Fachhochschule
Bonn-Rhein-Sieg
University of Applied Sciences

Fachbereich Informatik
Department of Computer Science

Abschlussarbeit

im Bachelor Studiengang

Optimierte Umsetzung eines Telekommunikationsprotokolls aus dem Bereich Internet Telefonie auf Mikrocontrollern

von

Sebastian Blumenthal

Erstbetreuer: Prof. Dr.-Ing. Norbert Jung
Zweitbetreuer: Prof. Dr. Thomas Breuer
Eingereicht am: 10.09.2007

Vorwort und Danksagung

Die folgende Bachelorarbeit wurde bei der Gesellschaft für Netzwerk- und Automatisierungstechnologie mbH (N.A.T GmbH) durchgeführt.

Die N.A.T. GmbH beschäftigt sich seit ihrer Gründung 1990 mit der Entwicklung und Vermarktung von Netzwerktechnologie für industrielle Anwendungen. Die entwickelten Produkte stellen als solche Komplett-Lösungen dar, d. h. der Kunde kann für eine bestimmte Problemstellung in der Kommunikation sowohl die Hardware als auch die dazu benötigte Software aus einer Hand beziehen. Die Bandbreite der Lösungen reicht von einfachen Standardlösungen bis hin zu auf den jeweiligen Kunden zugeschnittene Individuallösungen, die Bandbreite der Technologie von einfachen Lösungen im LAN-Bereich bis hin zu komplexen Lösungen im WAN-Bereich.

An dieser Stelle möchte ich mich bei allen Mitarbeitern der N.A.T. GmbH für die freundliche Unterstützung während meiner Bachelorarbeit bedanken. Besonderer Dank gilt Herrn Helmut Laufkötter und Herrn Heiko Körte, die mir bei Fragen und Problemen stets hilfreich zur Seite standen.

Danken möchte ich auch Herrn Prof. Dr.-Ing. Norbert Jung für die Betreuung der Bachlorarbeit seitens der Fachhochschule Bonn-Rhein-Sieg, sowie Herrn Prof. Dr. Thomas Breuer als Zweitbetreuer.

Inhaltsverzeichnis

Abkürzungsverzeichnis

CCITT	Comité Consultatif International Téléphonique et Télégraphique
cPCI	Compact PCI
DSP	Digital Signal Processor
IETF	Internet Engineering Task Force
IP	Internet Protocol
ISDN	Integrated Services Digital Network
ISUP	ISDN User Part
ITU	International Telecommunication Union
IVR	Interactive Voice Response
MG	Media Gateway
MGC	Media Gateway Controller
MOS	Mean Opinion Score
PCI	Peripheral Component Interconnect
PCM	Pulse Code Modulation
PSTN	Public Switched Telephone Network
RFC	Request For Comments
RTCP	Real Time Control Protocol
RTP	Real Time Protocol
SDP	Session Description Protocol
SG	Signalling Gateway
SIP	Session Initiation Protocol
TCP	Transport Control Protocol
TDM	Time Devision Multiplexing
UAC	User Agent Client
UAS	User Agent Server
UDP	User Datagram Prtotocol
URL	Universal Ressource Locators
VoIP	Voice over IP

Tabellenverzeichnis

Abbildungsverzeichnis

1 Einleitung

1.1 Motivation

Neben der herkömmlichen, klassischen Telefonie, hat sich Voice over IP (VoIP) als weitere Sprachkommunikationstechnologie entwickelt, deren Bedeutung immer stärker wächst. Gerade im Hinblick auf die zunehmende Konvergenz der Telekommunikationsnetze besteht eine besondere Anforderung an die Protokolle darin, verschiedenartige Netze, insbesondere paketvermittelnde IP- und leitungsvermittelnde Telefonnetze, zu koppeln.

Um die Kommunikation mit VoIP zu ermöglichen, werden bestimmte Protokolle und Architekturen benötigt. Es gibt bereits einige Protokolle, die zwei Teilnehmer befähigen ein Telefonat mittels VoIP, auch über heterogene Kommunikationsnetze, zu führen. Dazu zählen unter anderem der H.323 Standard, ein proprietäres Protokoll von CISCO mit dem Namen „Skinny“, das Media Gateway Control Protocol (MGCP) und das aus MGCP entwickelte Nachfolgeprotokoll „MEGACO“.

Zwischen den unterschiedlichen Netzen residieren als Vermittlungsknoten spezielle *Gateways*. Um diese effizient zu implementieren, wird leistungsfähige, spezialisierte „embedded“ Hardware benötigt. Von den zuvor genannten Protokollen bietet sich MEGACO als am geeignetsten an, um als Kommunikationsprotokoll der Gateways zu dienen. Dieses Protokoll hat den Vorteil, das es weniger komplex, als zum Beispiel H.323, ist und es wird auf Ressourcenmanagement geachtet, welches die begrenzten Ressourcen einer embedded Plattform berücksichtigt.

Um der wachsenden Bedeutung von Voice over IP gerecht zu werden und Kunden der Gesellschaft für Netzwerk- und Automatisierungstechnologie mbH (N.A.T GmbH) zukünftig VoIP Produkte anbieten zu können, beschäftigt sich diese Bachlorarbeit mit einem Realisierungskonzept und der Implementation des MEGACO-Protokolls auf einer spezialisierten Telekommunikationshardwareplattform.

1.2 Einführung

Die Telefonie ist seit je her eine der wichtigsten technischen Kommunikationsmöglichkeiten. Die Entwicklung dieser Technologie, die etwa vor 150 Jahren begann, reicht von einfachen analogen Telefonen bis hin zu Telefonie über das Internet. [BADA2005, S.2]
Dieses Kapitel behandelt zunächst die Historische Entwicklung der Telefonie und führt anschließen in die Thematik von Voice over IP, sowie die damit verbundenen Problemstellungen, Vorteile und Realisierungsmöglichkeiten ein.

1.2.1 Historie der Telefonie

Eines der ersten kommerziell nutzbaren analogen Telefone wurde von dem US-Amerikaner Alexander Grahem Bell 1876 entwickelt. Bei der analogen Telefonie wird die Sprache, die in den Telefonapparaten mittels eines Mikrophons in elektrische Signale umgewandelt wird, über eine physikalische Leitung von einem Telefon zum einem anderen Telefon transportiert. Über einen Lautsprecher kann der Empfänger die Sprache vernehmen. Hat man mehrere Teilnehmer in einem Telefonnetz, muss bei einem Anruf zunächst eine exklusive Verbindung hergestellt werden, anschließend können Anrufer und angerufene Person sprechen. Nach Ende des Telefonats wird die Verbindung wieder frei gegeben. Dieses Prinzip, welches als leitungsvermittelndes Netzwerk oder *Public Switched Telephone Network* (PSTN) bezeichnet wird, ist bei analoger Technik recht einfach zu realisieren, und ist bei einer geringen Anzahl von Teilnehmern kostengünstig. Allerdings birgt analoge Technik auch einige Nachteile. Zum einen können analoge Signale durch Störeinflüsse beeinträchtigt werden, welche die Sprachqualität mindern. Solche Störungen nennt man auch *Rauschen*. Zum anderen müssen, um eine Verbindung zu ermöglichen, physikalische Leitungen aufgebaut werden. Hierzu benutzt man Relais als elektromechanische Schalter. Das Problem ist, dass Relais teuer und nur schwer zu warten sind, gerade bei den hohen Stückzahlen, in denen sie eingesetzt werden müssen, um große Telefonnetze zu realisieren. [GURL2002, S.6]

Um die Schwierigkeiten der analogen Telefonie zu beheben, wurde von dem Comité Consultatif International Téléphonique et Télégraphique (CCITT[1]) in den USA ein Standard, für ein digitales, als *Integrated Services Digital Network* (ISDN) bezeichnetes, Telefonnetz entwickelt. Ab den 1980er Jahren wurde ISDN eingeführt. In Deutschland würde zunächst von der Deutschen Telekom AG der proprietäre Standard 1TR6 benutzt, der einen Maßstab für ISDN-Leistungsmerkmale setzte. 1989 wurden die unterschiedlichen Standards

1 heutzutage: International Telecommunication Union (ITU)

der europäischen Länder in der Norm (E)-DSS1 miteinander harmonisiert. [KÖHL2002, S.36f] Das Sprachsignal wird nun nicht mehr in analoger Form übermittelt, sondern zunächst im Telefonapparat in ein digitales Signal umgewandelt, dann über einen 64kbit/s schnellen Datenkanal übertragen und anschließend im Empfangsgerät wieder in ein analoges Signal zurück gewandelt. Genau wie bei der analogen Technologie basiert ISDN auch auf dem PSTN Prinzip, also Verbindungsaufbau, Sprachübertragung und Verbindungsabbau. Eine Besonderheit von ISDN ist, dass die Kanäle zum Verbindungsaufbau und -Abbau getrennt von dem Kanal zur eigentlichen Audioübertragung sind. Letzteres wird von dem so genannten „*B-Kanal*“ übernommen während der „*D-Kana*l“ für die Verbindungskontrolle zuständig ist, die auch mit S*ignalisierung* bezeichnet wird. [BADA2005, S. 4ff] Die verschiedenen Kanäle befinden sich nicht auf verschiedenen physikalischen Leitungen, sondern werden zeitlich gemultiplext. Man spricht von *Time Division Multiplexing* (TDM). [GURL2002, S. 7]
Die Fähigkeiten von ISDN gehen über die normale Telefonie hinaus. Es werden auch „intelligente Dienste“ angeboten, die mit Hilfe von speziellen Servern an den Vermittlungsstellen des Telefonnetzes realisiert sind. Ein solcher Dienst kann zum Beispiel eine Televoting Funktionalität sein, oder ein Routing Dienst mit dem der Herkunftsort eines Anrufers ermittelt werden kann, sodass ein Anruf zu einer Firma, mit verschiedenen Zweigstellen, immer zur am nächstliegenden Stelle vermittelt wird. [BADA2005, S. 6f.]

Der nächste wichtige Schritt in der Entwicklung der Telekommunikationstechnologie, ist die Telefonie über das Internet. Bekannt als *Voice over IP* (VoIP) werden hierbei die Sprachdaten nicht über das herkömmliche Telefonnetz, sondern über ein IP basiertes Datennetzwerk transportiert. Handelt es sich bei diesem Netzwerk um das Internet, spricht man von *Internet-Telefonie.* Ein entsprechendes Telefon kann einerseits ein Telefon mit VoIP Technologie (*IP-Telefon)*, anderseits ein Computer mit einem Mikrofon, einem Lautsprecher und entsprechender VoIP Software sein. [BADA2005, S. 8]

1.2.2 Voice over IP und Probleme

Obwohl die Idee, Sprache einfach über ein Datennetzwerk zu versenden, sich zunächst simpel anhört, handelt es sich doch um eine recht komplexe Materie. Es ist die Natur eines IP-Netzwerks, dass Informationen in kleine Pakete aufgeteilt werden und zum Empfänger einzeln verschickt werden. Dabei ist es gut möglich, dass verschiedene Pakete auch verschiedene Routen benutzen. Daraus können zwei Situationen resultieren, die beide für eine Sprachübertragung ungünstig sind. Erstens können Datenpakete auf

schnelleren Wegen solche auf langsameren oder weiteren überholen. Es ist keinesfalls garantiert, dass die korrekte Reihenfolge eingehalten wird. Man kann sich leicht vorstellen, dass Sprache die in Pakete zerlegt, übertragen und in der falschen Reihenfolge wieder zusammengesetzt wird, vom Empfänger nur schwer zu verstehen ist. Das andere Problem ist, dass Sprache, die in Pakete zerlegt und zu äquidistanten Zeitpunkten übertragen wird, in den gleichen äquidistanten Zeitpunkten rekonstruiert werden muss, damit man sie verstehen kann. Die unterschiedlichen Übertragungszeiten der unterschiedlichen Routen widersprechen dieser Anforderung. Um diesen Effekten entgegen zu kommen gibt es ein spezielles Protokoll: *Real Time Protocol* (RTP). Im Kapitel 2.2 wird darauf genauer eingegangen. [GURL2002, S. 10ff.]

Dies sind nicht die einzigen Probleme, denn das IP-Netzwerk ist ein „Best-Effort[2]“ Netz, das bedeutet das IP-Pakete auf dem Weg zum Ziel verloren gehen können. Für den Datentransport benutzt man ein Protokoll, das einen zuverlässigen Bitstrom von einem Endpunkt zu einem anderen bereitstellen kann: *Transport Control Protocol* (TCP). Unter anderem werden verlorene Pakete erneut versendet. [TANE2003, S. 580ff.] Für den Sprachtransport kommt TCP nicht in Frage, weil das erneute Versenden der Sprachpakete zu lange dauert. Es besteht die Problematik der *Signallaufzeit (Delay)*. Mit Signallaufzeit ist die Zeit gemeint, die benötigt wird, bis die Information vom Sender bis zum Empfänger angelangt ist. Ist diese Zeit zu groß (oder treten zusätzliche Verzögerungen auf), ist ein Dialog über das Telefon sehr lästig, weil man immer kurz warten muss, bis man eine Antwort des anderen Telefonteilnehmers hören kann. [KÖHL2002, S. 101ff.]
Um Datenverlust vorzubeugen und die Signallaufzeit möglichst klein zu halten muss das Netzwerk richtig dimensioniert werden. Derartige Betrachtungen sind ein umfassendes Thema und sollen hier nicht weiter behandelt werden. Nähere Informationen findet man zum Beispiel in [GURL2002]: Kapitel 7 oder [BADA2005]: Kapitel 4.

Schließlich darf nicht vergessen werden, dass es im Internet zwar eindeutige Adressen für Computer, die IP-Adressen gibt, aber keine Telefonnummern im herkömmlichen Sinne. Gerade im Hinblick auf eine sinnvolle Nutzung von VoIP muss ein herkömmliches PSTN-Telefon in der Lage sein, ein IP-Telefon durch Eingabe einer klassischen Telefonnummer zu erreichen. Dieser Problemstellung widmet sich das Session Initiation Protocol (SIP), welches im Kapitel 2.3 ausführlicher behandelt wird. [BADA2005, S. 10]

2 IP-Pakete werden weitergeleitet, solange die Kapazitäten für die Übertragung ausreichen. Ist das Netzwerk überlastet, so werden Pakete verworfen. [TANE2003, S. 324]

Voice over IP wirft auch einige Fragen bezüglich der Sicherheit im Internet auf. Um zum Beispiel die Vertraulichkeit der Informationen zu wahren, ist Verschlüsselung auf der IP-Schicht denkbar. [BLAC2000, S. 229ff.] Dieses Thema wird jedoch in dieser Bachelorarbeit nicht weiter vertieft.

1.2.3 Nutzen von Voice over IP

Bei all den Schwierigkeiten, die Voice over IP bei der Telefonie hat, stellt sich die Frage, warum man eigentlich diese Technologie nutzen sollte? Dafür gibt es einige gute Gründe. Zunächst einmal ist die Internet-Telefonie günstig. Laut [BLAC2000] ist VoIP um etwa ein viertel bis drittel günstiger (konservative Einschätzung) als die klassische PSTN Telefonie. Ein zweiter wesentlicher Aspekt ist die Verschmelzung von den historisch unterschiedlich gewachsenen leitungsvermittelnden PSTN- und paketorientierten IP-Netzen. Ziel ist es, ein einheitliches „Next Generation Network“ aufzubauen, das alle möglichen Arten von Diensten anbieten kann: Sprache, Daten, Fernsehen, etc. [BADA2005, S. 1] Die Wahl als generisches Netz fällt dabei auf das IP-Netz, nicht unbedingt weil IP die perfekte Technologie hierfür ist (IP-Netze sind für den Datentransport konzipiert worden), sondern weil es schlichtweg vorhanden und gut verfügbar ist. [BLAC2000, S. 7]

Der Nutzen von VoIP ist nicht nur auf die reine Telefonie beschränkt, sondern es sind auch weitere interessante Anwendungen denkbar. Zum einen lässt sich VoIP gut in eine Call-Center Anwendung integrieren, denn neben den eigentlichen Gesprächsdaten müssen hier auch andere Informationen transportiert werden (Anrufverteilung auf Fachpersonal, Zuordnung der Kundendaten eines Anrufers, etc). Zum anderen könnte man auch eine Voice Mail-Box realisieren, bei der erfolglose Anrufe ähnlich wie E-Mails abgespeichert werden, die dann später einzeln abgehört werden können. [KÖHL2002, S. 19ff.]

1.2.4 Realisierungen von Voice over IP

Damit man VoIP effektiv nutzen kann, muss man auch die klassische PSTN Telefonie einbeziehen, die zur Zeit vorherrscht. Es werden spezielle Übergangspunkte, *Gateways* genannt, zwischen den beiden historisch gewachsenen Technologien benötigt. Für die Art und Weise, wie diese Gateways zwischen den unterschiedlichen Netzen vermitteln oder wie sie untereinander kommunizieren, gibt es eine Reihe von Protokollen: Der H.323 Standard, das von CISCO entwickelte Protokoll „Skinny“, das Media Gateway Control Protocol (MGCP), und das MEGACO-Protokoll, welches aus MGCP hervorging.

Während der H.323 Standard mit einem großen Funktionalitätsumfang aufwarten kann und daher auch komplex ist, wurde beim Entwurf des MEGACO-Protokolls auch auf Ressourcenmanagement geachtet. Das macht es besonders interessant für den Einsatz auf dedizierter „embedded“ Telekommunikationshardware. [GURL2002, S. 5]

MEGACO folgt dem Ansatz einer Client-Server Struktur, wobei eine zentrale Instanz, der *Media Gateway Controller* (MGC), eine Anzahl von *Media Gateways* (MG) verwaltet. Der Media Gateway Controller ist in der Lage einen Übergang für die unterschiedlichen Signalisierungsinformationen der Telefonnetze zu schaffen, währen ein Media Gateway die verschiedenen Nutzdaten in einander überführt.

Um diese Architektur effizient zu implementieren wird leistungsfähige, spezialisierte Hardware benötigt. Daher beschäftigt sich diese Bachlorarbeit mit der Implementierung des MEGACO-Protokolls auf einer speziellen für den Telekommunikationssektor ausgelegten Hardwareplattform aus der Produktpalette der N.A.T. GmbH.

1.3 Abgrenzung der Arbeit und Stand der Technik

Die Aufgabenstellung und Abgrenzung dieser Arbeit sind in den Kapiteln 3.1 sowie 3.2 (und 4.2) beschrieben. Zum Verständnis der Aufgabenstellung wird das Wissen aus dem Grundlagenkapitel benötigt, daher wird hier nicht näher darauf eingegangen.

Die bereits verfügbaren Technologien, auf denen aufgebaut werden kann, werden ebenfalls erst in Kapitel 3.4.5 vorgestellt.

2 Grundlagen

Das Kapitel enthält die Grundlagen, die zum Verständnis des Konzepts und der Implementation wichtig sind. Zunächst werden die Eigenschaften und Probleme von Voice over IP dargestellt und die daraus resultierenden Anforderungen für eingebettete Systeme. Weil das MEGACO-Protokoll unter anderem RTP und SIP benötigt, werden diese vorgestellt, bevor auf MEGACO genauer eingegangen wird.

2.1 Eingebettete Systeme und VoIP

Ziel von Voice over IP ist es, Sprache über ein Datennetzwerk zu transportieren. Daraus ergeben sich eine Reihe von Anforderungen an die Systeme und Protokolle. Vor allem spielen Signallaufzeit (Delay), Bandbreite und der Rechenaufwand eine wichtige Rolle.

Bei der Signallaufzeit darf das **2-Wege-Delay**[3] ein gewisse Maximalzeit nicht überschreiten, andernfalls ist ein flüssiger Dialog zwischen zwei Gesprächsteilnehmern nicht möglich. Durch die Verzögerung bekommt der Empfänger die Worte des Senders später zu hören und der Sender muss ebenfalls die Dauer der Signallaufzeit auf die Antwort warten. Werden diese Wartezeiten zu lange, kann das als sehr störend für den Gesprächsverlauf empfunden werden. Signallaufzeiten werden einerseits durch die endliche Ausbreitungsgeschwindigkeit in den Übertragungsmedien und andererseits durch die Verzögerungen in Netzwerkknoten bestimmt. Jeder einzelne Knoten erhöht die Verzögerung, vor allem weil Netzwerkpakete zwischengepuffert werden. [BLAC2000] gibt (auf Seite 26) die maximale Zeit für das 2-Wege-Delay mit 300ms an.

Für die Sprachkommunikation über ein Datennetzwerk werden kleine **Puffer** benötigt, damit die Verzögerung durch die Netzwerknoten gering bleibt. Kleine Puffer bergen das Risiko, das sie überlaufen, wenn zu viele Pakete gespeichert werden sollen. Das führt zu verworfenen Paketen.

Im Gegensatz zur Datentransmission sind einzelne verlorene Pakete für Voice-Applikationen unkritisch. Allgemein zeigen sich Sprachübertragungen fehlertoleranter als Datenübertagungen, weil einzelne verfälschte Bits kaum wahrnehmbare Änderungen in der Wiedergabe des Sprachsignals bewirken. Sogar einzelne verlorene Paket können verkraftet werden, solange der Paketverlust weniger als 10 Prozent beträgt. [BLAC2000, S. 24ff]
Der für die Kommunikation verwendete „Codec“ hat Einfluss auf die Anforderungen an

3 Zeit die benötigt wird um ein Paket zu einem Ziel und wieder zurück zur Quelle zu senden.

VoIP Systeme. Unter dem Begriff Codec oder auch Sprachkodierung versteht man eine bestimmte Art der Kodierung der Sprachdaten. Dabei werden die Samples, die bei der Quantisierung[4] der analogen Signale generiert werden, möglichst kompakt in Datenworten gespeichert. Es gibt verschiedene Algorithmen zur Kodierung, aber alle haben gemeinsam, dass je höher der Komprimierungsgrad ist, desto stärker sinkt die Übertragungsqualität.

Tabelle 1 soll den Zusammenhang zwischen Sprachqualität und benötigter Bandbreite verdeutlichen. Die Sprachqualität wird mit dem *Mean Opinion Score* (MOS) gemessen. Dieser wird empirisch von repräsentativen Testpersonen aus dem Höreindruck ermittelt. Die Skala reicht von 1: schlecht (trotz Anstrengung keine Verständigung), bis 5: exzellent (keinerlei Anstrengung zum Verständnis der Sprache notwendig; totale Entspannung möglich). Zum Vergleich erreicht analoge Telefonie einen Wert von 3,4 bis 4,0. ISDN erzielt (unter Verwendung des typischen ISDN Codecs), mit 4,5 einen besseren Wert. [BADA2005, S. 147ff.] Des weiteren steigt durch einen „effizienten“ Kodieralgorithmus auch die Komplexität und damit die benötigte Rechenleistung.

Verfahren	***Bitrate***	***MOS-Wert***	***Komplexität***
PCM (G.711)	64	4,3 - 4,5	1
ADPCM (G.726)	16/24/32/40	3,4/3,6/3,9/4,2	10
CS-ACELP (G.729)	8/6,4	4,0/3,8	15-30
LD-CELP (G.728)	16	4,0 - 4,1	50
ACELP (G.723.1)	5,3	3,5	25
MP-MLQ (G.723.1)	6,3	3,7	keine Angabe

Tabelle 1: MOS-Werte der Sprachkodierung[5][BLAC2000, S.74]

Ein in der Telekommunikation häufig verwendeter Codec ist der Codec G.711. Er wird unter anderem bei ISDN als Sprachkodierung eingesetzt. Bei G.711 kommt das Pulse Coding Modulation (PCM) Verfahren zum Einsatz, das heißt das analoge Signal wird 8000 mal in der Sekunde abgetastet und nach einem speziellen, nichtlinearen Prinzip quanti-

4 Ein analoges Signal wird zu äquidistanten Zeitpunkten abgetastet. Der zu dem Zeitpunkt erfasste Amplitudenwert des Signals wird als ein bestimmtes Datenwort abgespeichert.

5 PCM: Pulse Code Modulation, ADPCM: Adaptive Differential Pulse Code Modulation, CS-ACELP: Conjugate Structure Algebraic Code Excited Predictive Linear Coding, LD-CELP: Low Delay Code Excited Predictive Linear Coding: , ACELP: Algebraic Code Excited Predictive Linear Coding, MP-MLQ: Multipulse Maximum Likehood Quantization
Weitere Informationen zu den verschiedenen Codecalgorithmen findet der Leser in [BADA2005, S.145ff.] oder [BLAC2000]: Kapitel 4.

siert. Die Empfindlichkeit des menschlichen Gehörs nimmt logarithmisch mit der Lautstärke ab. Daher benutzt man eine logarithmische Aufteilung der Quantisierungsintervalle. Jeder Abtastwert wird als 8-Bit Datenwort gespeichert. Daraus ergibt sich eine Bitrate von 8000*8 Bit, also 64kbit/s[6]. [BADA2005, S.138f./S. 146]
Der G.711 Codec unterteilt sich in zwei leicht unterschiedliche Versionen, dem in Europa verwendeten G.711 „µ-Law“ und dem in Amerika und Japan eingesetzten „A-Law“. Trotz der geringen Unterschiede sind sie inkompatibel zu einander. Damit Sprachverbindungen zwischen Amerika und Europa möglich sind, sind in dem Standard auch die Umkodierungsregeln festgehalten. [BADA2005, S.140f.]

Bei VoIP Anwendungen müssen Aspekte des Netzwerks berücksichtigt werden. Auch die Art des verwendeten Codecs fließt in diese Betrachtungen mit ein.
Das zugrunde liegenden IP-Netzwerk muss genügend **Bandbreite** zur Verfügung stellen, um die anfallenden Daten für die Sprachkommunikation hinreichend schnell zu transportieren. Dabei muss beachtet werden, dass nicht nur die reinen Sprachdaten, sondern auch die Header der Schicht 2, IP-Header, UDP-Header, Header auf der Anwendungsschicht und gegebenenfalls Header für die Sprachkodierung der Codecs übertragen werden müssen.

Innerhalb der Netzwerkknoten muss der **Rechenaufwand** betrachtet werden. Dazu tragen die Kodierung, die Übertragung und Dekodierung bei. Gerade bei der Umformung von einem Sprachcodec in einen anderen, wird unter Umständen viel Rechenleistung benötigt. Typischerweise werden diese Rechenleistungen in *Millionen Instruktionen pro Sekunde* (MIPS), als grobe Richtwerte gemessen. Der Netzwerkknoten, insbesondere das embedded System welches ein Bestandteil des Knotens ist, muss die von ihm verlangte Mindestleistung (z. B. eine Mindestleistung an MIPS) aufweisen können. [BLAC2000 , S. 24ff]

Um die Mindestleistung bereit zu stellen, bietet sich der Einsatz von Digital Signal Processors (DSPs) an. DSPs sind auf die Verarbeitung von Signalen spezialisiert. Paketierte Sprache benötigt eine beachtliche Menge an sogenannten „Multiplier-Accumulator“ (MAC) Funktionen, um zum Beispiel das Frequenzspektrum eines Signals zu ermitteln, und das in einer sehr kurzen Zeit. DSPs sind konzipiert, um genau diese Anforderungen zu erfüllen. Sie übernehmen dann Aufgaben, wie zum Beispiel die Paketierung, Komprimierung,

6 Das Einheitenpräfix „k“ ist hier als Binärpräfix gemeint, also 1k = 1024

„Voive Activation Detection[7]“ oder „Echo Cancelation[8]“. Die Anzahl der gleichzeitig bearbeiteten Sprachkanäle, in Bezug auf die Codecoperationen, hängt von mehreren Faktoren ab. Im wesentlichen sind sie von der benötigten Rechenleistung (MIPS) aber auch der Größe des Speichers abhängig. Allgemein betrachtet könnte man DSPs auch als „Motoren“ der Sprach(um)kodierer und Modems bezeichnen.

In der Regel arbeitet der DSP als Ergänzung zu einem normalen Prozessor. Dieser normale, nicht spezialisierte, Prozessor ist für Kommunikation mit den Netzwerken und die Kontrolle des Systems verantwortlich.

Neben der hohen Geschwindigkeit für bestimmte Aufgabenbereiche sind im embedded Bereich eingesetzte DSPs meistens klein und stromsparend. Dies macht sie sehr attraktiv für die Internet Telefonie. [BLAC2000, S. 52ff.]

2.2 RTP als Sprachübermittlungsprotokoll

Da die Sprache vom Sender bis zum Empfänger in einer bestimmten maximalen Antwortzeit (< 300ms) übertragen werden muss, kann man das als Echtzeitanforderung auffassen. Ein IP-Netzwerk, respektive das Internet ist aber für Datenkommunikation konzipiert und die Protokolle beinhalten keine Unterstützung für Echtzeitkommunikation. Daher wurde ein spezielles Echtzeitprotokoll, das sogenannte *Realtime Transport Protoco*l (RTP), entwickelt. Es ist auf die Übermittlung von Sprache, Audio und Video über IP-Netzwerke spezialisiert. Die aktuellste Version (von 2003) dieses Protokolls, welches ursprünglich 1996 von der Internet Engineering Task Force (IETF) entwickelt wurde, ist im RFC 3550 zu finden. In diesem Standard ist auch das Schwesterprotokoll *Real Time Control Protocol* (RTCP) spezifiziert. RTCP ist für den Austausch von Kontroll- und Statusinformationen zwischen den Endpunkten zuständig. [BADA2005, S.148f]

RTP ist unabhängig von den darunterliegenden Transportschichten, aber in Verbindung mit IP-Netzen, setzt RTP auf dem verbindungslosen UDP auf. Abbildung 1 zeigt die Einordnung in den IP-Stack. [NÖLL2003, S. 97]

7 Nur wenn wirklich gesprochene Worte sich im Sprachkanal befinden, sollen diese als Pakete übertragen werden. Gesprächspausen sollen erkannt und nicht übertragen werden. In der Regel werden die „fehlenden“ Pakete beim Empfänger durch ein synthetisches Rauschen simuliert, um „knacksende“ Geräusche zu vermeiden.

8 Echo: Nebeneffekt der wegen einer Rückkopplung, also einer Rückspeisung der Sprache von dem Lautsprecher in das Mikrofon eines Telefons, entsteht und die Qualität mindert.

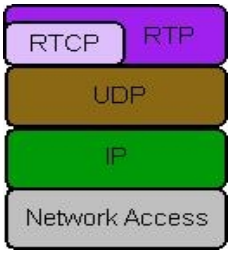

Abbildung 1: RTP und RTCP im IP-Stack

2.2.1 RTP

Ziel von RTP ist es, ein Echtzeitmedium[9] zwischen zwei oder mehr Teilnehmern zu transportieren. Sind die Parameter, wie zum Beispiel der verwendete Codec festgelegt, so besteht mit dem Austausch von RTP-Paketen ein logischer Übertragungskanal, der als *RTP-Session* bezeichnet wird. Ein weiterer Bestandteil der Session ist ein Kanal für die RTCP-Nachrichten.

Bevor ein Telefongespräch über ein IP-Netzwerk zustande kommen kann, muss eine Verbindung aufgebaut beziehungsweise nach dem Gespräch wieder abgebaut werden können. Diesen Verbindungsauf- und Abbau bezeichnet man als *Signalisierung*. Diesen Regeln folgt auch der Kommunikationsaufbau für RTP. Das bedeutet, die RTP-Session muss auf- und nach Beendigung wieder abgebaut werden. Das RTP-Protokoll ist allerdings ein reines Transportprotokoll und es sind keine Mechanismen für die *Signalisierung* vorgesehen. Darum müssen zusätzliche Signalisierungsprotokolle eingesetzt werden. Die gängigsten Signalisierungsprotokolle sind H.225 und H.245 aus der H.323 Protokollfamilie und das Session Initiation Protocol (SIP). Auf SIP wird im Kapitel 2.3 genauer eingegangen.

Eine Session benötigt immer einige Parameter der teilnehmenden Endpunkte, gewissermaßen eine Beschreibung der Session. Die Syntax, wie eine Session beschrieben wird, ist im *Session Description Protocol* (SDP) festgelegt (RFC 2327). H.323 verwendet zur Beschreibung eigene Regeln, währen SIP das Session Description Protocol benutzt.

Das Realtime Transport Protocol benötigt bestimmte Eigenschaften, um einen Multimediadatenstrom mit Echtzeitanforderung über ein IP-Netzwerk zu gewährleisten. Eine Ei-

9 Der Begriff Medium soll im Folgenden, wenn nicht anders vermerkt, als Überbegriff für Multimediainhalte verwendet werden (z. B. Video oder Sprache).

genschaften von RTP ist die **Übermittlung von Echtzeitmedien** in RTP-Paketen. Das Echtzeitmedium wird in einzelnen Paketen, die eine zusammenhängende Folge bilden, übertragen. Dies stellt den logischen Übertragungskanal dar.

Eine weitere Eigenschaft ist die Garantie der **Reihenfolge von RTP-Paketen**. Die einzelnen Pakete werden mit Sequenznummern versehen. Werden während der Übertragung Pakete vertauscht, so können sie beim Empfänger mittels der Nummerierung wieder in die korrekte Reihenfolge gebracht werden.

Ebenfalls muss die **Garantie der Isochronität** gegeben sein. Es werden Zeitstempel vergeben, sodass die gleichen Zeitabstände beim Empfänger wieder hergestellt werden können so wie sie beim Versendern der Pakete beim Absender waren. Ist die Isochronität nicht gegeben, hat ein Gesprächsteilnehmer bei einer Audiokommunikation, den Eindruck, der Gesprächspartner würde mal langsamer, mal schneller sprechen. Eine Abweichung von der Gleichartigkeit der Zeitabstände wir als „Jitter" bezeichnet.

Das Echtzeitprotokoll muss in der Lage sein, **unterschiedlicher Formate** zu transportieren. Mit Hilfe so genannter "Profiles" (RFC 3551), können verschiede Arten von Medien, die übertragen werden können, beschrieben werden. Die möglichen Formate erstrecken sich von VoIP, über Videokonferenzen bis hin zu Multimedia.

RTP sieht die Unterstützung einer **Translator**- und Mixer-**Funktion** vor. Translator heißt hier, dass ein Kodierungsformat in ein anderes überführt werden kann. Gerade im Hinblick auf die sich ähnelnden, aber inkompatiblen G.711 A-Law und µ-Law Codecs ist diese Fähigkeit wichtig. Mit der Mixer-Funktionalität ist das Zusammenmischen eines Datenstroms aus mehreren Datenquellen gemeint.

Die einzelnen RTP-Pakete bestehen aus einem Header und der Nutzlast (Payload). Im Header (vgl. Abbildung 2) lassen sich verschiedene Parameter finden, um die zuvor genannten Eigenschaften zu realisieren:

- Payload Type: Identifier für den Typ des Echtzeitmediums
- Timestamp: In dem Zeitstempel wird die Uhrzeit eingetragen. Beginnt mit einem zufälligen Anfangswert. In folgenden Paketen muss der Wert um die tatsächliche Zeitdifferenz zum vorherigen Stempel erhöht werden. Ziel ist die Kompensation der Jitter.
- Sequence Number: 16Bit-Nummer, um Verluste zu identifizieren oder Vertauschungen

zu korrigieren.

- SSRC (Synchronisation Source Identifier)

  Pakete sind nach Quellen gruppiert, die das gleiche Prinzip der Zeitstempelvergabe (Timing) haben. Quellen können Mikrophone, Mixer oder Kameras sein. Sequenznummern gelten nur für Quellen mit der gleichen SSRC
- CSRC (Contributing Source Identifier)

  CSRC ist optional, falls die Nutzlast nicht von der ursprünglichen Quelle stammt (z. B. von einem Mixer als Zwischenstation) [BADA, S. 151ff.]

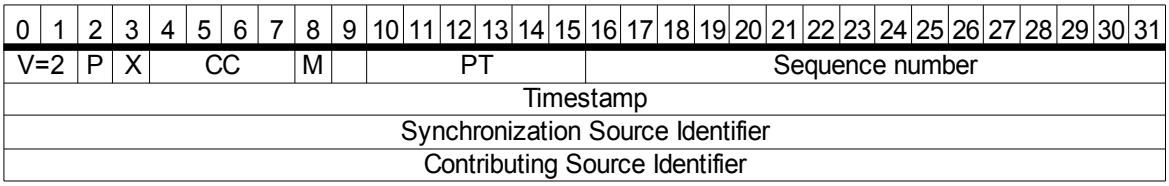

Abbildung 2: RTP-Header [RFC3550, S. 12]

2.2.2 RTCP

RTCP wird benutzt, um Informationen über die Verbindung der RTP-Session, insbesondere der Verbindungsqualität, zu erhalten. Hierfür werden regelmäßig so genannte Reports als Metadaten zwischen den Endpunkten versendet. Die Hauptfunktionen sind:

- Überwachung der Übertragungsqualität

  RTCP überwacht die Qualität des Netzwerkes. Eine Applikation die RTP/RTCP benutzt könnte sich an die Netzbedingungen anpassen (zum Beispiel die Bitrate ändern). Dazu werden *Sender* und *Receiver Reports* ständig ausgetauscht.
- Identifikation der Quelle

  Die eindeutige Identifikation Quelle bleibt auch erhalten, wenn ein Mixer eingesetzt wird (dieser würde die SSRC ändern)
- Unterstützung der Mehrpunkt-Kommunikation

  Der Austausch der Kontrollinformationen ermöglicht die Erkennung verschiedener Teilnehmer, die beispielsweise zu einer Konferenzsitzung gehören. Neue und ausscheidende Teilnehmer können dynamisch hinzugefügt oder entfernt werden. [BADA2005, S. 159]

2.3 SIP als Signalisierungsprotokoll

Um Verbindungsauf- und Abbau für Echtzeitkommunikation über ein IP-Netz, zum Beispiel für RTP zu ermöglichen, werden Signalisierungsprotokolle benötigt. Es haben sich zwei Protokolle zur Signalisierung durchgesetzt, einmal der im H.323 enthaltene Signalisierungsmechnismus, der sehr komplex ist und das auf Textnachrichten basierende, dem HTML-Protokoll ähnliche, *Session Initiation Protocol* (SIP). Dieses von IETF entwickelte Protokoll ist in seiner aktuellsten Form im RFC 3261 zu finden. [NÖLL2003, S. 70f.]

SIP kann sowohl auf TCP als auch auf dem verbindungslosen UDP aufsetzen (vgl. Abbildung 3). Mit UDP verläuft der Aufbau einer Session zwischen zwei Endpunkten schneller. Da Nachrichten innerhalb des SIP mit einer Bestätigung beantwortet werden, kann dies auch als Sicherungsmechanismus genutzt werden (der im UDP nicht vorhanden ist). [BADA2005, S. 248] Um die Eigenschaften des zu übertragenen Mediums zu übermitteln, benutzt SIP das Session Description Protocol (SDP).

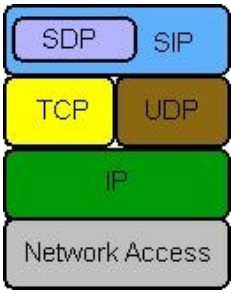

Abbildung 3: SIP im IP-Stack

Zusätzlich zu den Protokollen auf den verschiedenen Schichten, die die Kommunikation zwischen den einzelne Elementen ermöglichen, müssen auch die Komponenten, die während der Signalisierung adressiert werden können, selbst näher beschrieben werden.

2.3.1 Komponenten

Die wesentlichen Komponenten von SIP, die während der Signalisierung involviert sind, umfassen User Agent Clients (UAC), User Agent Server, Proxy-Server, Registrar-Server und Redirect-Server.

User Agents sind die Endgeräte einer Kommunikationssession. Ziel ist das Herstellen einer Kommunikation zwischen ihnen. Dazu verfolgt das Session Initiation Protocol einen Client-Server Ansatz, wobei der **User Agent Client** eine Instanz ist, die etwas bei dem

User Agent Server anfordert. Dieser kann dann die Anfrage bearbeiten, weiterleiten oder ablehnen. Die User Agents können sowohl einen User Agent Client (z. B. zum Initiieren eines Anrufs), als auch einen User Agent Server (z. B. um einen Anruf entgegen zunehmen) beinhalten.

Der **Proxy Server** leitet die SIP-Nachrichten zu anderen User Agent Servern weiter. Man kann ihn als eine Art Zwischenstation betrachten, die Nachrichten weiterreicht.

Mit Hilfe von „REGISTER“-Nachrichten kann sich ein Teilnehmer bei einem **Registrar** anmelden und mitteilen, an welchem Standort er verfügbar ist. Durch diesen Mechanismus erhält man die Möglichkeit, eine vom Standort unabhängige Adressierung zu realisieren. Ein „Location Server“ hält diese Informationen vor, um bei Bedarf abgerufen zu werden.

Üblicherweise kommt die Signalisierung zwischen User Agents über Proxy-Server zustande. Anstatt zum Beispiel eine Rufaufbau-Nachricht, transparent über mehrere Proxy Server zu leiten, kann hier der Client auch direkt über eine ermittelte IP-Adresse zum Ziel Kontakt aufnehmen. Um die IP-Adresse zu ermitteln, wird ein **Redirect-Server** eingesetzt. Er hat die Aufgabe einen User Agent Client über die IP-Adresse des gewünschten zu anrufenden Teilnehmers zu informieren. [NÖLL2003, S. 73]

In der Rufsignalisierung legt SIP das genaue Zusammenspiel der Komponenten fest, beziehungsweise wie die Nachrichten untereinander ausgetauscht werden.

2.3.2 Rufsignalisierung

Abbildung 4 stellt einen vereinfachten Rufaufbau, Kommunikation über RTP und anschließendem Verbindungsabbau dar. Die Rufsignallisierung beginnt mit einer *INVITE* Message (vgl. Abbildung 5), die einen anderen Anrufer auffordert, an einer Session teilzunehmen. In der Nachricht sind außerdem die Parameter für die RTP-Session hinterlegt (gemäß des Protokolls SDP). Falls das angerufene Endgerät die Parameter unterstützt, antwortete es mit einem *Ringing* SIP-Response, um das Klingeln einzuleiten. Sobald abgehoben wird, empfängt der Anrufer eine *OK* Nachricht, die mit einem ACK (Acknowledgement) quittiert wird. Danach ist der Anrufaufbau abgeschlossen. Es folgt das Gespräch mittels RTP (vgl. Kapitel 2.2). Um die Verbindung nach der Kommunikation wieder zu beenden, sendet eine Seite die *BYE* Nachricht. Nach der Antwort *OK* beenden beide Teilnehmer die RTP-Session. [BADA2005, S. 250f.]

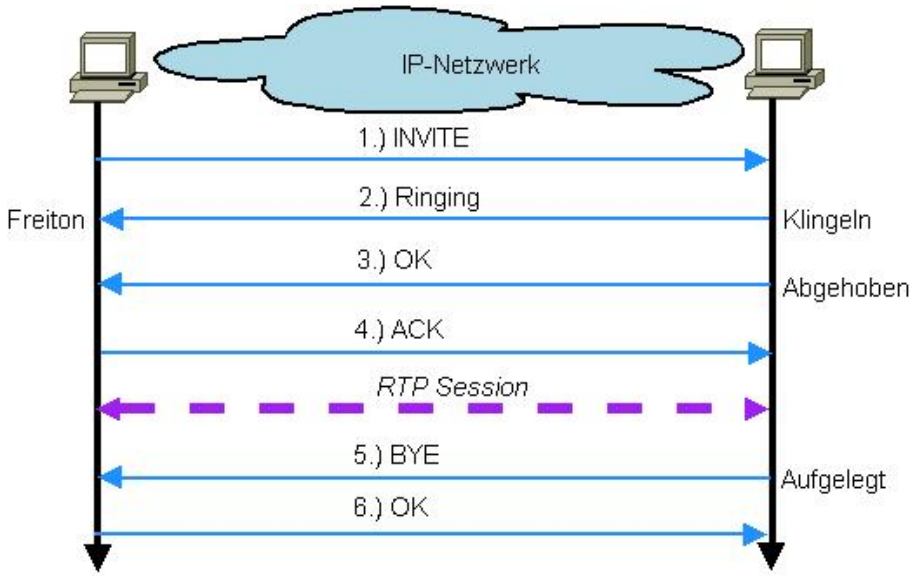

Abbildung 4: einfache Rufsignalisierung mit SIP [BADA2005, S. 250]

Damit auch Teilnehmer in nicht SIP-basierten Netzwerken für oder von SIP-Telefonen aus erreichbar sind, kann man (z. B. über einen Gateway) die Rufsignalisierung in die entsprechende andere Signalisierung umwandeln. Tatsächlich weisen SIP und ISDN einen sehr ähnlichen Verlauf der Signalisierung auf. Die Umwandlung von SIP-Nachrichten in ISDN-Signalisierungsnachrichten oder ISUP[10]-Nachrichten für ein SS7-Netz (die identisch zu den „normalen“ ISDN Nachrichten sind), ist im RFC 3398 spezifiziert. Aber auch die Interaktion von SIP und H.323 ist möglich. [BADA2005, S. 280f.]

```
INVITE sip:john@192.190.132.31 SIP/2.0
Via: SIP/2.0/UDP 192.190.132.20:3456
Call-ID: a2e3a@192.190.132.20
From: sip: mary@192.190.132.20
To: sip: sip: john@192.190.132.31
Cseq 1 INVITE
Content-type: application/sdp
Content-Length: 98

V=0
o=mary 3123 121231 IP IP4 192.190.132.20
c=IN IP4 192.190.132.20
m=audio 49170 RTP/AVP 0
```

SIP-Header

Body: SDP

Abbildung 5: SIP INVITE Nachricht [GURL2002, S. 126]

10 ISDN User Part

2.3.3 Weitere Eigenschaften

Ebenso wie die klassischen Telefone brauchen die Endgeräte in einem SIP-Netzwerk eine eindeutige Adressierung. SIP benutzt hierfür URLs (Universal Ressource Locators), die der E-Mail-Syntax sehr ähnlich sind. Die Adressen sind von der Form user@host. „User“ könnte ein Vor- oder Nachname, „Host“ eine numerische IP-Adresse oder ein Domainname sein. [DETK2002, S. 326]

SIP verfügt auch über Mechanismen für Anrufweiterleitung oder -verzweigung. Außerhalb der SIP-Spezifikation sind Konferenzen, Multicastadressen und Netzwerk-Ressourcenreservierung. SIP ist als Hilfsmittel für andere Dienste gedacht, dadurch ist es sehr flexibel. [DETK2002, S. 325 /NÖLL2003, S. 72]

SIP ist in der Lage die Signalisierung in einem IP-Netz zu realisieren. Möchte man allerdings verschiedene Signalisierungsprotokolle und/oder verschiedenartige Netze miteinander verbinden, braucht man einen erweiterten Ansatz. MEGACO ist eine Möglichkeit für solch einen Ansatz.

2.4 MEGACO

MEGACO (**Me**dia **Ga**teway **Co**ntrol) ist ein Protokoll, dass sich mit der Kommunikation zwischen so genannten *Gateways* befasst. SIP ist ein Protokoll für eine Endpunkt-zu-Endpunkt Signalisierung. Gatewayprotokolle verfolgen ein anderes Ziel: Hier soll die Intelligenz eines Netzwerkes an einem zentralen Punkt konzentriert werden. An den Rändern des Netzes befinden sich Instanzen, die von der zentralen Instanz gesteuert werden.

2.4.1 Allgemeines

Die ersten Protokolle, die sich mit Multimediagateways beschäftigten, IP Device Control (IPDC) und Simple Gateway Control Protocol (SGCP) wurden von der Industrie 1998 entwickelt. Aus den beiden Standards erarbeitete die IETF das Media Gateway Control Protocol (MGCP). Im Jahr 2000 entstand, zusammen mit dem Media Device Control Protocol (MDCP), dann daraus der Nachfolger MEGACO. Dieses Protokoll wurde nicht nur von IETF, sondern auch von der ITU-T unterstützt. Spezifiziert wurde es von der IETF als RFC 3015 und in dem wortgleichen ITU-T Standard H.248. Mittlerweile ist das RFC 3015 veraltet und wurde durch das RFC 3525 (2003) abgelöst. Hier wurden Klarstellungen und Fehlerkorrekturen eingearbeitet. [NÖLL2003, S. 112f.]

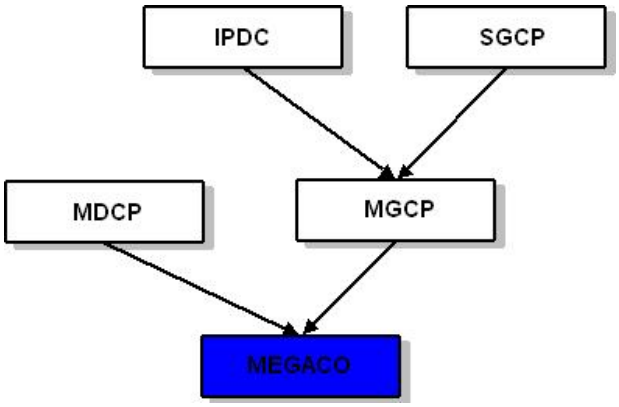

Abbildung 6: Evolution der Gatewayprotokolle [NÖLL2003, S. 113]

MEGACO steht von der Funktionalität her in Konkurrenz zum H.323 Standard. Ein wichtiger Unterschied besteht in der Komplexität. H.323 ist ein umfassender Multimediastandard in dem alles möglichst vollständig beschrieben ist. Das MEGACO-Protokoll arbeitet unabhängig von Signalisierungsprotokollen, es können daher Signalisierungsinstanzen von anderen Protokollen (z. B. SIP oder der die Signalisierungskomponente des H.323 Standards) implementiert werden. Damit bleibt es flexibel. [NÖLL2003, S. 76/S. 114]
Ein weiter Vorteil von MEGACO ist der bedachte Umgang mit den zur Verfügung stehenden Ressourcen. Dies wird unter anderem gut in der Spezifikation sichtbar: es gibt Parameter in den Nachrichten (die sich *ReserveGroup* und *ReserveValue* nennen), mit denen das Verhalten des Ressourcen-Managements gesteuert wird. Zum Beispiel kann ein Media Gateway (eine Instanz im MEGACO-Protokoll, die für das Transportieren von Echtzeitmedien zuständig ist) angewiesen werden, für alle Kommunikationsarten Ressourcen zu reservieren oder nur für eine bestimmte Art der Kommunikation. Auch in Fehlerfällen oder bei Nichtnutzung werden Ressourcen sofort freigegeben. [RFC3525, S. 31ff.] Gerade für embedded Hardwareplattformen, denen nur begrenzte Ressourcen zur Verfügung stehen, auf denen MEGACO implementiert werden soll, ist das wichtig.

2.4.2 Architektur von MEGACO

MEGACO basiert auf einer Client-Server Architektur (vgl. Abbildung 7). Die zentralen Serverinstanzen werden als ***Media Gateway Controller*** (MGC) bezeichnet. Ihre Aufgaben umfassen die Kontrolle und Steuerung der Clientelemente, den so genannten *Media Gateways* (MGs), und die Rufsignalisierung. Bezüglich der Signalisierung sind MGCs multiprotokollfähig, also je nach Implementierung wird SIP, H.323 oder ein anderes Protokoll unterstützt. Die Kommunikation zwischen mehreren MGCs erfolgt dann mit genau diesen

Protokollen. Unterstützt ein MGC mehrere Arten der Signalisierung, können verschiedene Typen von Kommunikationsnetzen zusammengeführt werden, zum Beispiel IP basierte Netze und PSTN Netze.

Die **Media Gateways** sind für den Transport der Echtzeitmedien, wie zum Beispiel Sprache, zuständig. Zwischen MGs wird das Medium über einen RTP-Kanal übertragen. Wenn ein MG an ein nicht auf IP basierendes Netz angeschlossen ist, so muss der Medienstrom in das entsprechende Format des anderen Netzwerks konvertiert werden. Ein MGC kommuniziert mit „seinen“ MGs über das MEGACO-Protokoll. Weiterhin ist es möglich, analoge Telefone direkt an einen MG anzuschließen. Die MGs unterstützen die entsprechenden Wahltöne, die für die analoge Telefonie benötigt werden. [NÖLL2003, S114f.]

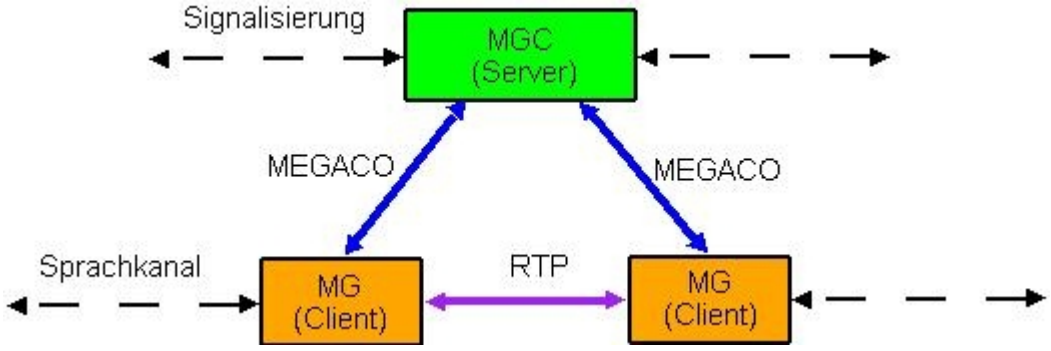

Abbildung 7: MEGACO-Architektur

Im Grunde hat ein MGC die MGs so anzusteuern, dass ein Kommunikationskanal zwischen zwei Endpunkten an den MGs zustande kommt. Die Client-Server Architektur wird treffend durch folgende Metapher beschreiben: Der MGC ist das Gehirn, während die MGs die Muskeln in dem System sind.

Wie in Abbildung 7 skizziert, decken unterschiedliche Protokolle verschiedene Aufgabenbereiche der Architektur ab.

2.4.3 Einordnung der Protokolle

Die bereits in den vorangegangenen Kapiteln erläuterten Protokolle RTP und SIP stehen in folgendem Zusammenhang mit MEGACO: mittels RTP werden die Echtzeitmedien zwischen den Gateways transportiert. Möglicherweise befinden sich ein oder mehrere Endgeräte in einem IP Netzwerk, dann würde der RTP-Kanal über den MG hinaus bis zum entsprechendem Endpunkt bestehen. Als Signalisierungsprotokoll kann zum Beispiel

H.323 oder SIP benutzt werden. Im Vorgriff auf das Konzept Kapitel (Kapitel 3) wird nur SIP und nicht H.323 als Signalisierungsprotokoll für das Konzept benötigt. Daher soll im Folgenden nur noch SIP betrachtet werden.

Neben einem Transport- und einem Signalisierungsprotokoll wird das MEGACO-Protokoll benötigt, um zwischen den Gateways zu vermitteln. MEGACO kann Nachrichten per TCP oder UDP übertragen. Bei letzterem muss allerdings ein *Application Layer Framing* (ALF) Sicherungsmechanismus implementiert werden. Genauere Informationen zu dieser Anforderung sind im Kapitel „ANNEX D - Transport over IP“ zu finden ([RFC3525, S.150ff,]).

Laut [RFC 3525], S. 72 muss der MGC die Kommunikation über TCP und UDP, der MG hingegen nur eines der beiden Transportprokolle unterstützen.

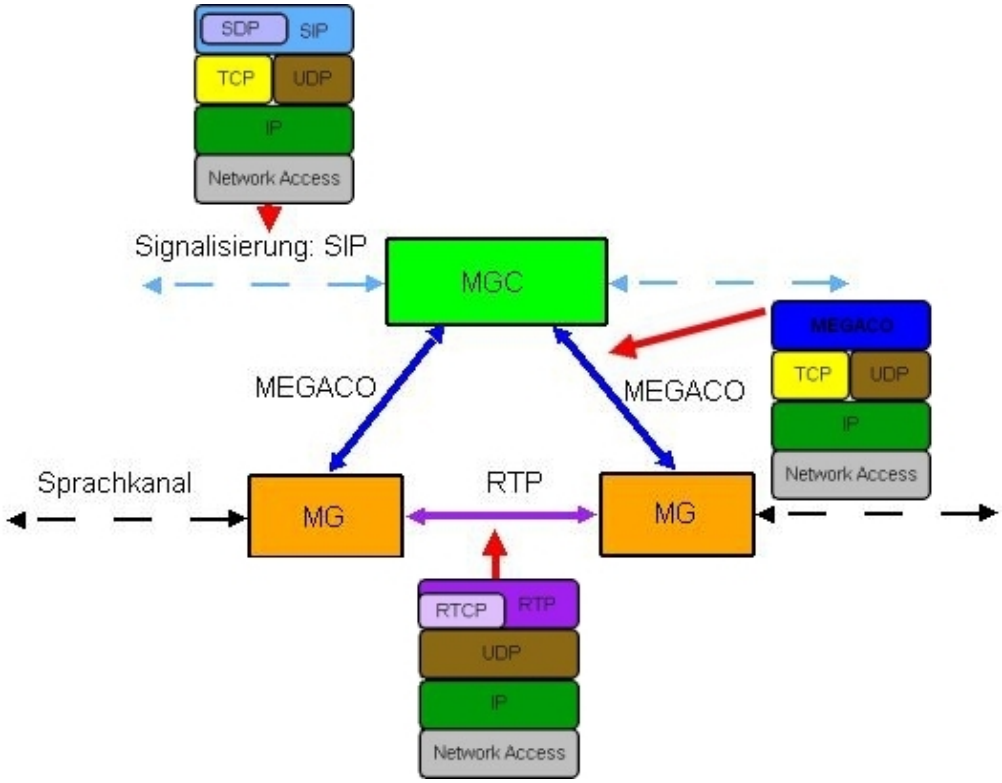

Abbildung 8: MEGACO und relevante Protokolle

Für eine feingranulare Kontrolle der Elemente, die untereinander mit einer Reihe verschiedenster Protokolle kommunizieren, enthält MEGACO eine abstrakte Modellierung.

2.4.4 Das abstrakte Modell

Das MEGACO-Protokoll verallgemeinert die Kommunikation auf einer abstrakten Ebene. Die logischen Endpunkte einer Verbindung, also Anschlüsse an den Gateways, werden durch so genannte *Terminations* repräsentiert. Die Verbindung zwischen zwei oder mehreren *Terminations* in einem Media Gateway besteht genau dann, wenn sie in einem Kontext, der als *Context* bezeichnet wird, zu einander stehen. Gewissermaßen werden werden die Ports an einem MG miteinander verschaltet.

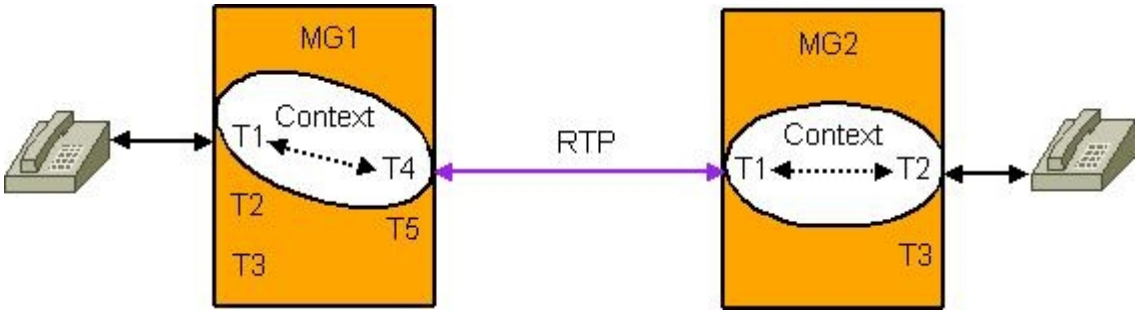

Abbildung 9: Verbindungen mit Terminations und Contexts

Mit zwei Gesprächsteilnehmern würde die Kommunikation über beispielsweise zwei MGs derart aussehen, dass in dem jeweiligen MG die involvierten *Terminations* in einen *Context* gesetzt werden, um die etablierte Verbindung anzuzeigen. In Abbildung 9 sind im MG1 die *Termination* T1, die das Telefon repräsentiert und T4 für das eine RTP-Session existiert, miteinander verbunden. Ebenso im MG2, hier werden der RTP-Strom via T1 und das Telefon via T2 in einen *Context* zueinander gesetzt.

Terminations sind die Quellen und Senken einer Kommunikation. Die „physical" *Terminations* sind Endgeräte, wie zum Beispiel Telefone. Nicht physikalische *Terminations* werden „emphemeral" *Termination* genannt. Sie werden unter anderem gebraucht, um den Endpunkt einer RTP-Session zu beschreiben. Sie entstehen und verschwinden dynamisch zur Laufzeit. Üblicherweise hält ein Media Gateway mehrere *Terminations* vor.
Die *Termination* wird beschrieben durch: allgemeine Eigenschaften (*Properties*), mögliche Ereignisse (*Events*), mögliche Signale (*Signals*) und Statistiken (*Statistics*). Zusammengenommen ist das ein *Package*. Es sind mehrere Pakages pro *Termination* möglich. Da im Laufe einer MEGACO-Kommunikation zwei (oder mehr) *Terminations* miteinander verbunden werden sollen, müssen die Eigenschaften zusammenfasst und übertragen werden können. Dazu gibt es *Descriptors*, die als Parameter einer MEGACO-Nachricht angesehen werden können. [BADA2005, S.300/RFC3525, S. 17]

Ein weiteres Abstraktionselement in dem MEGACO-Modell ist der ***Context***. Neben der Zuordnung, welche *Terminations* miteinander verbunden sind, werden auch die Topologien beschrieben, also, ob eine Verbindung uni- oder bidirektional ist.
Terminations die nicht verbunden sind, gehören dem „NULL-*Context*“ an. Sofern eine *Termination* ein Ereignis registriert, ist sie in der Lage, durch eine geeignete Message, den MGC zu informieren, der dann zum Beispiel einen Anrufaufbau initiiert. [RFC3525, S. 13ff.]
Das Kontextprinzip ist eine Neuentwicklung des MEGACO-Protokolls, im Gegensatz zum Vorgänger MGCP. Unter anderem zeigt sich hieran, warum MEGACO inkompatibel zu MGCP ist. [NÖLL2003, S.113]

Das erste Beispiel von Abbildung 10 zeigt eine Dreier-Konferenzschaltung, an der ein Teilnehmer aus einem paketorientierten Netzwerk und zwei aus einem leitungsvermittelnden stammen. Im zweiten Beispiel befindet sich eine *Termination* im NULL-*Context*, sie ist also nicht verbunden. Beispiel Drei illustriert eine Punkt-zu-Punkt Verbindung zwischen zwei Endgeräten.
Es sind auch Szenarien mit wartenden Anrufen möglich. So kann eine anrufende *Termination* in einem *Context* solange warten, bis die bereits etablierte Kommunikation der gewünschten *Termination* beendet ist und die dann ihrerseits aus dem „alten“ *Context* in den *Context* der wartetenden *Termination* wechselt. [RFC3525, S14ff.]

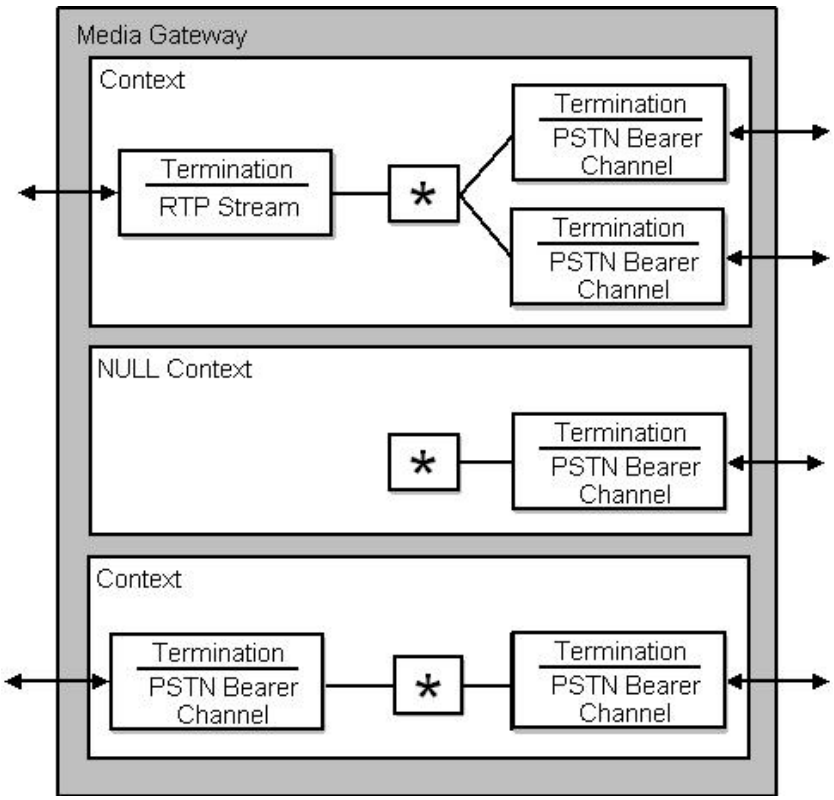

Abbildung 10: MEGACO-Verbindungsmodell [RFC3525, S.14]

Um die *Terminations* in *Contexts* zu setzen oder sie daraus zu lösen, sieht das MEGACO-Protokoll bestimmte Kommunikationsnachrichten vor.

2.4.5 Kommunikationsnachrichten

Für die Kommunikation zwischen Media Gateway und Media Gateway Controller gibt es eine Anzahl unterschiedlicher Nachrichten, die:

- das Manipulieren von *Terminations* und *Context* ermöglichen,
- zur Signalisierung benötigt werden,
- Statistiken übertragen können,
- oder ein generelle Steuerung eines Media Gateways zulassen.

Tabelle 2 zeigt eine Übersicht der Kommunikationsnachrichten.

Nachrichten	*Zweck*
Add, Subtract, Move	Manipulation der logischen Elemente (*Termination*/*Context*)
Notify, Modify	Signalisierung
AuditValue, AuditCapabilities	Statistiken
ServiceChange	Generelles Management

Tabelle 2: Übersicht der Kommunikationsnachrichten

Terminations und *Contexts* können mit den Nachrichten **Add**, **Subtract** und **Move** gesteuert werden. Die *Add* Nachricht fügt eine *Termination* zu einem *Context* hinzu. Add wird beim Verbindungsaufbau verwendet. *Subtract* löscht *Terminations* aus einem *Context*. So können bestehende Verbindungen beendet werden. Werden alle *Terminations* aus einem *Context* entfernt, so wird auch der *Context* gelöscht. Wenn eine *Termination* von einem *Context* in einen anderen wechseln soll, kann dies mit *Move* erreicht werden.

Währen der Signalisierung müssen Ereignisse registriert und bestimmte Parameter (z. B. verwendeter Codec) der aufzubauenden Verbindung festgelegt werden. Wenn Ereignisse, wie zum Beispiel das Abnehmen eines Hörers detektiert werden, muss ein Media Gateway das an den Media Gateway Controller weiterleiten. **Notify** ist die entsprechende Benachrichtigung. Über das **Modify** Kommando können Eigenschaften, Signale oder Events in einer *Termination* geändert werden.

MEGACO unterstützt die Möglichkeit Eigenschaften, Ereignisse oder Statistiken der Gateways zu erfragen. Solche Anfragen werden mit der Nachricht **AuditValue** vorgenommen. Über **AuditCapabilities**-Nachrichten kann ein MGC alle zulässigen Eigenschaften, Ereignisse oder Statistiken der MGs in Erfahrung bringen.

Sobald eine *Termination* in oder außer Betrieb genommen wird oder bei Änderungen von Eigenschaften, wird dies dem MGC mit einer **ServiceChange** Nachricht mitgeteilt. Eine Registrierung eines ganzen MGs bei einem MGC erfolgt ebenfalls über diese Nachricht. [RFC3525, S. 26f.]

Die vorgestellten Kommunikationsnachrichten werden auch *Commands* genannt. Es ist gut möglich, dass viele Kommandos gleichzeitig an einen MG abgesetzt werden müssen. Damit nicht jedes Kommando in einer einzelnen Nachricht versendet werden muss, werden mehrere Anweisungen zusammengefasst. MEGACO benutzt dafür ein hierarchisches

Prinzip: Mehrere Commands werden in *Actions* gruppiert. Mehrere Actions sind ihrerseits wieder ein *Transactions* gruppiert (vgl. Abbildung 11). [RFC, 3525, S.66ff.]
Muss nur ein einzelnes Command übertragen werden, so enthält die Transaction lediglich eine Action und diese nur das eine Command.

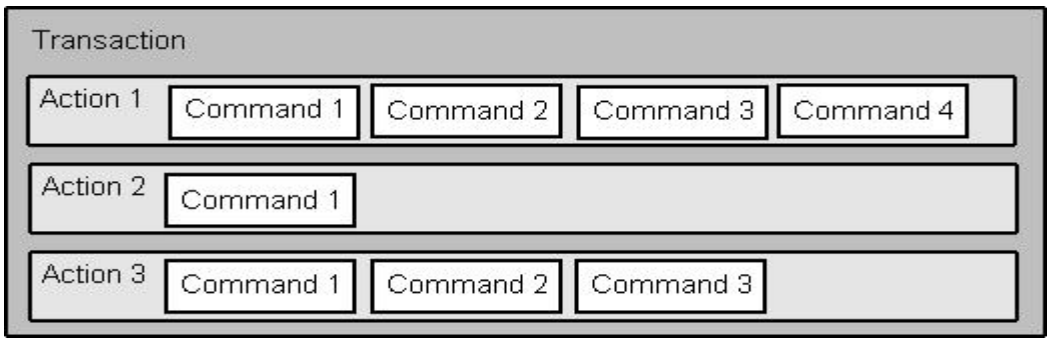

Abbildung 11: Transaction [RFC3525, S. 67]

Die verschiedenen Arten der Nachrichten müssen auf einheitliche Weise kodiert werden, damit sie über ein Netzwerk übertragen und an der Empfangsseite richtig interpretiert werden können.
Es gibt im MEGACO-Protokoll zwei Arten der Kodierung, einmal binär und einmal textbasiert. Die binäre Kodierung folgt der *Abstract Syntax Notation One* (ASN.1) Spezifikation[11]. Mit ASN.1 werden Datentypen und Werte definiert. Aus primitiven Datentypen können komplexere Typen zusammengesetzt werden:

```
IP4Address ::= SEQUENCE
{
      address OCTET STRING (SIZE(4)),
      portNumber INTEGER(0..65535) OPTIONAL
}
```

Hier wird der Typ `IP4Address` definiert, der aus einem vier Byte langen Datenwort als Adresse (`address`) und einem Integer für die Portnummer besteht. Der zweite Parameter ist zudem optional und hat einen definierten Wertebereich von 0 bis 65535. Das Beispiel ist ein Auszug aus dem MEGACO-Protokoll: [RFC3525, S. 94].
Über *Basic Encoding Rules* (*BER*)[12] werden die Datentypen kodiert und dekodiert.

Eine Kodierung über Textnachrichten erfolgt nach der im RFC 2234 definierten *Augmented Bakus-Naur Form* (ABNF). ABNF ist eine Metasprache, die Regeln enthält mit denen

11 [X.680]
12 [X.690]

man mit primitiven Elementen durch Wiederholungen, Auswahl aus Alternativen und Sequenzen komplexere Elemente erschaffen kann. Die zuvor genannte Definition einer IP-Adresse lautet in ABNF wie folgt:

```
IPv4address = V4hex DOT V4hex DOT V4hex DOT V4hex
V4hex = 1*3(DIGIT) ; "0".."255"[13]
```

Die Adresse besteht also aus einer Zeichenfolge, die aus 4 Dezimalzahlen besteht, mit je einem Wertebereich von 0 bis 255, die durch einen Punkt abgetrennt sind. Es handelt sich um die punktuelle Notation von IP-Adressen, wie zum Beispiel „192.168.0.1“.

ABNF ist unabhängig von der unterliegenden Architektur, da Nachrichten „nur“ aus Text bestehen (solange die Textkodierung allen Kommunikationsteilnehmern bekannt ist). ASN.1 ist nicht unabhängig von den darunteliegenden Schichten, aber es ist performanter. Es benötigt weniger Bytes um Nachrichten zu kodieren und es benötigt keinen Textinterpreter, der die Textnachrichten auswerten oder ein Modul, das sie aufsetzen kann.

Media Gateway Controller sollen beide Arten der Kodierung unterstützen. Media Gateways können beide Arten unterstützen, allerdings ist eine hinreichend. [RFC3525, S. 66]

2.4.6 Verbindungsauf- und Abbau

Die Abbildung 12 soll einen Anrufaufbau verdeutlichen. Beide, in diesem Fall analoge Endgeräte, sind direkt an die Media Gateways angeschlossen (welche sich in einem IP-Netz befinden). Man spricht auch von „Residential Gateways[14]“. In Abbildung 12 wird auch das Command-Reply Schema sichtbar, das heißt jedes Kommando wird direkt mit einem *Reply* beantwortet.

13 [RFC3525, S. 117]

14 Ein Residential Gateway vermittelt zwischen analogen Telefonen und paketvermittelnden Netzen. [RFC3525, S. 11]

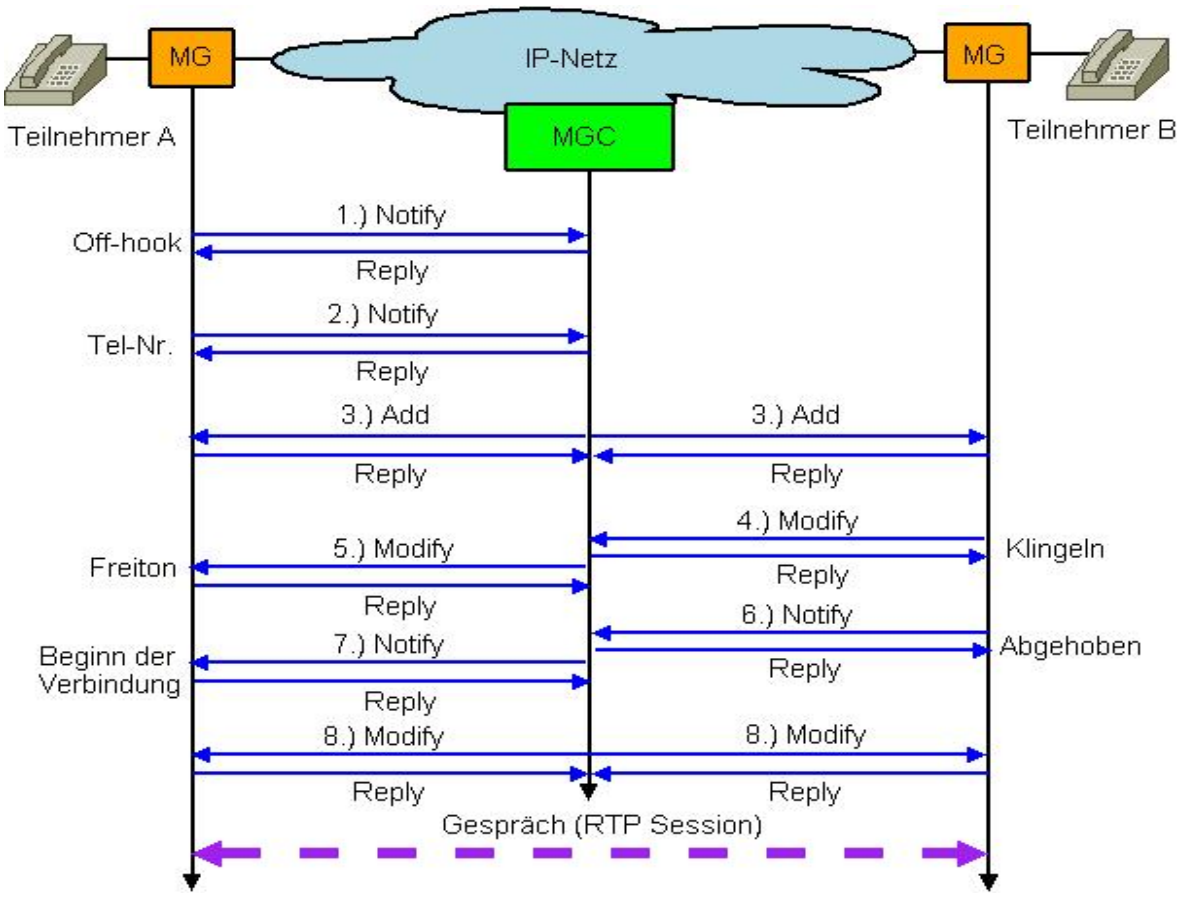

Abbildung 12: Verbindungsaubau nach MEGACO [BADA2005, S. 302]

Der Aufbau verläuft wie folgt:

1.) Das Abnehmen (Off-hook) des Hörers wird am MG detektiert und mit dem Kommando *Notify* dem MGC signalisiert.

2.) In einem zweiten *Notify* werden die Wahlziffern übermittelt. Der MGC muss dann anhand der Telefonnummern die IP-Adresse das Zielteilnehmers ermitteln.

3.) An beide MGs wird das *Add* Kommando gesendet. Jeder MG errichtet einen neuen *Context* (die Assoziation eines analogen Ports mit einem RTP-Stream). In der *Reply* Nachricht werden Informationen über die Sprachformate der *Terminations* übermittelt. Nur wenn sie kompatibel sind, wird der Rufaufbau fortgesetzt.

4.) *Modify* startet das "Klingeln". Die möglichen Sprachformate werden übermittelt und der MG entscheidet sich für ein Format. Dies wird im Reply dem MGC kommuniziert.

5.) Der andere MG erhält ebenfalls ein *Modify*, um das Freizeichen zu generieren. Zusätzlich enthält die Nachricht Informationen über das gewählte Sprachformat.

6.) Ein *Notify* signalisiert das Abheben des Hörers am angerufenen Endgerät.

7.) Der MGC reicht das *Notify* an den MG des anrufenden Teilnehmers weiter.
8.) Es werden MGs mit Hilfe der *Modify* Nachricht aufgefordert, das Auflegen eines Telefonhörers am jeweiligen MG zu signalisieren. Im Anschluss ist ein Gespräch über eine RTP-Session möglich.

Der Verbindungsabbau verläuft ähnlich. Das Auflegen eines Teilnehmers wird über eine *Notify* Nachricht bekannt gemacht. Daraufhin weist der MGC die teilnehmenden MGs mit dem Subtract Kommando an, die *Terminations* aus den *Contexts* zu löschen. Das beendet die RTP-Session. Der andere Teilnehmer sendet ein *Notify*, sobald auch er auflegt. Zum Schluss werden beide MGs per *Modify* aufgefordert, ein erneutes Abheben eines Hörers zu signalisieren. [BADA2005, S. 302ff.]

Ein wichtiges Einsatzgebiet von MEGACO ist die Vermittlung zwischen verschiedenartigen Netzen. Die nächste Illustration (Abbildung 13) zeigt den Verlauf eines Rufaufbaus zwischen einem Telefon an einem Residential Gateway im IP-Netzwerk (aus dem vorherigen Beispiel) und einem ISDN-Telefon. Man kann erkennen, dass der Verlauf im IP-Netzwerk ähnlich dem aus Abbildung 12 ist. Der Übersicht halber sind die Reply-Antworten nicht enthalten. Neu im zweiten Media Gateway ist eine als Signaling Gateway (SG) bezeichnete Instanz, die die MEGACO-Kommandos in die entsprechenden ISDN Befehle für die Signalisierung und umgekehrt übersetzt (SETUP: Beginn einer Verbindung, ALERT: es klingelt, CONN signalisiert Abheben des Hörers und das Akzeptieren der Verbindung, CONN ACK: bestätigt CONN). [BADA2005, S. 304f.]

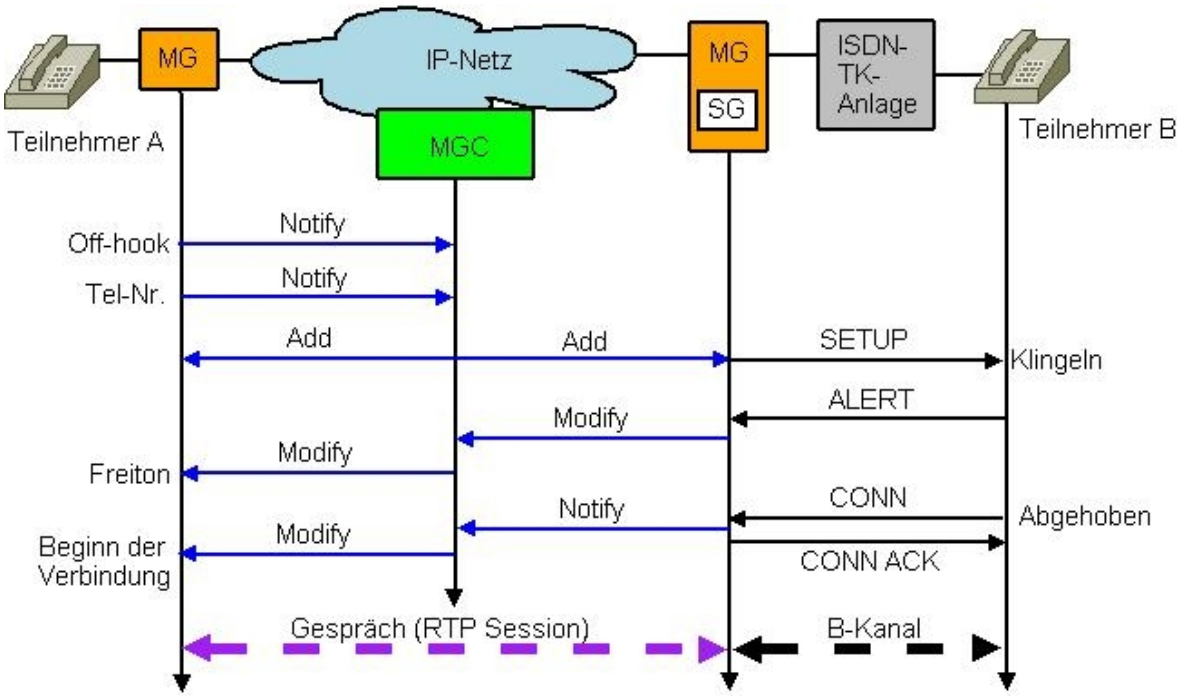

Abbildung 13: Integration von VoIP und ISDN [BADA2005, S. 304]

2.4.7 Anwendungsszenarien

Neben den im Kapitel 2.4.6 vorgestellten Residential Gateway Konzept sind auch weitere Szenarien möglich. Zum Beispiel könnte MEGACO als Vermittlungsstelle zwischen einem auf SIP oder H.323 als Signalisierung basierendem IP-Netz und einem ISDN-Netz wirken (vgl. Abbildung 14). Man bezeichnet dies auch als „Trunking Gateway“.

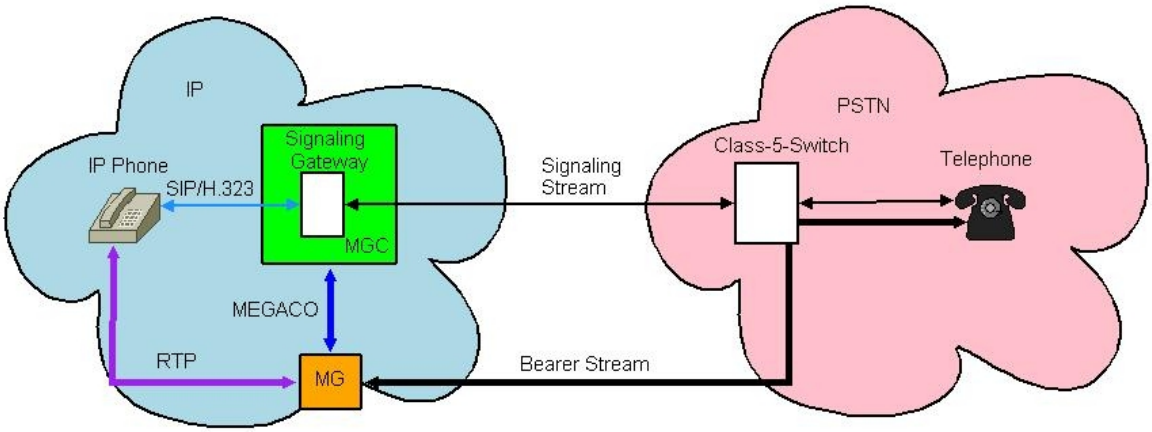

Abbildung 14: Trunking Gateway [RADVISION, S. 4]

Ein weitere Einsatzmöglichkeit für MEGACO, ist der Betrieb als Interactive Voice Re-

sponse (IVR) Server. Falls sich ein Teilnehmer mit einem IP-Telefon verwählt, kann der MGC einen MG veranlassen, einen RTP-Kanal zu dem Telefon aufzubauen, um eine Sprachnachricht der Art „Sie haben sich leider verwählt“ zu übermitteln (siehe Abbildung 15).

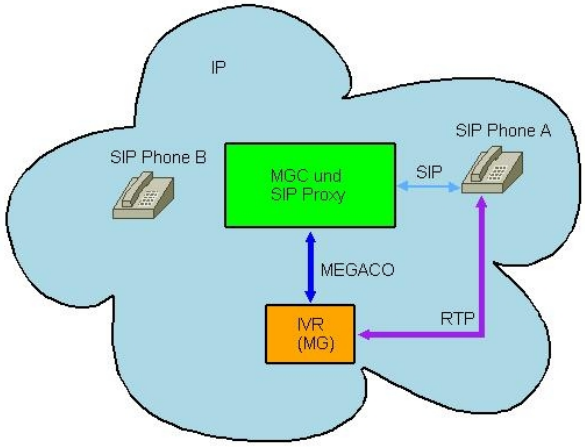

Abbildung 15: Interactive Voice Response Server [RADVISION, S. 7]

3 Konzept

In diesem Kapitel soll ein Konzept für die Implementierung des MEGACO-Protokolls für ein spezielles Einsatzszenario vorgestellt werden. Anschließend wird die passende Hardware aus dem Telekommunikationsbereich diskutiert, um das Szenario zu realisieren.

3.1 Aufgabenstellung

Ziel ist die Implementation der Kernkomponenten des MEGACO-Protokolls, so dass ein „Trunking Gateway“, also ein Vermittlungsgateway zwischen IP und PSTN-Netzen, auf ausgewählten Hardwarekomponenten realisiert werden kann. Das genaue Einsatzszenario für den Gateway ist im folgenden Kapitel (3.2) beschrieben. Die zu implementierenden Kernkomponenten umfassen die Realisierung der MGC und MG Instanzen und der grundlegenden Kommunikation untereinander.

3.2 Einsatzszenario

Im Hinblick auf die Verschmelzung von IP und PSTN-Netzen spielen Gateways zwischen diesen Netzen während der Übergangsphase eine wichtige Rolle. Daher soll ein so genanntes „Trunking Gateway“ realisiert werden. Ziel ist die Verbindung eines ISDN-Telefons und eines PCs, der über ein Headset und eine VoIP-Telefonie-Software[15] verfügt. Das Signalisierungsprotokoll soll SIP sein. Das ISDN Telefon ist an einer Testbox angeschlossen, um mehrere Telefone zu simulieren. Die Testbox ist über einen Primärmultiplexanschluss[16] mit dem Gateway verbunden. Somit können auch größere Telefonanlagen simuliert werden.
MEGACO soll das Verbindungsstück, angedeutet durch die schwarze Box auf Abbildung 16, zwischen den beiden unterschiedlichen Netzen sein.

15 Für einen Linux PC kann zum Beispiel „Ekiga“ benutzt werden. Es unterstützt SIP, sowie alle wichtigen Sprachcodecs .
16 Datenleitung mit 2.048 Mbits/s in Europa. Kann für ISDN benutzt werden, um 30 B-Kanäle zu bündeln.

Abbildung 16: Einsatzszenario

Um das Konzept zu vereinfachen, werden zunächst keine Konferenzsitzungen (ein Telefonat mit mehr als zwei Teilnehmern) unterstützt. Ebenfalls zur Vereinfachung, werden die zulässigen Codecs in dem IP Netz auf G.711 beschränkt. Da ISDN ebenfalls G.711 benutzt, folgt hieraus, dass keine Umwandlungen von einem Codec in einen anderen notwendig sind.

3.3 Konzept für einen Trunking Gateway mit MEGACO

3.3.1 Softwareelemente

Als Einsatzszenario ist ein Trunking Gateway vorgegeben. Abbildung 14 im Grundlagen Kapitel stellt dieses Szenario dar. Es werden ein Media Gateway für die Sprachübertragung und ein Media Gateway Controller zur Steuerung der MGs für die Signalisierung benötigt. Beide Instanzen sollen als unabhängige Prozesse auf einer einzigen Plattform laufen.

Der **MGC** benötigt einen Signaling Gateway, um die SIP-Nachrichten in entsprechende D-Kanal Nachrichten und umgekehrt umwandeln zu können. SIP und D-Kanal Protokollstack müssen hierfür verfügbar sein.

Der **MG** muss die Sprachdaten vom B-Kanal in die Nutzdaten einer RTP-Session überführen (und umgekehrt). Dafür ist eine Implementierung des RTP- und B-Kanal Protokollstapels notwendig.

Es gibt zwei Ansätze, um den MGC und den MG miteinander kommunizieren zu lassen. Die erste Möglichkeit ist eine Interprozesskommunikation. Das bedeutet, es werden lokale Funktionen aufgerufen, um die MEGACO-spezifischen Aktionen auszulösen. Eine Kodierung der Nachrichten ist nicht notwendig und sie müssen auch nicht durch den Netzwerkprotokollstack umgeformt werden (u. a. durch das Anfügen und Entfernen der Header).
Die andere Möglichkeit ist die Verwendung des TCP/IP Protokoll-Stacks über sein **lokales Loopback-Interface**[17], das heißt die MEGACO-Nachrichten werden aufgesetzt und mittels TCP/UDP an die eigene Plattform gesendet und anschließend ausgewertet. Diese Methode ist weniger performant, aber sie hat den entscheidenden Vorteil, dass im Zuge einer Weiterentwicklung leicht ein verteiltes Szenario realisiert werden kann. Das bedeutet, der MGC läuft als einzige Instanz auf dem System und ein oder mehrere MGs residieren auf anderen Systemen. In diesem Fall würde das Loopback-Interface durch ein „echtes“ Netzwerk ausgetauscht werden. Aufgrund der höheren Flexibilität wird die zweite Alternative gewählt.

In Abbildung 17 ist die das Konzept von einer MGC und einer MG Instanz auf einer gemeinsamen Plattform illustriert.

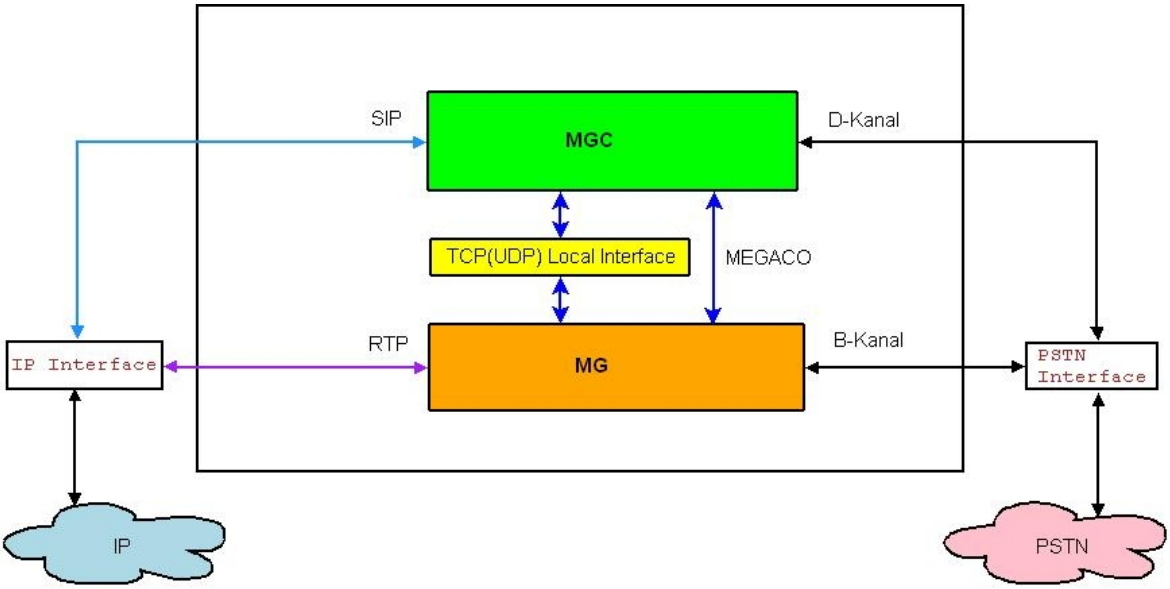

Abbildung 17: Softwarekonzept

Neben den beiden Gateway-Instanzen müssen auch Funktionen zum Verwalten der ***Terminations*** und ***Contexts*** vorhanden sein. Da sich *Terminations* und *Contexts* in den MGs

17 Quell- und Zielsystem sind identisch. IP-Pakete werden gewissermaßen an „sich selbst“ gesendet.

befinden, sind sie ein Teil des MG Moduls.

Die Softwareelemente und deren Schnittstellen zu den Protokollstacks, kann man in eine Softwarearchitektur eingliedern.

3.3.2 Softwarearchitektur

Die Erstellung einer Softwarearchitektur, also die Unterteilung des Gesamtprojektes in kleinere Moduleinheiten und Beschreibung des Zusammenspiels derer, soll das gesamte Projekt übersichtlicher und durch die Modularisierung besser wartbar gestalten.

Die Gesamtsoftware lässt sich in mehrere Schichten unterteilen (vgl. Abbildung 17). Die unterste Ebene ist eine Hardware- und Treiberschicht. Darauf aufbauend befinden sich die Netzwerkprotokollstacks. Bis auf RTP und SIP sind alle Komponenten dieser beiden Schichten bereits implementiert. Die Funktionalitäten der Protokollstapel sollen in Interfacemodulen gekapselt sein. MEGACO ist in der Applikationsschicht zu finden und greift auf besagte Interfacemodule zu. Die MEGACO-Applikation muss ein MGC-, ein MG- und ein Message-Modul besitzen, um Nachrichten zwischen den Gateway-Instanzen austauschen zu können. Das Nachrichtenmodul muss sowohl die Nachrichten kodieren, als auch dekodieren können.

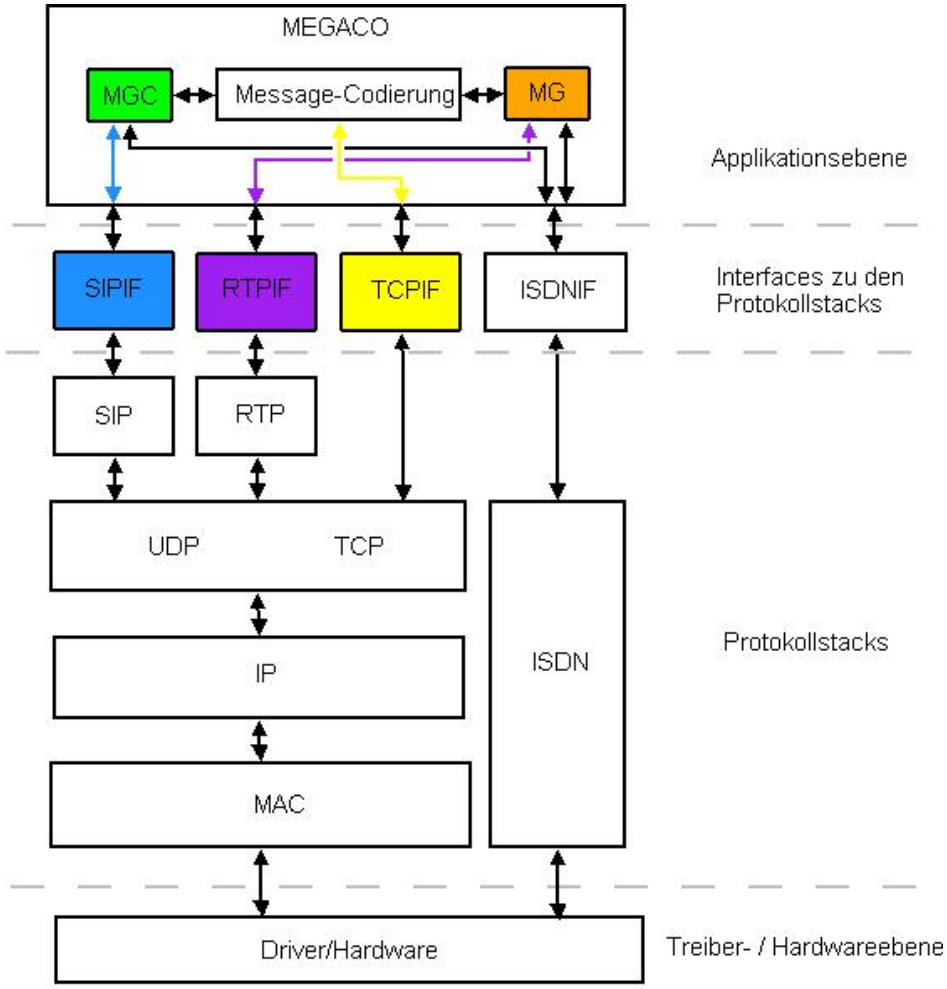

Abbildung 18: Softwarearchitektur

3.4 Zielhardware für einen Trunking Gateway

Um einen leistungsfähigen Trunking Gateway zu erhalten, soll die MEGACO-Software auf spezieller embedded Telekommunikationshardware der Produktpalette der N.A.T. GmbH laufen. Aus dem gestellten Einsatzszenario ergeben sich Anforderungen an die Hardwareplattform, auf der MEGACO implementiert werden soll:

1.) Interface zu einem IP-Netz
2.) Interface zu einem ISDN-Netz
3.) Rechenkapazität, also die Fähigkeit Maschinencode hinreichend schnell auszuführen, für das MEGACO-Protokoll, inklusive MGC, MG, dem logischen Modell und der Protokollabhandlung

Außerdem soll die Hardware in einem cPCI[18] System eingesetzt werden. Diese Anforderung resultiert nicht aus dem Einsatzszenario, sondern begründet sich darin, dass der Gateway für die auf cPCI basierende Produktlinie vertrieben werden soll. Dies stellt Anforderung 4.) dar.

3.4.1 NVTP1001

Bei dem NVTP1001-Board von der N.A.T GmbH, handelt es sich um eine cPCI-Trägerkarte (Carrier-Board). Auf der Karte befindet sich ein, für ein embedded System, leistungsstarker MPC8280 PowerQUICCII Prozessor von Motorola[19]. Er kann von 333 bis 450MHz betrieben werden. Als Hauptspeicher sind 32 bis 256MB SDRAM[20] verfügbar. Die CPU soll das MEGACO-Protokoll abwickeln. Gateway-Instanzen müssen ebenfalls bearbeitet werden. Vor dem Hintergrund dieser speziellen Anwendung als Telekommunikationsgateway, dürften Prozessor und Hauptspeicher hinreichend leistungsfähig sein.

Abbildung 19: NVTP1001

Dieses Board hat zwei Steckplätze für so genannte (optionale) PMC-Module[21] (daher auch der Name *Carrier*-Board). Es kann mit den zusätzliche Hardwarekomponenten an spezielle Aufgabenbereiche angepasst werden. Der Prozessor kann mit seinem internen PCI-Bus über eine PCI-zu-PCI Brücke auf den zweiten internen PCI-Bus und damit auf die Funktionen der PMC-Module zugreifen. [NVTP1001]

Die gesamte Platine wird in ein Gehäuse mit einem cPCI-Rückwandbus eingeschoben

18 Eine auf dem PCI(Peripheral Component Interconnect)-Bus basierende Industriecomputer-Architektur. cPCI Karten werden in ein Gehäuse mit einem cPCI-Rückwandbus eingeschoben; sie sind über den cPCI-Bus mit einander verbunden (Backplane).

19 *Motorola* ist seit 2004 unter dem Namen *Freescale Semiconductor* bekannt. Das Referenzdatenblatt bezieht sich allerdings noch auf *Motorola*

20 Synchronous Dynamic Random Access Memory (flüchtiger Speicher)

21 PCI Mezzanine Card

(auch als „Rack“ bezeichnet). Über den cPCI-Bus können weitere cPCI-Boards mit dem NVTP1001-Board kommunizieren und zum Beispiel auch auf die Funktionen der PMC-Module zugreifen. Das NVTP1001 System enthält einen *82545 Gigabit Ethernet Controller* des Herstellers Intel. Dieser erlaubt die Kommunikation mit einem IP-Netz und stellt das Interface zur paketorientierten Domäne dar.

Darüber hinaus ist ein Anschluss für ein serielles Kabel vorhanden. Über diese serielle Schnittstelle können Debug[22]-Informationen ausgelesen werden.

Mit dem Ethernet-Baustein, der CPU und dem cPCI Anschluss sind die Anforderungen 1.), 3.) und 4.) erfüllt.

Die rot markierten Bereiche der Abbildung 20 stellen die für das MEGACO Konzept relevanten Elemente dar.

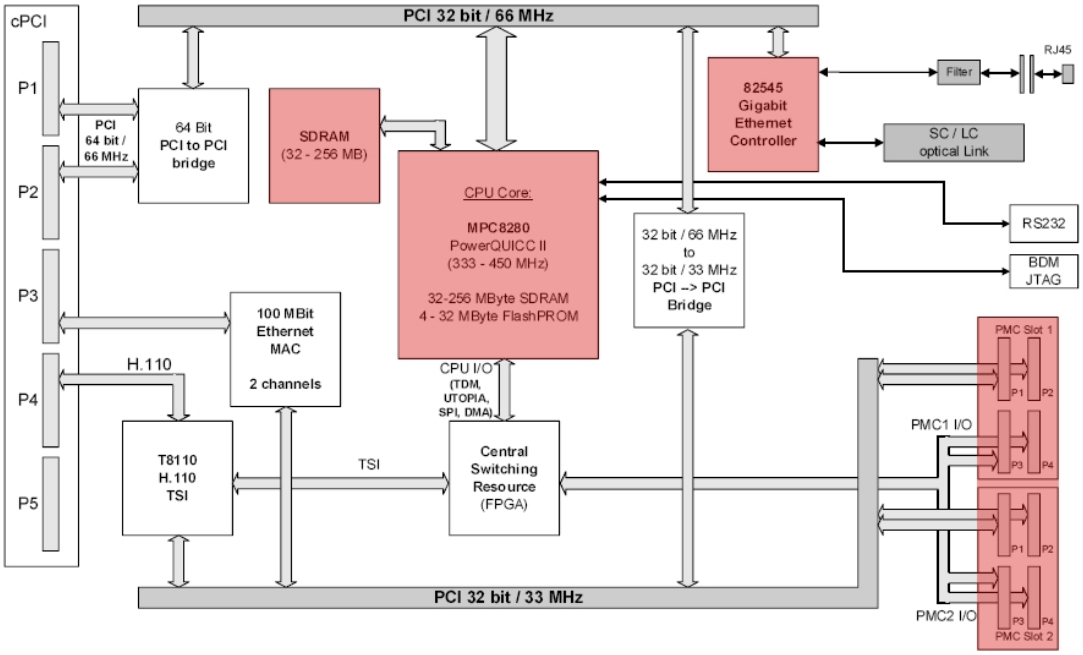

Abbildung 20: Blockdiagramm des NVTP1001-Boards [NVTP1001, S. 13]

22 Fehlersuche

3.4.2 NPMC-8E1

Um die Anforderung 2.) zu bedienen, wird noch ein ISDN-Interface benötigt. Daher kommt ein als NPMC-8E1 bezeichnetes PMC-Modul zum Einsatz. Es besitzt acht E1[23] Primärmultiplexanschlüsse.

Das NPMC-8E1-Board wird auf einen der freien Steckplätze des NVTP1001-Carrier-Bords platziert. Die Kommunikation erfolgt über einen PCI-Bus.

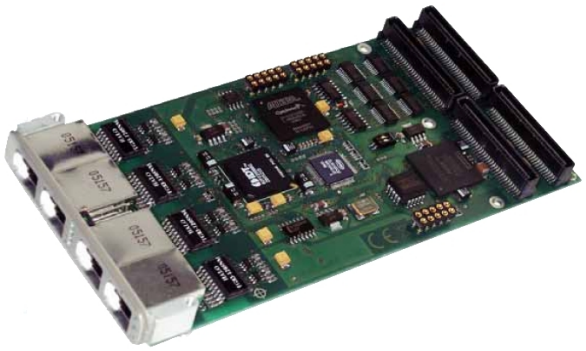

Abbildung 21: NPMC-8E1

3.4.3 NPMC-DSP

Zusätzlich kann ein NPMC-DSP Modul eingesetzt werden. Das NPMC-DSP ist ein PMC-Modul mit einer DSP-Farm (acht DSPs). Die DSPs können die Aufgabe der Codekonvertierung übernehmen. Da im Einsatzszenario die Verwendung der Sprachcodecs auf einen Codec festgelegt wurde, ist diese Karte nicht zwingend notwendig und als optional zu betrachten. Im Ausblick auf einen Einsatz der Hardware in einem ausgebauten VoIP-Umfeld ist diese Option sinnvoll.

Abbildung 22: NPMC-DSP

23 In Europa gebräuchliche Bezeichnung für einen Primärmultiplexanschluss

3.4.4 Gesamtsystem

Zusammengefasst soll ein NVTP1001-Board, bestückt mit einem NPMC-8E1 PMC Modul und optional mit einem NPMC-DSP Modul eingesetzt werden. Das gesamte Carrier-Board wird in ein cPCI-Rack eingeschoben. Obwohl es dadurch an einen cPCI-Bus angeschlossen ist und mit weiteren Trägerkarten kommunizieren könnte, wird dies nicht genutzt, weil der gesamte Gateway auf dieser einen Karte läuft. Man kann dies als eine „Standalone" Anwendung betrachten. Das Rack versorgt die NVTP1001 Karte mit Strom. In Anlehnung an Abbildung 16 soll die folgende Skizze die Einordnung der Hardwarekomponenten in das Einsatzszenario anschaulich machen.

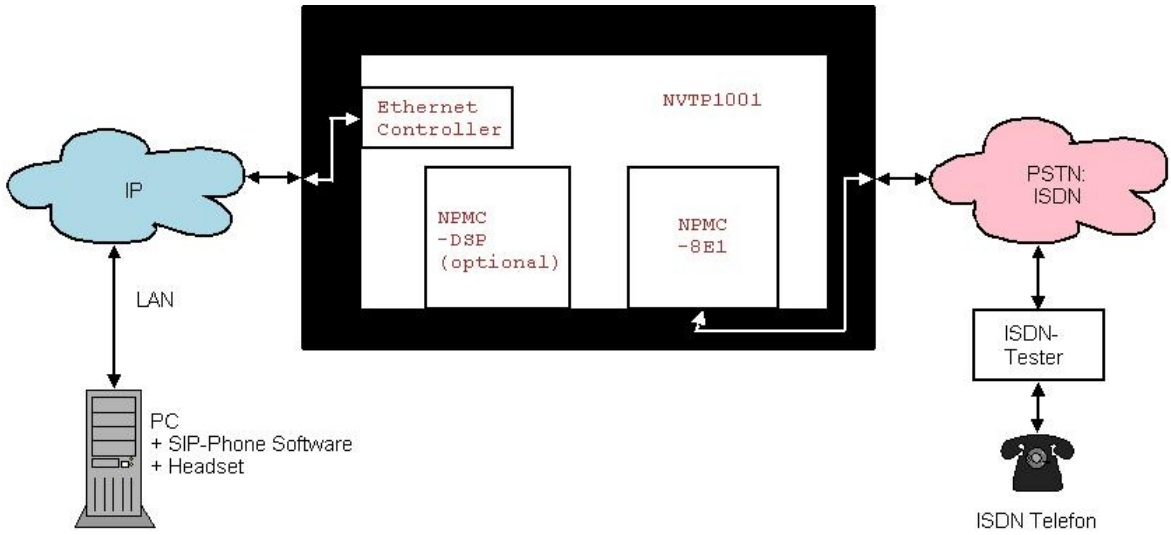

Abbildung 23: Gesamtsystem

3.4.5 Softwareumgebung

Für das NVTP1001 System ist ein proprietärer, von der N.A.T. GmbH entwickelter Echtzeit-Micro-Kernel[24] mit dem Namen *OK1* (Open Kernel One) verfügbar. Der Kernel übernimmt viele wichtige Funktionen, wie zum Beispiel Speichermanagement, Zeitmanagement, Signal- und Interruptabhandlung und Taskscheduling. Es handelt sich um ein nichtpreemptives Scheduling, das heißt, dass der Prozessor nicht den einzelnen Prozessen entzogen werden kann (außer bei Interrupts). Die maximale Anzahl der Tasks ist nur durch die zur Verfügung stehenden Ressourcen begrenzt. Zwischen den Prozessen kann über Signale oder Semaphoren kommuniziert werden. [OK1, S. 7ff.]

Neben diesen Grundfunktionen sind Routinen der C-Standard-Bibliothek verfügbar. Über verschiedene Treiber können Funktionalitäten der anderen installierten Hardwarekompo-

24Ein Betriebssystem mit minimaler Funktionalität wird als „Micro-Kernel" bezeichnet.

nenten genutzt werden, wie zum Beispiel der Gigabit Ethernet-Controller. Sowohl der ISDN- als auch der TCP/UDP-Stack sind bereits implementiert. Das Socket-Konzept[25], wie es bei vielen Betriebssystemen zu finden ist, ist ebenfalls verfügbar und erleichtert die Programmierung.

3.5 MEGACO auf der Zielhardware

Um einen Trunking Gateway zu realisieren, müssen die Hardware- und Softwarekomponenten miteinander zusammenarbeiten.

Die beiden Instanzen des MEGACO-Protokolls, MGC und MG, werden auf der CPU der NVTP1001-Plattform prozessiert. Diese sollen möglichst unabhängig von einander behandelt werden, damit die Flexibilität erhalten bleibt, die Instanzen zukünftig auf unterschiedlichen Plattformen laufen zu lassen. Um das zu realisieren, wird für jede Instanz ein eigener Task, mit Hilfe des OK1 Kernels, generiert.

Für die Kommunikation zwischen MGC und MG wird ein TCP/IP Loopback-Interface benötigt. Der OK1 Kernel bietet ein Socket-Konzept, auf das zurück gegriffen wird. Das heißt, jede Instanz nutzt ein Socket, um darüber MEGACO-Nachrichten abzusetzen.
Die Sockets können weiterhin verwendet werden, um den auf dem NVTP1001-Board verbauten Ethernet-Controller zu benutzen. Es können dann, über die Sockets, Server für RTP und SIP erstellt oder zu einem entfernen (RTP-/SIP-) Server eine Verbindung aufgebaut werden.

Damit der MGC den D-Kanal und der MG den B-Kanal für ISDN nutzen können, greifen sie auf die Funktionalitäten der NPMC-8E1 Karte zu. Der dafür benötigte ISDN Stack ist in dem OK1 Kernel bereits enthalten und kann über entsprechende Funktionsaufrufe genutzt werden.

In Abbildung 24 ist die Kombination der gewählten Hardwarekomponenten in Zusammenspiel mit Softwareinstanzen dargestellt.

25 Softwareschnittstelle, um Netzwerkprotokolle zu nutzen

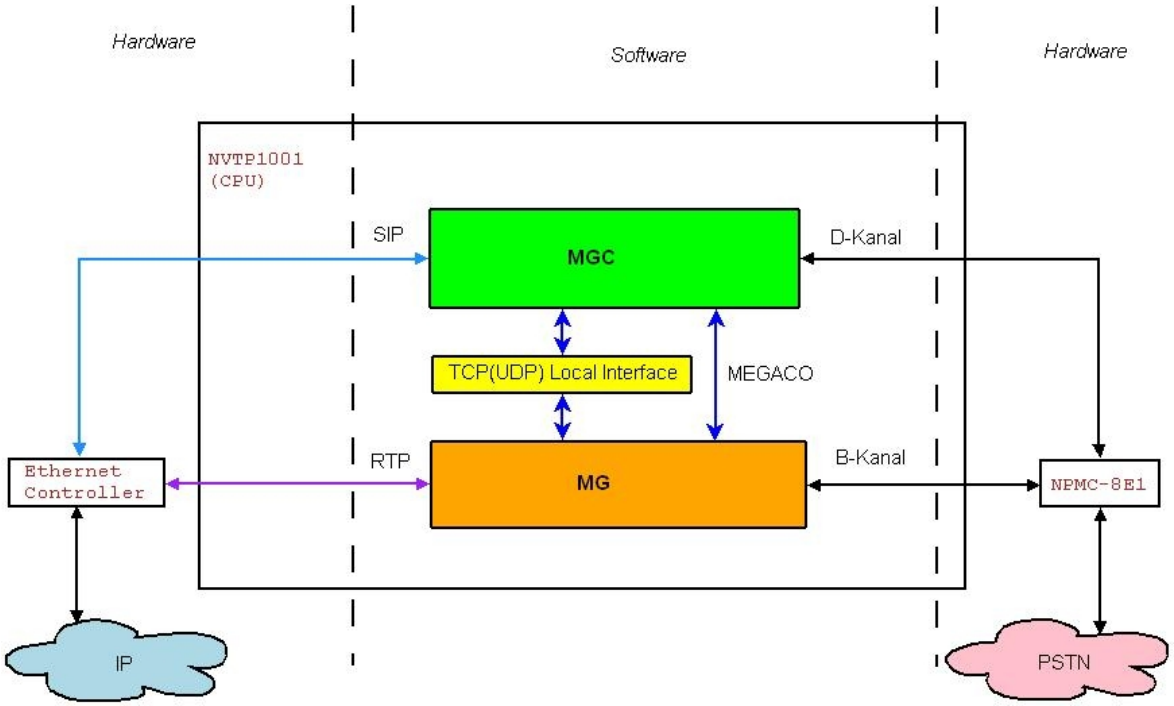

Abbildung 24: Soft- und Hardwarekonzept

4 Implementation

Dieses Kapitel beschäftigt sich mit der Implementierung des in Kapitel 3 vorgestellten Konzepts. Das beinhaltet die Wahl der Programmiersprache und die Festlegung des genauen Umfangs, der von dem Konzept realisiert wurde. Anschließend werden die implementierten Elemente genauer diskutiert. Das Kapitel endet mit einer Darstellung der aufgetretenen Schwierigkeiten.

4.1 Wahl der Programmiersprache

Bei der Implementation wurde die Programmiersprache C benutzt. C ist die für embedded Systeme bevorzugte Sprache. Sie ist mächtig, im Sinne der Möglichkeit direkt auf Speicher zugreifen zu können, und ermöglicht performanten Code zu entwickeln. Außerdem unterstützt das OK1 Betriebssystem nur C.

Die Softwaremodule wurden in einem Texteditor erstellt und mit einem für die Zielhardware passenden Compiler übersetzt. Der so generierte binäre Code kann von einem Bootloader auf dem NVTP1001-Board über ein lokales Netzwerk auf die Plattform kopiert und anschließend nach einem Neustart ausgeführt werden.

Der Aufbau des Quellcodes folgt den Programmierrichtlinien der N.A.T. GmbH. Damit ist der Code für die anderen Entwickler übersichtlicher und leichter verständlich.

4.2 Implementationsumfang

Um das Einsatzszenario aus Kapitel 3.2 zu realisieren, müssen SIP, RTP (inklusive RTCP), einer ISDN B- und D-Kanal-Ansteuerung sowie das komplette MEGACO-Protokoll implementiert werden. Zusammengenommen ist das nicht im Rahmen einer Bachlorarbeit möglich. Daher wird sich diese Arbeit auf den Kern des MEGACO-Protokolls beschränken. Die Implementation soll die Instanzen MGC und MG umfassen sowie die grundlegende Kommunikation zwischen ihnen. Dafür wird auch Funktionalität des lokalen TCP/UDP Interfaces benötigt.

Die Einordnung des Implementationsumfangs in den Gesamtkontext ist in Abbildung 25 dargestellt. Die rote Linie kennzeichnet die realisierten Module (die unteren Schichten sind nicht in dem Bild enthalten).

Der MEGACO-Standard schreibt die binäre und textuelle Kodierung der Nachrichten vor (für den MGC). Da die Interoperabilität zunächst nur zu einem MG (im gleichen System) gegeben sein muss, wird nur eine Variante implementiert. Die Wahl fällt auf die performantere binäre Kodierung.

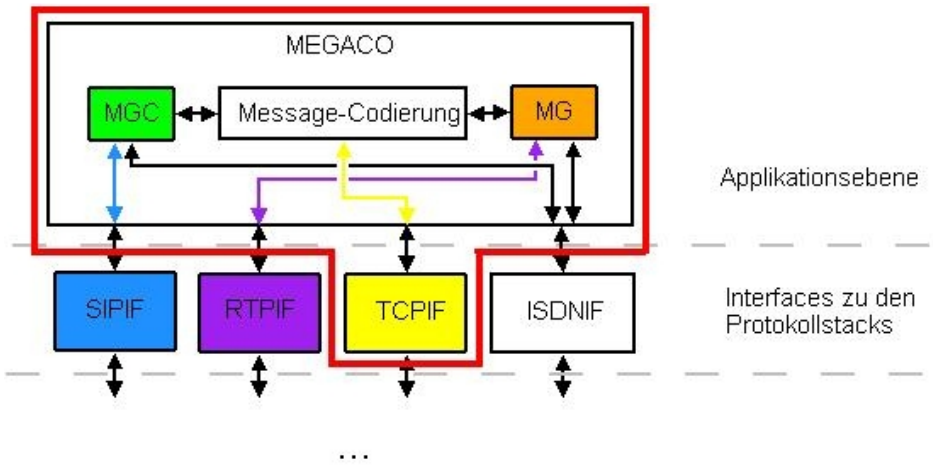

Abbildung 25: Implementationsumfang

4.3 MEGACO

4.3.1 Das lokale TCP/IP Interface

Die Kommunikation zwischen MGC und MG erfolgt über TCP (bzw. UDP). Daher ist das Modul „TCPIF[26]“ von zentraler Bedeutung. Es soll die für die Kommunikation zwischen MGC und MC benötigten Funktionalitäten in möglichst komfortable Funktionen kapseln, die eigentliche TCP/UDP Kommunikation realisieren und diese Funktionalitäten initialisieren.

Wie in Kapitel 2.4.2 dargestellt, basiert MEGCO auf einem Client-Server Modell. Diese Architektur wird auf ein Client-Server Modell im TCP/UDP abgebildet. Der MCG wird also als Server fungieren und der MG als Client. Damit sowohl TCP als auch UDP für die Übertragung von MEGACO-Nachrichten möglich sind, stellt der MGC für die beiden Verbindungsarten je einen Server bereit. Zum Aufgabenbereich des MGCs gehört auch die Umsetzung von Signalisierungsinformationen. Damit Teilnehmer aus einem IP-Netzwerk, mit SIP als Signalisierungsprotokoll, eine Verbindung zu dem Gateway aufnehmen können, muss auch ein Server für SIP Anfragen bereitgestellt werden. Als weitere Serverfunktionalitäten werden zwei UDP Sockets für RTP und RTCP benötigt, damit der MG RTP-Sessions annehmen kann. In Tabelle 3 sind die Server und ihre Funktionalitäten zu-

26 beinhaltet die Quelldateien tcpif.c und tcpif.h

sammengefasst.

Instanz	*Typ*	*Portnummer*	*Zweck*
MGC	TCP	2945	Kommunikation mit MG(s)
MGC	UDP	2945	Kommunikation mit MG(s)
MGC	UDP	5060	SIP
MG	UDP	4000	RTP
MG	UDP	4001	RTCP

Tabelle 3: Übersicht der Server

Um die Server zu verwalten, existiert eine Struktur, in der die wichtigsten Informationen für die Server hinterlegt sind. Sie enthält unter anderem: das Sockethandle[27], das benötigt wird, um die verschiedenen Sockets zu unterscheiden, die eigene IP-Adresse, die eigene Portnummer und auch den Pufferspeicher für eingehende Nachrichten. Für jedes Serversocket existiert eine Instanz dieser Struktur. Alle Instanzen sind ihrerseits in einem Array zusammengefasst. Vordefinierte Makros[28] mit eindeutiger Bezeichnung erleichtern den Zugriff auf das Verwaltungsarray.

Während der Initialisierungsphase werden Tasks erzeugt, damit jedes Serversocket eigenständig Anfragen annehmen und bearbeiten kann. Der OK1 Kernel liefert hierzu die entsprechenden Funktionen. Innerhalb der Tasks werden die Sockets erzeugt und entsprechend den Werten in dem Verwaltungsarray konfiguriert. Danach wird auf Anfragen gewartet.
Der für TCP und UDP konfigurierte Server für MEGACO-Nachrichten befinden sich in einer ständigen Schleife (solange kein Fehler auftritt) und reicht eintreffende Nachrichten an das MGC Modul weiter.

Ein Client kann sich über eine spezielle Methode (`megaco_open_conn`), unter Angabe von Ziel-IP und Port, mit einem Server verbinden. Ein Flag[29] steuert, ob die Verbindung mit TCP oder UDP aufgebaut werden soll. Für die Flags stehen Makros zur besseren Lesbarkeit zur Verfügung (`TCP_PROT`, `UDP_PROT`). Alle Parameter werden in einer Struktur abgelegt. Die Struktur wird bei dem Verbindungsaufbau in einem internen Array abgespei-

27 Dieser Parameter identifiziert ein Socket. Er wir zum Beispiel bei Lese- und Schreiboperationen benötigt.
28 Makro: Programmcode, der vom C-Präprozessor verarbeitet wird. Unter anderem kann man so aussagekräftige Namen für Konstanten verwenden, die dann später vom Präprozessor durch die echten Konstanten ersetzt werden.
29 Steuerparameter

chert. Wird die Verbindung beendet (`megaco_close_conn`), wird der Eintrag in der Tabelle gelöscht. Der Platz ist für nachfolgende oder weitere Verbindungen wieder frei. Die maximale Anzahl der gleichzeitigen Verbindungen wird nur durch die Größe des Arrays bestimmt. Die Größe kann über das Makro `MAX_CONN` gesteuert werden. Zunächst sind nur wenige Verbindungen notwendig; im wesentlichen die Verbindung zwischen MG und MGC, daher ist der aktuelle Wert von 64 hinreichend.

Nach erfolgreicher Beendung der Verbindungsaufbau-Methode wird der Index der Tabelle zurückgegeben. Im folgenden können die Lese- und Schreibfunktionen (`megaco_send_msg`, `megaco_read_msg`) diesen Index als Handle benutzen, um die gewünschte Verbindung zu identifizieren.

Standardmäßig wird während des Bootvorgangs der Ethernet-Controller für die IP-Kommunikation konfiguriert, nicht aber das benötigte Loopback-Interface. Die Routine, die das lokale Interface startet, ist durch eine kleine Abwandlung der ursprünglichen Methode für die Initialisierung des Ethernet-Controllers realisiert. Bei jedem Neustart werden jetzt sowohl der Ethernet-Controller als auch das Loopback-Interface gestartet.

Als Quellen und Senken der Kommunikation werden in MEGACO *Terminations* als Abstraktion benutzt.

4.3.2 Das abstrakte Modell

MEGACO benötigt *Terminations* und *Contexts*. Das heißt, es muss eine Möglichkeit geben, diese abstrakten Elemente zu erzeugen, zu verwalten und zu beenden. *Terminations* müssen mit Hilfe eines *Contexts* in Beziehung miteinander stehen können (vgl. Kapitel 2.4.4).

4.3.2.1 Termination

Terminations repräsentieren die logischen und physischen Quellen und Senken einer Kommunikation. Sie stellen aus Sicht eines MGs die Ein- und Ausgänge an seinem Gateway dar. Die *Terminations* sind als C-Strukturen innerhalb des MGs realisiert. Sie beinhalten eine `TerminationID`, um verschiedene *Terminations* identifizieren zu können. Neben der `TerminationID` ist auch eine `ContextID` vorhanden, um anhand derer ermitteln zu können, welchem *Context* sie zugeordnet ist. Die Angabe der `ContextID` innerhalb der *Termination* ist zwar nicht in der MEGACO-Spezifikation vorgeschrieben, aber

sie verhindert ein unnötiges Durchsuchen aller *Context*-Instanzen, um festzustellen, in welcher sich eine *Termination* befindet.

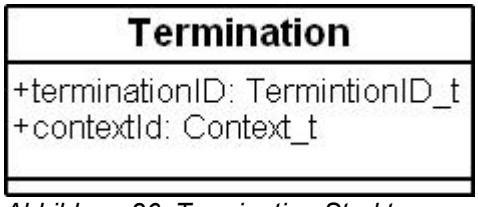

Abbildung 26: Termination Struktur

Die `TerminationID` ist ein 64 Bit lange Variable und wird in einem 8 Byte langen Datenvektor abgelegt. Der MG ist für die Erzeugung zuständig und innerhalb seines Gültigkeitsbereiches muss die ID eindeutig sein. Dazu gibt es eine separate Funktion (`mg_generate_terminationId`), die bei jedem Aufruf aus einem globalen `TerminationID`-Vektor eine `TerminationID` erzeugt und anschließend den globalen Wert inkrementiert. So kann bei einem erneuten Aufruf eine neue, eindeutige Identifikationsnummer generiert werden. Bei einem Überlauf der globalen Variable wird diese auf den initialen Wert (0_d) zurückgesetzt. Dieser Fall ist sehr unwahrscheinlich, weil zuvor 2^{64} *Terminations* erzeugt worden sein müssen. In Anbetracht der dynamisch erzeugten *Terminations* für RTP-Sessions ist das zwar möglich, aber dafür müsste der MG über einen sehr langen Zeitraum[30] im Betrieb sein.

Wird eine neue *Termination* erstellt (`mg_create_termination`), so wird eine Instanz aus der *Termination*-Struktur erzeugt, mit einer eindeutigen `TerminationID` versehen und eine Referenz auf die Instanz in einem Array abgespeichert. Die Größe dieses Arrays ist mit dem Makro `MAX_TERMINATIONS` definiert. Laut MEGACO-Spezifikation ist die maximale gleichzeitige Anzahl an *Terminations* eine Eigenschaften eines MGs, daher ist eine über ein Makro definiert Konstante, ein geeigneter Parameter. [RFC3525, S. 17]

Weitere Eigenschaften einer *Termination* müssen noch implementiert werden. Es fehlt zum Beispiel die genaue Art der Termination (RTP-Session, ISDN-Interface, etc). Um solche Informationen aufzunehmen, kann die *Termination*-Struktur nachträglich erweitert werden.

30 Bei 1000 neuen *Terminations* pro Sekunde würde der Überlauf nach erst nach ca. 600Mio Jahren erfolgen.

4.3.2.2 Context

Wie die *Termination*, wurde ein *Context* ebenfalls als Struktur implementiert. Enthalten sind die `ContextID`, je ein Konstrukt, um eine Sequenz von *Terminations* und Topologien aufnehmen zu können, eine optionale Variable für verschiedene Prioritätsstufen und eine ebenfalls optionale „emergency“ Variable, die einen Notruf anzeigen kann. [RFC3525, S. 17] Die in der *Context*-Struktur enthaltenen Variablen sind in Abbildung 27 dargestellt.

Werden *Contexts* erstellt (`mg_create_context`), residieren die Instanzen, analog zu den *Terminations*, in einem Array. Auch hier begrenzt die Größe des Arrays die maximale Anzahl an *Contexts*. Bei der Erstellung werden automatisch die `ContextIDs` angelegt.

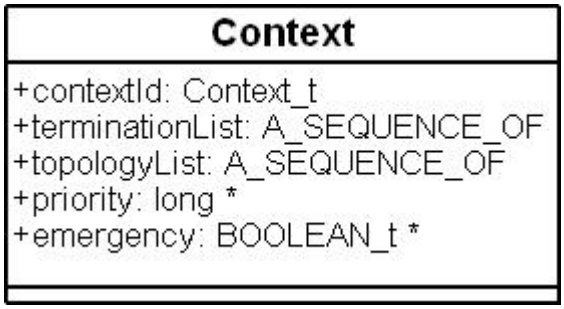

Abbildung 27: Context Struktur

Um der Anforderung gerecht zu werden, *Terminations* einem *Context* hinzuzufügen, existiert eine entsprechende Funktion (`mg_add_to_context`). Als Übergabeparameter werden eine `TerminationID` und eine `ContextID` verwendet. Zunächst muss die *Termination* zur dazugehörigen `TerminationID` ermittelt werden, insbesondere der Index des Arrays, in dem die *Termination* abgespeichert ist. Für das andere Array mit den *Context*-Instanzen, muss der Index nicht erst ermittelt werden, denn die `ContextID` ist der Index. Das ist möglich, weil eine `ContextID` lediglich 4 Bytes benötigt und damit als 4 Byte Integer implementiert ist. [RFC3525, S. 96]

Sind die benötigten Objekte ermittelt, so wird die *Termination* in die Sequenz-Struktur des *Contexts* aufgenommen.

Die Löschung einer *Termination* erfolgt analog, indem sie aus der Sequenz-Struktur gelöscht wird. Das hat außerdem zur Folge, dass die *Termination* zu dem *Null-Context* verschoben wird, um zu signalisieren, dass sie nicht mehr mit einer anderen *Termination* verbunden ist. Mit dem Entfernen der letzten *Termination* aus einem *Context*, wird der *Con-*

text ebenfalls entfernt.

Besagter *Null-Context* unterscheidet sich aus Sicht der Implementierung nicht von anderen *Contexts*, außer das er über die `ContextID` $0x00000000_h$ erreichbar ist. Aufgrund der automatischen Vergabe der `ContextIDs` muss in der Initialisierungsphase zuerst der *Null-Context* angelegt werden.
Das Hinzufügen oder Löschen einer *Termination* zu/aus einem *Context* in einem Media Gateway wird durch die Protokollnachrichten *Add*, *Subtract* oder *Move* ausgelöst.

4.3.3 MEGACO Protokollnachrichten

4.3.3.1 Aufbau

In der MEGACO-Spezifikation ist der Nachrichtenaufbau für die binäre Kodierung in der ASN.1 Syntax beschreiben. Die dort definierten Elemente sollen zunächst in eine für Media Gateways und Media Gateway Controller intern nutzbare Struktur gebracht werden. Die Generierung der C-Strukturen erleichtert der Einsatz eines Codegenerator-Werkzeugs. Es handelt sich dabei um den Open-Source ASN.1 Compiler für die Programmiersprache C: *asn1c*. Er kann auf Computern mit Linux als Betriebssystem installiert werden.

Der Zweck des ASN.1 Compilers besteht darin, eine Spezifikationen in ASN.1 Syntax in eine andere (Programmier-)Sprache zu konvertieren. In diesem Fall C oder C++. Der Compiler liest die Spezifikation ein und erzeugt eine Reihe von Strukturen (z. B. structs, unions, enums), die den ASN.1 Typen entsprechen. Ebenfalls sind die Umwandelung der Strukturen in einen übertragbaren Bytestrom und umgekehrt enthalten. Man spricht auch von Serialisierung und Deserialisierung. [WALK2006, S. 7]

Die Eingabedatei für den Compiler lässt sich aus dem *ANNEX A - Binary encoding of the protocol* ab Seite 90 [RFC3525] erstellen. Nach dem Aufruf des Compilers entstehen viele separate Headerdateien. Für jedes in der Spezifikation genannte Element je ein einzelner Header. Um die Übersichtlichkeit innerhalb des MEGACO-Projekts zu gewährleisten, wurden alle Einzeldateien in einem einzigen Header (`megaco_msg.h`) für die Nachrichten zusammengefasst. Bei der Gelegenheit wurden überflüssige Elemente entfernt. Unter anderem wurden die *Constraints*, also Elemente, um den Datenbereich einer Variablen einzuschränken und diese auf die Einhaltung der Grenzen zu testen, beseitigt. Zusätzlich wurden einige primitive Variablen den Eigenschaften des Hostsystems (NVTP1001) ange-

passt. In der Headerdatei befinden sich ebenfalls die im Implementations-Kapitel genannten Strukturen für *Contexts* und *Terminations*.

Ein Modul, das diesen Header inkludiert, ist in der Lage, alle Arten von MEGACO-Nachrichten in Form von Strukturen aufzusetzen. Ebenso werden empfangene Nachrichten in dieser Form aufbereitet. Die im Kapitel 2.4.5 vorgestellten Kommunikationsnachrichten sind hier vollständig wiederzufinden.

In der Spezifikation, und somit auch in dem Nachrichtenheader, taucht häufig das Prinzip einer Sequenz auf. Sequenz bedeutet hier, dass eine beliebige Anzahl von Elementen als Parameter eines anderen Elementes vorhanden ist. Um dem Prinzip einer Sequenz Rechnung zu tragen, gibt es eine Struktur (`A_SEQUENCE_OF`), die eine variable Anzahl an weiteren Strukturen aufnehmen kann. Sie enthält Parameter für die Anzahl und die Gesamtgröße der aktuellen Sequenz. Die Referenzen auf die beherbergten Strukturen sind in einem Array abgespeichert. Für den vereinfachten Umgang mit Sequenzen befinden sich in einem Hilfsmodul (megaco_utils.c) Funktionen zum Erstellen, Hinzufügen und Entfernen von Elementen. Optional kann der Parameter `free` als Funktionspointer benutzt werden. Er verweist auf eine Funktion, die aufgerufen werden soll, sobald ein Element aus der Liste entfernt wird.

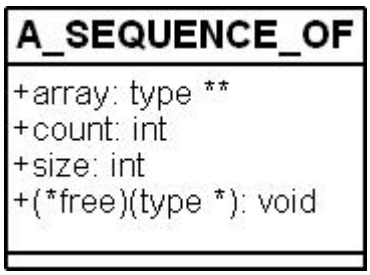

Abbildung 28: Sequenz-Struktur

(Die in Kapitel 4.3.2.2 vorgestellten *Contexts* enthalten eine Sequenz von *Terminations* und greifen daher auch auf die Hilfsstruktur der Sequenz zurück.)

Nach dem Erstellen der Nachrichten müssen diese kodiert werden, damit sie versendet werden können.

4.3.3.2 Kodierung

Bevor die Nachrichten, die in Form von Strukturen existieren, über ein Netzwerk übertragen werden können, müssen sie in einen Bytestrom umgewandelt werden. Die Prozedur, eine Nachricht von einem Media Gateway Controller zu einem Media Gateway oder umgekehrt zu senden, geschieht wie folgt: Zunächst wird die Nachricht aufgesetzt, das heißt Speicher wird für die Strukturen alloziert, die Komponenten werden initialisiert. Müssen dabei Sequenzen von MEGACO-Elementen erstellt werden, kann dazu die spezielle Sequenz-Struktur und deren Hilfsfunktionen benutzt werden. Danach muss die Nachricht serialisiert werden. Die dann in einem Datenpuffer vorliegende Nachricht kann mit den im Kapitel 4.3.1 vorgestellten Sende- und Empfangsfunktionen verschickt werden. Der Empfänger erhält die Nachricht und speichert sie ein einem Empfangspuffer. Der Inhalt des Puffers muss deserialisiert werden. Das bedeutet, es werden die benötigten Strukturen erzeugt und mit dem in der Nachricht enthaltenen Inhalt befüllt. Die Nachricht hat danach die gleiche Form, wie sie vor dem Serialisieren des Senders vorlag. Der Empfänger ist nun in der Lage die Nachricht auszuwerten und entsprechend zu reagieren. Die Abbildung 29 skizziert den Verlauf einer Nachrichtenübertragung.

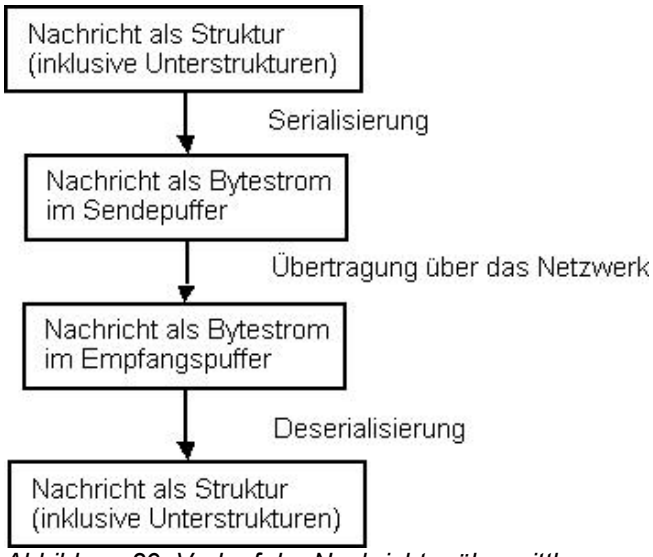

Abbildung 29: Verlauf der Nachrichtenübermittlung

Die Serialisierung und Deserialisierung sind in einem eigenen Modul (`megaco_msg_convert.c`) ausgelagert. Es enthält eine Sendefunktion, die den Zeiger auf eine Nachrichtenstruktur bekommt, die Nachricht zunächst serialisiert und anschließend sendet. Während des Serialisierens wird die Struktur systematisch durchlaufen und entweder werden die Parameter in einen globalen Puffer kopiert oder, falls sich in einer Struktur eine weitere Struktur befindet, wird eine Unterfunktionen aufgerufen. Für die unterschiedlichen

Strukturen gibt es entsprechende Unterfunktionen, die das betreffende Element serialisieren. Stößt der Serialisierungsprozess auf eine Sequenz, so muss die korrespondierende Unterfunktion zum Serialisieren entsprechend der Anzahl der Elemente in der Sequenz oft aufgerufen werden.
Die Deserialisierung folgt dem gleichen Prinzip, nur in umgekehrter Reihenfolge. Der Datenpuffer wird ausgelesen und anhand des Inhalts kann erkannt werden, um welche Strukturen (beziehungsweise bei einer Sequenz, um wie viele Elemente des jeweiligen Typs) es sich handelt. Wird eine spezielle Struktur während des Durchlaufens des Datenpuffers erkannt, so wird die entsprechende Unterfunktion zum Deserialisieren aufgerufen.
Wenn die Deserialisierung abgeschlossen ist und keine Fehler aufgetreten sind, dann ist die übertragene Nachricht in der gleichen Form, wie vor der Serialisierung zu finden. Die Empfangsfunktion gibt einen entsprechenden Zeiger auf die Nachrichten-Struktur zurück. Der Empfänger könnte nun seinerseits eine Antwortnachricht erstellen, die dann den gleichen Ablauf erfahren würde wie die vorherige, empfangene Nachricht.

Die zuvor genannten Serialisierung- und Deserialisierunsmechanismen sind im Zuge des Projekts entwickelt worden. Aufgrund der großen Menge an verschiedenen Nachrichten und deren darin enthaltenen Unterstrukturen, ist die Konvertierung nicht vollständig implementiert. Die Nachrichten sind soweit kodierbar, um einen groben Kommunikationsablauf zeigen zu können.

Es besteht auch die Möglichkeit, die Serialisierung und Deserialisierung durch den vom ASN.1 Compiler generierten Code übernehmen zu lassen. Das bedeutet, dass neben den Strukturen für die Nachrichten auch Funktionen zum Umwandeln der Strukturen in die *BER* Kodierung und umgekehrt generiert werden. Diese Funktionen konnten nicht in des MEGACO-Projekt eingebunden werden, weil nicht alle Fehler während des Compliervorgangs behoben werden konnten. Die Nutzung dieses Codes blieb daher erfolglos.

Der Funktionsumfang ist mit der jetzigen Implementierung noch nicht vollständig. Für die Weiterentwicklung hilft die Tatsache, dass das gesamte Kodierungsmodul austauschbar ist. So ist es möglich, die textbasierte Kodierung anstatt der binären zu nutzen, in dem man das Modul austauscht (oder beide Arten, wenn die Sende- und Empfangsfunktionen mit einem Flag ausgestattet werden, um die Kodierungsart zu wählen).

Mit dem Kodierungsmodul, in Verbindung mit dem lokalen Interface, sind die Instanzen MGC und MG in der Lage mit einander zu kommunizieren.

4.3.4 MGC und MG als Instanzen

4.3.4.1 Realisierung

Wie in dem Konzeptkapitel 3.5 erläutert, müssen auf der NVTP1001-Plattform sowohl ein Media Gateway als auch eine Media Gateway Controller als Instanzen vorhanden sein.

Der MGC ist über die Server-Tasks realisiert, das bedeutet, jedesmal wenn eine Anfrage eines MG an den MGC erfolgt, wird eine entsprechende Funktion im MGC-Modul (`megaco_mgc.c`) zur Behandlung aufgerufen, um so z. B. ein Abnehmen eines Hörers oder das Eintreffen einer SIP-Signalisierungsnachricht zu signalisieren. Damit kann der Mechanismus zum Aufbau eines Gespräches angestoßen werden.
Der MG ist als eigenständiger Task implementiert. Nachdem er gestartet wurde, nimmt er mit dem MGC Kontakt auf und versucht sich zu registrieren. Laut MEGACO-Spezifikation sind eine primäre IP-Adresse und eine oder mehrere sekundäre IP-Adressen vorgesehen. Die sekundären Adressen sollen benutzt werden, falls der MGC unter der primären Adresse nicht erreicht werden kann. [RFC 3525, S. 64] In dieser Implementation ist die primäre Adresse als Makro hart einkodiert (nämlich die eigene IP-Adresse). Ein alternativer MGC existiert in diesem Einsatzszenario nicht, deshalb sind weder eine sekundäre IP, noch ein Mechanismus, der die Erreichbarkeit überprüft und gegebenfalls eine sekundäre IP-Adresse auswählt, vorhanden.
Für einen künftigen, weitergehenden Einsatz in einem verteiltem Szenario (MGC und MGs befinden sich auf verschiedenen Plattformen) könnte man die primäre und die sekundäre(n) IP-Adresse(n) im persistenten Flash-Speicher des NVTP1001-Boards abspeichern.

Nach einer erfolgreichen Registrierung wartet der MG auf Anweisungen des MGCs. Damit zwischenzeitlich Ereignisse am MG detektiert werden können, wacht ein Hilfs-Task (`mg_event_listener`) über den Zustand des Gateways. Er ist in der Lage Variablen oder andere Elemente zyklisch zu überprüfen und damit auf Veränderungen zu reagieren. Tritt ein Ereignis ein, so kann der Hilfs-Task eine Nachricht an den MGC senden.

Bevor die MGC und MG Instanzen betriebsbereit sind, müssen sie initialisiert werden.

4.3.4.2 Initialisierung

Die Initialisierung ist der Ladevorgang, in dem Speicherplatz für Programmelemente angelegt und Variablen oder Puffer auf Startwerte gesetzt werden. Ist auf dem NVTP1001-Board der OK1 Kernel installiert, so wird in der Initialisierungsphase zunächst der EchtzeitMicro-Kernel gestartet. Dann können die Initialisierungsroutinen für die Applikationen aufgerufen werden. Die Instanzen MG und MGC werden gestartet. Zuletzt startet der Task-Scheduler, um die verschieden Tasks zu verwalten. [OK1, S. 10]

Der aus dem Modul `app.c` generierte Maschinencode, welcher standardmäßig beim Anstarten der NVTP1001-Plattform ausgeführt wird, dient als Einstiegspunkt der Initialisierung. Es werden Funktionen für das Aktivieren der Hard- und Firmware, sowie eine Funktion, die alle Angaben zur aktuellen Konfiguration und Version von Hard- und Software ausgibt, gestartet. Es folgt der Aufruf der MEGACO-Initialisierungsfunktion (`megaco_init`) aus dem Modul `megaco.c`.
Zuerst werden die TCP und UDP Server als Tasks, entsprechend ihrer Konfiguration, gestartet (vgl. Kapitel 4.3.1). Dabei dienen die globalen Makros `MG` und `MGC` als Steuerparameter. Nur wenn sie gesetzt sind, werden die jeweiligen Servertasks für MG oder MGC gestartet.

Als weiterer Teil der MEGACO-Initialisierungsroutine wird, falls der Parameter `MG` gesetzt ist, die Media Gateway Instanz initialisiert. Dazu muss der globale Datenvektor für die `TerminationID` (vgl. Kapitel 4.3.2) auf den Startwert (0_d) gesetzt werden. Dieser wird dann später inkrementiert, um eindeutige IDs zu erzeugen. Eine *Context*-Instanz für den *Null-Context* muss erzeugt werden. Da er als erster *Context* aufgerufen wird, erhält er automatisch die `ContextID` 0_d. Nun müssen alle physisch vorhandene *Terminations* erzeugt und konfiguriert werden. In diesem Fall eine für den B-Kanal des ISDN Interfaces. Die *Terminations* für die RTP-Sessions entstehen erst zur Laufzeit.
Ist die Initialisierung des MGs abgeschlossen, so wird ein neuer Task für den MG erzeugt und ebenso ein Hilfs-Task, der auf Ereignisse des Gateways reagieren kann. Die Initialisierungssequenz ist im Anhang, in Abbildung 31 als Flussdiagramm dargestellt.

4.3.4.3 Protokollfluss

Nach den Initialisierungen sind die MGC und MG Instanzen bereit für die Kommunikation. Die erste Handlung des MGs ist, sich bei einem MGC zu registrieren. Für die Registration gibt es keine spezielle Nachricht, sondern es wird die *ServiceChange* Message verwendet

(vgl. Kapitel 2.4.5). Diese Nachricht signalisiert normalerweise Änderungen einer oder mehrerer *Terminations*. Zum Beispiel kann ein Ereignis, dass eine *Termination* nicht mehr oder wieder verfügbar ist, mitgeteilt werden. Da bei dem Anmelden eines MGs, ein neuer Gateway samt seinen *Terminations* einem MEGACO-Netz hinzugefügt wird, kann man dies als das Verfügbarwerden aller Terminators an dem Gateway ansehen und daher ist die Verwendung der *ServiceChange* Nachricht nachvollziehbar.

Die *ServiceChange* Nachricht, die zur Registrierung benutzt wird, zeichnet sich durch einige spezielle Parameter aus. Zum einen ist die `TerminationID` auf $FFFFFFFFFFFFFFFF_h$ festgelegt. Diese ID bedeutet, dass nicht die *Terminations*, sondern der Media Gateway als Gesamtheit gemeint ist. Zum anderen wird als Grund "Service Restored" (`reason_code = 900`) in dem dafür vorgesehenen Feld angegeben.

Für jede Nachricht, insbesondere für eine Transaktion muss eine TransaktionsID vergeben werden. MEGACO arbeitet nach einem Anfrage-Antwort Schema, das heißt auf eine Anfrage wird eine Antwort zurückgeschickt. Die TransaktionsID ist für eine Anfrage und deren Antwort die gleiche. Für jede neue Transaktion wird die TransaktionsID inkrementiert. Prinzipiell können sowohl der MGC als auch der MG eine neue Transaktion anstoßen (MG: z. B. „Off-Hook" wird detektiert, MGC: z. B. Aufforderung eine bestimmte *Termination* in einen bestimmten *Context* zu setzen). Das bedeutet, beide Instanzen müssen die zuletzt verwendete TransaktionsID vorhalten und immer beim Senden und Empfangen inkrementieren.
Als Startwert soll für jede Kommunikation zwischen dem MGC und einem MG eine zufällige Zahl vergeben werden, damit die verschieden Kommunikationsbeziehungen zwischen einem MGC und mehrere MGs auch mittels der TransactionID besser unterschieden werden können. In dieser Implementation liegt nur eine Kommunikationsbeziehung vor und der Startwert ist als willkürliche Wahl (9000_d) fest einkodiert.

Nach Erhalt der Transaktion schickt der MGC seine Antwort auf die Registrierungs-Nachricht. Er hat nun die Gelegenheit über ein Modify Eigenschaften der *Terminations* zu ändern. Das ist allerdings noch nicht implementiert. Der Ablauf für die Registrierung ist in Abbildung 30 zusammengefasst.

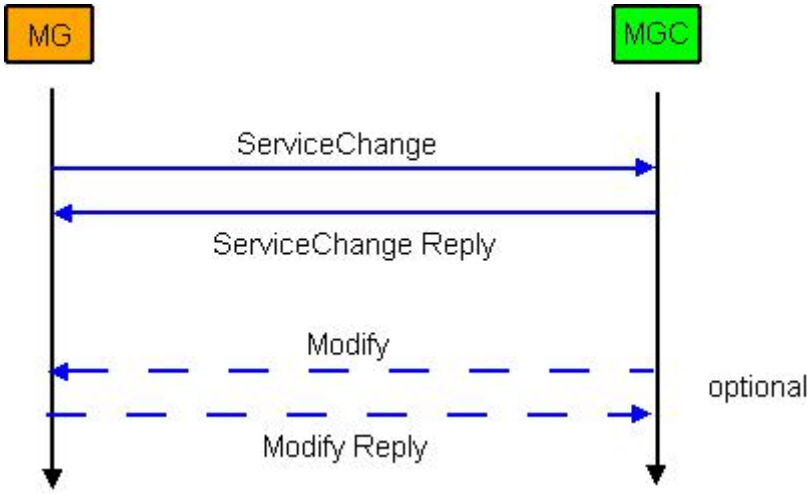

Abbildung 30: Ablauf der Registrierung eines MGs

4.4 Schwierigkeiten

Während der Implementierungsphase tauchten verschiedene Arten von Problemen auf. Es gab technische Schwierigkeiten mit dem Loopback-Interface und möglicherweise defekter Hardware. Bei der Implementation bereiteten die teilweise recht abstrakten Standards Mühe.

Nach der Implementation der Server für SIP, MEGACO und RTP stellte sich heraus, dass die Kommunikation mit TCP/UDP nicht über das Loopback-Interface funktionierte. Sowohl die vom TCP/UDP Stack ausgewertete Adresse 127.0.0.1[31], als auch die eigene IP-Adresse für das Ethernet-Interface zeigten keine Funktion. Eine Verbindung von einem externen Test-PC zu den Servern, ließ sich problemlos aufbauen. Sogar in umgekehrter Richtung - das heißt auf dem Test-Rechner lief eine Serverinstanz und die NVTP1001-Plattform versuchte darauf zu verbinden - gab es keine Probleme.
Der Grund für dieses Verhalten war eine fehlende Initialisierung des Loopback-Interfaces. Um dieses zu aktivieren, wurde eine neue Routine (`ifconfig_loopback.c`) geschrieben, die bis auf wenige Änderungen aus dem Quellcode zur Initialisierung des Ethernet-Interfaces besteht. Allerdings kann die Loopback-Adresse 127.0.0.1 immer noch nicht benutzt werden. Die Kommunikation über die eigene IP-Adresse funktioniert. Die genaue Ursache für dieses Phänomen ist unbekannt. Da die Verwendung der eigenen IP eine problemlose Fortführung des MEGACO-Projekts ermöglicht, soll dies nicht näher untersucht werden.
Die Benutzung der „echten“ Loopback-Adresse könnte allerdings einen Performancevor-

31 Für IP (in der Version 4) fest definierte Adresse für den Gebrauch als Loopback-Adresse

teil bedeuten, weil der Datenstrom nicht auf Hardwareebene von dem Sende- in den Empfangspuffer kopiert werden muss, so wie es bei der Verwendung der eigenen IP-Adresse der Fall ist. Dies kann als Optimierungsoption für eine künftige Weiterentwicklung gesehen werden.

Weitere Schwierigkeiten ergaben sich zu Beginn bei der Inbetriebnahme der Hardware. Das erste Problem war ein nicht funktionierendes Ethernet-Interface. Es konnte also auch kein neu entwickelter und compilierter Code auf das NVTP1001-Board geladen und anschließend ausgeführt werden. Das zweite Problem war, dass das System gelegentlich den Bootvorgang nicht ordnungsgemäß durchführte. Die Debug-Ausgaben deuteten auf eine fehlerhafte Initialisierung des PCI-Bus hin. Ohne die Probleme näher einzugrenzen, fand die Entwicklung auf Ersatzhardware statt.

Neben den technischen Problemen, erwies sich die Umsetzung der MEGACO-Spezifikation als Herausforderung. Die Nachrichten werden in den abstrakten Formen der ASN.1- und ABNF-Schreibweise notiert. Aus Sicht eines Implementierers ist eine direkte Kodierung (wie z. B. Der RTP-Header in Abbildung 2) leichter in Nachrichten umzuwandeln. Ohne den Einsatz des ASN.1 Compilers hätte sich die Implementation der Nachrichten zu einem sehr zeitaufwendigem Modul entwickelt. Aber auch mit seinem Einsatz war die Anpassung des Headers (`megaco_msg.h`), der ca. 1500 Zeilen Quellcode (inklusive Leerzeilen für Formatierungen und Kommentare) umfasst, eine langwierige Arbeit.

Der überwiegende Teil der Spezifikation ist nur in Textform beschrieben. So ist zum Beispiel keine Skizze für einen Protokollautomaten vorhanden. Die Elemente *Context* und *Termination* sind ebenfalls nur im Text beschrieben. Wären beide Elemente wenigstes in ASN.1 und ABNF beschrieben (auch wenn sie nicht Element der Nachrichtenkodierung sind), wären einige Sachzusammenhänge sicher klarer gewesen.

Ein weitere Schwierigkeit bestand in einer bestimmten Art von Programmierfehlern, genauer der Konsequenz, die zur Laufzeit auftritt. Es handelt sich dabei um Pointer, die auf einen nicht initialisierten Speicherbereich zeigen. Wird auf diese im Programmablauf zugegriffen, könnte das zu einer sogenannten Exception[32] führen, welche dann den Task, in dem ein Fehler ausgelöst wurde, beendet.
Das eigentliche Problem ist weniger der Fehler an sich, sondern vielmehr die Art, ihn einzukreisen und zu finden. Möchte man bestimmte Codesegmente auskommentieren, um

32 Fehlerfall

den Fehler einzugrenzen, bedeutet das auch, dass der Compiler einen um einige Zeilen gekürzten Maschinencode erzeugt. Dadurch, dass der Maschinencode „verschoben“ wird, befindet sich an der Speicheradresse hinter dem falsch initialisierten Pointer ein anderer Code. Abhängig davon, was für ein (undefinierter) Code sich dahinter verbirgt, wird eine Exception ausgelöst oder nicht. Es kann also passieren, dass durch das alleinige Auskommentieren einer unproblematischen Coderegion keine Exception mehr ausgelöst wird. In einem besonders schwerwiegendem Fall half die JTAG[33]-Schnittstelle des Prozessors auf dem NVTP1001-Board, um den Fehler zu finden.

Zum Abschluss soll noch eine konzeptionelle Schwierigkeit erwähnt werden. Wie in Kapitel 2.2.1 beschreiben, benötigt man für die RTP-Header Zeitstempel. Für die Zeitstempel wiederum benötigt man eine Echtzeituhr. Solch eine Uhr ist auf dem NVTP1001 System nicht vorhanden. Ein Lösung für dieses Problem könnte die Verwendung der internen Timer (die von dem OK1 Kernel verwaltet werden) sein. Die Auflösung von einer hundertstel Sekunde eines Timers ist für die Anforderungen der RTP-Spezifikation hinreichend. Da das RTP-Modul nicht im Implementationsumfang enthalten ist, wurde eine Echtzeituhr auch nicht implementiert.

33 Standardisierte Möglichkeit zum Testen und Debuggen auf elektronischer Hardware. (IEEE 1149.1)

5 Tests

Das in Kapitel 3.2 vorgestellte Szenario, kann aufgrund der Tatsache, dass nicht alle Komponenten für den Trunking Gateway implementiert sind, nicht als Gesamtheit getestet werden. Allerdings kann man die Untermodule einzeln testen.

5.1 Das lokales TCP/ IP Interface

Dieses Modul enthält drei wesentliche Elemente, die benötigt werden um die volle Funktionalität zu gewährleisten. Erstens muss ein Server bereitstehen, auf den sich Clients verbinden können, zweitens muss die Fähigkeit vorhanden sein, sich mit einem bestimmten Server verbinden zu können, und drittens muss man über das lokale Loopback-Interface eine Verbindung zu einem lokalen Server aufbauen können.

Die ersten beiden Testfälle wurden mit einem in der Vorlesung „Verteilte und parallele Systeme“ vorgestellten Programm getestet. Das Programm stellt eine Möglichkeit dar, einen Client und einen Server zu erstellen, die einfache, über die Tastatur eingelesene, Textnachrichten versenden.

Nachdem die Server auf der NVTP1001-Plattform initialisiert sind, kann man mit dem Clientprogramm (`udp_client.c` oder `tcp_client.c`) eine Verbindung zu dem Server aufbauen. Solange die Debug-Ausgaben für MEGACO aktiviert sind, kann man erkennen, dass eine eingetippte Testnachricht, wie zum Beispiel „Test123“ erscheint. Die Debug-Ausgaben werden über die serielle Schnittstelle des NVTP1001-Boards ausgelesen. Die Server können sowohl mit TCP, als auch mit UDP getestet werden. Abbildung 32 im Anhang zeigt eine Erfolgreiche Testung des UDP und des TCP Servers.

Um die Fähigkeit zu überprüfen, ob sich das Modul mit anderen Servern in Verbindung setzen kann, wurde auf dem Test-PC das Server-Testprogramm gestartet. Nach der Initialisierung von MEGACO wird mit einer Testfunktion (`megaco_test_connection_test`) eine Verbindung zum Testserver (`udp_server.c` oder `tcp_server.c`) aufgebaut und anschließend die Testnachricht „Test123“ versendet. Das Testprogramm empfängt die Nachricht und gibt sie auf dem Terminalfenster aus. Es sind UDP oder TCP als Verbindungsart möglich. Der Verbindungstest für UDP ist in Abbildung 33 und der Test für TCP in Abbildung 34 im Anhang zu finden.

Um das Loopback Interface zu testen, existiert eine Funktion (`megaco_test_loopback`), die eine Verbindung zu einem eigenen Server aufnimmt und ebenfalls eine Testnachricht schickt. Die Bestätigung, ob die Nachricht erfolgreich angenommen wird, ist in der Debug-Ausgabe sichtbar. Die Debug-Ausgabe zu dem Test ist in Abbildung 35 im Anhang wiederzufinden.

5.2 Terminations und Contexts

Ob Terminations oder Contexts erfolgreich erstellt werden, kann mit Hilfe der Debug-Ausgaben nachverfolgt werden. Nachdem alle benötigten Terminations und Contexts erstellt sind, kann man eine Termination einem Context hinzufügen. Wenn dies erfolgreich funktioniert hat, kann man auf die Termination über den Context zugreifen. Das Auslesen der `TerminationID` zeigt die korrekte Aufnahme der Termination.

Der Testablauf beinhaltet das Erstellen eines *Contexts* und zweier *Terminations*, die dem *Context* hinzugefügt werden. Die `TerminationIDs` aller *Terminations* in dem *Context* werden ausgelesen. Danach wird erst die erste, dann die zweite Termination wieder herausgelöst. Jeweils nach dem Lösen wird überprüft, welche der *Terminations* sich noch in dem *Context* befindet (anhand der `TerminationID`). Der Testverlauf kann in Abbildung 36 im Anhang nachvollzogen werden.

5.3 Registrierungsablauf

Um den korrekten Ablauf der Registrierungssequenz zu überprüfen, muss keine besondere Testfunktion aufgerufen werden. Die Registrierung erfolgt automatisch während des Startvorgangs. Es müssen lediglich die Debug-Ausgaben aktiviert sein.

Wie in Abbildung 37 im Anhang dargestellt, zeigen die Debug-Ausgaben, dass der MG zunächst eine *ServiceChange* Nachricht schickt, um sich zu registrieren. Der MGC erkennt diese Nachricht und sendet ein entsprechendes *ServiceChange Relpy*. Zusätzlich sendet der MGC eine *Modify* Nachricht, allerdings ohne Konfigurationsoptionen, weil der MG zum jetzigen Stand der Implementation diese *Modify* Nachricht noch nicht interpretieren kann.

6 Schluss

6.1 Zusammenfassung

Gegenstand dieser Arbeit war die Implementation der Kernelemente des MEGACO-Protokolls mit dem Aspekt einen Trunking Gateway zu realisieren. Dies wurde auf spezieller Telekommunikationshardware getestet.

Die Behandlung der Grundlagen umfasste allgemeine Informationen über VoIP und relevante Protokolle, wie RTP, SIP und MEGACO. Für die Erstellung eines Konzepts und dessen Implementation war eine intensive Auseinandersetzung mit der MEGACO-Spezifikation notwendig. Dazu gehörte auch die Einarbeitung in die ASN.1 Syntax.
Erst wurde ein Konzept für die Softwareelemente und deren Zusammenspiel in einer Gesamtarchitektur vorgestellt. Anschließend wurden die Hardwarekomponenten ausgewählt, die für einen Trunking Gateway benötigt werden.
Die Implementation beinhaltet die Realisierung der Kommunikation, die Realisierung der abstrakten Elemente *Termination* und *Context*, die Realisierung der Serialisierung und Deserialisierung, die Behandlung von MG und MGC als Instanzen sowie einem groben Protokollablauf bei der Registrierung eines MGs bei einem MGC. Einzeltests der Module konnten die Funktionalität der implementierten Elemente von MEGACO zeigen.

6.2 Ausblick

Diese Bachlorarbeit kann als Basis für eine Implementation eines Trunking Gateways nach MEGACO betrachtet werden. Die implementierten Kernelemente müssen dann um einige Elemente erweitert werden.

Zunächst ist es wichtig, dass MEGACO in seiner Gesamtheit implementiert wird. Das beinhaltet den kompletten Protokollautomaten sowie die vollständige Serialisierung und Deserialisierung aller Kommunikationsnachrichten, auch in der textbasierten ABNF Syntax. Ebenso fehlen den *Terminations* noch Eigenschaften, insbesondere die genauen Parameter, die eine *Termination* beschreiben.
Die MEGACO-Spezifikation fordert für die Übertagung der Nachrichten über UDP einen Sicherungsmechanismus, der zur Zeit nicht implementiert ist, genauso wie das Verhalten im Fehlerfall (Failover).

Damit die NVTP1001-Plattform als funktionstüchtiger Trunking Gateway fungieren kann, ist die Implementation des RTP- und SIP-Protokollstacks notwendig. Möglicherweise könnte der Einsatz von Open-Source[34] Software hier hilfreich sein. Außerdem muss die Ansteuerung der ISDN B- und D-Kanäle realisiert werden.

Ist der Trunking Gateway vollständig implementiert, kann über weitergehende Anwendungen nachgedacht werden. Man kann zum Beispiel ein verteiltes Szenario verwirklichen. Das bedeutet, dass der Media Gateway Controller und der Media Gateway auf verschiedenen Telekommunikationsplattformen laufen können. Da die Kommunikation bereits über TCP/UDP abgehandelt wird, müsste man lediglich die entsprechenden IP-Adressen ändern oder einen Mechanismus entwickeln, der dynamisch die IP-Adresse eines MGCs für einen MG ermittelt.

Neben dem Trunking Gateway sind weitergehende Anwendungsszenarien wie zum Beispiel ein Interactive Voice Response Server denkbar.

34 Software mit öffentlich verfügbarem Quellcode. Sie ist häufig kostenfrei erhältlich.

Literaturverzeichnis

[BADA2005] Badach, Anatol: Voice over IP: Die Technik, 2. Auflage, Carl Hanser Verlag, München/Wien, 2005

[BLAC2000] Black, Uyless: Voice Over IP, Prentice Hall PTR, New Jersey, 2000

[DETK2002] Detken, Kai-Oliver: Echtzeitplattformen für das Internet, Addison-Wesley, München, 2002

[GURL2000] Gurle, David; Hersent, Olivier; Petit, Jean-Pierre: IP Telephony: Paket-based multimedia communications systems, Pearson Education Limited, Großbritannien, 2000

[KÖHL2002] Köhler, Rolf-Dieter: Voice over IP, 1. Auflage, mitp-Verlag, Bonn, 2002

[NÖLL2003] Nölle, Jochen: Voice over IP, VDE Verlag GmbH, Berlin/Offenbach, 2003

[NVTP1001] N.A.T. GmbH (Hrsg.): NVTP1001 - Technical Reference Manual V1.5, 2005, Download im Juli 2007 unter www.nateurope.com

[OK1] N.A.T. GmbH (Hrsg.): N.A.T. OK1 Reference Maunual, Version 1.1, o. V., o. O., 2006

[RADVISION] Radvision (Hrsg.): Implementing Media Gateway Control Protocols – A RADVISION White Paper, 1/2002, Download im August 2007 unter http://radvision.com/

[RFC3525] Groves, C. et al.: Gateway Control Protocol Version 1, Network Working Group, RFC 3525, Internet Engineering Task Force, 6/2003

[RFC3550] Schulzrinne, H. et al.: RTP: A Transport Protocol for Real-Time Applications, RFC 3550, Audio-Video Transport Working Group, Internet Engineering Task Force, 7/2003

[TANE2003] Tanenbaum, Andrew S.: Computernetzwerke, 4. überarbeitete Auflage, Pearson Studium, München, 2003

[WALK2006] Walkin, Lev: Using the Open Source ASN.1 Compiler, 18/2006, Download im Juli 2007 unter http://lionet.info/asn1c/documentation.html

[X.680] ITU-T Recommendation X.680 (1997), Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation.

[X.690] ITU-T Recommendation X.690 (1997), Information Technology - ASN.1 Encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER).

Anhang

A1 Flussdiagramm der Initialisierung

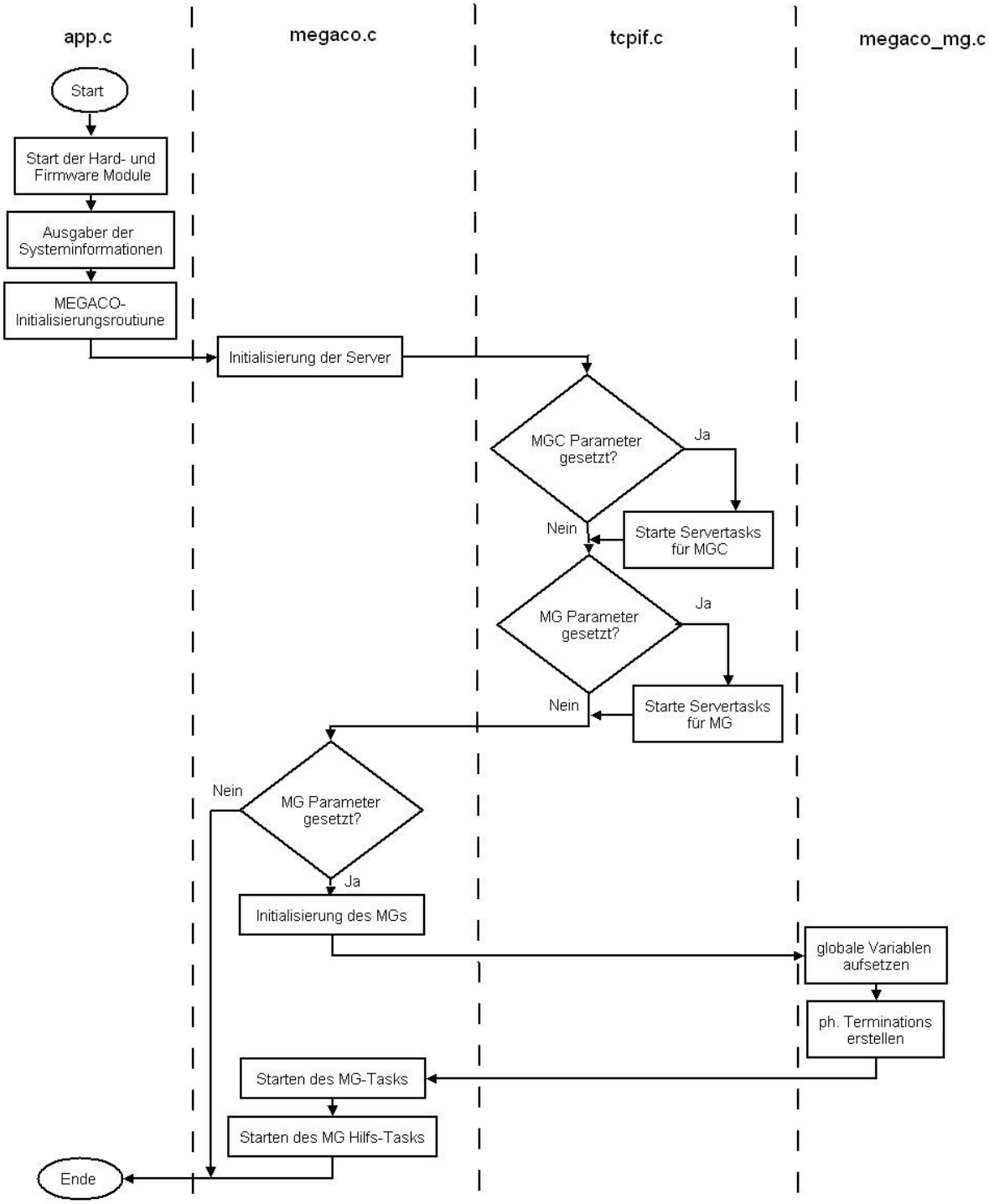

Abbildung 31: Ablauf der Initialisierung

A2 Testresultate

Die relevanten Stellen sind rot hinterlegt.

Erklärung zu Abbildung 32:

1.) Senden der Nachricht *Test123* mittels UDP
2.) Empfang der Nachricht *Test123* am UDP-Server
3.) Senden der Nachricht *Test123* mittels TCP
4.) Empfang der Nachricht *Test123* am TCP-Server

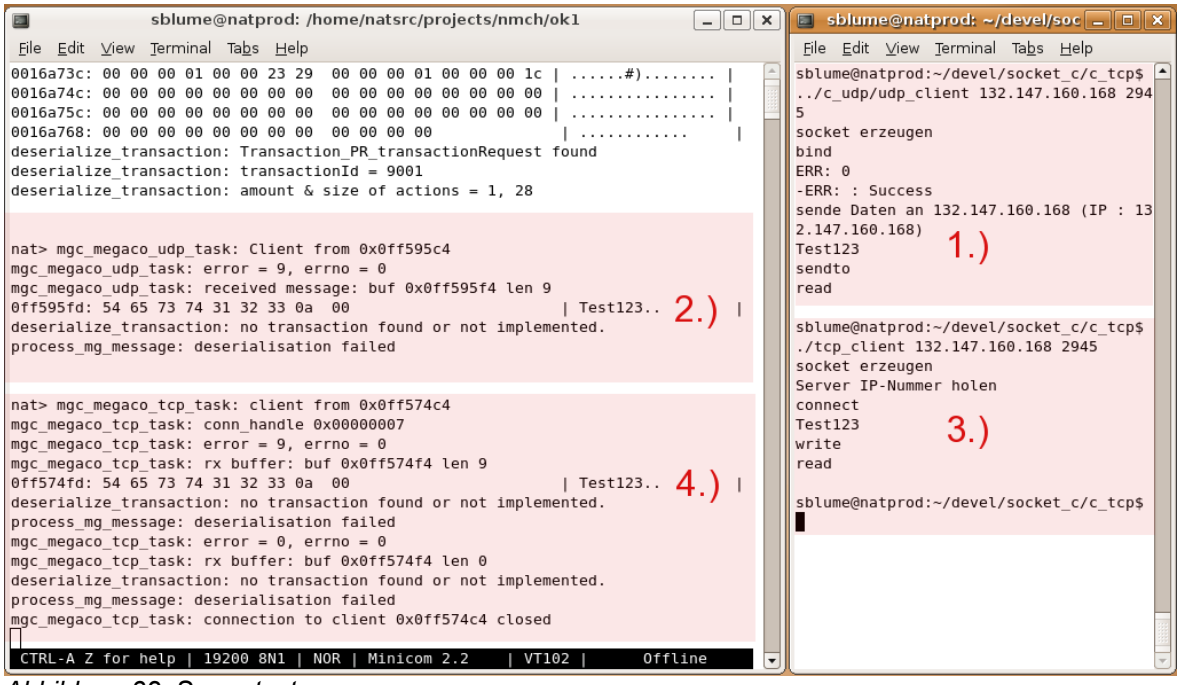

Abbildung 32: Servertest

Erklärung zu Abbildung 33:

1.) Senden der Nachricht *Test123* an Testserverprogramm mittels UDP
2.) Empfang der Nachricht *Test123* am Testserver

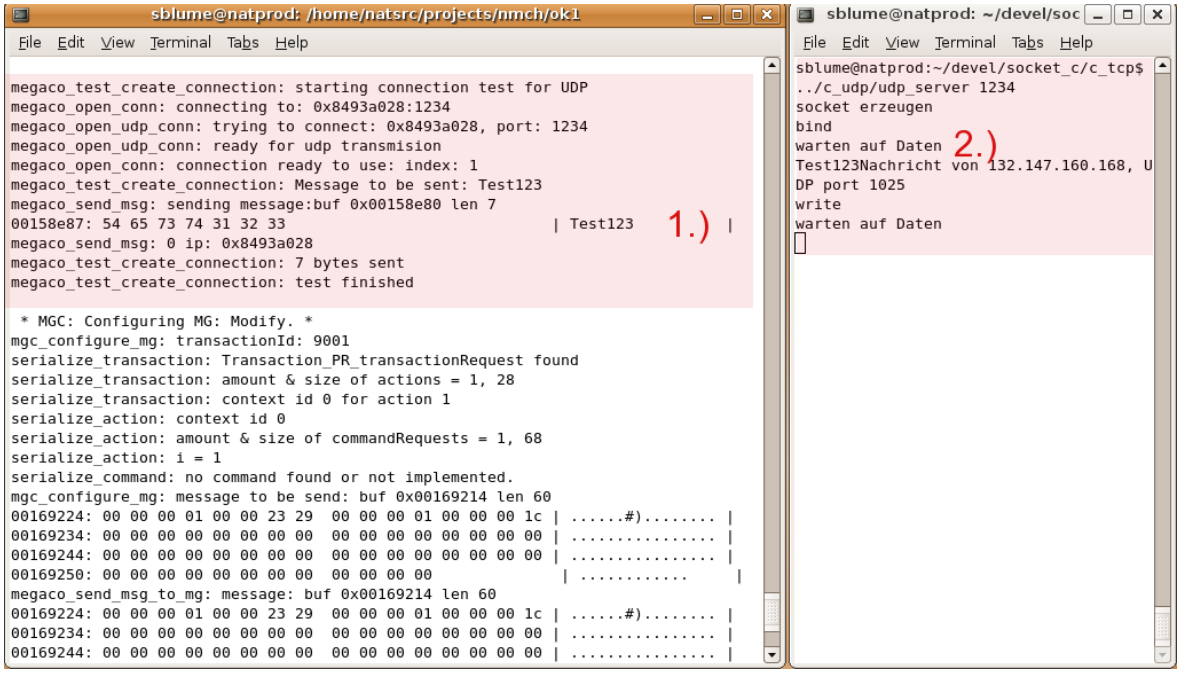

Abbildung 33: Verbindungstest zu UDP-Server

Erklärung zu Abbildung 34:

1.) Beginn der Testroutine, die aber von dem Task-Scheduler unterbrochen wird

2.) Senden der Nachricht *Test123* an Testserverprogramm mittels TCP

3.) Empfang der Nachricht *Test123* am Testserver

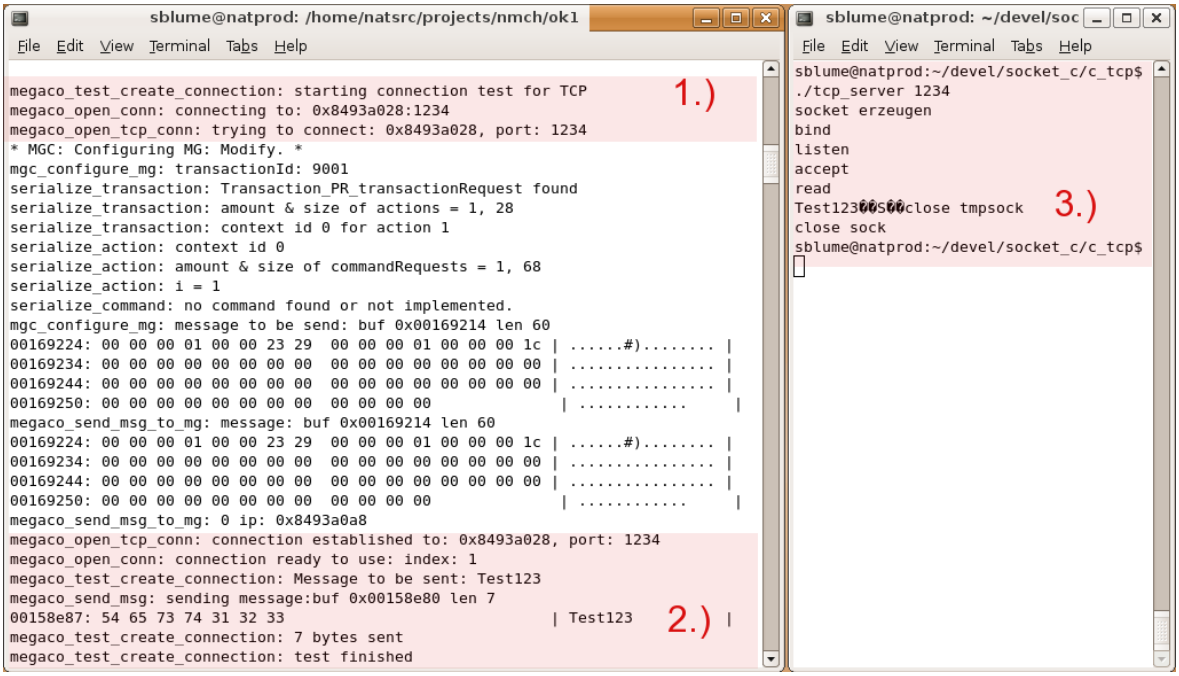

Abbildung 34: Verbindungstest zu TCP-Server

Erklärung zu Abbildung 35:

1.) Senden der Nachricht *Test123* mittels UDP an eigene Adresse
2.) Empfang der Nachricht Test *123* am eigenen UDP-Server
3.) Senden der Nachricht *Test123* mittels TCP an eigene Adresse
4.) Empfang der Nachricht Test *123* am eigenen TCP-Server

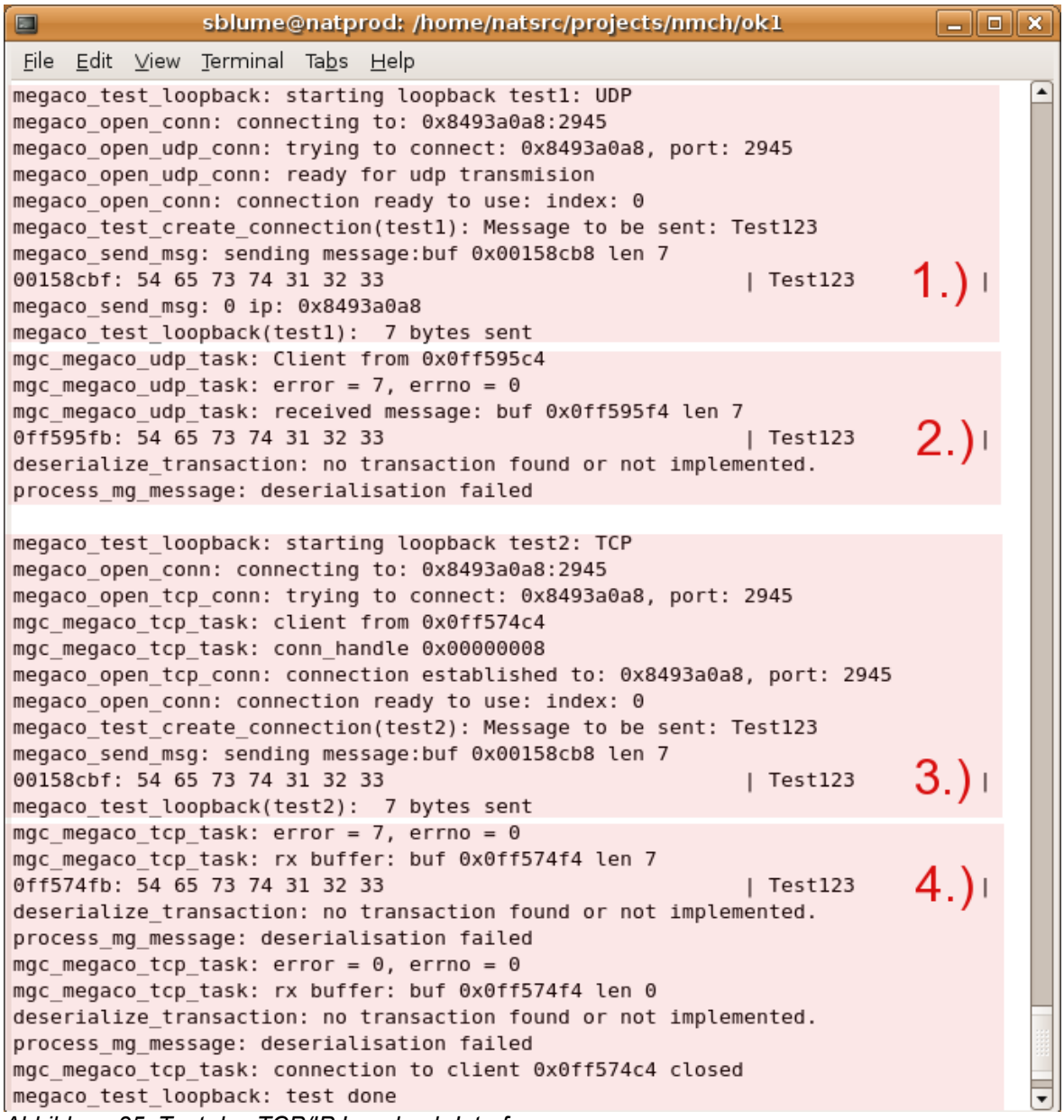

Abbildung 35: Test des TCP/IP Loopback-Interfaces

Erklärung zu Abbildung 36:

1.) *Termination* 1 wird erzeugt (`TermiationID` 0x0000000000000003)
2.) *Termination* 1 wird zum Test-*Context* hinzugefügt

3.) *Termination* 2 wird erzeugt (`TermiationID` 0x0000000000000004)

4.) *Termination* 2 wird zum Test-*Context* hinzugefügt

5.) Inhalt des *Contexts* wird überprüft: es sind sind zwei *Terminations* enthalten (mit `TermiationID` 0x0000000000000003 und 0x0000000000000004)

6.) *Termination* 2 wird aus *Context* gelöscht

7.) nur noch ein *Termination* ist vorhanden (0x0000000000000003)

8.) *Termination* 1 wird aus *Context* gelöscht

9.) es sind keine *Terminations* in dem *Context* mehr vorhanden

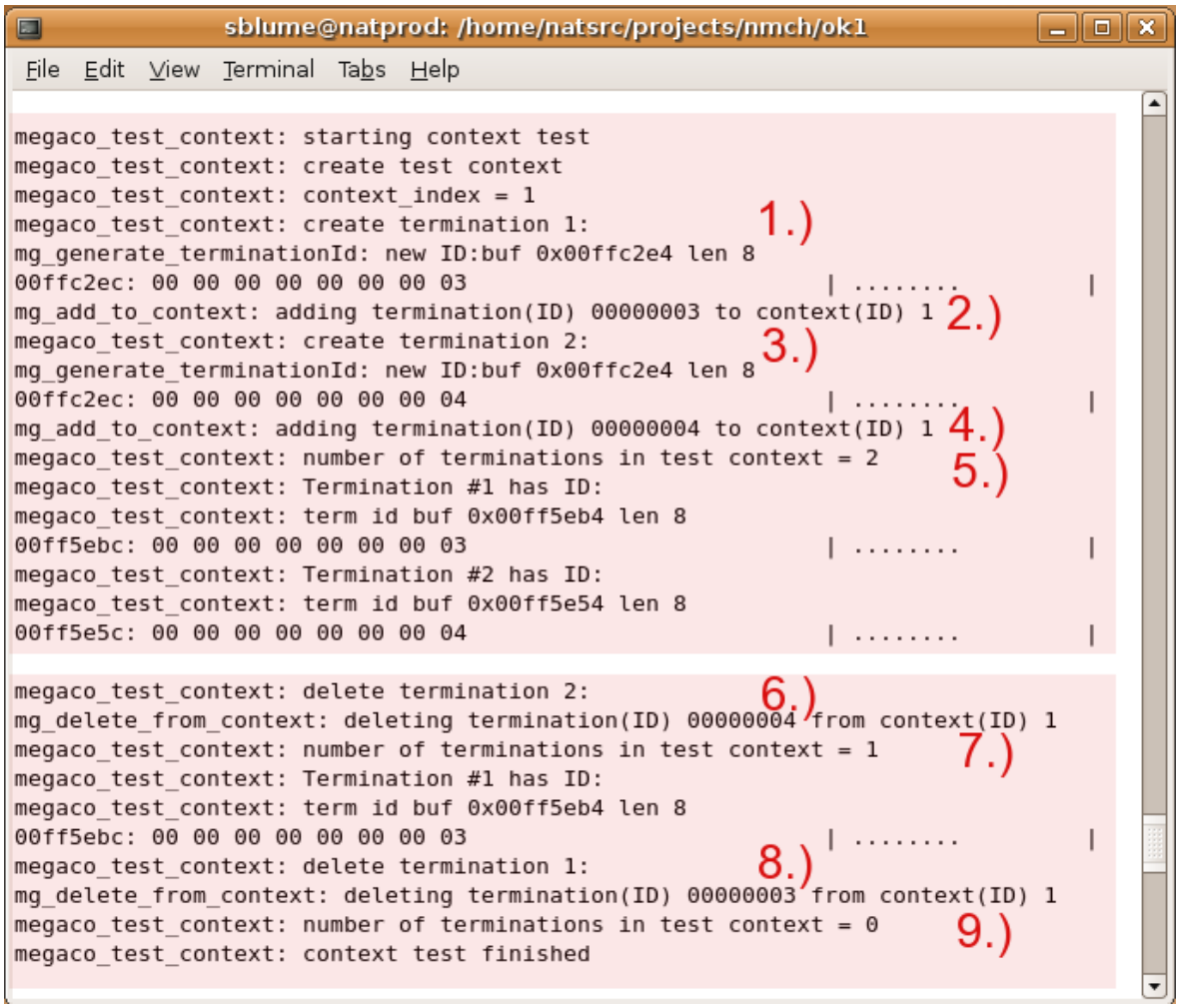

Abbildung 36: Test von Context und Terminations

Erklärung zu Abbildung 37:

1.) der MG schickt eine *ServiceChange* Nachricht, um sich zu registrieren

2.) der MGC erhält die Nachricht

3.) der MGC antwortet auf die Anfrage

4.) der MG erhält die Antwort

5.) der MGC versendet eine *Modify* Nachricht (allerdings noch ohne Parameter zur Konfiguration des MGs)

6.) der MG erhält die Nachricht, kann sie aber nicht auswerten, weil die Behandlung der Nachricht noch nicht implementiert ist

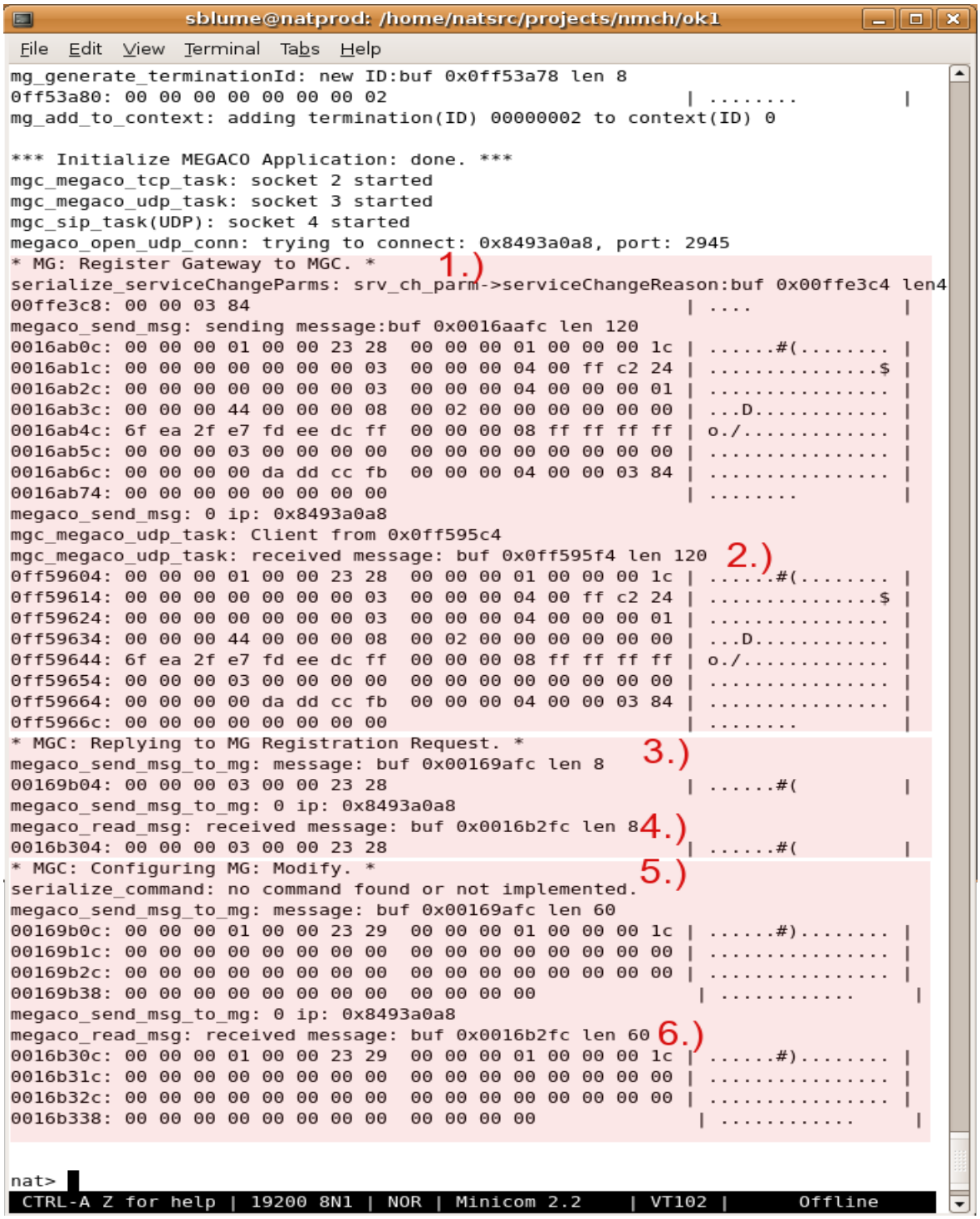

Abbildung 37: Test des Registriegungsablaufs

A3 Zuordnung der Softwaremodule zu den Quellcode Dateien

Die folgenden Quellcode Dateien wurden im Rahmen dieser Bachelorarbeit erstellt beziehungsweise angepasst:

Startpunkt der MEGACO-Applikation:

`megaco.c`

`megaco.h`

Media Gateway und Media Gateway Controller:

`megaco_mg.c`

`megaco_mg.h`

`megaco_mgc.c`

`megaco_mgc.h`

Nachrichtenstrukturen sowie Serialisierung und Deserialisierung:

`megaco_msg_convert.c`

`megaco_msg_convert.h`

`megaco_msg.h`

Lokales TCP/IP Interface:

`ifconfig_loopback.c`

`tcpif.c`

`tcpif.h`

Hilfsmodul:

`megaco_utils.c`

`megaco_utils.h`

Einsprungspunkt für MEGACO:

`app.c`

Globale Konfiguration:

`project.h`

Versions- und Konfigurationsinformationen:

`megaco_rel.c`

Testmodul:

`megaco_test.c`

`megaco_test.h`

Diese Dateien wurden auf einer CD zusammengestellt. Die CD wurde für die Bewertung der Bachelorarbeit zusammen mit dieser beim Prüfungsausschuss eingereicht. Aus recht-

lichen Gründen ist eine darüber hinausgehende Verwendung oder Veröffentlichung des auf der CD befindlichen Quellcodes nur nach einer vorhergehenden, schriftlichen Zustimmung durch die Firma N.A.T. GmbH zulässig.

Erklärung

Ich versichere an Eides statt, die von mir vorgelegte Arbeit selbständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeiten benutzt habe, sind angegeben.

Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keinem anderen Professor und auch keiner anderen Prüfungsbehörde vorgelegen.

Datum und Unterschrift des Studierenden: _____