gefunden wurde, alle Suchergebnisse an den Telefonie-Server zurückgegeben werden. Dafür eignet sich eine Liste von Routen-Informationen, in der alle Daten zu einer Route (d.h. ein möglicher Weg zu einer Telefonnummer) enthalten sind.

Für die Adreßsuche wird für die Kommunikation zwischen Telefonie-Server, IP-Exchange und den Vermittlungsmodulen neben der gesuchten Adresse auch das gewünschte Protokoll und die gewünschte Adreßfamilie benötigt. In den optionalen Parametern können dann Wünsche für bestimmte Tarife etc. ausgedrückt werden.

### 4.1.3 Konfiguration

*TRIP* benötigt für jedes Protokoll, das der Telefonie-Server benutzt (bzw. akzeptiert), Angaben welcher Computer auf welchem Port als Server agieren soll. Auch kann beim *TRIP*-Modul angegeben werden, ob es nur senden, nur empfangen oder beides machen soll (siehe 2.2.1, Seite 15ff).

Auch bei *Annex G* muß das Vermittlungsmodul wissen, auf welchen Computern und Ports die Server zu finden sind. Dabei sind zwar nicht unterschiedliche Vermittlungsprotokolle zu berücksichtigen, aber *Annex G* erlaubt es, Informationen über unterschiedliche Transportprotokolle zu verschicken (siehe 2.6 auf Seite 24ff).

Da im Moment nicht bekannt ist, welche Möglichkeiten weitere Protokolle bieten werden, ist es auch nicht möglich zu sagen, ob und welche der Funktionen konfigurierbar sein werden.

Eine einfache Lösung zur Konfiguration ist, daß jedes der Vermittlungsmodule auf andere Schlüsselwörter des Telefonie-Servers reagiert - wobei gleiche Funktionen auch durch dieselben Schlüssel verwendet werden können. Es muß dann beim Telefonie-Server zu jedem Protokoll eine Liste der möglichen Konfigurationen mitgeführt und bei jedem neuen Vermittlungsprotokoll erweitert werden - aber dann muß der Telefonie-Server an dieses neue Protokoll angepaßt werden.

#### 4.1.4 Status-Informationen

Für den Telefonie-Server kann es in bestimmten Fällen interessant sein, zu erfahren, in welchem Zustand IP-Exchange oder deren Vermittlungsmodule gerade sind, dies kann z.B. die Anzahl der bekannten Adressen sein, oder die Anzahl der Verbindungen zu anderen Programmen mit denen Adressen ausgetauscht werden. Die Abfrage des Status ist protokollspezifisch und es ist möglich, daß einzelne Protokolle keinen sinnvollen Datenbestand pflegen und somit keine Informationen für den Statusbericht liefern können.

## 4.2 Der Aufbau einer gemeinsamen Schnittstelle

Die Hauptfunktion von IP-Exchange ist es, für einen Telefonie-Server Telefonnummern (bzw. Adressen) auflösen zu können. Einzelne Module implementieren Protokolle wie *TRIP* oder *Annex G*. Diese Module können dabei in Kontakt mit anderen **Location Servers** (bei *TRIP*) oder **BorderElements** (bei *Annex G*) stehen und Daten austauschen, oder die benötigten Daten in einer vorhandenen Datei suchen. Bei Bedarf — einer Anfrage des Telefonie-Servers — muß IP-Exchange dann, falls möglich, die entsprechende gewünschte Antwort geben.

Aus diesem Grund muß die Kommunikation aller Vermittlungsmodule mit IP-Exchange über eine gemeinsame Schnittstelle ablaufen. Daten für Adreßeinträge und Informationsanfragen werden über diese Schnittstelle an die Module weitergereicht. Damit sind später keine Änderungen an IP-Exchange selber notwendig, wenn eines der Module an eine neue Protokollversion angepaßt werden muß.

Zuerst muß bei den bestehenden Modulen betrachtet werden, welche Informationen sie mit der Außenwelt austauschen können bzw. welche Informationen dafür notwendig sind.

*TRIP* ermöglicht es, Adressen (bzw. die Adressen der zuständigen Server) zwischen den einzelnen *Location Servern* auszutauschen. Da jeder Telefonie-Server (und auch Telefon-Endstellen) unterschiedliche Protokolle verstehen und akzeptieren, teilt *TRIP* immer mit, welches bei den einzelnen Adressen verwendet werden kann. Aus diesem Grund muß das Vermittlungsmodul für *TRIP* neben der Adresse auch erfahren, welche Protokolle der entsprechende Telefonie-Server verstehen kann.

*Annex G* ermöglicht es, zwischen den entsprechenden *BorderElementen* die für *Annex G* typischen Informationen wie Telefonnummern oder Adressen im URL-Format, Tarifinformationen oder Lebensdauer der Anfrage auszutauschen.

Wenn der Telefonie-Server über IP-Exchange Informationen über Telefonie-Endgeräte haben will, muß IP-Exchange bei den Vermittlungsmodulen nachfragen ob, und wenn ja welche, Informationen verfügbar sind.

### 4.3 Bereitgestellte Schnittstellen

Da IP-Exchange erweiterbar sein soll müssen alle Vermittlungsmodule folgende Schnittstellen verwenden. Dabei ist darauf zu achten, das jedes Vermittlungsmodul Informationen, die es nicht verwenden kann, ignorieren können muß. IP-Exchange benutzt die Schnittstellen, wenn die Kommunikation mit dem Telefonie-Server dies erforderlich macht.

Dabei werden immer alle Vermittlungsmodule aktiviert. Dies ist notwendig, da IP-Exchange die möglichen Funktionalitäten und Konfigurationsmöglichkeiten der einzelnen Vermittlungsmodule nicht kennt.

Durch die Verwendung optionaler Parameter, in denen andere — noch unbekannte — Parameter bzw. Anforderungen aufgelistet werden können, ist es auch die Verwendung zukünftiger Vermittlungsmodule möglich.

### 4.3.1 Konfiguration der Vermittlungsmodule

Der Aufruf dieser Schnittstelle ermöglicht es, Vermittlungsmodule zu konfigurieren. Die Vermittlungsmodule sollten so eingerichtet werden, daß sie am Programmstart — bevor der Telefonie-Server Telefonnummern in die Datenbank einträgt — und auch während des Betriebes (re-)konfiguriert werden können.

Kommando	<b>change_rc</b>
Parameter	Eine Variable vom Typ OptPar, in der alle Konfigurationsdaten stehen
Rückgabewert	Keiner

Tabelle 4.1: Schnittstelle zur Modul-Konfiguration

Die Liste der Konfigurationsbefehle wird vor dem Aufruf der Schnittstelle in eine Variable des Typs OptPar geschrieben, wobei die einzelnen Strings die Syntax **Feld=Wert** haben. Vermittlungsmodule sollen unbekannte Felder ignorieren, dann kann der jeweilige Telefonie-Server neue Konfigurationsmöglichkeiten nutzen, falls ein (neues) Vermittlungsmodul diese bietet. Jedem Vermittlungsmodul steht es frei, sich eine eigene Konfigurationsdatei einzurichten und diese mit den von dem Telefonie-Server angegebenen Werten abzuspeichern.

### 4.3.2 Eintragen von Adressen

Jede Telefon-Endstelle hat eine Adresse — sei es eine Telefonnummer oder eine Adresse im URL-Format. Für die IP-Telefonie ist es notwendig, daß eine Zuordnung einer Adresse zu einer IP-Adresse bei Bedarf möglich ist. Dafür muß die Zuordnung im Internet bekannt gemacht werden, welches die Aufgabe der Vermittlungsmodule ist. Sie geben über von ihnen benutzte Protokolle bekannt, welcher Server für die Adresse zuständig ist (bzw. unter welcher IP-Adresse er zu finden ist).

Über diese Schnittstelle (Tabelle 4.3.2) wird den Vermittlungsmodulen mitgeteilt, welche *Adresse* in die Datenbasis eingetragen werden soll. Da die Adresse auf verschiedene Weisen dargestellt werden kann (z.B. in dezimaler oder pentadezimaler Schreibweise) steht in *Familie*, welche Darstellung für

Kommando	<b>set_Addr</b>
Parameter	<i>vector</i> <unsigned char> Adresse, eine Liste von Zeichen in der die einzutragende Telefonnummer bzw. Adresse steht
	<i>int</i> <i>Protokoll</i> - Das Protokoll (bzw. die Protokolle), die das Endgerät mit der angegebenen Adresse versteht
	<i>int</i> <i>Familie</i> , die Adrefamilie, die für die Kodierung der Adresse benutzt wird
	<i>OptPar</i> <i>Parameter</i> - eine Variable der Klasse <i>OptPar</i> , die alle optionalen Parameter in einer Liste enthält
Rückgabewert	<i>int</i> Die Nummer(n) des Protokolls, das nicht in die Datenbank eingetragen werden konnte

Tabelle 4.2: Schnittstelle zum Eintragen von Adressen

die Adresse gültig ist. In *Protokoll* steht, welche Protokolle der Telefonie-Server für die Adresse akzeptiert. Die Vermittlungsmodule müssen dann eigenständig entscheiden, ob sie diese Telefonnummer in ihre Datenbasis eintragen und dann bekannt machen wollen. Im Rückgabewert zeigen die Vermittlungsmodule an, für welche Protokolle die Adresse nicht in die Datenbasis übernommen werden konnte.

Wert	Protokoll, bei dem die Adresse nicht eingetragen bzw. gelöscht werden konnte
0	Kein Fehler
1	SIP
2	H.323-H.225.0-Q.931
4	H.323-H.225.0-RAS
8	H.323-H.225.0-Annex G

Tabelle 4.3: Rückgabewerte und ihre Bedeutung

Der Rückgabewert und der Wert im Protokoll-Feld bestehen aus Bitvektoren, die sich wie in den Tabellen 4.3 und 4.4 zusammensetzen. Ein Protokoll-Wert von 6 sagt dann aus, daß die Adresse in den Protokollen H.323-H.225.0-Q.931 (=2) und H.323-H.225.0-RAS (=4) eingetragen werden soll. Ein Rückgabewert von 9 sagt aus, daß die Adresse beim Protokoll 1 (=SIP) und 8 (=Annex

G) nicht eingetragen werden konnte.

Wert	Protokoll	Wert	Familie
1	SIP	1	Dezimale Telefonnummern (0..9)
2	H.323-H.225.0-Q.931	2	Pentadezimale Telefonnummern (0..E)
4	H.323-H.225.0-RAS	3	E.164 Telefonnummern (0..9)
8	H.323-H.225.0-Annex G	4	Adressen im URL-Format
16	H.323-Party Number		

Tabelle 4.4: Mögliche Werte für Protokoll- und Familie-Felder

### 4.3.3 Löschen von Adressen

Über eine Schnittstelle wird den Vermittlungsmodulen mitgeteilt, welche *Adresse* in der jeweiligen Datenbasis gelöscht werden soll. Dabei grenzen *Protokoll* und *Familie* die zu löschende Information eventuell ein. Als Rückgabewert kann jedes Vermittlungsmodul angeben, für welches Protokoll die Löschung gescheitert ist. Dies passiert, wenn die Adresse mit dem/der angegebenen Protokoll oder Adressfamilie vom Vermittlungsmodul nicht verarbeitet wird.

### 4.3.4 Anfrage von Adreßinformationen

Diese Schnittstelle ermöglicht IP-Exchange, eine Nachfrage bei den Vermittlungsmodulen zu stellen, ob eine Adresse mit den angegebenen Parametern bekannt ist. Dabei kann die Adresse selber durch die Benutzung von „Wildcards“ abgekürzt werden. Es können auch mehrere Protokolle angegeben werden. Wenn eines (oder auch mehrere) zutrifft, gibt jedes Vermittlungsprotokoll eine Liste zurück, in der aufgeführt ist, bei welchem Telefonie-Server die gesuchte Telefonnummer gefunden werden kann. Diese Liste wird aus den Daten der Vermittlungsmodule aufgebaut, in denen alle Informationen über den Telefonie-Server der Endstelle des Gesprächspartners aufgelistet werden. Wenn verschiedene Module einen passenden Eintrag finden, werden alle in die Liste aufgenommen. Der eigene Telefonie-Server kann dann entscheiden, welches Protokoll bzw. welchen Server er für den Anruf verwenden will.

Kommando	<b>remove_Addr</b>
Parameter	<i>vector &lt;unsigned char&gt;</i> Adresse, eine Liste von Zeichen, in der die zu löschende Telefonnummer bzw. Adresse steht
	<i>int Protokoll</i> - Das Protokoll (bzw. die Protokolle) der Adressen, die in den Datenbanken der Vermittlungsmodule gelöscht werden sollen
	<i>int Familie</i> , die Adressfamilie, die für die Kodierung der Adresse benutzt wird
	<i>OptPar Parameter</i> - eine Variable der Klasse OptPar, die alle optionalen Parameter in einer Liste enthält
Rückgabewert	<i>int</i> , Die Nummer(n) des Protokolls, bei dem die Adresse nicht in der Datenbank gefunden werden konnte

Tabelle 4.5: Schnittstelle zum Löschen von Adressen

### 4.3.5 Statusabfrage des Vermittlungsmoduls

Mit dieser Schnittstelle kann IP-Exchange bei den Vermittlungsmodulen nach deren Status nachfragen. IP-Exchange fragt in bestimmten Zeitintervallen den Status aller Vermittlungsmodule ab und gibt Änderungen an den Telefonie-Server weiter.

### 4.3.6 Datenbank löschen

Mit dieser Schnittstelle können die Vermittlungsmodule von IP-Exchange angewiesen werden, alle internen, d.h. vom eigenen Telefonie-Server bekannt gegebene, Adressen zu löschen. Dies ist bei einem Fehler oder Absturz des Telefonie-Servers notwendig. Sobald IP-Exchange bemerkt, daß der Telefonie-Server die Verbindung unterbrochen hat, werden seine Telefonnummern gelöscht. Wenn die Verbindung wiederhergestellt wird, müssen alle Telefonnummern neu angemeldet werden.

Kommando	<b>clean</b>
Parameter	Keine
Rückgabewert	Keiner

Tabelle 4.8: Schnittstelle zur Löschung der Datenbank

Kommando	<b>get_Addr</b>
Parameter	<i>vector</i> <unsigned char> Adresse, eine Liste von Zeichen, in der die gesuchte Telefonnummer bzw. Adresse steht
	<i>int</i> Protokoll - Das Protokoll (bzw. die Protokolle), das für die gesuchte Adresse akzeptabel ist
	<i>int</i> Familie, die Adreßfamilie, die für die Kodierung der Adresse benutzt wird
	<i>OptPar</i> Parameter - eine Variable der Klasse OptPar, die alle optionalen Parameter in einer Liste enthält
Rückgabewert	<i>vector</i> <Route_inf>, eine Liste von Routeninformationen die für die gesuchte Nummer wichtig sind. Darin enthalten sind der zuständige Server und etwaige optionale Parameter

Tabelle 4.6: Schnittstelle zur Adreß-Suche

Kommando	<b>get_status</b>
Parameter	keiner
Rückgabewert	Eine Variable vom Typ Status, die eine Liste von Strings enthält

Tabelle 4.7: Schnittstelle zur Statusabfrage eines Vermittlungsmoduls

## 4.4 Aufbau eines Vermittlungsmoduls

Einer der Vorteile von IP-Exchange ist, daß der Anschluß neuer Vermittlungsmodule einfach zu bewerkstelligen sein soll. Die wichtigste Eigenschaft - neben der Implementierung obiger Schnittstellen - eines neuen Moduls muß es sein, unbekannte Parameter zu ignorieren. Dadurch wird es möglich, daß jedes Vermittlungsmodul nur die Informationen von IP-Exchange benutzt, die es für sich als relevant erachtet.

Wie genau ein neues Vermittlungsmodul implementiert wird, ist für IP-Exchange unwichtig, aber es wird dennoch ein Eintrag in IP-Exchange benötigt damit das Modul angesprochen werden kann. Dafür muß der Quell-Code des Vermittlungsmoduls bei IP-Exchange (exchange.cc) eingetragen und eine neue Klasse definiert werden. Zusätzlich müssen bei bestimmten Prozeduren (einfügen oder löschen von Adressen, Modul-Konfiguration, Adreß-Suche und Status-Abfrage) den Schnittstellendefinitionen entsprechende Methoden

der neuen Klasse aufgerufen werden. Auch sollte beim Einbau des Moduls darauf geachtet werden, daß der zuständige Telefonie-Server das Modul auch benutzen kann. Dazu muß er das neu implementierte Protokoll benutzen und das Vermittlungsmodul auch entsprechend konfigurieren können.

In den Tabellen 4.9 und 4.10 wird ein Programmgerüst für ein *Annex G* BorderElement gezeigt.

```
#include "BE.hh"
BorElem::BorElem(){
}

BorElem:: BorElem(){
}

int BorElem::set_Addr(vector < unsigned char> Addr, int proto,
int fam, OptPar p){
// proto==8 is Annex G
if (proto<8) return proto;
else return (proto-8);

// implementation

return 0;
}
int BorElem::rm_Addr(vector <unsigned char> Addr, int proto,
int fam, OptPar p){
// proto==8 is Annex G
if (proto<8) return 8;

// implementation
return 0;
}

status BorElem::get_status(){
status s;
// s containing number of addresses, connections
// (and their addresses)
return s;
}
```

Tabelle 4.9: Programm-Skelett für ein BorderElement

```

vector <Route_inf> BorElem::get_Addr
(vector <unsigned char> Addr,
int proto, int fam,
OptPar p){
vector ¡Route_inf¿ s(0);
// implementation

return s;
}

void BorElem::clean(){
// removing all addresses from the gatekeeper
}

void BorElem::change_rc
(OptPar new_configuration){
// just use what you need. }

```

Tabelle 4.10: Programm-Skelett für ein BorderElement (Fortsetzung)

## 4.5 Implementierung

Die einzelnen Vermittlungsmodule (siehe 6 für die Beispielimplementierung eines *TRIP*-Location Servers) von IP-Exchange müssen, je nach implementiertem Protokoll und Aufbau der Datenbasis, große Datenmengen verwalten und durchsuchen. Dafür muß eine möglichst schnelle Programmiersprache benutzt werden, d.h. sie sollte kompiliert sein und nicht interpretiert werden müssen. Da gleichzeitig mehrere Verbindungen zu verschiedenen Kommunikationspartnern aufrecht erhalten werden müssen, ist eine weitere Voraussetzung, daß mehrere Aufgaben unabhängig von einander ausgeführt werden können (**Multithreading**). Da die Größe der Datenbasis variiert, wird der Speicher im Laufe der Zeit immer stärker fragmentiert. Ein Aufräumen des Hauptspeichers — Garbage-Collection genannt — wäre deswegen wünschenswert.

Aus der Fülle an vorhandenen Programmiersprachen boten sich für die Im-

plementierung *Java* und *C++* an.

**Java** ist eine interpretierte Sprache, d.h. die Ausführung des Programms ist langsamer als bei kompilierten Sprachen, dafür ist sie sehr fehlertolerant und plattformunabhängig. JAVA unterstützt Threads, und kann somit Aufgaben unabhängig voneinander bearbeiten, darüber hinaus bietet es eine Garbage-Collection an.

**C++** dagegen ist eine Compiler-Sprache, d.h. die C++-Befehle werden in Maschinencode umgesetzt und werden dann schneller ausgeführt als JAVA-Instruktionen. Allerdings ist C++ fehlerintolerant — ein Programmfehler führt (fast) immer zu einem Programmabbruch — und muß auf die verschiedenen Plattformen portiert werden. Auch C++ unterstützt Threads, bietet aber keine Defragmentierung des freien Speichers an. Der Geschwindigkeitsvorteil bei der Ausführung des Programms hat mich dann dennoch dazu bewogen, IP-Exchange in C++ zu schreiben.

# Kapitel 5

## Externe Schnittstellen von IP-Exchange

Während, wie im Kapitel 4 beschrieben, IP-Exchange die einzelnen Vermittlungsmodule fest integrieren muß, um sie ansprechen zu können ist die Kommunikation zum zuständigen Telefonie-Server über *Mbus*-Nachrichten realisiert. IP-Exchange selbst hat keine Vorstellung von der es umgebenden (Internet-)Welt. Auf dem *Message Bus* (Mbus) versendete Nachrichten werden abgehört, und wenn sie IP-Exchange betreffen, werden die den Nachrichten entsprechenden Aktionen ausgeführt. IP-Exchange ist darauf ausgelegt, das es in einem lokalen System mit einem Telefonie-Server zusammenarbeitet.

In diesem Kapitel wird erläutert, welche *Mbus*-Nachrichten von IP-Exchange verstanden werden und warum sie für die Kommunikation mit dem Telefonie-Server notwendig sind.

### 5.1 Mbus-Nachrichten

Als Ergebnis des vorherigen Kapitels wurden die folgenden Nachrichten entwickelt, um eine Kommunikation zwischen IP-Exchange und einem Telefonie-Server zu ermöglichen. Diese Nachrichten werden von IP-Exchange verarbeitet und über die entsprechenden Schnittstellen (siehe Kapitel 4) an die Vermittlungsmodule weitergegeben.

Für die Kommunikation zwischen IP-Exchange und einem Telefonie-Server werden vor allem *Mbus-RPC*-Kommandos verwendet. Sie bestehen aus einem

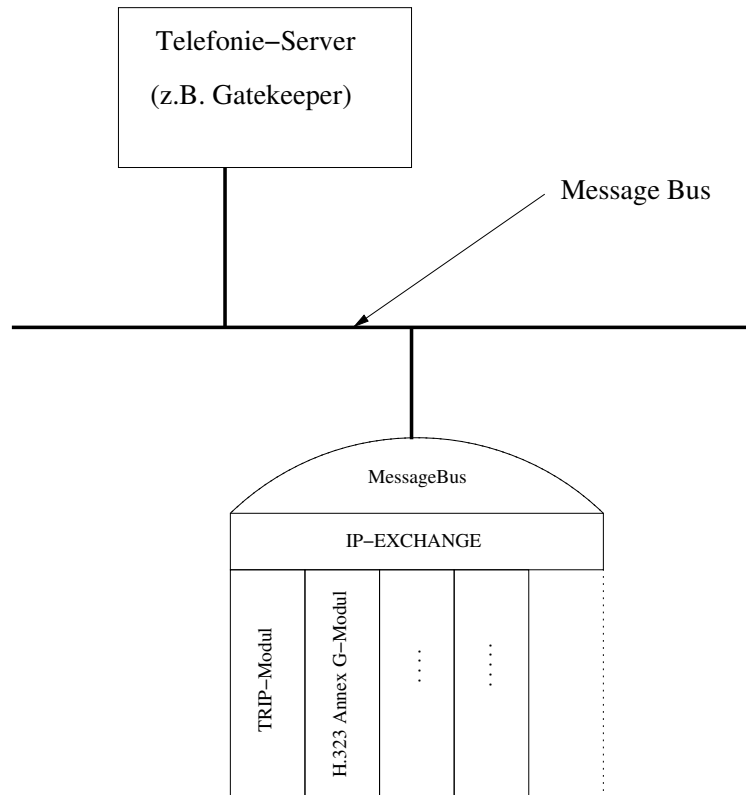


Abbildung 5.1: Mbus-Anbindung von IP-Exchange

Kommandopaar. Ein Kommando — mit Parametern — löst beim Kommunikationspartner eine Aktion aus, und dieser schickt eine Nachricht mit dem Resultat zurück. Der erste Parameter des Kommandos enthält die *RPC-ID*, die beim Sender eindeutig sein sollte, und den *RPC-Typ* — im Fall von IP-Exchange „UNICAST“.

Für die Rückantwort wird der selbe Kommandoname benutzt und „.return“ angehängt. Im ersten Parameter wird die *RPC-ID* angegeben, damit der Empfänger das Resultat zuordnen kann, und der *RPC-Status*, in dem bestätigt wird, das die gewünschte Prozedur ausgeführt - oder aber nicht gefunden - wurde. Der nächste Parameter enthält Informationen über das Ergebnis der Prozedur. Mögliche Informationen sind „OK“ oder „FAILED“, zusätzlich wird die Information in Textform ausgedrückt. Die weiteren Parameter enthalten dann das Ergebnis. Bei jeder Nachricht sind zur Veranschaulichung einige Beispiele angegeben.

Die Nachrichten zum Hinzufügen, Löschen und Suchen von Adressen sowie zum Konfigurieren der Vermittlungsmodule und zur Statusabfrage werden „zuverlässig“ versendet, d.h. wenn eine Nachricht nicht beantwortet wird, schickt der Sender seine Nachricht noch einmal, da er annimmt, daß sie beim Empfänger nicht angekommen ist.

### 5.1.1 Hinzufügen und Löschen von Adressen

telephony.routing.add bzw. telephony.routing.remove	
<b>Kommandotyp</b>	RPC
<b>Parameter</b>	<i>int</i> protocol
	<i>int</i> family
	<i>string</i> address
<b>Optionale Parameter</b>	<i>string</i> Liste von Schlüssel=Wert

Tabelle 5.1: Hinzufügen und Löschen von Adressen - Nachrichtenformate der Kommandos

Das Hinzufügen und Löschen von Adressen wird über das selbe Nachrichtenformat bewerkstelligt. Bei jeder Nachricht gibt der Telefonie-Server an, welche Adresse angemeldet bzw. gelöscht werden soll, wobei mit jeder Nachricht nur eine Adresse bearbeitet wird.

Zusätzlich muß angegeben werden, welche Protokolle der für die Endstelle zuständige Telefonie-Server akzeptiert und in welcher Adreßfamilie die Adresse kodiert ist.

Folgende Protokolle werden von IP-Exchange akzeptiert, wobei ohne Probleme weitere Protokolle hinzugefügt werden können - jedes Vermittlungsmodul soll nur die Protokolle auswählen, die es verarbeiten kann, die restlichen sollen ignoriert werden. Dabei ist es auch möglich, mehrere Protokolle gleichzeitig anzugeben (z.B. der Wert 5 im Protokoll-Feld würde bedeuten, daß für eine Adresse als Protokolle SIP und H.323-H.225.0-RAS akzeptiert werden).

- **1** = SIP
- **2** = H.323-H.225.0-Q.931

- 4 = H.323-H.225.0-RAS
- 8 = H.323-H.225.0-Annex G

Bekannt sind folgende Adreßfamilien - wobei H.323 eine ganze Fülle von Adreßfamilien kennt, die hier nicht alle vollständig vorgestellt werden. Falls neue - oder auch alte - Protokolle neue Adreßfamilien benötigen können diese einfach angefügt werden.

- 1 = Dezimale Telefonnummern (0..9)
- 2 = Pentadezimale Telefonnummern (0..9,A..E); Telefonnummern in den USA werden in diesem Format abgelegt
- 3 = E.164-Telefonnummern (0..9)
- 4 = Adressen im URL-Format (z.B. `prelle@tzi.org`)
- 5 = Die H.323-**PartyNumber** enthält mehrere Felder wie z.B. tatsächliche Adresse oder Typ der Party (wie z.B. Private). Anstelle für jeden dieser Typen eine eigene Adreßfamilie zu reservieren wird bei dieser Familie angezeigt, das der genaue Typ der *PartyNumber* und die zusätzlich notwendigen Informationen in den optionalen Parametern angegeben sind.

Die optionalen Parameter ermöglichen es, zusätzliche Informationen an die Vermittlungsmodule weiterzugeben. Jedes Vermittlungsmodul muß dann entscheiden ob es die Parameter verwenden kann bzw. soll. Mit diesen Parametern können auch zukünftige Vermittlungsmodule ohne große Probleme an IP-Exchange angebunden werden.

Für die Adreßfamilie 5 (*PartyNumber*) müssen die optionalen Parameter, je nach PartyTyp unterschiedliche Einträge beinhalten. Mögliche Typen sind `PublicPartyNumber`, `PrivatePartyNumber`, `MobileUIM`, `Endpoint` und `ExtendedAliasAddress`. Jeder der Typen benötigt eine unterschiedliche Anzahl von Unterparametern, die aber über die IP-Exchange zwischen der Vermittlungsstelle und den Vermittlungsmodulen ausgetauscht werden können.

Die Rückantwort enthält eine Liste von Tupeln, die das Format (Vermittlungsmodul, Ergebnis des Adreßeintrages) haben. Das Ergebnis des Adreßeintrages ist 0, wenn keine Fehler aufgetreten sind. Ansonsten steht dort die Protokoll-Nummer (oder die Protokoll-Nummern), für die die Telefonnummer nicht eingetragen bzw. gelöscht werden konnte.

telephony.routing.add.return bzw. telephony.routing.remove.return	
<b>Kommandotyp</b>	RPC-Antwort
<b>Parameter</b>	Liste von Tupeln <MSymbol Vermittlungsmodul, int Ergebnis>

Tabelle 5.2: Hinzufügen und Löschen von Adressen -  
Nachrichtenformat der Rückantwort

### Beispiele für Adressen hinzufügen

1. `telephony.routing.add.rpc (2 1 ''+49421300005'')`

Hier soll die Telefonnummer *+49421300005* mit dem Protokoll *H.225.0-Q.931* und der dezimalen Adreßfamilie in die Datenbasis eingetragen werden. Eine mögliche Antwort würde wie folgt aussehen:

```
telephony.routing.add.rpc.return ((TRIP 0) (Annex_G 2))
```

Da ein *Annex G*-Modul *Q.931* nicht akzeptiert, wird dort angemerkt das die Adresse nicht eingetragen werden konnte. Das *TRIP*-Modul hingegen wird die Adresse eintragen können. Insgesamt wurde die Nachricht erfolgreich abgeschlossen, da die Adresse in mindestens einem Vermittlungsmodul eingetragen werden konnte.

2. `telephony.routing.add.rpc (9 2 ''+49421300002'')`

Hier soll die Telefonnummer *+49421300002* mit den Protokollen *SIP* und *it Annex G* in der pentadezimalen Adreßfamilie in die Datenbank eingetragen werden. Die Antwort sollte wie folgt lauten:

```
telephony.routing.add.rpc.return ((TRIP 0) (Annex_G 1))
```

Während *TRIP* beide Protokollarten unterstützt, fehlt beim *Annex G*-Modul die Unterstützung für *SIP*.

3. `telephony.routing.add.rpc (12 4 ''prelle@tzi.de'')`

Bei dieser Nachricht soll eine Adresse im URL-Format für die Protokolle *Annex G* und *H.225.0-RAS* in die Datenbank eingetragen werden. Eine Antwort würde so aussehen:

```
telephony.routing.add.rpc.return ((TRIP 12) (Annex_G 4))
```

*TRIP* erlaubt keine Adressen im URL-Format - deswegen wird diese Nachricht komplett abgelehnt. *Annex G* dagegen kennt das *RAS*-Protokoll nicht und lehnt nur dieses ab.

### Beispiele für Adressen löschen

1. `telephony.routing.remove.rpc (2 1 ''+49421300005'')`

Die Adresse *+49421300005* soll aus den Datenbanken ausgetragen werden. Allerdings nur das Protokoll *Q.931* in der dezimalen Adressfamilie. Andere Einträge der Adresse mit einem anderen Protokoll (bzw. einer anderen Adressfamilie) sollen davon nicht berührt werden. Eine Antwort könnte wie folgt aussehen:

```
telephony.routing.remove.rpc.return ((TRIP 0) (Annex_G 2))
```

Da *Annex G* das Protokoll *Q.931* nicht benutzt teilt das Vermittlungsmodul mit, daß die Adresse für diese Protokoll nicht gespeichert ist und damit nicht gelöscht werden kann.

2. `telephony.routing.remove.rpc (9 2 ''+49421300008'')`

Hier soll die Telefonnummer *+49421300008* bei den Protokollen *SIP* und *Annex G* in der pentadezimalen Adressfamilie gelöscht werden. Die Antwort sieht wie folgt aus:

```
telephony.routing.remove.rpc.return ((TRIP 0) (Annex_G 1))
```

*Annex G* ist dafür ausgelegt H.323-Adressen auszutauschen. Adressen des *SIP*-Protokolls kennt es nicht, und gibt einen entsprechenden Fehler-Code zurück, da dieses Protokoll für die Adresse nicht gelöscht werden konnte.

3. `telephony.routing.remove.rpc (13 3 ''+49421300002'')`

Die Adresse *+49421300002* soll für die Protokolle *SIP*, *RAS* und *Annex G* und der E.164-Adressfamilie gelöscht werden. Ein Rückgabewert wie folgt würde bedeuten:

```
telephony.routing.remove.rpc.return
((TRIP 4) (Annex_G 13))
```

Die Adresse konnte im *TRIP*-Modul für die Protokolle *SIP* und *Annex G* gelöscht werden, für das Protokoll *RAS* jedoch nicht. Ein Fehler-Code, wie im Beispiel für das *Annex G*-Modul angegeben, würde bedeuten, dass die angegebene Adresse für keines der Protokolle gelöscht werden konnte, weil entweder nicht vorhanden oder nicht unterstützt.

### 5.1.2 Bestätigen von Adressen

telephony.routing.commit	
<b>Kommandotyp</b>	RPC

Tabelle 5.3: Bestätigen von Adressen -Nachrichtenformat des Kommandos

telephony.routing.commit.return
---------------------------------

Tabelle 5.4: Bestätigen von Adressen -Nachrichtenformat der Rückantwort

Da in der Kommunikation zwischen IP-Exchange und einem Telefonie-Server es zu Häufungen von Nachrichten kommen kann — besonders wenn ein *Gateway* große Mengen an *PSTN*-Nummern bekannt gibt oder löscht — sammelt IP-Exchange alle Nachrichten, die Adressen hinzufügen oder löschen und verarbeitet sie in den Modulen erst dann, wenn der Telefonie-Server sie durch diese Nachricht freigibt.

Ein Beispiel erübrigt sich hier, da bei dieser Nachricht keine Parameter angegeben werden können.

### 5.1.3 Suchen von Adressen

Während des Aufbaus eines Telefongesprächs braucht der Telefonie-Server die Information, wo der gewünschte Gesprächsteilnehmer zu finden ist, d.h. bei welchem Telefonie-Server sein Telefon angemeldet ist und über welches

telephony.routing.query	
<b>Kommandotyp</b>	RPC
<b>Parameter</b>	<i>int</i> protocol <i>int</i> family <i>string</i> address
<b>Optionale Parameter</b>	<i>string</i> Liste von Schlüssel=Wert

Tabelle 5.5: Suchen von Adressen - Nachrichtenformat des Kommandos

Protokoll es kommunizieren kann bzw. will. Die Angaben zu Protokoll, Adressfamilie und optionalen Parametern sind analog zu Abschnitt 5.1.1.

Die Rückantwort besteht aus einer Liste von Tripeln im Format (Protokoll, Adresse der Vermittlungsstelle, optionale Informationen). Wenn ein Vermittlungsmodul zu einer Adresse mehrere Einträge findet, werden sie alle für die Rückantwort verwendet. Die optionalen Informationen enthalten Daten, die das jeweilige Protokoll zur Verfügung stellt - wie bei z.B. *Annex G* die Tarifinformationen.

telephony.routing.query.return	
<b>Parameter</b>	Liste von Tripeln < <i>int</i> Protokoll, <i>string</i> Adresse der Vermittlungsstelle, <i>string</i> optionale Informationen>

Tabelle 5.6: Suchen von Adressen - Nachrichtenformat der Rückantwort

### Beispiele für Adressen suchen

1. telephony.routing.query.rpc (2 1 ''+4930232345'')

Hier wird die Telefonnummer *+4930232345* mit dem Protokoll *Q.931* und der dezimalen Adressfamilie gesucht. Die Antwort könnte wie folgt aussehen:

```
telephony.routing.query.rpc.return
(2 ''Q-Gate.Berlin.de'')
```

## 2. telephony.routing.query.rpc (5 3 ''+73272214615'')

Die Telefonnummer *+73272214615* wird für die Protokolle *SIP* und *RAS* für die E.164-Familie gesucht. Mögliche Antworten wären:

```
telephony.routing.query.rpc.return
    ((1 ''Sip_Gate.somewhr.ru'')
     (4 ''anywhere.ru''))
```

oder

```
telephony.routing.query.rpc.return ((4 ''RAS-Gate.ru''))
```

oder

```
telephony.routing.query.rpc.return
    ((1 ''Sip-Gate.where.ru''))
```

Jede der Rückantworten stellt dar, welche Nachricht IP-Exchange an einen Telefonie-Server zurückgeben kann.

## 3. telephony.routing.query.rpc (13 1 ''+4989453455'')

Hier wird die Telefonnummer *+4989453455* für die Protokolle *Annex G*, *RAS* und *SIP* für die dezimale Adreßfamilie gesucht. Die Anzahl der möglichen Antworten ist bei so vielen möglichen Protokollen schon recht groß (hier nur Teilweise wiedergegeben):

```
telephony.routing.query.rpc.return
    ((1 ''Sip_Gate.Berlin.de'')
     (4 ''Ras.Berlin.de'')
     (8 ''H.323-Gate.Berlin.de''))
```

oder

```
telephony.routing.query.rpc.return
    ((4 ''RAS-Gate.de''))
```

oder

```
telephony.routing.query.rpc.return
    ((1 ''Sip-Gate.Berlin.de''))
```

Je mehr Protokolle erlaubt sind, desto mehr Antwortmöglichkeiten hat IP-Exchange, und die Vermittlungsstelle hat dann mehr Möglichkeiten, die Verbindung herzustellen.

### 5.1.4 Konfigurieren von Vermittlungsmodulen

Es können Situationen eintreten, in denen der Telefonie-Server Einstellungen bei einem Vermittlungsmodul verändern muß. Mit dieser Nachricht kann er die bestehenden Einstellungen abändern bzw. beim Start der Verbindung neu konfigurieren.

telephony.routing.configure	
<b>Kommandotyp</b>	RPC
<b>Parameter</b>	Liste von Strings

Tabelle 5.7: Konfiguration von Vermittlungsmodulen - Nachrichtenformat des Kommandos

Diese Nachricht enthält eine Liste von Strings, jeder String enthält eine Anweisung für die Vermittlungsmodule. Jedes Vermittlungsmodul erhält die gesamte Nachricht und muß aussortieren, welche der Anweisungen sie betreffen. In Anhang A.2 findet man die möglichen Konfigurations-Kommandos für das *TRIP*-Modul von IP-Exchange.

Eine Rückantwort auf diese Nachricht wird nicht gegeben.

#### Beispiele für Konfiguration von Vermittlungsmodulen

1. `telephony.routing.configure.rpc (''G_Port=4711''  
''G_Server=dustbin'' ''SIP_Port=1932'')`

Beim *TRIP*-Modul wird der *Annex G*-Port auf 4711 und der Server auf "Dustbin" gesetzt. Der *SIP*-Port wird auf 1932 gesetzt. Bei einem *Annex G*-Modul werden der Port und der Server für *Annex G* ebenfalls gesetzt. Die Anweisung für *SIP* wird ignoriert. Als Rückantwort wird immer zurück gegeben, daß die Nachricht angekommen ist und abgearbeitet wurde.

2. `telephony.routing.configure.rpc  
(''SIP_Server=bse.informatik.uni-bremen.de''  
''SIP_Port=1932'')`

Bei dieser Nachricht werden *SIP*-Server und *SIP*-Port neu gesetzt. Dies betrifft aber nur Module, die *SIP* unterstützen.

```
3. telephony.routing.configure.rpc (''Itad=4711''
  ''Port=6065'' ''Mode=Send/Receive'')
```

Diese Nachricht setzt beim *TRIP*-Modul die **ITAD** — dem Administrativen Bereich, dem das Modul zugehört — auf 4711, den Port, auf den es lauschen soll, auf 6065 und die Art der Kommunikation mit anderen *TRIP-Location Servern* auf “Send/Receive”. Ein *Annex G*-Modul würde diese Kommandos ignorieren.

### 5.1.5 Übermitteln von Statusinformationen

telephony.routing.status	
<b>Kommandotyp</b>	RPC

Tabelle 5.8: Übermitteln von Statusinformationen - Nachrichtenformat des Kommandos

Mit dieser Nachricht fordert der Telefonie-Server einen Statusbericht von IP-Exchange an.

Die Rückantwort enthält eine Liste von Statusinformationen, die den Telefonie-Server interessieren können. Die Liste besteht aus Tupeln, bei denen der erste Eintrag definiert, für welches Modul die Statusinformation gilt. Der zweite Eintrag besteht aus einer Liste von “Key=Value”-Paaren, die in Anhang A.3 definiert werden. Falls ein Modul keine Statusinformationen liefern kann, z.B. weil das Protokoll keine Datenbasis benutzt und keine Verbindungen zu Kommunikationspartnern hat, bleibt der zweite Eintrag leer.

telephony.routing.status.return	
<b>Parameter</b>	Eine Liste von Tupeln im Format (MSymbol Vermittlungsmodul, Liste von <string> mit Statusinformation)

Tabelle 5.9: Übermitteln von Statusinformationen - Nachrichtenformat der Rückantwort

**Beispiel für das Übermitteln von Statusinformationen**

Mit dem folgenden Befehl fordert der Telefonie-Server Statusinformationen der einzelnen Vermittlungsmodule bei IP-Exchange an:

```
telephony.routing.status.rpc ()
```

Die Rückantwort enthält die Angaben der einzelnen Vermittlungsmodule. In diesem Fall kennt das *TRIP*-Modul 25 Adressen und hat 3 Kommunikationspartner, die ebenfalls benannt werden. Das *Annex G*-Modul hat nur 3 Adressen im Speicher und nur eine lokale Verbindung offen.

```
telephony.routing.status.return
(
  (
    (TRIP)(''Addresses=25''
          ''Connections=3''
          ''Connection=134.102.218.68''
          ''Connection=bse.informatik.uni-bremen.de''
          ''Connection=localhost'')
  )
  (
    (Annex_G)(''Addresses=3''
              ''Connections=1''
              ''Connection=localhost'')
  )
)
```

**5.1.6 Beenden von IP-Exchange**

Wenn der Telefonie-Server der Meinung ist, daß IP-Exchange (vorläufig) nicht mehr benötigt wird, kann er mittel `mbus.quit` IP-Exchange explizit zum Programm-Ende auffordern.

mbus.quit
-----------

Tabelle 5.10: Beenden von IP-Exchange

Nach Erhalt dieser Nachricht schließt IP-Exchange alle offenen Verbindungen, fährt die Vermittlungsmodule herunter, meldet sich mittels `mbus-bye`

beim *Mbus* ab und beendet sich dann.

mbus.bye
----------

Tabelle 5.11: Ende-Nachricht von IP-Exchange

Auf Beispiele für diese einfachen Nachrichten wird hier verzichtet.

## 5.2 Schlußfolgerung

Die hier vorgestellten *Mbus*-Nachrichten ermöglichen eine ausreichende Kommunikation zwischen einem Telefonie-Server und IP-Exchange. So ist es IP-Exchange möglich, Adressen effektiv zu verwalten, und dem Telefonie-Server bei Bedarf Informationen für einen Anruf oder zum Aufbau einer Statistik zur Verfügung zu stellen.



# Kapitel 6

## Der TRIP-Location Server

Einen Teil dieser Arbeit stellt die Implementierung eines Vermittlungsmoduls für das *TRIP*-Protokoll dar, das im Kapitel 2.2 vorgestellt wurde.

*TRIP* verbreitet Telefon-Routing-Informationen innerhalb und zwischen *Administrativen Bereichen*. Dafür werden **Location Server** benutzt. *TRIP* ist dafür ausgelegt, daß die ausgetauschten Informationen von unterschiedlichen Signalisierungsprotokollen stammen können. Die Informationen werden in einer Datenbasis bereit gehalten und aktualisiert.

In diesem Kapitel wird nun Implementierung eines *Location Servers* erläutert.

### 6.1 Kommunikation mit IP-Exchange

Der *Location Server* benutzt die dafür vorgesehenen Schnittstellen von IP-Exchange. Dadurch wurde es notwendig, daß der *LS* Adressen seiner Datenbasis hinzufügen oder aus dieser löschen können muß. Auch ein Durchsuchen der Datenbasis soll möglich sein. Da *TRIP* erlaubt, daß Adressen in verschiedenen Protokollarten kodiert sein dürfen, muß der *LS* beim Eintragen, Löschen oder Suchen achten, welches Protokoll die gesuchte Adresse haben darf.

Der *Location Server* kann von IP-Exchange über einen entsprechenden Aufruf einer Schnittstelle konfiguriert werden. Sonst werden Standard-Werte benutzt oder — falls verfügbar — die Werte, die IP-Exchange bei der letzten Konfiguration bekannt gegeben hat (siehe Anhang A.2).

## 6.2 Datenklassen

Die für *LS* wichtigste Datenklasse stellt alle vorhandenen **Routeninformationen** bereit. Dort wird die Telefonnummer gespeichert und auch welcher Server für sie zuständig ist. Auch werden die weiteren *TRIP*-Eigenschaften, wie z.B. die Liste der *ITADs*, die die Adresse durchlaufen hat oder auch durch die sie gerouted werden soll, dort gespeichert. Jede der Klassen bietet Methoden an, die Daten zu kodieren und zu dekodieren. Dadurch kann die Repräsentation der Daten innerhalb der Klassen ohne Probleme geändert werden.

Die Datenbank von *LS* besteht aus einer Baumstruktur. Jede Ziffer einer Telefonnummer stellt einen Ast des Baums dar. Am Blatt, dort wo die gesamte Nummer auf Äste verteilt wurde, stehen die dazugehörigen Informationen. Dieses System hat den Vorteil, daß das Anlegen, Löschen oder Suchen von der Größe der Datenbasis unabhängig ist und der Aufwand linear zur Länge der Telefonnummer wächst. Der Bedarf an Speicher richtet sich nach der Länge der einzelnen Telefonnummern, wobei nebeneinanderliegende Telefonnummern weniger Speicher verbrauchen. Durch die Baumstruktur bedingt benötigt jeder Eintrag auch bei großen Mengen von Telefonnummern nur geringe Mengen an Speicher, insgesamt könnte aber auch ein großer Speicher nicht ausreichend sein (siehe Kapitel 7.1 für genauere Angaben). Für sehr große Mengen an Telefonnummern ist es notwendig, daß jeder *Location Server* möglichst viele Telefonnummern zusammenfaßt (Route Aggregation, siehe Kapitel 2.2.5).

## 6.3 Programmablauf

Beim Aufrufen eines Vermittlungsmoduls durch IP-Exchange sollte zuerst nachgesehen werden, ob eine Konfigurationsdatei vorhanden ist. Wenn keine vorhanden ist, werden die für den Betrieb des Vermittlungsmoduls notwendigen Variablen mit einem Standardwert initialisiert, sonst werden die Daten aus der Datei verwendet. Um Daten mit anderen *LS* austauschen zu können, müssen deren Adressen bekannt sein. Diese werden - falls sie in der Konfigurationsdatei stehen - in eine Klasse eingelesen. Die möglichen Einstellungen in der Konfigurationsdatei sind unter Anhang A.2 zu finden.

Danach werden zwei *Threads* gestartet. Der eine wartet auf Programmaufrufe von der IP-Exchange. Der andere versucht, Verbindungen zu allen eingele-

senen Kommunikationspartnern aufzubauen. Für jede gelungene Verbindung wird wiederum ein eigener *Thread* zur Verfügung gestellt. Falls die Verbindung zu einem Partner abbricht bzw. abgebrochen wird, versucht *LS* nach einer Weile, die Verbindung wieder herzustellen.

Jede einzelne Verbindung wird dem Protokoll entsprechend aufgebaut und unterbrochen, wenn ein Fehler auftritt. Nachdem die Kommunikation initialisiert wurde, wird auf eine Reihe von Ereignissen gewartet. Erlaubt sind Verbindungsabbau (NOTIFICATION), Informationen über Telefonnummern (UPDATE) oder Verbindung aufrechterhalten (KEEPALIVE) des Kommunikationspartners oder das Versenden eigener, durch andere **Threads** gesammelte, Daten. Ein erneuter Aufbauwunsch der Verbindung wäre nicht protokollkonform und muß ein Verbindungsabbruch auslösen. Bei einem UPDATE werden die neuen Informationen in die Datenbasis eingearbeitet.

Jeder der *Threads* ist in der Lage, die Datenbasis gemäß seinen Informationen zu verändern, d.h. Routing-Informationen zu löschen oder hinzuzufügen. Daher ist es notwendig, daß zu einem Zeitpunkt jeweils nur ein *Thread* die Datenbasis verändern bzw. abfragen darf. Dies wird durch **Semaphoren** bewerkstelligt. Sobald einer der *Threads* Aktionen durchführt, die die gemeinsame Datenbasis betreffen, versucht er diese zu sperren — dabei muß er evtl. auf die Freigabe durch einen anderen *Thread* warten. Erst nach erfolgter Sperrung kann die Datenbasis geändert werden.

Aufrufe von IP-Exchange betreffen in fast allen Fällen die Datenbasis, in der die Telefoninformationen festgehalten werden. Es können Daten eingetragen, gelöscht oder es können Informationen zu einer bestimmten Telefonnummer gesucht werden. Zusätzlich kann der *LS* von IP-Exchange konfiguriert werden und nach Statusinformationen befragt werden, wobei hier die Anzahl der Adressen und Verbindungen zu anderen *LS* zurückgegeben werden.

## 6.4 Das Programm lctest

Derzeit sind keine aktuellen, frei verfügbaren Implementierungen von *Location Servern* vorhanden. Die von **Vovida**[9] zur Verfügung gestellte Implementierung stammt aus dem Jahre 2000 und ist vollkommen veraltet. Deswegen wurde es notwendig ein Programm zu schreiben, das einen *Location Server* zur Verfügung stellt, aber nicht mit IP-Exchange verbunden ist. Mit diesem Programm wurde es dann möglich, verschiedenste Konfigurationen und

*ITAD*-Topologien (IP Telephony Administrative Domain) zu testen.

Im folgenden werden die Topologien vorgestellt, für die eine Überprüfung der Korrektheit erforderlich schien. Die Sterne stellen einzelne **ITADs** dar. Jede *ITAD* kann eine beliebige Anzahl von **Location Servern** betreiben. Die Verbindungen der *LS* innerhalb der *ITADs* wird hier weggelassen, da Informationen externer *Location Server* innerhalb einer *ITAD* weitergeleitet werden, ohne daß eine Verarbeitung der Daten stattfindet. Bei diesen Beispielen wird davon ausgegangen, daß die jeweiligen Informationen der *LS* bei den Partnern schon vorliegen.

Zusätzlich wird gezeigt, wie sich die Änderungen in der Topologie auf die **Route Aggregation** auswirken.

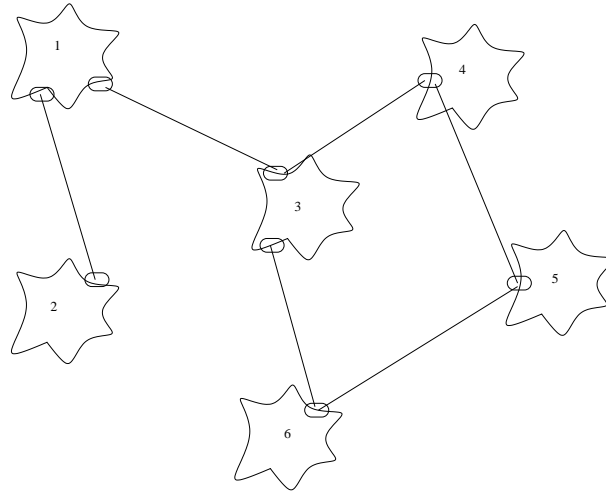


Abbildung 6.1: ITAD-Topologie mit Ring

### ITAD-Topologie mit Ring

Die erste Topologie (Abb. 6.1) enthält eine Ringverbindung. Sie kann entstehen, da die einzelnen *Location Server* bzw. die zuständigen *ITADs* den anderen nicht mitteilen, zu welchen anderen *LS* sie Kontakt halten. Eine Route, die ein *LS* verbreitet, erreicht die anderen *Location Server* bis zu zweimal. Hier muß erreicht werden, daß jede Information nur einmal in die aktuelle Datenbank aufgenommen wird, die doppelte Information aber zwischengespeichert wird — falls der Nachbar-*LS* ausfällt, dessen Information

benutzt wurde kann die doppelte dann verwendet werden.

Eine Adresse, die bei *LS 6* ihren Ursprung hat, kann auf unterschiedlichen Wegen bei *LS 4* eintreffen (über *LS 3* und 5). Wenn später *LS 3* ausfällt müssen alle seine Adressen gelöscht werden, auch diejenigen aus *LS 6*. Dann kann aber *LS 4* die redundante Information von *LS 4* benutzen.

Wenn zwei *LS* einen Adreßraum abdecken, z.B. wenn *LS 1* die Ziffern 0 bis 4 und *LS 2* die Ziffern 5 bis 9 eines Bereichs der Telefonadressen abdecken, wird daraus eine neue, kürzere Telefonadresse generiert und nur diese weiter verbreitet (**Route Aggregation**), der genaue Ablauf wird in Kapitel 2.2.5 beschrieben. In diesem Fall wird entweder *LS 1* oder *LS 2* die Route zusammenstellen.

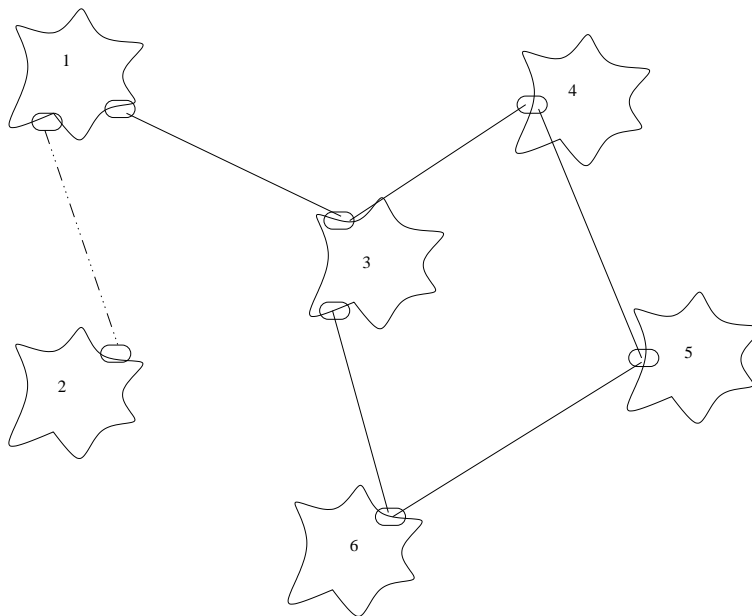


Abbildung 6.2: ITAD-Topologie mit abgetrennter ITAD

### ITAD-Topologie mit abgetrennter ITAD

Bei dieser Topologie (Abbildung 6.2) ist ein *Location Server* — *LS 2* — ausgefallen. Sobald *LS 1* dies bemerkt, muß er, gemäß des *TRIP*-Protokolls, bei allen bestehenden Verbindungen die Informationen von 2 entfernen, auch

wenn die Telefon-Endpunkte, die bei *LS 2* angemeldet sind, selber noch erreichbar sind. In diesem Fall betrifft das *LS 1*. Dieser entfernt dann wiederum die Informationen von *LS 2* bei seinen Verbindungen. Sollte *LS 2* wieder arbeiten, meldet er alle seine Informationen bei *LS 1* wieder an und dieser leitet sie dann weiter.

Wenn *LS 2* Telefonnummern bereitstellt, die für die *Route Aggregation* verwendet wurden, muß die generierte Telefonadresse ebenfalls gelöscht werden und die übriggebliebenen Adressen des (der) anderen neu bekannt gemacht werden.

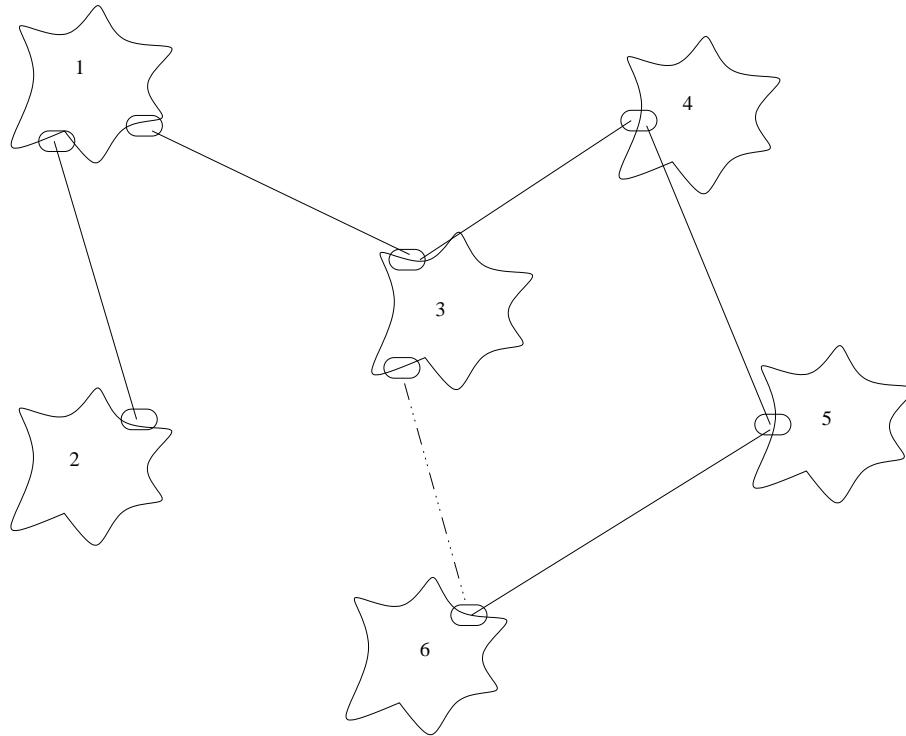


Abbildung 6.3: ITAD-Topologie mit abgetrennter ITAD(2)

Die Topologie in (Abbildung 6.3) zeigt eine unterbrochene Verbindung zwischen *LS 6* und *LS 3*. Dadurch ist der Ring zwar aufgebrochen, aber alle Informationen erreichen - evtl. über Umwege - alle *Location Server*.

Wenn diese beiden *LS* Telefonadressen für die *Route Aggregation* bereit gestellt haben, muß die generierte Telefonadresse zunächst gelöscht werden. Beide bemerken den Verbindungsabbruch sofort und löschen die Adressen des jeweiligen anderen. Da aber die Adressen der beiden über die *LS 4* und *5*

weitergeleitet werden, kann die kürzere Telefonadresse etwas später wieder generiert werden. Allerdings ist es ungewiß, bei welchem *Location Server* die Informationen zuerst komplett vorliegen, da dies von den Laufzeiten der einzelnen Nachrichten zwischen den *LS* abhängt. Von *LS* 3 bis *LS* 6 kann es jeder sein.

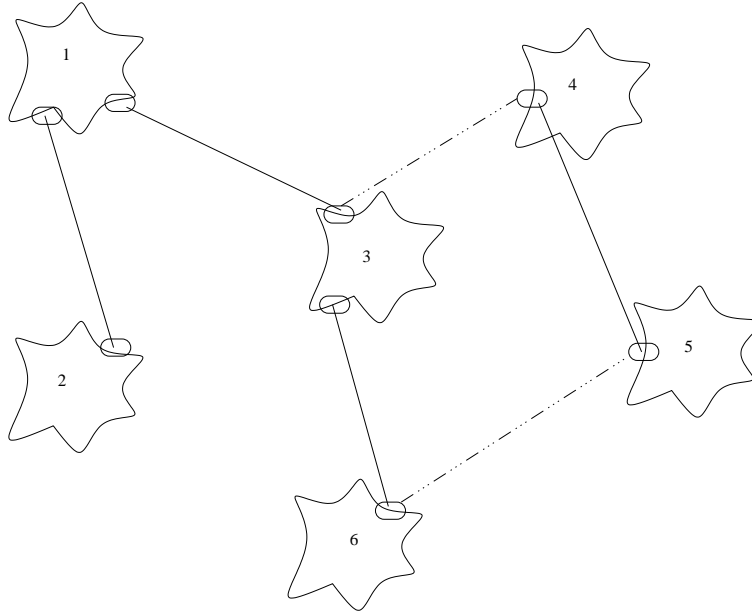


Abbildung 6.4: ITAD-Topologie in zwei Teile getrennt

### ITAD-Topologie in zwei Teile getrennt

Bei dieser Topologie (Abbildung 6.4) wird der Ring in zwei unabhängige Netze getrennt. In jedem der Teilnetze dürfen nur die Informationen erhalten bleiben, die von den *Location Servern* in dem Teilnetz ausgehen.

Wenn die *Route Aggregation* innerhalb eines der Teilnetze stattfand - also entweder zwischen *LS* 4 und *LS* 5, oder zwischen den anderen - ändert sich nichts. Ansonsten muß die generierte Telefonadresse gelöscht werden.

### Testumgebung

Für die Testläufe wurden sechs verschiedene Computer benutzt, die jeweils eine eigene *ITAD* dargestellt haben. Jeder der Rechner war ein Pentium-II

mit 400 Mhz. Obwohl keine großen Datenmengen verfügbar waren — im realen Betrieb werden Tausende von Adressen verwaltet — wurde deutlich, daß das Verhalten des implementierten *Location Server* den Erwartungen entsprach. Das reale Verhalten der einzelnen *LS* entsprach den Überlegungen, und es deutet alles darauf hin, daß der *Location Server* einsatzbereit ist.

Trotzdem besteht bei Testläufen, die nur das eigene Programm beinhalten, das Problem, daß Fehler der Implementierung nicht immer aufgedeckt werden da sie sich gegenseitig aufheben können. Wenn z.B. ein Fehler in der Kodierung einer Nachricht (zufällig) durch einen Fehler in der Dekodierung der Nachricht verdeckt wird, sieht das Ergebnis fehlerfrei aus. Allerdings werden bei *TRIP* Fehler in den einzelnen Nachrichten, falls sie bemerkt werden, zurückgegeben, bevor die Verbindung aufgelöst wird. Daher sollten Fehler im Betrieb schnell gefunden und behoben werden können.

# Kapitel 7

## Verwendung der IP-Exchange

In diesem Kapitel wird kurz erläutert, wie **IP-Exchange** betriebsbereit gemacht werden kann.

### 7.1 Systemanforderungen

**IP-Exchange** wurde in einer Linux-Umgebung entwickelt. Getestet wurde es mit Distributionen von Red-Hat und SuSE. Die Portierung auf Windows, sollte ohne Probleme möglich sein, da alle benutzten Programm-Bibliotheken (wie z.B. *Mbus*) inzwischen dafür übersetzt wurde. Die Verwendung andere Plattformen bedingt vor allem das Vorhandensein des *Mbus*.

Der genaue Bedarf von IP-Exchange an System-Ressourcen ist schwer abzuschätzen, da nicht genügend reale Daten vorhanden sind. Für den Austausch von Nachrichten zwischen IP-Exchange und dem Telefonie-Server auf der einen, sowie zwischen den Vermittlungsmodulen und ihren Kommunikationspartnern wird eine gewisse Bandbreite benötigt. Bei den Heute verfügbaren Kapazitäten sollten die Nachrichten keine Probleme bereiten.

Zusätzlich wird für den Aufbau einer Datenbasis Speicherkapazität benötigt, z.B. kann beim **TRIP**-Vermittlungsmodul eine Routeninformation maximal 4 KB groß sein. Mit 256 MB Hauptspeicher könnten dann theoretisch 64.000 Routeninformationen verwaltet werden. Tatsächlich wird aber eine Information nur wenige hundert Bytes verbrauchen. Zusätzlich benötigt jeder Eintrag der Routeninformationen in die Datenbasis etwas Speicher, bei 64.000 Einträgen wären das ca. 250 KB.

Die Anzahl der zu erwartenden Adressen ist bisher auch nicht genau bekannt. Zwar sind allein in Deutschland, bei ca. 25 Millionen Haushalten und ca. 50

Millionen Handys, gut 70 Millionen Telefone vorhanden. Da aber *Gateways* ganze Bereiche registrieren (z.B. alle Telefonnummern im Bereich 0421) hängt die Anzahl der zu verwaltenden Adressen von der Anzahl der *Gateways* und ihrer jeweiligen Einstellungen ab.

Die Geschwindigkeit des Prozessors bestimmt mit, wie schnell eine Datenbasis durchsucht werden kann, ein Prozessor mit 700 Mhz sollte ausreichend sein.

An Software benötigt **IP-Exchange** eine Implementierung des *Message Bus*, damit Nachrichten mit dem Telefonie-Server ausgetauscht werden können.

Da IP-Exchange mit einem Telefonie-Server zusammenarbeitet, muß dieser ebenfalls vorhanden und ansprechbar sein, da ansonsten IP-Exchange zwar funktionsfähig ist, aber keine Funktion ausführt.

## 7.2 Interaktion mit dem Uni-Gatekeeper

Durch die Verwendung des *Message Bus* ist IP-Exchange in der Lage, mit dem **Gatekeeper** der Universität Bremen zu arbeiten. Zeitgleich zu dieser Arbeit wurde der *Gatekeeper* so erweitert, daß die in Kapitel 5 definierte Kommunikation zwischen Telefonie-Server und IP-Exchange ermöglicht wird.

Der Aufbau der Kommunikation zwischen *Gatekeeper* und IP-Exchange läuft automatisch ab. Nach Programmstart wartet IP-Exchange darauf, daß der *Gatekeeper* die gemeinsame Kommunikation beginnt.

## 7.3 Installation

Nach dem Entpacken des Archivs sollen im Verzeichnis von *IP-Exchange* die Befehle „./configure“ und „./make all“ ausgeführt werden. Danach kann *IP-Exchange* mit dem Befehl „./exchange“ gestartet werden.

## 7.4 Konfiguration

IP-Exchange selbst enthält keine Komponenten die konfiguriert werden könnten. Die einzelnen Vermittlungsmodule könnten dagegen zusätzliche Informa-

tionen benötigen, die in einer Konfigurationsdatei stehen. Diese sind versteckt im Verzeichnis von IP-Exchange zu finden. Zusätzlich muß der *Mbus* konfiguriert werden, damit IP-Exchange arbeiten kann. Eine Übersicht der Dateien findet sich in Anhang A.

## 7.5 Fehlerhandhabung

Bei C++-Programmen ist es leider nicht immer möglich, exakt zu ermitteln, wo genau ein Fehler aufgetreten ist. Im allgemeinen bricht das Programm ab, und es gibt keinen Hinweis welche Funktion bzw. welcher Programmaufruf den Fehler ausgelöst hat.

Aus diesem Grund schreibt z.B. das *TRIP*-Modul in eine **Log**-Datei, was gerade ausgeführt wird. Bei einem Programmabbruch kann dann in der *Log*-Datei ungefähr ersehen werden, bei welcher internen Funktion der Fehler lag. Protokoll-Fehler, die bei der Kommunikation mit anderen *TRIP*-Modulen auftreten, führen zu einem Abbruch dieser Kommunikation, aber nicht zu einem Programmabbruch.

Die *Log*-Datei enthält Text-Einträge die neben dem Zeitpunkt die aktuelle Aktion des Programms enthalten, wie z.B. Daten versendet, erhalten oder dekodiert.



# Kapitel 8

## Zusammenfassung und Ausblick

Mit der Entwicklung von *IP-Exchange* wurde eine Kommunikationslücke zwischen Telefonie-Servern und einzelnen Vermittlungsmodulen geschlossen. Durch die Verwendung des *Message Busses* und des modularen Aufbaus der Vermittlungsmodule kann *IP-Exchange* einfach erweitert werden und mit verschiedenen Telefonie-Servern zusammenarbeiten. Durch die jederzeit mögliche Verwendung von noch unbekanntem Parametern ist *IP-Exchange* ohne Probleme erweiterbar.

*IP-Exchange* arbeitet im Moment nur mit dem in der AG Rechnernetze benutzten *H.323-Gatekeeper* zusammen. Durch die entwickelten *Mbus*-Schnittstellen kann *IP-Exchange* auch mit anderen Telefonie-Servern interagieren, und durch die festgelegten internen Schnittstellen können neue Vermittlungsmodule in *IP-Exchange* integriert werden.

Durch den modularen Aufbau ist es möglich, verschiedene Systeme des Sammelns von Adressen zu realisieren. Während einige Module alle vorhandenen Daten an Gesprächspartner weiter geben (**Push**), wie z.B. *TRIP*, geben andere die Daten erst bei Nachfrage weiter (**Pull**), wie z.B. *Annex G*. Durch die Verwendung dieser Systeme entsteht der Vorteil, daß, wenn eine Adresse noch nicht vorhanden ist und damit durch ein *TRIP*-Modul nicht aufgelöst werden kann, diese über *Annex G* evtl. in Erfahrung gebracht werden kann. Wenn sie aber vorhanden ist, kann Zeit gespart werden.

Durch die Möglichkeit neue, noch nicht bekannte Steuerinformationen vom Telefonie-Server an die Vermittlungsmodule weiterzuleiten, kann die Grundstruktur von *IP-Exchange* beliebig lange verwendet werden.

*IP-Exchange* selbst ist ebenfalls erweiterungsfähig und kann neue Funktionen

zur Verfügung stellen. Denkbar wäre eine Abschaltung der einzelnen Vermittlungsmodule durch den Telefonie-Server. Diese Funktion könnte durch *Mbus*-Nachrichten erreicht werden.

Auch wäre es realisierbar, das ein Telefonie-Server mehrere IP-Exchange-Instanzen benutzt, wobei jede eine andere Aufgabenstellung hat. Da jede *Mbus*-Entity eine eigene Adresse hat, können die IP-Exchange-Instanzen damit unterschieden werden.

Damit wäre es dann auch möglich, das mehrere lokale Telefonie-Server IP-Exchange gemeinsam nutzen. Dafür wäre es dann zusätzlich notwendig, daß jede lokale Adresse genau einem Telefonie-Server zugeordnet werden kann, um den die Datenbasis konsistent halten zu können.

Auch könnte der Einbau bzw. Einsatz neuer Vermittlungsmodule automatisiert werden. Dies erfordert zwar einige Änderungen im Code von IP-Exchange (die einzelnen Schnittstellenaufrufe der Vermittlungsmodule werden in eine Meta-Klasse geschrieben, die dann alle gefundenen Module anspricht), aber dadurch wird es IP-Exchange dann Möglich neue Vermittlungsmodule ohne Code-Änderungen zu benutzen.

Da die Entwicklung in der IP-Telefonie sich gerade am Anfang befindet, ist es schwer abzuschätzen welche Richtung sie einschlagen wird. Schon jetzt gibt es Protokolle (siehe Kapitel 2.3) die mit einzelnen Vermittlungsprotokollen (in diesem Fall *TRIP*) konkurrieren, und es ist noch nicht absehbar, ob — und wenn ja, welche — Protokolle sich durchsetzen werden.

Gerade zu dieser Zeit bietet IP-Exchange die Möglichkeit, unabhängig von aktuellen Entwicklungen arbeiten zu können. Durch die definierten Schnittstellen und *Mbus*-Kommandos ist die Kommunikation zwischen Telefonie-Servern und Vermittlungsmodulen zum ersten mal konkretisiert worden. Die Kommunikation ist einheitlich, egal wie Telefonie-Server und Vermittlungsmodule dann tatsächlich implementiert werden.

IP-Exchange wird, dank modularer Bauweise und flexiblen Nachrichtenformaten, auch in Zukunft nutzbar bleiben, egal wie die weitere Entwicklung aussehen mag.

# Anhang A

## Konfigurationsdateien

Obwohl IP-Exchange die Möglichkeit bietet, die einzelnen Vermittlungsmodule zur Laufzeit zu konfigurieren, ist es oft sinnvoll, bestimmte langfristige Daten — wie z.B. gewünschte Kommunikationspartner — so abzulegen, daß sie nicht bei jedem Neustart von IP-Exchange (oder auch des Telefonie-Servers) neu konfiguriert werden müssen. Dazu bietet es sich an, diese Informationen in eine Datei abzulegen, die dann beim Starten von IP-Exchange ausgelesen wird. Da die Kommunikation zum Telefonie-Server über den *Mbus* erfolgt muß auch dafür eine Konfiguration erfolgen.

### A.1 Konfiguration der IP-Exchange

Die IP-Exchange selbst hat bisher keine Einstellungsmöglichkeiten. Da IP-Exchange mit einem Telefonie-Server auf lokaler Ebene kommuniziert werden keine Angaben benötigt, wo dieser zu finden ist. Daher ist eine Konfiguration zum Programmanfang nicht notwendig.

Mögliche Einstellungen für eine Konfigurationsdatei wären die explizite Angabe des Kommunikationspartners, oder welche Vermittlungsmodule IP-Exchange verwenden soll und welche ausgeschaltet werden können. Diese Funktionen müssen dann aber in IP-Exchange noch implementiert werden.

## A.2 Konfiguration des TRIP-Location Servers

Die Konfigurationsdatei - die im gleichen Verzeichnis wie IP-Exchange erwartet wird - ermöglicht es, dass bestimmte Parameter des *LS* schon beim Programmstart initialisiert werden. Die Konfigurationsdatei enthält Texteinträge, bei denen jede Zeile einen Parametereintrag darstellt. Mögliche Parameter werden in einer Liste von "Schlüssel=Wert"-Paaren[19] aufgelistet. Dieselben Parameter können zur Laufzeit von einem Telefonie-Server über IP-Exchange an den *Location Server* übermittelt werden.

- **Itad=Int** : Die eigene **ITAD**-Nummer.
- **Intern=IP-Addr/Hostname** : interne *LS*, die Teil der eigenen *ITAD* sind, und mit denen kommuniziert werden soll.
- **Extern=IP-Addr/Hostname** : externe Kommunikationspartner, d.h. *LS* die außerhalb der eigenen *ITAD* liegen.
- **Holdtime=Int** : Die maximale Zeit, die zwischen zwei Nachrichten vergehen darf.
- **Mode=SEND\_ONLY/SEND&RECEIVE/RECEIVE\_ONLY** : der für den *LS* gültige Sendemodus.
- **G\_Server=IP-Addr/Hostname** : Die Angabe, unter welcher IP-Adresse der Telefonie-Server auf *Annex G*-Nachrichten wartet.
- **G\_Port=Int** : Die Port-Nummer des Telefonie-Servers für *Annex G*-Nachrichten.
- **SIP\_Server=IP-Addr/Hostname** : Die Angabe, unter welcher IP-Adresse der Telefonie-Server auf *SIP*-Nachrichten wartet.
- **SIP\_Port=Int** : Die Port-Nummer des Telefonie-Servers für *SIP*-Nachrichten.
- **RAS\_Server=IP-Addr/Hostname** : Die IP-Adresse des Telefonie-Servers für *H.225.0-RAS*-Nachrichten.
- **RAS\_Port=Int** : Die Port-Nummer des Telefonie-Servers für *H.225.0-RAS*-Nachrichten.

- **Q\_Server=IP-Addr/Hostname** : Die IP-Adresse des Telefonie-Servers für *H.225.0-Q.931*-Nachrichten.
- **Q\_Port=Int** : Die Port-Nummer des Telefonie-Servers für *H.225.0-Q.931*-Nachrichten.

Damit der Telefonie-Server eigene Adressen über *TRIP* verbreiten lassen kann ist es notwendig, daß Angaben gemacht werden, welche Protokolle von ihm verwendet werden — dies wird durch die *Mbus*-Nachrichten zum Eintragen von Adressen realisiert. Zusätzlich muß *TRIP* noch wissen, wo (Adresse und Port) die jeweils zuständigen Server zu finden sind, wobei natürlich nur die Server benannt werden müssen, die dann auch wirklich verwendet werden.

Da sich diese Informationen selten Ändern bietet es sich an, sie in die Konfigurationsdatei zu übernehmen. Sollte *TRIP* später erweitert werden, müssen für jedes neue Protokoll die Angabe des Servers und des Ports hinzugefügt werden.

Unbekannte Schlüssel werden ignoriert. Falls IP-Exchange Änderungen an der Konfiguration vornimmt, wird die Datei entsprechend angepaßt.

## A.3 Mögliche Statusinformationen

Die *Mbus*-Status-Nachricht kann folgende Einträge enthalten. Wie immer kann diese Liste erweitert werden, ohne das die Funktion bestehender Module eingeschränkt wird.

- **Addresses=Int** : Die Anzahl der lokalen und externen Adressen, die dem Vermittlungsmodul bekannt sind
- **Connections=Int** : Die Anzahl der im Augenblick offenen Kommunikationsbeziehungen
- **Connect=IP-Adresse/Hostname** : Die Adresse einer offenen Kommunikationsbeziehung

## A.4 Mbus-Konfiguration

Eine genaue und ausführliche Beschreibung der **Mbus**-Konfiguration ist in [18] zu finden. Im folgenden Auszug des Drafts können die notwendigen An-

gaben für die Datei ebenfalls entnommen werden.

Die Konfigurationsdatei soll den Namen ".mbus" haben und im HOME-Verzeichnis des Benutzers liegen. Die Datei muß so angelegt werden, daß andere Benutzer diese Datei weder lesen noch überschreiben dürfen. Die Datei sollte folgende Felder enthalten, wobei CONFIG\_VERSION, HASHKEY und ENCRYPTIONKEY auf jeden Fall enthalten sein müssen. Die einzelnen Einträge sind dann in der Form von "Schlüssel=Wert"-Paaren[19] abgelegt.

```
mbus_topic      = ''[MBUS]''
version_info    = ''CONFIG_VERSION= x''
hashkey_info    = ''HASHKEY= key_value''
encrkey_info    = ''ENCRYPTIONKEY= key_value''
scope_info      = ''SCOPE= scope''
port_info       = ''PORT= port''
address_info    = ''ADDRESS= address''
```

Ohne die Konfigurationsdatei kann keine Anwendung gestartet werden, die den *Mbus* verwendet. Damit eine Kommunikation mit einem Telefonie-Server möglich ist, sollte die Datei von beiden Programmen genutzt werden.

# Anhang B

## Verwendete Mbus-Nachrichten

Die folgenden Tabellen geben nochmals eine Übersicht welche, Kommandos für die Kommunikation über den *Message Bus* zwischen einem Telefonie-Server und IP-Exchange bereitstehen.

telephony.routing.add	
<b>Kommandotyp</b>	RPC
<b>Parameter</b>	<i>int</i> protocol
	<i>int</i> family
	<i>string</i> address
<b>Optionale Parameter</b>	<i>string</i> Liste von Schlüssel=Wert

telephony.routing.add.return	
<b>Kommandotyp</b>	RPC-Antwort
<b>Parameter</b>	Liste von Tupeln < <i>string</i> Vermittlungsmodul, <i>int</i> Ergebnis>

telephony.routing.remove	
<b>Kommandotyp</b>	RPC
<b>Parameter</b>	protocol
	family
	address
<b>Optionale Parameter</b>	<i>string</i> Liste von Schlüssel=Wert

telephony.routing.remove.return	
<b>Kommandotyp</b>	RPC-Antwort
<b>Parameter</b>	Liste von Tupeln < <i>string</i> Vermittlungsmodul, <i>int</i> Ergebnis>

telephony.routing.query	
<b>Kommandotyp</b>	RPC
<b>Parameter</b>	<i>int</i> protocol
	<i>int</i> family
	<i>string</i> address
<b>Optionale Parameter</b>	<i>string</i> Liste von Schlüssel=Wert

telephony.routing.query.return	
<b>Parameter</b>	Listen von Tripeln < <i>int</i> Protokoll, <i>string</i> Adresse der Vermittlungsstelle, <i>string</i> optionale Informationen>

telephony.routing.configure	
<b>Kommandotyp</b>	RPC
<b>Parameter</b>	Liste von Strings

telephony.routing.status	
<b>Kommandotyp</b>	RPC

telephony.routing.status.return	
<b>Parameter</b>	Eine Liste von Tupeln im Format (MSymbol Vermittlungsmodul, Liste von < <i>string</i> > mit Statusinformation)

telephony.routing.commit	
<b>Kommandotyp</b>	RPC

telephony.routing.commit.return
---------------------------------

mbus.quit	
<b>Kommandotyp</b>	Mbus-Nachricht für Programmende

mbus.bye	
<b>Kommandotyp</b>	Mbus-Benachrichtigung beim Programmende



# Literaturverzeichnis

- [1] Wess, Holger. *Geschichte der Telegraphie*  
<http://www.devcon3.de/telegraphie/phone.html>
- [2] Y. Rekhter and T. Li *Border Gateway Protocol 4 (BGP-4)*  
IETF RFC 1771, März 1995.
- [3] Rosenberg, Salama, Squire *Telephony Routing over IP*  
IETF RFC 3219, Januar 2002
- [4] J. Moy *Open Shortest Path First Version 2*  
IETF RFC 2328, 1998.
- [5] ITU-T Empfehlung *H.323.H.225.0 Annex G*  
[http://standard.pictel.com/ftp/avc-site/till\\_0012/9905\\_San/H.225.0\\_-Annex\\_G990528.zip](http://standard.pictel.com/ftp/avc-site/till_0012/9905_San/H.225.0_-Annex_G990528.zip)
- [6] Ott, Kutscher, Perkins *The Message Bus: A Platform for Component-based Conferencing Applications*  
[http://www.mbus.org/mbus\\_cbg2000.pdf](http://www.mbus.org/mbus_cbg2000.pdf)
- [7] Ott, Perkins, Kutscher *A Message Bus for Local Coordination*  
<http://www.mbus.org/drafts/draft-ietf-mmusic-mbus-transport-06.html>
- [8] Kutscher, Dirk *The Message Bus: Guidelines for Application Profile Writers*  
<http://www.mbus.org/drafts/draft-ietf-mmusic-mbus-guidelines-00.html>
- [9] Vovida, <http://www.vovida.org/protocols/downloads/trip/>
- [10] ITU-T Empfehlung *H.323v4*  
[http://standard.pictel.com/avc-site/till\\_0012/0011\\_Gen/H323v4-final\\_010206.zip](http://standard.pictel.com/avc-site/till_0012/0011_Gen/H323v4-final_010206.zip)

- [11] Handley, Schulzrinne, Schooler & Rosenberg *SIP: Session Initiation Protocol*  
IETF RFC 2543, März 1999
- [12] ITU-T Empfehlung *H.225.0*  
[http://standard.pictel.com/ftp/avc-site/till\\_0012/0011\\_Gen/H2250v4-final\\_010317.zip](http://standard.pictel.com/ftp/avc-site/till_0012/0011_Gen/H2250v4-final_010317.zip)
- [13] Zimmermann, D. *The finger user information protocol*  
IETF RFC 1288, Dezember 1991
- [14] ITU-T Empfehlung *H.245*  
[http://standard.pictel.com/ftp/avc-site/till\\_0012/0011\\_Gen/H245Version7.zip](http://standard.pictel.com/ftp/avc-site/till_0012/0011_Gen/H245Version7.zip)
- [15] ITU-T Empfehlung *T.120* <http://www.itut.int/>
- [16] ITU-T Empfehlung *G.711* <http://www.itut.int/>
- [17] ITU-T Empfehlung *H.261* <http://www.itut.int/>
- [18] Ott, Perkins & Kutscher *A Message Bus for Local Coordination*  
<http://www.mbus.org/drafts/draft-ietf.mmusic-mbus-transport-06.txt>
- [19] Crocker & Overell *Augmented BNF for Syntax Specifications: ABNF*  
IETF RFC 2234, November 1997
- [20] Mealling & Daniel *The Naming Authority Pointer (NAPTR) DNS Resource Record*  
IETF RFC 2915, September 2000
- [21] R. Faltstrom *E.164 numbers and DNS*  
IETF RFC 2916, September 2000
- [22] *Siemens Online Lexikon*  
[http://w3.siemens.de/solutionprovider/\\_online\\_lexikon.html](http://w3.siemens.de/solutionprovider/_online_lexikon.html)
- [23] *Die Erfindung des Telefons*  
<http://www.sultan-zonk.de/history/telefon/vermittlung.html>
- [24] Prella, Stefan *Entwurf und Implementierung eines H.323-Gatekeepers zur Ressourcenverwaltung und Zugangsregelung für IP-Telefonie-Dienste*, Oktober 1999
- [25] *Leitfaden IP-Telefonie*  
<http://www.swyx.de/basics/leitfaden-inhalt.html>

- [26] Ludwig, Martin B. *IP-Telefonie im Reifeprozess*  
[http://www.captiva.ch/media/fachbeiträge\\_pdf/fb\\_PSL\\_VoIP.pdf](http://www.captiva.ch/media/fachbeiträge_pdf/fb_PSL_VoIP.pdf)