



Fachhochschule
Frankfurt am Main
University of
Applied Sciences

Optimierung eines Media Servers auf Basis der Open Source Lösung GStreamer zur Bereitstellung von Audio/Videokonferenzen

Verfasser:

Michael NIEMAND

Matrikel-Nr.:

837778

Abgabedatum:

31.07.2012

Referent:

Prof. Dr.-Ing. Ulrich TRICK

Koreferent:

Prof. Dr. Manfred JUNGKE

22. Juli 2012

Erklärung

Hiermit erkläre ich, dass die Diplomarbeit von mir eigenhändig, selbstständig und nur mit der angegebenen Literatur verfasst wurde. Andere Hilfsmittel, die zum Einsatz gekommen sind, wurden explizit erwähnt. Aus der angegebenen Literatur übernommene Textpassagen wurden gekennzeichnet. Weiterhin wurde diese Arbeit in dieser oder ähnlicher Form keiner anderen Prüfungsbehörde, die für die Vergabe eines akademischen Grades zuständig ist, vorgelegt.

Frankfurt am Main, den 22. Juli 2012

(Unterschrift des Verfassers)

Ich danke meiner Familie, ohne deren Unterstützung mein Studium nicht möglich gewesen wäre, dabei insbesondere meiner Mutter für ihre finanzielle Unterstützung und ihr Lektorat, meinem Vater für seine ansteckende Begeisterung für alles ingenieurmäßige und meinem Bruder, der mich überhaupt erst auf die Idee für diesen Studiengang gebracht hat.

Mein Dank geht darüber hinaus an meinen Betreuer Armin Lehmann, der immer für meine Fragen Zeit hatte, und meinen sehr guten Freund Oleg Klassen für die Gestaltung des Covers.

Inhaltsverzeichnis

1	Einleitung	5
2	Theoretische Grundlagen	7
2.1	SIP	7
2.1.1	SIP-Nachrichten	8
2.1.2	SIP-Transaktionen	10
2.1.3	SIP-Netzelemente	11
2.2	SDP	11
2.3	RTP	12
2.3.1	RTP-Headerfelder	13
2.3.2	RTCP (Real-Time Transport Control Protocol)	14
2.4	UDP	15
2.5	GStreamer	15
2.5.1	GStreamer Werkzeuge	15
2.5.2	GStreamer Elemente	16
2.6	Digitales Audio	19
2.6.1	Abtastung	19
2.6.2	Quantisierung	19
2.6.3	Codierung	20
2.7	Digitales Video	21
2.7.1	Abtastung/Rasterung	21
2.7.2	Quantisierung	21
2.7.3	Codierung	22
2.8	Java	22
2.8.1	Objektorientierung	22
2.8.2	Virtuelle Maschine	23
2.8.3	Errorhandling	23
2.8.4	Events	24
2.8.5	Multithreading	24
2.8.6	Config Files	25
2.8.7	Interfaces	25
2.8.8	JAIN-SIP	26
2.8.9	GStreamer-Java	26
2.9	GNU/Linux	26
2.9.1	Philosophie	26
2.9.2	Lizenz	28

3 Anforderungsanalyse	29
3.1 Einleitung	29
3.1.1 Zweck	29
3.1.2 Umfang	29
3.1.3 Erläuterungen zu Begriffen und Abkürzungen	29
3.1.4 Verweise auf sonstige Ressourcen oder Quellen	30
3.1.5 Übersicht	31
3.2 Allgemeine Beschreibung	31
3.2.1 Produktperspektive	32
3.2.2 Produktfunktionen und Anwendungsfälle	33
3.2.3 Benutzermerkmale	34
3.2.4 Annahmen und Abhängigkeiten	34
3.3 Spezifische Anforderungen	34
3.3.1 Funktionale Anforderungen	35
3.3.2 Nicht-funktionale Anforderungen	36
3.3.3 Sonstige Anforderungen	37
4 Realisierung	38
4.1 Entwicklungsumgebung	38
4.2 Projektierung	38
4.3 Vorarbeiten	39
4.3.1 GStreamer	39
4.4 Entwicklung	45
4.4.1 Vorüberlegungen	45
4.4.2 Klassen	51
4.4.3 Erstmals etablierte Videokonferenz	72
4.4.4 Entfernen von Teilnehmern	72
4.4.5 Gesamtfunktionstest	72
5 Fazit und Ausblick	74
5.1 Fazit	74
5.1.1 Erfüllung der Anforderungsanalyse	74
5.1.2 Persönliche Erfahrungen	75
5.2 Ausblick	75
5.2.1 Interne Verbesserungen	75
5.2.2 Weitere Features	77

Kapitel 1

Einleitung

„Man kann nicht nicht kommunizieren“ - Paul Watzlawick

Die Menschen haben eine Sprache entwickelt, die es ihnen ermöglicht, komplexe Sachverhalte zu beschreiben und sich auszutauschen. Durch die Erfindung der Schrift gelingt es sogar, diese Informationen zu speichern und später wieder abzurufen. Mit Sprache und Schrift hat der Mensch es geschafft, Geschichte und Geschichten zu schreiben und so Tatsächliches oder Erdachtes festzuhalten und anderen Menschen an anderen Orten, ja sogar anderen Zeiten mitzuteilen. Die Menschen können nach knapp 2 Jahrhunderten lesen, wie Napoleon Bonaparte zum Kaiser der Franzosen wurde und Professoren können lernwilligen Studierenden erklären, wie ein Verbrennungsmotor funktioniert.

Und doch: Immer wieder gibt es Missverständnisse. Staatsmänner geraten in Streit, ebenso wie Frau und Mann. Nicht zuletzt auf Grund von Kommunikationsproblemen.

Kommunikation besteht eben aus mehr als Worten, aus mehr als dem, was gesagt oder geschrieben wird. Oft reicht der Blick einer Person oder eine Geste und das Gegenüber versteht, ohne dass auch nur ein einziges Wort gewechselt wird. Nonverbale Kommunikation stellt einen nicht unerheblichen Teil menschlicher Kommunikation dar.

Gerade der Teil der Kommunikation, der zwischen den Zeilen geschieht, der nicht mit Worten ausgedrückt werden *kann*, der Teil, der im Menschen berührt, was über die reine Kognition hinausgeht, findet über Mimik und Gestik statt.

Mit den Mitteln der frühen elektrischen Fernkommunikation, angefangen im 18. Jahrhundert mit dem Telegraphen, über das Telefon im 19. bis zur Erfindung der Email im 20. Jahrhundert, können diese wichtigen Informationen nicht oder nur sehr begrenzt (etwa über den Tonfall beim Telefon) übertragen werden. Ohne, dass es Manchem bewusst ist, wird ein großer Teil der Information verworfen.

Schon in frühen Visionen der Zukunft kam deswegen immer wieder die Übertragung von Bewegtbildern des Ferngesprächspartners vor, wie etwa in dieser Illustration aus dem Jahre 1922 von Frank R. Paul (Abbildung 1.1), die das Leben im Jahre 1972 zeigen soll.



Abbildung 1.1: Vision der Zukunft aus dem Jahr 1922

1964 bringt AT&T das erste Videotelefon auf den Markt, das Interesse ist jedoch verhalten. So bleibt diese Vision lange Zeit eine Vision [GAIJ01].

Dank der Fortschritte in der Digitalisierung von Mediendaten und der immer schneller werdenden Internetverbindungen sind Übertragungen von Bewegtbildern von einem Ort zu einem anderen auf der Erde in Lichtgeschwindigkeit heute kein Problem mehr. Preisgünstig und ohne spezielle Hardware kann heute jeder, der Zugriff auf einen Computer mit Internetzugang und einer Webcam hat, mit jedem anderen Menschen auf der Welt, der die gleichen Voraussetzungen erfüllt, diese Vision wahr werden lassen.

Proprietäre Anwendungen wie Skype haben schnell viele Nutzer gefunden. Sie sind einfach zu benutzen und ermöglichen einen komfortablen Zugang zur Welt der Videotelefonie. Nachteile dieser Software röhren aus ihrer geschlossenen Infrastruktur. Niemand, außer der Firma, die diese Software erstellt, weiß, was die Software genau macht. So kann auch niemand wissen, welche Informationen von Dritten gespeichert oder gar censiert werden. Um diese Nachteile zu vermeiden, kann Software entwickelt werden, die auf Standards basiert, also fest definierten Grundlagen, die für jeden Interessierten nachzulesen sind, oder gar als Open Source Software, deren Quelltext ebenso jedem Nutzer frei zur Verfügung steht.

Als Sonderfall von Videotelefonie gelten Videokonferenzen, die es ermöglichen, mehr als 2 Parteien in ein Gespräch zu involvieren. Videokonferenzen haben vielfältige Einsatzmöglichkeiten, etwa für international tätige Unternehmen, die Reisekosten sparen oder die Umwelt schonen wollen und ihre Meetings daher über derartige Systeme abhalten oder auch für Familien und Freunde, die miteinander reden und sich dabei sehen wollen, ohne physisch am gleichen Ort sein zu können.

Die Funktionsweise und die Erstellung eines Videokonferenzsystems, das nur auf Standards und Open Source-Software basiert, soll in dieser Diplomarbeit untersucht und dokumentiert werden.

Kapitel 2

Theoretische Grundlagen

In diesem Kapitel werden die theoretischen Grundlagen der Diplomarbeit behandelt. Es wird auf die zum Einsatz kommenden Netzwerkprotokolle und das verwendete Media-Framework GStreamer eingegangen sowie Aspekte, wie Digitalisierung von Audio- und Videosignalen, das Open Source Betriebssystem GNU/Linux und die Java-Plattform behandelt.

2.1 SIP

Im Internet existieren zahlreiche Anwendungen, die das Erstellen einer Sitzung erfordern, wobei der Begriff „Sitzung“ als Austausch von Daten zwischen zusammengehörigen Teilnehmern zu verstehen ist. Die Implementierung dieser Anwendungen wird erschwert, da Nutzer ihren Standort verändern, unter mehreren Pseudonymen erreichbar sein und auch mehrere verschiedene Medientypen austauschen können - oft sogar gleichzeitig. SIP (Session Initiation Protocol) ist ein Application Layer-Protokoll, um Sitzungen mit ein oder mehr Teilnehmern zu erstellen, zu modifizieren und zu beenden. In Abbildung 2.1 findet sich die Einordnung in das OSI-Referenzmodell (Open Systems Interconnection).

Es stehen zahlreiche Protokolle zur Verfügung, um verschiedene Medientypen, wie Sprache, Video oder Textnachrichten zu transportieren. SIP arbeitet mit diesen Protokollen zusammen, indem es Internet-Endpunkten (sog. User Agents) ermöglicht, sich zu finden und über den Charakter einer Sitzung, die sie zueinander aufbauen wollen, zu einigen.

Um dem Umstand der Standortveränderung der Nutzer Rechnung zu tragen, ermöglicht SIP darüber hinaus die Bereitstellung sog. Proxy/Registrar-Server, an die User Agents ihre Anfragen senden können. SIP arbeitet unabhängig von dem darunter liegenden Transportprotokoll und ist nicht an die Art von Sitzung gebunden, die damit erstellt wird [RFC3261].

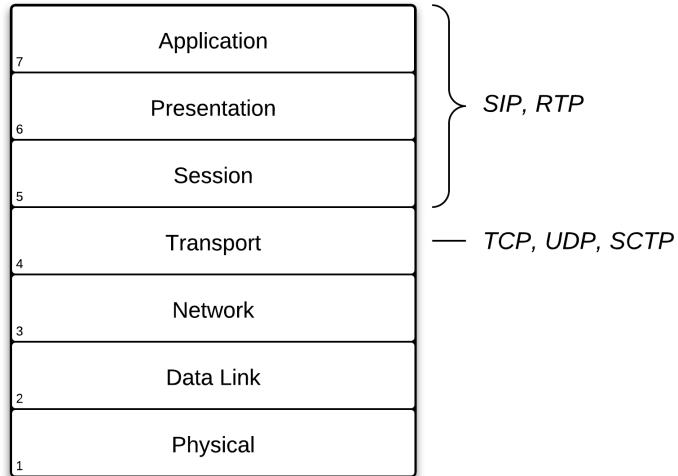


Abbildung 2.1: OSI-Referenzmodell

2.1.1 SIP-Nachrichten

Kommunikation mit SIP wird über SIP-Nachrichten eingeleitet. Alle Nachrichten in SIP sind in Anfragen (Requests) und Antworten (Responses) organisiert. Anfrage und Antwort wird Transaktion genannt [PERE08].

Nachrichten können unter anderem dazu dienen, eine Session auf- oder abzubauen, zu modifizieren oder Informationen über die aktuelle Session abzufragen. Am Anfang einer Session steht oft eine INVITE-Nachricht. Sie dient dem Austausch von Verbindungsinformationen (im Header der SIP-Nachricht) und dem Aushandeln von Medienparametern (im Body der SIP-Nachricht). Ein Beispiel für eine SIP-INVITE findet sich in Abbildung 2.2.

SIP-Requests

Es gibt 6 verschiedene Arten von Grundmethoden in SIP [RFC3261]:

- REGISTER
- INVITE
- ACK
- CANCEL
- BYE
- OPTIONS

In weiteren RFCs gibt es darüber hinaus noch die erweiterten Methoden

- REFER
- PRACK

- UPDATE
- INFO
- SUBSCRIBE
- MESSAGE
- NOTIFY
- PUBLISH

Im Folgenden werden die 6 Grundmethoden erläutert:

REGISTER Der Teilnehmer kann hierdurch eine Verknüpfung zwischen temporärer und ständiger SIP URI hinterlegen, um später durch andere Teilnehmer erreichbar zu sein. Der Teilnehmer sendet dazu initial und periodisch REGISTER-Nachrichten an den Registrar-Server. Der Registrar-Server speichert diese Informationen, die auch von einem etwaigen Proxy Server genutzt werden, in einer Datenbank, die auch Location Service genannt wird. Die Unterscheidung zwischen Registrar und Proxy Server ist logischer, nicht physischer Natur [RFC3261].

INVITE Eine SIP INVITE-Nachricht (engl. *to invite* - einladen) wird so genannt, weil damit der anfragende User Agent den Server bittet, den angefragten User Agent einzuladen. In der INVITE-Nachricht sind bestimmte Header-Felder enthalten, die Informationen über die Nachricht enthalten. Unter diesen Informationen befinden sich ein einmaliger Identifizierer für den Anruf (Call-ID), die Zieladresse (To) und die Adresse des initiiierenden User Agents (From) sowie Informationen über die Art von Verbindung, die aufgebaut werden soll (Content-Type) [RFC3261].

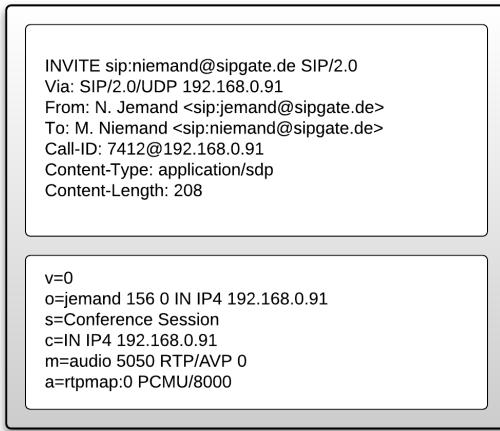


Abbildung 2.2: INVITE-Nachricht mit SDP-Body

ACK Damit eine Instanz weiß, ob eine finale Antwort auf eine INVITE-Nachricht angekommen ist, muss der Erhalt dieser finalen Antwort nochmals bestätigt werden. Dies geschieht durch die ACK-Nachricht. Dieses Verfahren wird 3-Wege-Handshake genannt [RFC3261].

CANCEL Ein Nutzer, der eine Anfrage zum Aufbau einer Sitzung stellt, kann diese auch abbrechen. Dieser Abbruch fordert den Server auf, keine weitere Verarbeitung durchzuführen, zum Zustand vor der Anfrage zurückzukehren und eine entsprechende Fehlermeldung für diese Transaktion zu generieren [RFC3261].

BYE Wenn eine Sitzung beendet wird, generiert der beendende User Agent als Erstes eine BYE-Nachricht, diese wird direkt zu dem anderen teilnehmenden User Agent weitergeleitet, der diese mit einem 200 OK bestätigt, ein ACK wird nicht gesendet [RFC3261].

OPTIONS Die OPTIONS-Anfrage erlaubt einem User Agent die Abfrage der Fähigkeiten eines Proxy Servers oder eines anderen User Agents. Das gestattet einem User Agent, Informationen über die erlaubten Anfragen, Inhaltstypen, Codecs usw. einer Gegenstelle zu erhalten, ohne diesen anzurufen [RFC3261].

Responses

SIP Responses sind als Antworten auf SIP-Requests zu verstehen. Sie geben Auskunft über den Zustand des Systems, an das eine Anfrage gestellt wird. So gibt es beispielsweise die Codes 200 für „OK“, also die erfolgreiche Beantwortung der Anfrage oder 486 für „BUSY HERE“, also die Ablehnung der Anfrage mangels freier Kapazitäten auf der angefragten Instanz. Die Responses sind in Klassen eingeteilt. In Tabelle 2.1 findet sich eine Auflistung der möglichen Klassen.

Tabelle 2.1: Klassen von Responses

Code	Art	Zweck, Bedeutung
1xx	Vorläufig	Anfrage empfangen, weitere Bearbeitung im Gange
2xx	Erfolg	Anfrage empfangen, verstanden und akzeptiert
3xx	Umleitung	Informationen über neuen Standort des Nutzers oder andere Dienste, um die Anfrage zu beantworten
4xx	Client-Fehler	Anfrage enthält falsche Syntax oder kann von dem Server nicht bearbeitet werden
5xx	Server-Fehler	Server kann die scheinbar gültige Anfrage nicht erfüllen
6xx	Globaler Fehler	Anfrage kann von keinem Server erfüllt werden

2.1.2 SIP-Transaktionen

Eine SIP-Transaktion besteht aus einer Anfrage und einer Antwort auf diese Anfrage, die keine bis mehrere vorläufige Antworten und eine finale Antwort enthält [RFC3261].

2.1.3 SIP-Netzelemente

User Agents

„Als User Agent (UA) wird diejenige Software- und/oder Hardware-Komponente bezeichnet, die das Endgerät für die SIP-basierte Kommunikation darstellt.“ [TRIC09]. Es wird hierbei noch zwischen User Agent Clients (UAC) und User Agent Servern (UAS) unterschieden, wobei der UAC Requests sendet und auf Responses wartet und der UAS auf Requests wartet und Responses sendet.

Proxy Server

Ein Proxy Server oder kurz Proxy dient im SIP-Umfeld der Vermittlung von SIP-Sessions. Man unterscheidet zwischen Stateful und Stateless Proxy Servern. Ein Stateful Proxy ist, bezogen auf die SIP-Transaktion, ein aktives SIP-Netzelement. Er kennt den Zustand (engl. *state*) einer SIP-Transaktion und kann deshalb erkennen, wenn etwa ein Paket mit einer Anfrage wiederholt verschickt wurde oder wenn eine Antwort ausbleibt. Ein Stateless Proxy leitet dagegen Pakete nur weiter und speichert keinerlei Transaktionsinformationen. Ein Stateless Proxy kann deswegen auch beispielsweise einen etwaigen Fehlerzustand nicht erkennen [TRIC09].

Conference Server

Ein SIP Conference Server ermöglicht den Aufbau einer Konferenz, also der Kommunikation zwischen mehr als 2 Teilnehmern. Eine SIP-Session wird jedoch immer mit einem Dialog, also der Kommunikation von 2 Instanzen, aufgebaut. Daher muss es ein Element geben, das die Konferenz zur Verfügung stellt. Dieses Element wird Conference Server genannt. Ein Conference Server in SIP ist dabei logisch in die Elemente Mixer und Focus unterteilt. Der Mixer übernimmt dabei die Verarbeitung, also hauptsächlich das Mischen, der Mediendaten. Der Focus ist eine Sonderform des User Agents und ist für die SIP-Kommunikation zuständig. Focus und Mixer können sowohl gepaart, als auch getrennt auftreten, wobei es in der Regel einen Focus und mehrere Mixer gibt [TRIC09].

Session Border Controller

Ein Session Border Controller (SBC) ist eine Spezialform des Back-to-Back User Agents. Er befindet sich in der Regel an der Nahtstelle zwischen 2 Netzen, etwa zwischen Kern- und Access-Netz. Die möglichen Aufgaben eines SBC können die Signalisierung und die transportierten Medien betreffen und sind sehr vielseitig. Unter anderem zählen Traffic Control für die Signalisierung, Authentifizierung und Zugriffskontrolle oder auch Codec-Aushandlung dazu [TRIC09].

2.2 SDP

SDP (Session Description Protocol) ist ein Protokoll, um Multimedia-Sessions mit dem Ziel des Session-Aufbaus oder der Session-Veröffentlichung zu beschreiben [RFC4566]. SDP wird in Zusammenhang mit anderen Protokollen zur Session-Initiierung verwendet. Neben SIP sind das

etwa H.232 oder SAP (Session Announcement Protocol). SDP hilft bei der Aushandlung von Codecs, implementiert jedoch selbst keinen Mechanismus zur Aushandlung. Ein Beispiel für eine SDP-Nachricht (im Body einer SIP-Nachricht) findet sich in Abbildung 2.2.

Eine SDP-Nachricht enthält folgende Felder:

- v= Protokollversion, nach aktuellem Standard [RFC4566] 0
- o= Sitzungsanbieter und -identifizierer
- s= Sitzungsname
- i= Sitzungsinformationen (optional)
- u= URI (optional)
- e= E-mail-Adresse (optional)
- p= Telefonnummer (optional)
- c= Verbindungsinformationen, nicht erforderlich, wenn in Medienbeschreibung enthalten (optional)
- b= Bandbreiteinformationen (optional)
- Ein oder mehrere Zeitinformationen wie folgt
 - t= Zeit, in der die Sitzung aktiv ist
 - r= Null oder mehr Wiederholungen (optional)
- z= Zeitzonenanpassungen (optional)
- k= Verschlüsselungsschlüssel (optional)
- a= Null oder mehr Zeilen Sitzungsattribute (optional)
- Null oder mehr Medienbeschreibungen wie folgt
 - m= Medienname und Transportadresse
 - i= Medientitel (optional)
 - c= Verbindungsinformationen (optional, wenn auf Sitzungsebene definiert)
 - b= Bandbreiteinformationen (optional)
 - k= Verschlüsselungsschlüssel (optional)
 - a= Null oder mehr Medienattributzeilen (optional)

2.3 RTP

RTP (Real-Time Transport Protocol) ist ein Netzwerk-Protokoll für den Transport von Echtzeitdaten, wie etwa Audio-, Video- oder auch Simulationsdaten über Multi- oder Unicast-Netzwerkdienste [RFC3550]. Es ist im OSI-Referenzmodell im Application Layer angesiedelt (Siehe Abbildung 2.1). Die Möglichkeit, mit RTP Multicast (d.h. die Übertragung von Nachrichten von

einem Punkt zu einer Gruppe) zu nutzen, ist sehr hilfreich, sollen die gleichen Daten an mehrere Empfänger geschickt werden, ohne dass sich die Bandbreite beim Sender erhöht. Die Pakete werden in dem Fall von den entsprechenden Netzwerkelementen vervielfältigt.

2.3.1 RTP-Headerfelder

Der RTP-Header hat eine variable Länge. Es gibt einen festen Teil (die ersten 96 Bit) und einen darauf folgenden, variablen Teil (0 bis 256 Bit). Die Länge des variablen Teils wird im festen Teil indirekt definiert. Der RTP-Header ist schematisch in Abbildung 2.3 abgebildet.

Anmerkung: In der Grafik ist der Payload angedeutet, dieser gehört nicht zum Header.

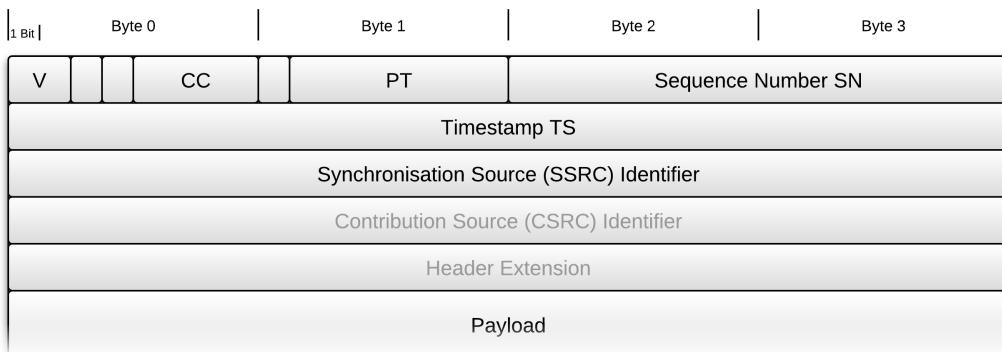


Abbildung 2.3: RTP-Header

Es folgt eine Beschreibung der Header-Felder.

Version (V), 2 bit Dieses Feld beinhaltet die Version des Protokolls, in der derzeitigen Spezifikation ([RFC3550]) ist dies eine 2.

Padding (P), 1 bit Wenn das Padding Bit gesetzt ist, enthält das Paket am Ende ein oder mehrere zusätzliche Oktette, die nicht zur Payload gehören. Dies kann etwa für einen Verschlüsselungsalgorithmus verwendet werden [RFC3550].

Extension (X), 1 bit Das Extension Bit gibt Auskunft darüber, ob auf den Header eine Header-Erweiterung folgt.

CSRC Count (CC), 4 bit Dieses Feld beinhaltet die Anzahl der CSRC (Contribution Source) Identifizierer, die nach dem festen Teil des Headers folgen.

Marker (M), 1 bit Das Marker Bit kann von verschiedenen Profilen genutzt werden. Es ist vorgesehen, um etwa die Grenzen von Frames festlegen zu können.

Payload Type (PT), 7 bit Der Payload Type gibt Auskunft über die Art des Payloads. Es gibt eine Reihe fest definierter Payload-Typen sowie einen dynamischen Bereich (Siehe Tabelle 2.2).

Tabelle 2.2: Einige Payload-Typen

PT Nr	Codec
0	PCMU
3	GSM
8	PCMA
9	G722
31	H261
34	H263
96-127	dynamisch

Sequence Number (SN), 16 bit Die Sequence Number wird mit jedem Paket um 1 erhöht. Sie kann vom Empfänger verwendet werden, um den Verlust von Paketen festzustellen. Die erste Zahl ist dabei zufällig und nicht vorher bestimmbar, um Angriffe bei Verschlüsselung zu erschweren.

Timestamp (TS), 32 bit Dieses Feld beinhaltet den Zeitpunkt des ersten Oktetts des Pakets. Auch hier ist der Startwert zufällig. Pakete, die etwa zum gleichen Videoframe gehören, können hier den gleichen Wert enthalten.

Synchronisation Source Identifier (SSRC), 32 bit Die SSRC, ebenfalls ein Zufallswert, dient der Identifikation der Synchronisationsquelle, also der Quelle eines Streams von RTP-Paketen [RFC3550].

Contribution Source Identifier (CSRC), 0 bis 15 Felder mit je 32 bit In einem Stream, der die gemischten Daten mehrerer Teilnehmer enthält, haben die Pakete die gleiche SSRC, nämlich die des Mixers. Um die Pakete Teilnehmer beim Empfänger dennoch auseinander halten zu können, kann dieses Feld verwendet werden.

Header Extension, 32 Bit, optional Ist das Extension Bit gesetzt, muss dieses Feld eine einzelne Header-Erweiterung enthalten. Diese kann verwendet werden, um Payload-unabhängige Funktionen zu implementieren, die zusätzliche Informationen erfordern [RFC3550].

2.3.2 RTCP (Real-Time Transport Control Protocol)

RTCP, das ebenfalls in [RFC3550] beschrieben wird, hat das Hauptziel, Feedback über die Qualität der Datenverteilung zur Verfügung zu stellen. Darüber hinaus enthält RTP einen dauerhaften Transport Layer Identifizierer, den sogenannten CNAME (Canonical Name). Dieser hilft, den tatsächlichen Teilnehmer zu identifizieren, da die SSRC sich ändern kann. RTCP-Pakete werden periodisch gesendet. Die Verwendung von RTCP ist laut [RFC3550] nicht obligatorisch. Daher senden nicht alle Soft- und Hardphones entsprechende Pakete.

2.4 UDP

UDP (User Datagram Protocol) ist ein schlankes Netzwerkprotokoll, um Nachrichten zwischen Anwendungen zu verschicken. Es enthält nur die Informationen, die für eine Verbindung wirklich nötig sind: Quell- und Zielport, die Länge des Paketes, inkl. des UDP-Headers selbst in Oktetts und eine Prüfsumme (Siehe Abbildung 2.4). Es arbeitet verbindungslos. UDP implementiert keine Flusskontrolle, was bedeutet, dass verloren gegangene Pakete nicht nochmals gesendet werden. Es setzt IP als unterlagertes Protokoll voraus. UDP ist im OSI-Referenzmodell im Transport Layer angesiedelt [RFC768].

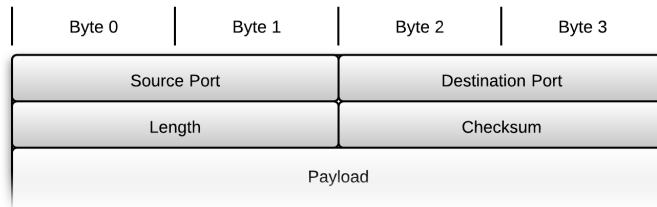


Abbildung 2.4: UDP-Header

Anmerkung: In der Grafik ist der Payload angedeutet, dieser gehört nicht zum Header.

2.5 GStreamer

GStreamer ist ein quell-offenes (Open Source) Media-Framework. Es stellt grundlegende Funktionen zur Verarbeitung von Mediendaten bereit. So können Mediendaten mit Hilfe der GStreamer Plugins etwa codiert, dekodiert, empfangen, verschickt oder anderweitig verarbeitet werden. Die Fähigkeiten von GStreamer sind sehr umfangreich und werden deswegen hier nur angerissen. Ausführlichere Informationen finden sich unter [GSTR01].

2.5.1 GStreamer Werkzeuge

gst-launch

Dieser Befehl dient zum Testen einer Pipeline in der Kommandozeile. Es lässt sich jede, über die API realisierbare Pipeline übergeben und so deren Funktion testen.

gst-inspect

Mit dem Befehl **gst-inspect** und einem danach übergebenem Element-Namen werden in der Kommandozeile Informationen über das Element angezeigt, wie etwa mit welchen Medientypen es umgehen kann oder welche Parameter das Element hat.

2.5.2 GStreamer Elemente

Pipeline

Eine Pipeline bezeichnet eine vollständige GStreamer-Verarbeitungskette. Nach dem Prinzip Eingabe, Verarbeitung, Ausgabe besteht eine sinnvolle Pipeline also mindestens aus 3 Elementen. Ein Beispiel wäre etwa: Laden einer unkomprimierten Audiodatei, Codieren der Datei ins MP3-Format und Speichern der Datei. Die Verarbeitung kann dabei auch viel mehr Schritte enthalten oder sich gar aufteilen, soll beispielsweise eine Videodatei gleichzeitig angezeigt und gespeichert werden (1 Quelle, 2 Senken) oder sollen eine Audio- und eine Videodatei zu einer Datei zusammengefügt werden (2 Quellen, 1 Senke).

Bin

Ein Bin (engl.; etwa: Behälter, Kasten) enthält mehrere Elemente, kann so gewisse Aufgaben vereinfachen und dann wiederum Teil einer Pipeline sein. So gibt es etwa „Playbin“, die das Abspielen von Videos vereinfacht oder „RTPBin“, die das Senden und Empfangen von RTP-Paketen wesentlich erleichtert.

Pad

Pads sind virtuelle Anschlüsse, um die einzelnen GStreamer Elemente zu einer Pipeline zusammenzufügen. Dabei gibt es Source- und Sink-Pads, also Quellen- und Senkenanschlüsse. Es gibt Elemente, die nur Source-Pads haben, etwa Elemente für das Öffnen einer Datei oder das Generieren eines Testbildes. Dann gibt es Elemente, die beide Arten von Pads haben (Quelle und Senke), in der Regel also Elemente für die Verarbeitung. Als dritte Möglichkeit gibt es Elemente, die nur eine oder mehrere Sink-Pads haben, zum Beispiel um eine Datei zu speichern.

Capabilities

Capabilities, zu deutsch Fähigkeiten, beschreiben die Medientypen, die Elemente oder ganze Pipelines verarbeiten können. Die Elemente handeln diese in der Regel unter sich selbst aus. Die Weitergabe kann jedoch innerhalb der Pipeline auf Medientypen bestimmter Parameter beschränkt werden.

Plugin

GStreamer Plugins sind all die einzelnen Elemente, die in GStreamer Eingabe, Verarbeitung oder Ausgabe übernehmen können.

Es gibt beispielsweise Plugins, um

- die Wellenform eines Audiosignals graphisch darzustellen,
- einen Strom von Medien in Pakete des RTP-Protokolls zu verpacken und ein Plugin, um diese Daten dann per UDP über das Netz zu versenden,
- aus einer Datei zu lesen oder in eine Datei zu schreiben,

- mehrere Audiosignale akustisch zu mischen
- uvm.

Das Framework selbst sowie seine Plugins sind in C geschrieben. Es gibt allerdings sog. Bindings, um GStreamer auch in anderen Programmiersprachen nutzen zu können, wie zum Beispiel Java.

Folgende GStreamer-Plugins und -Elemente sind für die vorliegende Arbeit von besonderem Interesse:

udpsource ist ein Plugin, um vom Netzwerk per UDP eingegangene Pakete zu empfangen und zu verarbeiten. Als Parameter kann beispielsweise der Port angegeben werden, auf dem das Plugin lauschen soll. Voreingestellt ist an der Stelle der Port 4951.

rtpbin ist ein Element, das mehrere andere Elemente in sich vereint. Es vereinfacht die Verarbeitung von RTP-Paketen. Es erkennt etwa, wenn RTP-Pakete verschiedener Sender empfangen werden und erstellt dementsprechend automatisch ein neues Source-Pad (Siehe Abschnitt 2.5.2) um diese dann abgreifen zu können.

rtppcmudepay depaketiert RTP-PCMU (Pulse Code Modulation μ Law)-Pakete (Siehe Abschnitt 2.6.3). Am Sink-Pad wird ein RTP-Strom „angeschlossen“ und an der Quelle wird ein PCMU-codierter Audiostream ausgegeben.

mulawdec decodiert den eingehenden PCMU-codierten Audiostream und gibt einen uncodierten Audiostream aus. Für die weitere Verarbeitung von Audiosignalen ist das unumgänglich.

tee ist ein „Mehrfachverteiler“ (engl. *tee* - T-Stück). Es besitzt eine Senke und N-Quellen, die alle das ausgeben, was an der Senke eingegeben wird. Die maximale Anzahl der theoretisch möglichen Quellen beträgt 2147483647 (32 Bit Integer mit Vorzeichen).

liveadder wird verwendet, um Audiosignale akustisch zu mischen. Der **liveadder** besitzt im Vergleich zu dem vergleichbaren Element **adder** den Vorteil, auch im laufenden Betrieb Audio-Quellen hinzufügen oder entfernen zu können.

mulawenc codiert ein Audiosignal mit dem PCMU-Codec.

rtppcmupay paketiert einen PCMU-codierten Audiostream für den Versand in RTP.

rtph264depay depaketiert einen in RTP „verpackten“ H.264-Videostrom.

ffdec_h264 decodiert einen in H.264-codierten Videostrom.

capsfilter verändert den Medienstrom selbst nicht, sondern schränkt lediglich die Weiterleitung auf bestimmte Formate ein.

videomixer mischt mehrere Videoströme.

ffmpegcolorspace konvertiert den Farbraum eines Videosignals.

videoscale kann ein Videosignal herauf- oder herunterskalieren, also dessen Auflösung vergrößern oder verkleinern. Dazu können verschiedene Algorithmen verwendet werden, wie etwa Pixelwiederholung oder ein bilinearer Algorithmus. Letzterer ist voreingestellt.

videobox fügt einem Video einen Rahmen hinzu oder beschneidet es. So kann einem Video etwa ein 2 Pixel breiter Rand hinzugefügt werden, der die äußeren 2 Pixel des Videos überdeckt.

textoverlay kann einen Text auf einem Videosignal einblenden.

x264enc ist ein Open Source Codierer für das H.264-Format.

rtpH264pay paketiert H.264-codiertes Video zum Versenden via RTP.

udpsink versendet die eingehenden Datenpakete mit dem Protokoll UDP über das Netzwerkinterface.

audiotestsrc generiert ein Audiosignal, für das unter anderem verschiedene Wellenformen und Frequenzen eingestellt werden können. Über den Parameter **wave** lässt sich die Wellenform bestimmen. Möglich sind unter anderem Sinus (voreingestellt, wenn nichts übergeben wird), Rechteck, Sägezahn, Stille oder Rauschen. Über den Parameter **freq** lässt sich die Frequenz (in Hertz) einstellen, 440 Hz sind voreingestellt.

videotestsrc generiert ein Videosignal. Eine mögliche Einstellung ist die Art des Testbildes (**pattern**), wie beispielsweise Schneegestöber (**snow**), Farbbalken (**smpte**, voreingestellt) oder Bluescreen (**blue**).

libvisual_lv_scope dient der Darstellung von Audiosignalen als Wellenform.

2.6 Digitales Audio

Dies ist ein extrem weites Feld, weswegen dieser Abschnitt nur oberflächlich bleiben kann. Mehr Details würden den Rahmen sprengen, aber nicht zu einem besseren Verständnis beitragen.

Um ein Audiosignal digital übertragen zu können, muss es zuvor digitalisiert werden. Dies geschieht in 3 Schritten:

1. Abtastung
2. Quantisierung
3. Codierung

2.6.1 Abtastung

Bei der Abtastung, auch Sampling (von engl. *to sample* - eine Probe entnehmen) genannt, werden die Amplitudenwerte eines zeit- und wertkontinuierlichen Audiosignals zum immer gleichen, wiederkehrenden Zeitpunkt gespeichert, zum Beispiel 10 mal pro Sekunde, was einer Abtastfrequenz von 10Hz entspräche. Siehe dazu Abbildung 2.5.

Resampling Beim sog. Resampling wird die Abtastrate eines digitalen Audiosignals im Nachhinein verändert.

Nyquist-Shannon-Abtast-Theorem Bezuglich der Abtastfrequenz ist zu beachten, dass diese immer mindestens doppelt so groß sein muss, wie die größte, noch zu erfassende Frequenz im Signal.

Diese Werte sind nun *zeitdiskret*, das heißt es gibt fest definierte Zeitpunkte, zu denen für den Amplitudengang der Amplitudenwert gespeichert wird. Allerdings sind diese noch *wertkontinuierlich*, was bedeutet die Amplitude kann zu einem bestimmten Zeitpunkt eine zwar *begrenzte* (zwischen größter und kleinster Amplitude), aber *stufenlose* und damit unendliche Anzahl an Werten annehmen. Um ein wertdiskretes Signal zu erhalten, muss dieses zusätzlich *quantisiert* werden.

2.6.2 Quantisierung

Bei der Quantisierung (von lat. *quantitas* – Menge) werden den während der Abtastung gemessenen, stufenlosen Amplitudenwerten zuvor fest definierte Stufen zugeordnet (wie zum Beispiel in Abbildung 2.5 die Stufen 0 bis 15). Wenn also das Signal zum Zeitpunkt 2 irgend einen Wert zwischen 10 und 11 hat, wird trotzdem nur entweder eine 10 oder eine 11 gespeichert.

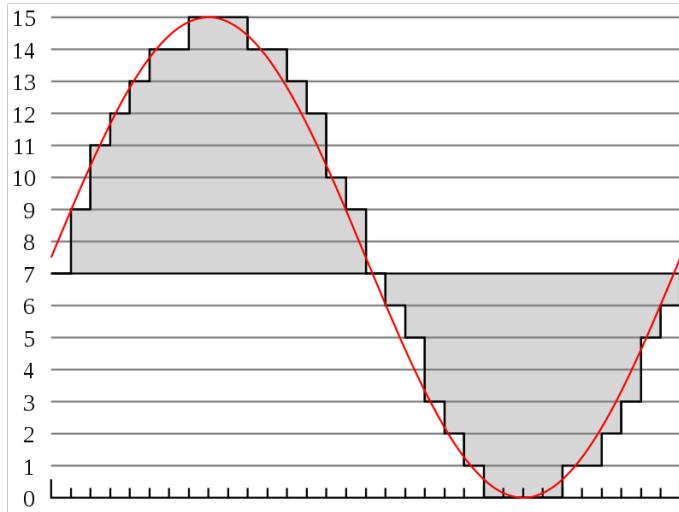


Abbildung 2.5: Digitalisierung eines kontinuierlichen Signals

Das Signal liegt nun als Zahlenfolge vor. Zur Übertragung und Speicherung wird das Signal noch codiert.

2.6.3 Codierung

Der Begriff Codec setzt sich aus den Worten *code* und *decode* zusammen, also codieren und decodieren. Beim Codieren werden die Amplitudenwerte eines digitalen zeit- und wertdiskreten Signals als Codeworte abgebildet, beim Decodieren wird dieser Vorgang wieder rückgängig gemacht.

PCMU (G.711 μ Law)

PCMU ist ein Codec, der sich sehr gut für das Codieren von Sprache eignet und daher in der IP-Telefonie häufig verwendet wird. Sein Name setzt sich aus 2 Teilen zusammen:

PCM PCM, ausgeschrieben Pulse-Code-Modulation, ist eine Methode, analoge Audiosignale digital abzubilden. Dabei werden, wie im Abschnitt 2.6.1 beschrieben, zeit- und wertkontinuierliche Audiosignale in zeit- und wertdiskrete Audiosignale umgewandelt, um sie digital speichern zu können.

μ Law-Verfahren Das μ Law-Verfahren ist eine Methode zur Quantisierung (Siehe Abschnitt 2.6.2) von Audiosignalen. Im Gegensatz zu linearer Quantisierung (gleich viele Stufen über den gesamten Dynamikbereich) kommt eine logarithmische Quantisierungskennlinie zum Einsatz. Diese ist bei niedrigeren Pegeln flacher (mehr Stufen) und bei höheren Pegeln steiler (weniger Stufen) als eine lineare Kennlinie, was zu einem besseren Signal-Rausch-Abstand führt. Darüber hinaus ist die Codierung so gestaltet, dass kontinuierliche 0-Folgen ausbleiben, was wiederum die Signallückengewinnung erleichtert.

Das codierte Signal wird mit folgender Formel aus dem linearen Signal erzeugt:

$$f_\mu(x) = \text{sgn}(x) \frac{\ln(1 + \mu * |x|)}{\ln(1 + \mu)} \quad (2.1)$$

mit 2.1 $\mu = 255$ wobei $\text{sgn}(x)$ = Vorzeichen von x . [CISC06]

2.7 Digitales Video

Dies ist ein noch viel weiteres Feld als 2.6, weswegen auch dieser Abschnitt nur oberflächlich bleiben kann. Mehr Details würden auch hier den Rahmen sprengen und ebenso nicht zu einem besseren Verständnis beitragen.

Ein Film, Video oder auch Bewegtbild ist die Abfolge von nacheinander folgenden Einzelbildern. Hierbei wird die Trägheit des menschlichen Auges ausgenutzt, um den Eindruck einer flüssigen Bewegung zu erzeugen. Ab 20 Bildern pro Sekunde kann ein Mensch keine Einzelbilder mehr wahrnehmen [SCHM08].

Wie auch bei der Digitalisierung von Audiosignalen sind mehrere Schritte nötig. Diese Schritte sind schematisch in Abbildung 2.6 dargestellt.

2.7.1 Abtastung/Rasterung

Wie auch bei der Abtastung von Audio in Abschnitt 2.6.1 werden zeitkontinuierliche Daten in zeitdiskrete und wertkontinuierliche in wertdiskrete Daten umgewandelt. Dazu werden in regelmäßigen Abständen, zum Beispiel 20, 25 oder 30 mal pro Sekunde - je nach Format - aufeinander folgende Einzelbilder aufgezeichnet. Der Sensor einer digitalen Kamera ist in ein Raster unterteilt (Siehe Abbildung 2.6 mittleres Bild). Auch hier gilt das **Abtast-Theorem von Shannon**: Das Raster muss mindestens so fein wie die halbe Größe des kleinsten aufzunehmenden Details des Bildes sein.

2.7.2 Quantisierung

Im Anschluss werden die einzelnen aufgezeichneten Punkte des Rasters quantisiert (vergl. 2.6.2), d.h. die real unendliche Farbtiefe wird auf eine definierte Menge begrenzt. Dabei gehen Farbinformationen verloren (Siehe Abbildung 2.6 rechtes Bild). In der Regel zeichnet eine Videokamera die Rot-, Grün- und Blau-Anteile des Bildes auf, aus denen sich alle anderen Farben mischen lassen. Diese Bildinformationen können äquivalent durch Luminanz (Helligkeitswerte; Y) und Chrominanz (Farbwerte; UV) ausgedrückt werden. Die Chrominanz, also die Farbwerte, können dabei ohne nennenswerten Qualitätsverlust im Verhältnis zur Luminanz reduziert werden. In diesem Zusammenhang tauchen in der Regel die Begriffe 4:2:2 oder 4:2:0 auf. Diese Angaben besagen, in welchem Faktor die Farbinformationen im Vergleich zu den Helligkeitsinformationen reduziert werden [TUDO95].



Abbildung 2.6: Digitalisierung eines Videosignals

2.7.3 Codierung

Die Anzahl möglicher Codecs für Video ist noch größer und unüberschaubarer als der von Audiocodecs. Jedoch ist allen gemein, dass sie versuchen, die Datenmenge zu verkleinern, indem sie redundante und/oder unwichtige Informationen aus dem Signal zu entfernen. Beispiele für solche redundanten oder unwichtigen Informationen sind etwa [TUDO95]:

- gleichbleibender Hintergrund in einer Szene über die Zeit
- Punkte gleicher Farbe nebeneinander im Bildraster
- Details, die vom menschlichen Auge i.d.R. nicht wahrgenommen werden

H.264

H.264 ist ein Codec der sehr vielfältig eingesetzt werden kann. H.264 bietet eine hohe Bildqualität bei niedriger Bandbreite.

2.8 Java

Meist wird der Begriff Java als Synonym für die Programmiersprache verwendet. Java bezeichnet aber eine ganze Umgebung, zu der neben der Programmiersprache auch die Java-Klassenbibliothek und die virtuelle Maschine gehören.

2.8.1 Objektorientierung

Java ist eine objektorientierte Programmiersprache. Objektorientierung ist ein Ansatz, bei dem reale Objekte als Objekte im Quellcode eines Programms abgebildet werden.

Im klassischen, sequentiellen Ansatz gibt es ein Hauptprogramm, das sich in Unterprogramme verzweigen kann (Siehe Abbildung 2.7).

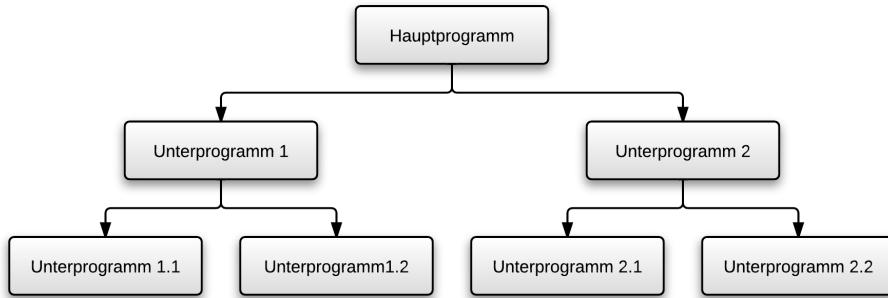


Abbildung 2.7: Sequentielle Programmierung

Eins der Hauptprobleme dabei ist, dass es zu Redundanzen kommt. Das Hauptprogramm kann auf Unterprogramm 1 und 2 zugreifen, Unterprogramm 2 jedoch nicht auf Unterprogramm 1.1 und 1.2. Soll nun Unterprogramm 2 auf Funktionen wie in 1.2 zugreifen können, so müssen diese mehrfach implementiert werden. Dadurch müssen Änderungen an mehreren Stellen durchgeführt werden, was wiederum die Fehleranfälligkeit erhöht.

Beim objektorientierten Ansatz gibt es einzelne Programmobjekte, die den realen Objekten entsprechen, die abgebildet werden sollen. Diese Objekte werden Klassen genannt. So gibt es etwa eine Klasse `auto` mit den Methoden `fahre()` und `bremse()`.

Vererbung

Darüber hinaus können Klassen gewisse Eigenschaften von anderen, generelleren Klassen erben. Das lässt sich am Besten an einem Beispiel verdeutlichen: Eine Klasse `pkw` und eine Klasse `flugzeug` erben etwa beide von der Klasse `fortbewegungsmittel` die Methode `zusteigen()`. Beide Klassen müssen diese Methoden also nicht redundant implementieren. Beide Klassen können sich darüber hinaus unterscheiden, so dass `pkw` eine Methode `fahre(geschwindigkeit)` und `flugzeug` eine Methode `fliege(geschwindigkeit, höhe)` haben.

2.8.2 Virtuelle Maschine

Während in anderen Programmiersprachen beim Kompilieren meist Maschinencode erzeugt wird, der für die Ausführung auf einer bestimmten Plattform vorgesehen ist, wird in Java ein Zwischencode (Bytecode) erzeugt. Dieser Bytecode wird dann von der virtuellen Maschine (VM) ausgeführt. Die VM muss zwar für jede Plattform angepasst werden, aber dafür ist ein einmal geschriebenes Java-Programm ohne Anpassung auf jeder Plattform lauffähig, für die es die VM gibt [GOLL01].

2.8.3 Errorhandling

Damit ein Programm bei einem Fehler nicht komplett beendet wird, müssen erwartbare Ausnahmefehler (sog. Exceptions) abgefangen werden. Wird zum Beispiel von dem Mediaserver ein SIP-Request (Siehe 2.1.1) empfangen, für den keine SIP-Transaktion (Siehe Abschnitt 2.1.2)

existiert, weil etwa der Mediaserver neu gestartet wurde, während noch eine Transaktion in Progress war, führt das zwangsläufig zu einem Fehler, der entsprechend behandelt werden muss, um unerwartetes Verhalten zu verhindern.

Dazu wird entweder der entsprechende Teil im Quelltext mit einem try-catch-Block umgeben, oder die Funktion um ein throws-Statement erweitert. Es können mehrere catch-Blöcke für verschiedene Fehlertypen oder auch ein Block für alle Fehler verwendet werden.

Wird eine Methode um das throws-Statement erweitert, so müssen alle Klassen, die diese Methode implementieren, entweder auch throws oder einen try-catch-Block für die entsprechende Funktion nutzen. Das throws-Statement informiert den Compiler über einen möglichen behandlungswürdigen Ausgang der Funktion [GOLL01].

2.8.4 Events

Oftmals besteht das Problem, dass ein Programmteil, also eine Funktion, ein Thread, eine Klasse, genau dann in Aktion treten soll, wenn ein anderer Programmteil einen bestimmten Zustand hat. Dies ist solange trivial, wie diese beiden Programmteile in einer Beziehung zueinander stehen und untereinander referenziert werden können. Ist das nicht der Fall, sind Events eine mögliche Lösung. Ein Event oder auch „Ereignis“ funktioniert im Grunde genommen so: Eine Partei A definiert ein Ereignis, eine andere Partei B „abonniert“ dieses über einen sog. „Listener“. Löst nun Partei A das Ereignis aus, wird Partei B davon unterrichtet und reagiert entsprechend dessen Listenern.

Soll beispielsweise eine bestimmte Klasse immer davon in Kenntnis gesetzt werden, wenn ein Wert einer anderen Klasse sich ändert, lässt sich das sehr gut mit Hilfe von Events realisieren.

2.8.5 Multithreading

Heutige Computer führen in der Regel mehrere Aufgaben gleichzeitig aus. Dabei muss zwischen Quasiparallelität und Hardware-Parallelität unterschieden werden. Quasiparallel bedeutet, dass die Aufgaben für den Nutzer den Anschein gleichzeitiger Ausführung erwecken, in Wirklichkeit jedoch nacheinander vom Prozessor verarbeitet werden. Bei echter Hardware-Parallelität werden die Prozesse von mehreren Prozessoren oder Prozessorkernen tatsächlich gleichzeitig ausgeführt.

Ein Prozess ist ein Programm mit all seinen Befehlen und Variablen, das sich gerade in Ausführung befindet. Das Prozessmodell basiert auf 2 Konzepten: der Bündelung von Ressourcen einerseits und der Ausführung andererseits. Wird nur das Konzept der Ausführung betrachtet und die sonst zum Prozess gehörigen Ressourcen, wie etwa geöffnete Dateien und erzeugte Kindprozesse, außen vorgelassen, wird von einem Thread oder auch Ausführungsfaden gesprochen. Ein Prozess kann dabei mehrere Threads enthalten, was Multithreading genannt wird. Diese Threads können wiederum quasiparallel oder hardwareparallel ausgeführt werden [TANN03].

Wurde bis vor wenigen Jahren noch versucht, die Rechengeschwindigkeit von Computern über die Taktfrequenz ihrer Prozessoren zu erhöhen, wird heute dazu übergegangen, Prozessoren mit mehreren Kernen zu bauen, wie etwa die Intel i-Serie oder AMD FX. Dieser Umstand ist von großem Vorteil, wenn eine Software exzessiven Gebrauch von Multithreading macht. Die Threads eines Prozesses werden Hardware-parallel ausgeführt und beschleunigen so die Ausführung des gesamten Prozesses.

Java selbst kann in seiner VM Multithreading auf Ein-Prozessor-System emulieren. Auf echten Mehrprozessorsystemen gibt die VM die Thread-Verwaltung an das Betriebssystem ab, es wird dann von nativen Threads gesprochen [ULLE06].

In Java gibt es mehrere Methoden, eigene Threads explizit zu starten. Eine Variante besteht darin, eine Klasse zu erzeugen, die das Interface (Siehe Abschnitt 2.8.7) Runnable implementiert, diese Klasse dann dem Konstruktor einer neuen Instanz der Klasse Thread als Argument zu übergeben und anschließend die Methode `start()` dieser Instanz der Klasse Thread aufzurufen. Eine weitere Möglichkeit besteht darin, ein Runnable-Objekt zu erstellen, dessen Methode `run()` zu überschreiben und dieses Objekt dann dem Konstruktor einer Instanz der Klasse Thread zu übergeben und daraufhin wieder dessen `start()`-Methode aufzurufen.

Wird mit Funktionen externer Bibliotheken wie GStreamer-Java gearbeitet, die - einmal gestartet - permanent ablaufen, kommt der Entwickler um explizites Multithreading überhaupt nicht herum.

2.8.6 Config Files

Manche Variablen im Programm sollten Initialwerte haben, die verändert werden können, ohne den Programmcode bearbeiten zu müssen. Daher ist es üblich und gute Praxis, diese Werte außerhalb des Programmcodes, etwa in einer Textdatei oder bei umfangreichen Daten gar einer Datenbank, zu speichern. So können Werte, wie zum Beispiel der gewünschte Port zum Empfang von Audioströmen, leicht mit Hilfe eines Texteditors bearbeitet werden. Dies kann auch von technisch weniger versierten Anwendern durchgeführt werden.

Es gibt die verschiedensten Möglichkeiten, die Variablen in einer Textdatei zu hinterlegen. Die meisten Möglichkeiten bietet XML, am einfachsten ist eine reine Textdatei, in der die einzelnen Werte im Format `[VARIABLENNAME]=[VARIABLENWERT]` gespeichert werden. In Java gibt es bereits Bibliotheken, um die Variablen dann einfach auszulesen oder auch zu schreiben. Aufwändige Datei- und String-Operationen werden so nicht nötig.

Zur Initialisierung des Programms werden dann kurzerhand die programminternen Variablen mit den Werten aus der Datei belegt.

2.8.7 Interfaces

Ein Interface ist ein programmiertechnisches Konzept, das die Interoperabilität von Programmteilen sicher stellt. Implementiert eine Klasse ein oder mehrere Interfaces, so muss diese Klasse alle Methoden des Interfaces überschreiben. Dadurch wird garantiert, dass alle Klassen, die ein bestimmtes Interface implementieren, gleich angesprochen werden können. Wie Interfaces genau funktionieren, unterscheidet sich geringfügig zwischen verschiedenen Programmiersprachen.

In Java kann eine Klasse, im Gegensatz etwa zu C++ oder Perl, nur von einer einzigen und nicht von mehreren Klassen gleichzeitig erben, dafür aber beliebig viele Interfaces implementieren. Daher gibt es in Java nur die Möglichkeit, mit Interfaces zu arbeiten, soll eine Klasse die Eigenschaften mehrerer anderer Klassen besitzen. Soll beispielsweise ein Objekt Auto die Felder und Methoden des Objektes Fortbewegungsmittel (etwa `fahre()` und `maxGeschwindigkeit`) bekommen, aber ebenso die des Objektes Produkt (zum Beispiel `verkaufen()` und `preis`), dann muss das über Interfaces realisiert werden.

2.8.8 JAIN-SIP

JAIN-SIP ist eine Java-Bibliothek, die es ermöglicht, auf Protokollebene mit SIP zu arbeiten. Durch diese tiefe Ebene gibt es keinerlei Beschränkung, welches SIP-Netzelement entwickelt werden kann. JAIN-SIP ist jedoch weniger gut für die Entwicklung eines Produkts geeignet, das schnell Marktreife erlangen soll, da die gesamte SIP-Logik von Hand implementiert werden müsste. JAIN-SIP erlaubt dafür aber direkten Zugriff auf SIP auf Protokollebene, wie etwa die Felder einzelner SIP-Nachrichten, und ermöglicht so eine große Kontrolle und hilft beim Verständnis für die Funktionsweise von SIP [PERE08].

2.8.9 GStreamer-Java

Als GStreamer-Java werden die sog. Bindings für GStreamer unter Java bezeichnet. Bindings erlauben den Zugriff auf Methoden einer in einer anderen Programmiersprache entwickelten Bibliothek. GStreamer ist in C geschrieben, die GStreamer-Java-Bindings ermöglichen deren Nutzung in Java.

2.9 GNU/Linux

GNU/Linux ist ein Unix-Derivat, aber im Gegensatz dazu frei und Open Source.

Was von den meisten Anwendern als Linux bezeichnet wird, sollte eigentlich korrekterweise GNU/Linux heißen. Das GNU-Projekt hatte sich lange vor Linus Torvalds vorgenommen, ein freies („im Sinne von Freiheit, nicht Freibier“ [GNU01]) Betriebssystem zu entwickeln. Heutige GNU/Linux Distributionen enthalten sogar wesentlich größere Anteile an GNU-Quellcode (28%) als Linux Quellcode (3%).

Was also als Linux bezeichnet werden sollte, ist daher nur der Kern des Betriebssystems, der sogenannte Kernel. Diesen Kernel hat Linus Torvalds 1991 begonnen zu entwickeln:

„I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones.“ - Linus Torvalds in der Usenet-Gruppe `comp.os.minix` am 16. August 1991

2.9.1 Philosophie

Da GNU/Linux ein Unix-Derivat ist, folgt es auch der Unix-Philosophie. Diese Philosophie zu verinnerlichen, erleichtert die Arbeit mit Unix-Derivaten, vor allem für Personen, die an andere Betriebssysteme und deren Philosophien gewöhnt sind.

„Alles ist eine Datei“

Diese Aussage betrifft die Organisation des Betriebssystems. Zur Vereinheitlichung von Schnittstellen und der so verbesserten Nutzerfreundlichkeit wird unter unix-artigen Betriebssystemen diese Abstraktion vorgenommen. Das bedeutet [KERN84]:

- Laufwerke, egal ob lokal oder im Netzwerk, werden über die gleiche Verzeichnisstruktur angesprochen.
- Dementsprechend werden auch virtuelle Laufwerke in ein und demselben Verzeichnisbaum erstellt und sind nur ein Verzeichnis darin.
- Gerätetreiber sind ebenfalls über das Dateisystem zugreifbar. Alle Gerätedateien finden sich im Verzeichnis `/dev` (von engl. *device* - Gerät).
- Der Zugriff auf Kernel-Daten geschieht über das Verzeichnis `/proc`.

„Mache nur eine Sache und mache sie gut.“

Unix-artige Systeme, also auch GNU/Linux, machen sehr stark Gebrauch von der Kommandozeile. Mittlerweile gibt es zwar eine ganze Reihe von grafischen Oberflächen und grafischen Anwendungen, ohne die Kommandozeile ist jedoch kaum der volle Funktionsumfang des Betriebssystems nutzbar. Dies hat seinen Ursprung in einem weiteren Aspekt der Philosophie, die unixoiden Betriebssystemen zugrunde liegt. Mike Gancarz hat diese in 9 Leitsätzen formuliert [GANC95]:

1. Klein ist schön
2. Jedes Programm soll genau eine Sache gut machen
3. Erstelle so schnell wie möglich einen Prototyp
4. Portabilität geht über Effizienz
5. Speichere numerische Daten in flachen ASCII-Dateien
6. Nutze die Hebelwirkung von Software zu deinem Vorteil
7. Shell-Skripte vergrößern Hebelwirkung und Portabilität
8. Vermeide Benutzeroberflächen, die den Benutzer gefangen halten
9. Gestalte jedes Programm als Filter

Diese Leitsätze werden oft zusammengefasst als „Mache nur eine Sache und mache sie gut.“. Außerdem können Programme auf ASCII-Text-Basis zusammen arbeiten. Soll also das Dateilisting eines Ordners mit einem weiteren Programm gefiltert und dessen Ergebnis dann in einer Datei gespeichert werden, ist das mit wenigen Befehlen zu erledigen:

```
ls | grep pdf > alle-pdfs.txt
```

Anmerkung: Die obere Zeile findet alle Dokumente, die „pdf“ im Dateinamen haben, also nicht nur als Endung, außerdem ist die Suche case-sensitiv. Dokumente, deren Dateinamen „PDF“ enthalten, werden also nicht gefunden.

Pipelines und Ausgaben umlenken Mit Pipes (vertikaler Strich |) lassen sich Kommandozeilenwerkzeuge verbinden. Was das Programm links von der Pipe ausgibt, dient dem Programm rechts davon als Eingabe (Siehe Codebeispiel in Abschnitt 2.9.1). Ebenso ist es möglich, die Ausgabe eines Programms umzuleiten, etwa in eine Datei, statt auf das Terminal. Dies geschieht über den Operator >.

sudo Der Befehl **sudo** („superuser do“) wird einem anderen Befehl vorangestellt, wenn dieser mit Superuser-Rechten ausgeführt werden soll. Das ist zum Beispiel dann vonnöten, wenn ein neues Programm installiert wird.

Der zuletzt eingegebene Befehl wird mit Superuser-Rechten durch die Eingabe von

```
sudo !!
```

wiederholt.

aptitude Aptitude ist eine von diversen Paketmanagern unter GNU/Linux. Sehr häufig, nämlich zur Installation eines Programms, wird der aptitude-Befehl **install** ausgeführt:

```
apt-get install sendip
```

Wobei nach dem Befehl **install** noch der Name des zu installierenden Programms eingegeben werden muss, im oberen Beispiel also das Programm **sendip**.

grep Mit **grep** lassen sich Texte filtern. Die Eingabe erfolgt dabei nach dem folgenden Schema:

```
grep [options] PATTERN [FILE...]
```

Wobei **PATTERN** ein Regulärer Ausdruck ist [GNU01].

sendip Mit diesem Programm lässt sich ein IP-Paket verschicken. Der Inhalt kann in Form einer Datei übergeben werden. Ein Codebeispiel findet sich im Kapitel Realisierung in Abschnitt 4.4.2, Absatz Zwischentest unter „Focus“.

2.9.2 Lizenz

Das Thema Lizenzierung ist eigentlich eher ein juristisches, nichtsdestotrotz soll es hier kurz angerissen werden, da es für Entwickler nicht gänzlich unwichtig ist.

Der Großteil der Software in der GNU/Linux-Welt steht unter der sogenannten GNU GPL (GNU General Public License). Es muss dabei zwischen Urheberrecht und Nutzungsrecht unterschieden werden. Diese beiden Begriffe werden oft falsch verwendet. Unter der GNU GPL behält der Urheber eines Werkes sein Urheberrecht, räumt jedoch Nutzern seines Werkes uneingeschränkte Nutzungsrechte ein. Lizenznehmer, die entsprechend lizenzierte Software vertreiben - ob modifiziert oder nicht - müssen ihren Kunden die gleichen Rechte einräumen [GNU01]. Das bedeutet, dass Software, die auf GNU GPL-lizenzierter Software basiert, ebenfalls unter GNU GPL veröffentlicht werden muss.

Kapitel 3

Anforderungsanalyse

Diese Anforderungsanalyse oder auch SRS (Software Requirements Specification) ist Teil der Diplomarbeit „Optimierung eines Mediaservers auf Basis der Open Source Lösung GStreamer zur Bereitstellung von Videokonferenzen“. Sie folgt soweit wie möglich dem Standard der ANSI/IEEE Std 830-1984 zum Aufbau von entsprechenden Dokumenten [IEEE830].

3.1 Einleitung

Dieser Abschnitt enthält die Einleitung der Anforderungsanalyse. Es wird auf Zweck und Umfang sowie Begriffsdefinitionen bezüglich der Anforderungsanalyse eingegangen.

3.1.1 Zweck

Der Zweck dieser Anforderungsanalyse ist die Beschreibung des Produkts. Sie enthält alle Anforderungen an den Soll-Zustand des Mediaservers. Abschnitt 1 und 2 (Einleitung und Allgemeine Beschreibung) beschreiben das Produkt aus Anwendersicht, Abschnitt 3 (Spezifische Anforderungen) ist hauptsächlich für den Entwickler von Belang.

3.1.2 Umfang

Dieses Dokument, also der Abschnitt Anforderungsanalyse der Diplomarbeit, umfasst die Beschreibung der Anforderungen an den Mediaserver.

3.1.3 Erläuterungen zu Begriffen und Abkürzungen

Die folgende Tabelle 3.1 enthält eventuell erläuterungsbedürftige Begriffe der Anforderungsanalyse.

Tabelle 3.1: Begriffe und Abkürzungen

Begriff	Erläuterung
Konferenzlösung/Mediaserver	Wird hier im Dokument von dem <i>Mediaserver</i> gesprochen, ist das Java-Programm gemeint. Wird hingegen von <i>Konferenzlösung</i> gesprochen, ist die gesamte Umgebung gemeint. Sprich: Der Computer mit dem laufenden Mediaserver und ggf. angeschlossene Clients bzw. SIP User Agents. <i>Media-Server</i> , also mit großem „S“, bezeichnet die Klasse Media-Server
Client	Als Client werden die Teilnehmer der Videokonferenz bezeichnet, also die SIP User Agents (Hard- oder Softphones mit Kamera, Display und Mikrofon), die sich mit dem Mediaserver verbinden
Focus	Als Focus wird der Teil des Mediaservers bezeichnet, der die Signalisierung per SIP übernimmt
Mixer/Bridge	Als Mixer bzw. Bridge wird der Teil des Mediaservers bezeichnet, der die Mediendaten selbst mischt bzw. aufbereitet. Der Mixer mischt die Audiosignale akustisch. Als (Video-) Bridge wird dagegen ein Systeme bezeichnet, das verschiedene Videobilder über- und nebeneinander darstellen kann. Zur Unterscheidung der Bridge zu einem tatsächlichen Videomixer siehe Abbildung 3.1

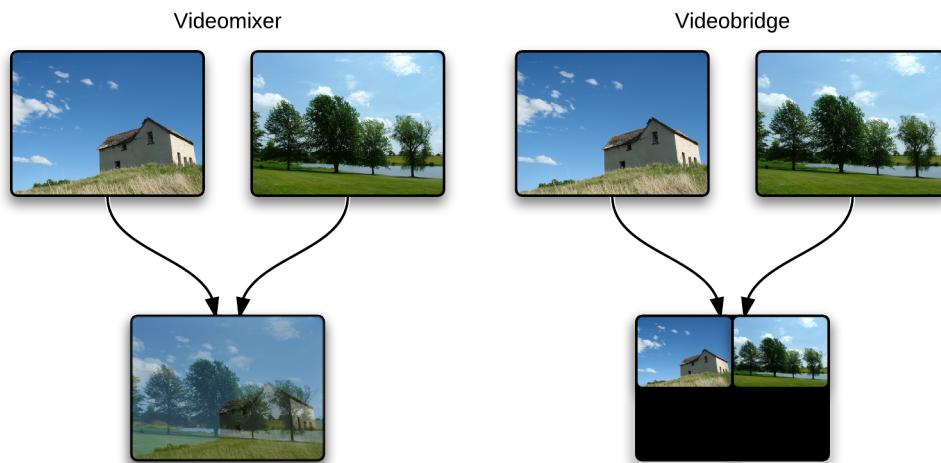


Abbildung 3.1: Unterschied zwischen einem Videomixer und einer Videobridge

3.1.4 Verweise auf sonstige Ressourcen oder Quellen

Die Diplomarbeit, in die diese Anforderungsanalyse eingebettet ist, enthält Angaben zu den technischen Grundlagen (Kapitel 2) sowie Details der Umsetzung (Kapitel 4).

3.1.5 Übersicht

Diese Anforderungsanalyse besteht aus den folgenden Teilen:

Einleitung Die Einleitung enthält eine Übersicht über dieses Kapitel (Anforderungsanalyse) der Diplomarbeit.

Allgemeine Beschreibung Dieser Teil enthält eine grobe Beschreibung des Mediaservers. Unter anderem: was soll das Produkt aus Nutzersicht können (Unterabschnitt Produktfunktionen und Anwendungsfälle), wer sind die zu erwartenden Nutzer (Unterabschnitt Benutzermerkmale) und welche Annahmen über die Umgebung, in der das Produkt zum Einsatz kommt, werden getroffen (Unterabschnitt Annahmen und Abhängigkeiten)?

Spezifische Anforderungen In diesem Abschnitt wird auf die technischen Details eingegangen, sprich: Welche genauen funktionalen und nicht-funktionalen Anforderungen werden an das Produkt gestellt, welche Schnittstellen stellt das Produkt zur Verfügung, welchen Qualitätsanforderungen muss das Produkt genügen?

3.2 Allgemeine Beschreibung

Das Produkt bietet die Möglichkeit, in einer SIP-Umgebung (Siehe Abbildung 3.2) eine Videokonferenz mit bis zu 4 Teilnehmern aufzubauen. D.h.: die Nutzer wählen von ihren Clients den Mediaserver an, worauf sie auf ihren Clients die Videos aller bereits verbundenen Teilnehmer sehen und die Audioströme aller Nutzer, außer sich selbst, hören können.

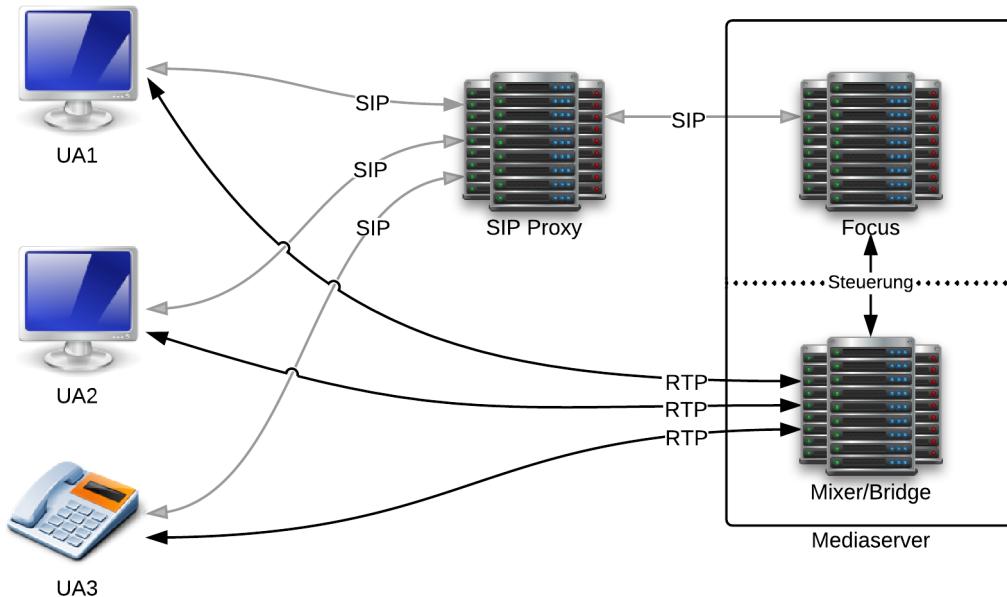


Abbildung 3.2: Umgebung Konferenzlösung

3.2.1 Produktperspektive

Der Mediaserver wird in Java entwickelt. Der Mediaserver verwendet die Bibliothek gstreamer-java, um Mediendaten verarbeiten zu können und die jain-sip-Bibliothek, um auf Protokollebene SIP-Nachrichten zu senden und zu empfangen. Die Bibliothek gstreamer-java stellt Zugriff auf das GStreamer Media-Framework bereit, das in C geschrieben ist.

Systemschnittstellen

In diesem Abschnitt wird aufgelistet, welche Schnittstellen, sowohl intern als auch extern, das Produkt implementiert. Für Details zu den Schnittstellen sind die Kapitel 2 und 4 der Diplomarbeit, in die diese Anforderungsanalyse eingebettet ist, heranzuziehen.

Interne Schnittstellen

- GStreamer API
- Steuerung Focus - Mixer (Siehe außerdem 3.3.1)

Externe Schnittstellen

- Session Initiation Protocol (SIP) mit Session Description Protocol (SDP)
- Realtime Transport Protocol (RTP)

Benutzerschnittstellen

Als Benutzerschnittstelle dient die Kommandozeile. Es werden Statusmeldungen ausgegeben, die Auskunft über den Zustand des Systems geben. Aus Gründen der Internationalisierung sind alle Statusmeldungen in Englisch verfasst.

Folgende Meldungen werden ausgegeben:

- Teilnehmer beigetreten, mit aktueller Teilnehmeranzahl
- Teilnehmer hat die Konferenz verlassen, mit aktueller Teilnehmeranzahl
- Fehlermeldungen
- evtl. interne Meldungen zur Kontrolle der ordnungsgemäßen Funktion

Darüber hinaus ist eine Logging-Funktion wünschenswert. Sprich: Das Generieren einer Textdatei, in die während des Betriebs des Systems Statusmeldungen geschrieben werden, um die ordnungsgemäße Funktion kontrollieren zu können und im Fehlerfall die Ursachenforschung zu erleichtern.

3.2.2 Produktfunktionen und Anwendungsfälle

Wie bereits erwähnt, ist die Funktion des Produkts im Wesentlichen die Bereitstellung von Videokonferenzen. Welche Anwendungsfälle (Use Cases) sich daraus ergeben, ist dem folgenden Use Case Diagramm 3.3 zu entnehmen.

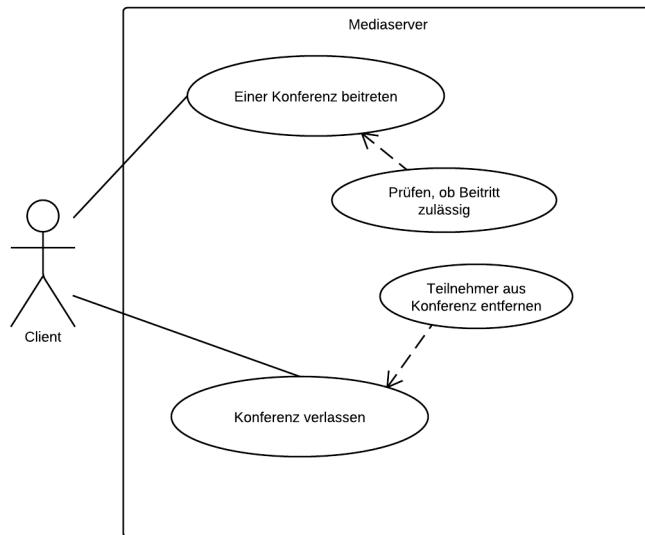


Abbildung 3.3: Use Cases Mediaserver

3.2.3 Benutzermerkmale

Die zu erwartenden Benutzer der Konferenzlösung beschränken sich auf Studenten im Telekommunikationslabor und Mitarbeiter der Forschungsgruppe für Telekommunikationsnetze an der Fachhochschule Frankfurt. Es kann daher von einem gewissen Grundverständnis der Funktionsweise von Software, Netzwerken bzw. Netzwerkprotokollen ausgegangen werden.

3.2.4 Annahmen und Abhängigkeiten

Notwendige Software

- GStreamer
- Java Virtual Machine
- installierte Bibliotheken:

gstreamer-java-1.5.jar
jain-sip-sdp-1.2.1995.jar
jna-3.2.4.jar
commons-configuration-1.8.jar
log4j-1.2.12.jar
commons-lang-2.6.jar
commons-logging-1.1.1.jar

Notwendige Hardware

- Ein Computer mit Netzwerkanschluss, auf dem der Mediaserver zum Laufen kommt. Bei der Dimensionierung ist zu beachten, dass der Computer in der Lage sein muss, mehrere Video- und Audioströme zu empfangen, zu mischen und wieder zu versenden, ohne eine allzu große Latenz (Zahlenwerte siehe Abschnitt 3.3.2) hinzuzufügen.
- SIP User Agents mit Kamera und Mikrofon

Notwendige Orgware Es wird vorausgesetzt, dass die Lösung in einer Netzwerkumgebung mit IPv4-fähigen und eingerichteten Netzwerkzugängen eingesetzt wird.

Sonstige Annahmen Es wird davon ausgegangen, dass die Clients Video- und Audiosignale senden. Die Möglichkeit, der Konferenz mit nur einem Audio- oder Videosignal beizutreten, wird außen vorgelassen.

3.3 Spezifische Anforderungen

In diesem Abschnitt werden die Anforderungen aus Abschnitt 3.2 genauer spezifiziert.

3.3.1 Funktionale Anforderungen

Verbindung zum Mediaserver

Aufbau einer Verbindung Der laufende Mediaserver lauscht an einem definierten Socket, ob ein Client eine Verbindung aufzubauen versucht.

1. Anfrage des Clients und Initialisierung der Verbindung per SIP
2. Hinzufügen des Teilnehmers in die GStreamer Pipeline

Anfrage des Clients und Initialisierung der Verbindung per SIP Der Client ist ein SIP User Agent. Daher findet der Aufbau der Verbindung über das Schema *INVITE - 200 OK - ACK* (Siehe Abschnitt 2.1) statt. Ebenso muss die Verbindung entsprechend abgebaut werden. Lehnt der Mediaserver die Verbindung ab, wird dies dem Client mit entsprechenden SIP-Statusmeldungen (4xx-6xx) signalisiert. Hierfür müssen passende Statusmeldungen im entsprechenden Standard [RFC3261] gefunden werden.

Hinzufügen des Teilnehmers in die GStreamer Pipeline Steht dem Hinzufügen eines neuen Clients zu Bridge und Mixer nichts entgegen, wird dies entsprechend durch den Focus per SIP signalisiert, woraufhin der Client der GStreamer Pipeline hinzugefügt werden kann.

Dazu wird das per UDP eingehende Videosignal in die Videobridge eingespeist, die dieses depaketiert, decodiert und mit den ggf. schon vorhandenen Videosignalen unter- und nebeneinander dargestellt ausgibt. Anschließend wird das Signal wieder H.264-codiert, paketiert und per UDP gesendet.

Die Verarbeitung des Audiosignals unterscheidet sich davon, weil die Teilnehmer ihr eigenes Audiosignal nicht hören sollen. Geht also das Audiosignal des neuen Clients per UDP ein, wird dieses depaketiert, decodiert und den ausgehenden Audiosignalen der bereits vorhandenen Clients akustisch hinzugemischt. Außerdem werden die Audiosignale der bereits existierenden Clients dem ausgehenden Audiosignal des neuen Clients hinzugefügt. Die ausgehenden Audiosignale der Clients werden dann wieder codiert, paketiert und per UDP gesendet, wobei im Gegensatz zum Video so viele Audiosignale resultieren, wie Clients verbunden sind. Als Audiocodec wird PCMU vorgesehen, die Implementierung weiterer, bei SIP üblicher Codecs ist wünschenswert.

Abbau einer Verbindung Der Abbau einer Verbindung wird ebenfalls über SIP signalisiert (*BYE - 200 OK*). Es ist zu beachten, dass eine außerordentliche Beendigung der Verbindung (z.B. Netzwerkverbindung geht verloren) abgefangen wird.

Elemente des Mediaservers

Der Mediaserver besteht aus den 3 Elementen Focus, Videobridge und Audiomixer (Siehe Abbildung 3.4), wobei Mixer und Bridge in gewisser Weise eine Funktionseinheit bilden, da sie für die Aufbereitung der Mediadaten zuständig sind.

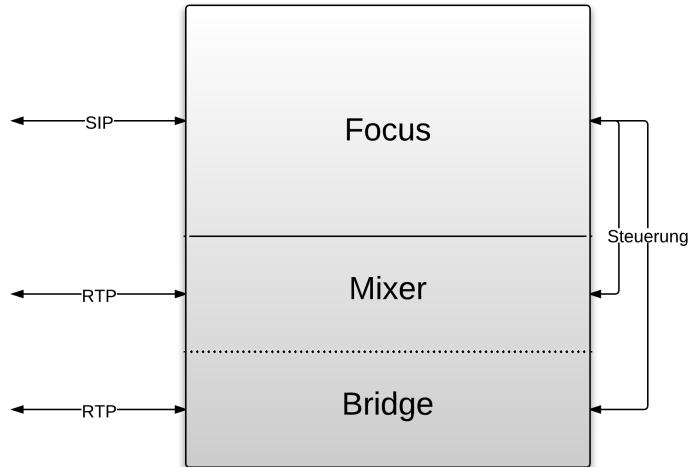


Abbildung 3.4: Schematischer Aufbau des Mediaservers

Focus Um mit den User Agents kommunizieren zu können, muss der Mediaserver Nachrichten des SIP-Protokolls empfangen und versenden können, und entsprechend reagieren. Diese Aufgabe übernimmt der Focus. Der Focus ist zuständig für die vollständige Abwicklung der SIP-Dialoge, entsprechend des Zustandes von Mixer und Bridge. Können Mixer und Bridge also beispielsweise keine neuen Teilnehmer hinzugefügt werden, weil die maximale Teilnehmeranzahl erreicht ist, kommunizieren Mixer und Bridge das dem Focus. Dieser wiederum meldet dem Client, der sich zu verbinden versucht hat, etwa ein *600 Busy Everywhere* oder ein *486 Busy* zurück, bzw. wickelt den entsprechenden kompletten SIP-Dialog ab. Verlässt hingegen beispielsweise ein Client die Konferenz mit einem *BYE*, meldet der Focus das an Mixer und Bridge, die den Teilnehmer aus der Konferenz entfernen und den Vollzug an den Focus zurückmelden. Dieser sendet dem Client dann das *200 OK* zurück.

Audiomixer Verbindet sich ein neuer Teilnehmer mit dem Mediaserver, wird er der Konferenz hinzugefügt, sofern die maximale Teilnehmerzahl noch nicht überschritten wurde. Die Audioströme der bereits vorhandenen Teilnehmer werden ihm hinzugefügt, sein Audiosignal wiederum den bereits vorhandenen Teilnehmern. So wird sichergestellt, dass jeder Teilnehmer die Audiosignale der jeweils anderen Teilnehmer hört, sich selbst jedoch nicht.

Videobridge Die Videobridge sorgt für die Darstellung der Videoströme. Sie ist bereits entwickelt und wird in die Gesamtlösung integriert.

3.3.2 Nicht-funktionale Anforderungen

Anforderungen an Performance

Da das Produkt nicht kommerziell eingesetzt wird, sondern im Rahmen einer Diplomarbeit als „proof-of-concept“ entsteht und nur unter Laborbedingungen zum Einsatz kommt, wird der Performance keine große Bedeutung beigemessen. Dennoch ist eine möglichst geringe Verzögerung

der Kommunikation (Signalverzögerungen von Mund zu Ohr unter 200 ms gelten als sehr gut, 200-300 ms als gut, 300-400 ms als noch akzeptabel [TRIC09]) und ein möglichst geringer Versatz zwischen Audioausgabe und Videodarstellung („lip-sync“) wünschenswert.

Qualitätsanforderungen

Für die Qualitätsanforderungen gilt Ähnliches wie für die Anforderungen an die Performance: Ein kommerzieller Einsatz ist nicht vorgesehen. Dennoch sollte das Produkt so stabil wie möglich laufen. D.h. bei einem bestimmungsgemäßem Betrieb laut der entsprechenden Definition in Abschnitt 3.2.4 sollte das Produkt nicht abstürzen.

3.3.3 Sonstige Anforderungen

Design

Der Quellcode soll mit ausreichend Kommentaren versehen und strukturiert sein, so dass er auch für sachkundige Personen, die nicht an dem Projekt mitarbeiten, gut lesbar und verständlich ist. Kommentare sind in englisch zu verfassen.

Wünsche

Wünschenswert ist die Möglichkeit, den Mediaserver ohne Änderung des Quelltextes konfigurieren zu können. Dies kann beispielsweise durch die Übergabe bestimmter Parameter beim Start in der Kommandozeile, Konfigurationsdateien, über ein Webinterface oder per SIP selbst erfolgen.

Kapitel 4

Realisierung

4.1 Entwicklungsumgebung

Die Entwicklungsumgebung besteht komplett aus kostenlosen Diensten bzw. Open Source-Software. Dies gibt anderen Studenten die Möglichkeit, die Ergebnisse kostenfrei zu nutzen und weiterzuentwickeln. Zur Organisation kommen Cloud-basierte Dienste, wie Diigo, Dropbox und Google Docs zum Einsatz, um das Risiko von Datenverlust zu minimieren und von überall auf die Daten zugreifen zu können.

Folgende Hard- und Software kommt zum Einsatz:

- 3 Computer mit Ubuntu / KUbuntu, der als Server fungierende Computer dabei mit einem Intel i5 Mehrkernprozessor
- Eclipse IDE zur Entwicklung in Java
- L^AT_EXmit KiLe, BIBTeX und einer umfangreichen Anzahl an Zusatzpaketen für den schriftlichen Teil
- Dropbox zur Sicherung und Verteilung aller Dokumente und Daten
- Diigo Bookmark Tool zur Sammlung und Organisation von Informationen und Weblinks
- Google Docs für allgemeine Notizen
- Lucidchart zur Visualisierung

4.2 Projektierung

Geschätzter Zeitaufwand:

1. Projektierung: 2 Wochen
2. Entwicklung Mediaserver Basisfunktionalität (GStreamer): 2 Monate
3. Entwicklung SIP-Implementierung (JAIN-SIP): 2 Monate

4. Refaktorierung: 1 Woche
5. Test: 1 Woche
6. Dokumentation: 1 Monat

An dem Dokument Diplomarbeit wird während der gesamten Zeit kontinuierlich gearbeitet.

4.3 Vorarbeiten

4.3.1 GStreamer

Zunächst wird GStreamer in der Kommandozeile (Siehe Abbildung 4.1) getestet, um sich mit dessen Funktionsweise und Features vertraut zu machen. In einer Testumgebung, die aus 3 Rechnern besteht, werden Audio- und Videoströme generiert und versendet, empfangen, gemischt, verarbeitet und dargestellt.

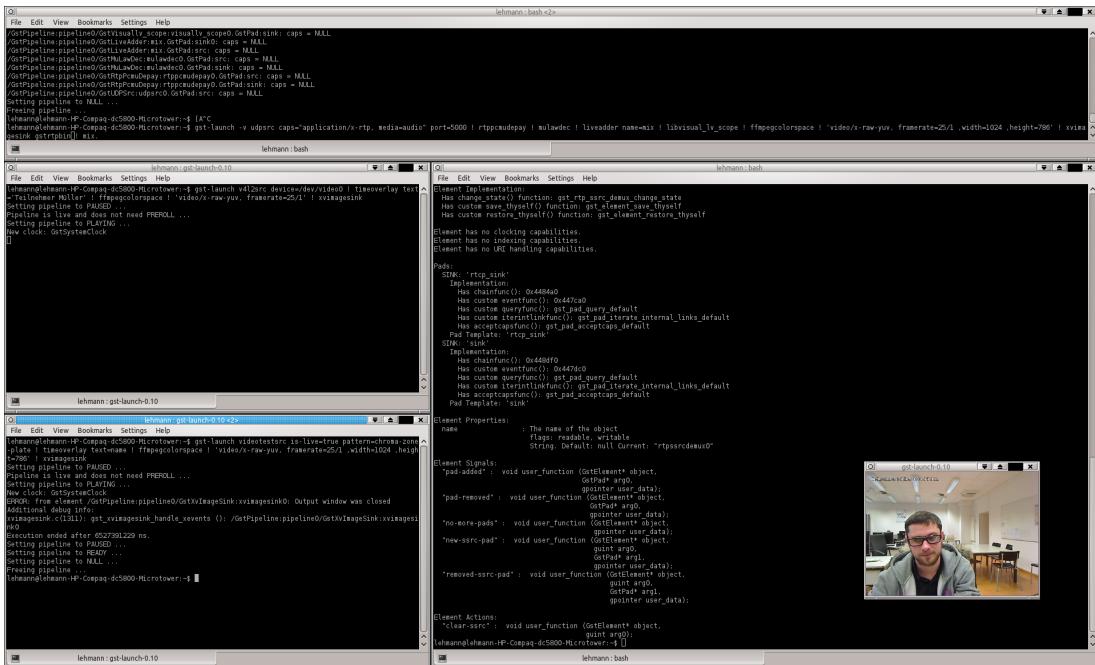


Abbildung 4.1: Testumgebung GStreamer per Kommandozeile

Folgende Szenarien werden getestet:

Video

- Testbild mit Texteinblendung
- Anzeigen eines Fensters, welches das von der Webcam aufgezeichnete Bild anzeigt

- Testvideo an eine Adresse im Netzwerk senden
- Empfangen dieses Videos an der Gegenstelle

Audio

- Audiotestsignal über das Netzwerk senden
- Über das Netzwerk empfangenes Audiotestsignal auf Lautsprecher ausgeben
- 2 Audiotestsignale mischen und über Lautsprecher wiedergeben
- 2 Audiotestsignale mischen und resultierende Wellenform als Video ausgeben
- 2 Audiotestsignale aus Netzwerkquelle empfangen, mischen und als Video ausgeben
- 2 Audiotestsignale aus Netzwerkquelle empfangen, mischen und über Lautsprecher wiedergeben

Exemplarisch hier einige Szenarien, von simpel bis kompliziert, im Detail:

Testbild mit Texteinblendung

Zunächst wird ein sehr einfaches Szenario getestet. Es soll ein Testbild angezeigt und ein Text eingeblendet werden.

```

1 gst-launch
2 videotestsrc pattern=smpTE !
3 timeoverlay text=TEST !
4 'video/x-raw-yuv, framerate=25/1 ,width=640 ,height=480' !
5 xvimagesink

```

Anmerkung: Bezuglich der Pipelines ist zu beachten, dass diese ohne Zeilenumbrüche und Einrückungen in der Kommandozeile einzugeben sind. Zeilenumbrüche sind hier nur aus Gründen der Lesbarkeit eingefügt worden. Die Pipelines sind als Skripte auf der beigelegten CD enthalten.

Hier die einzelnen Teile des Aufrufs im Detail; auf bereits erläuterte Elemente wird nicht nochmals eingegangen:

gst-launch ist der Aufruf des GStreamer-Kommandozeilen-Programms, dahinter können optional noch Parameter übergeben werden (hier nicht der Fall) und anschließend die Pipeline (Siehe Abschnitt 2.5.2).

videotestsrc ist das Quell-Element der Pipeline. Als Parameter wird **pattern=smpTE** übergeben, was eins der möglichen Muster ausgibt (Details gibt **gst-inspect [Elementname]** aus), hier im Beispiel ein Balkenmuster (Siehe Abbildung 4.2).

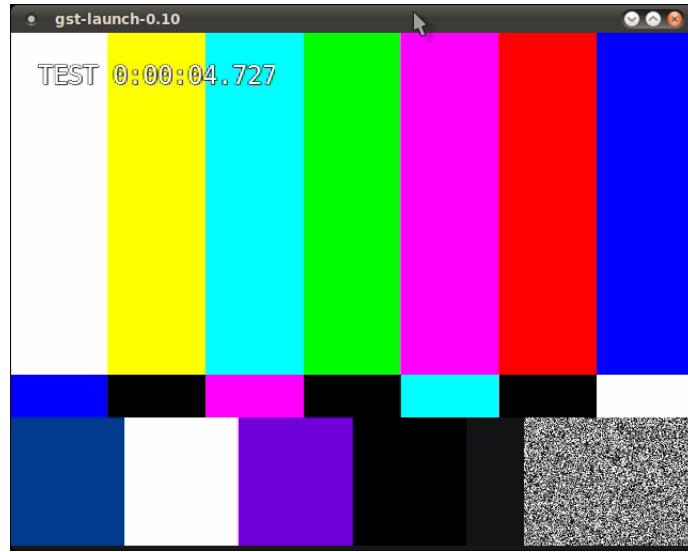


Abbildung 4.2: Ausgabe eines Testbildes mit videotestsrc

timeoverlay blendet über das auf der Senke eingegebene Video die verstrichene Zeit und einen optional übergebenen Text, hier im Beispiel „TEST“ mit dem Parameter `text=TEST`.

Caps oder auch Capabilities (engl. für Fähigkeiten) beschränken die Weitergabe von Mediendaten entlang der Pipeline, in diesem Beispiel hier auf Video mit YUV-Farbraum, der Framerate 25 Bilder pro Sekunde und der Auflösung 640 mal 480 Pixel.

xvimagesink ist das letzte, das Senken-Element der Pipeline. Es gibt das auf der Senke eingegebene Video in einem Fenster auf dem Bildschirm aus.

Audiotestsignal über das Netzwerk senden

Es soll ein Testsignal erzeugt, mit PCMU codiert, in RTP paketiert und per UDP über das Netzwerk versendet werden.

```

1 gst-launch -v
2
3 audiotestsrc freq=440 !
4   audioconvert !
5   audioresample !
6   mulawenc !
7   rtppcmupay !
8   udpsink host=192.168.0.91 port=5000

```

Anmerkung: Einrückungen und Zeilenumbrüche aus Gründen der Lesbarkeit eingefügt

gst-launch startet GStreamer, hier mit dem Parameter **-v** für „verbose“ (engl. für wortreich), der für detailliertere Ausgaben sorgt, was die Fehlersuche erleichtern kann.

audiotestsrc erzeugt ein Audio-Testsignal, hier einen Sinuston mit der Frequenz 440 Hertz (Kammerton A).

audioconvert konvertiert Audiodaten zwischen verschiedenen Formaten, zum Beispiel verschiedenen Bit-Tiefen und Samplingraten.

audioresample dient dem Resampling von Audiodaten.

mulawenc codiert das Audiosignal mit dem PCMU Codec.

rtpcmupay paketiert die mit dem PCMU codierten Daten in RTP.

udpsink verschickt die am Sink-Pad eingehenden Pakete an den Socket, der mit den Parametern **host** und **port** übergeben wird.

2 Audiosignale aus Netzwerkquelle empfangen, mischen und als Video ausgeben

Die im Abschnitt 4.3.1 beschriebene Pipeline wird auf 2 Rechnern gestartet und die Audiosignale sollen auf einem dritten Rechner empfangen, gemischt und deren Wellenform als Video ausgegeben werden.

```

1 GST_DEBUG_DUMP_DOT_DIR=/tmp/ gst-launch --gst-debug-level=2
2 gstrtpbin name=rtpbin
3 udpsrc name=udpsrc0 caps="application/x-rtp, media=audio, clock-rate=8000,
4 payload=0, encoding-name=PCMU" port=5000 !
5 rtpbin.recv_rtp_sink_0

```

gst-launch wird hier mit dem Parameter **--gst-debug-level=2** gestartet, was zusätzliche Meldungen ausgibt, die bei der Fehlersuche helfen. Die Tiefe der Meldungen reicht dabei von 0 (keine Meldungen) bis 5 (alle Meldungen) [GSTR01]. Die hier verwendete Meldungstiefe 2 gibt nur Warnungen und Fehler aus. Darüber hinaus wird dem Befehl ein Pfad übergeben (**GST_DEBUG_DUMP_DOT_DIR=/tmp/**), in dem eine DOT-Datei gespeichert wird, jedes Mal wenn der Zustand der Pipeline sich ändert. DOT ist eine Sprache, mit der Diagramme generiert werden können. Die so erstellte Datei kann dann mit dem Befehl

```
dot -Tpng DATEINAME.DOT -o AUSGABEDATEI.PNG
```

in eine PNG-Grafik konvertiert werden, die die Analyse der Pipeline bei Fehlern vereinfacht. Ein Beispiel für eine solche Grafik findet sich in Abbildung 4.8.

gstrtpbinsrc definiert eine RTP-Bin (Siehe Abschnitt 2.5.2) mit dem Namen `rtpbin`. Zu beachten ist, dass nach dieser Definition kein Ausrufezeichen folgt, der Ausgang der RTP-Bin bleibt also zunächst unverknüpft.

udpsrc ist eine Netzwerk-Quelle. Hier im Beispiel werden, direkt als Parameter, Caps übergeben. So wird zu Beginn der Pipeline sichergestellt, dass nur die entsprechenden Daten weitergeleitet werden, nämlich RTP-Pakete mit Audiodaten, Payload-Typ 0 (was PCMU entspricht [RFC3551]), mit einer Abtastrate von 8kHz, auf Port 5000.

rtpbin.recv_rtp_sink_0 verweist auf die zuvor definierte RTP-Bin und zwar explizit auf den `recv_rtp_sink_0`, was dem Sink-Pad für RTP-Daten mit der Nummer 0 entspricht. Das Ausrufezeichen am Ende des vorangestellten Elementes `udpsrc` verknüpft also das Source-Pad von `udpsrc` mit einem explizit definierten Sink-Pad von `rtpbin`. Hierauf folgt kein Ausrufezeichen, es findet also keine Verknüpfung mit dem Sink-Pad des Folgelements statt. Einrückungen und Zeilenumbrüche sind aus Gründen der Lesbarkeit eingefügt worden.

```

6 rtpbin. !
7   rtppcmudepay !
8   mulawdec !
9   liveadder name=mix1 !
10  libvisual_lv_scope !
11  ffmpegcolorspace !
12  'video/x-raw-yuv, framerate=25/1 ,width=1024 ,height=786' !
13  xvimagesink

```

rtpbin. verweist ebenfalls auf die zuvor definierte RTP-Bin. Diesmal jedoch auf das Element insgesamt. Das Ausrufezeichen rechts davon verknüpft es mit dem darauf folgenden Element. Werden, wie hier, keine Pads explizit angegeben, verknüpft GStreamer diese automatisch.

rtppcmudepay depaketiert die PCMU-Daten aus RTP-Bin.

mulawdec decodiert die PCMU-Daten zu unkomprimierten Audiodaten.

liveadder definiert einen Audiomischer mit dem Namen „mix1“. Das Element links davon liegt bereits an einem Sink-Pad des Mixers an. Das Element rechts davon an dessen Source-Pad.

libvisual_lv_scope stellt die Wellenform des am Sink-Pad eingegebenen Audiosignals dar und generiert ein entsprechendes Videosignal, das auf dem Source-Pad ausgegeben wird.

ffmpegcolorspace konvertiert den Farbraum des eingegebenen Videosignals.

xvimagesink gibt das eingegebene Videosignal auf dem Bildschirm aus. Zu beachten ist hier wieder, dass nach diesem Element kein Ausrufezeichen folgt, da das Element das letzte Element in der Pipeline ist. Die nachfolgenden Elemente befinden sich an früherer Stelle innerhalb der Pipeline (Siehe Abbildung 4.3; Source-Pads sind in weiß, Sink-Pads in grau dargestellt).

```

14 rtpbin. !
15   rtppcmudpay !
16   mulawdec !
17   mix1.

```

rtpbin. verweist auf die zuvor definierte RTP-Bin, darauf folgen wieder Elemente zum Decodieren (**rtppcmudpay** und **mulawdec**)

mix1. greift auf den bereits definierten Audiomischer zu und belegt so das zweite Sink-Pad des Mischers mit dem Element links von ihm.

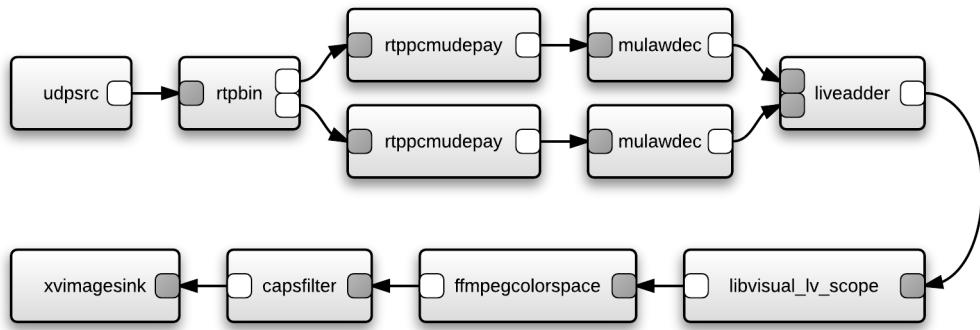


Abbildung 4.3: Pipeline zum Mischen zweier, über das Netz empfangener Audioströme

An der grafischen Ausgabe der Sinustöne ist gut erkennbar, wie beide Sinustöne unterschiedlicher Frequenz gemischt werden (Siehe Abbildung 4.4). Es ist jedoch auch deutlich zu erkennen, wie die Spitzen der Signale abgeschnitten werden. Da beide Testsignale mit maximalem Pegel generiert werden, ist deren Summe, an Stellen, an denen 2 Wellenberge addiert werden, größer als der maximale Pegel. Dieses Phänomen wird Clipping (von engl. *to clip* - abschneiden) genannt.

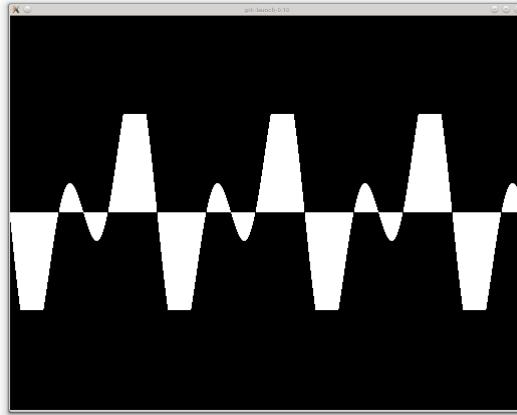


Abbildung 4.4: Gemischte Sinustöne

Um Clipping zu verhindern, kann das GStreamer-Plugin `leveler` verwendet werden, das den Pegel der einzelnen Audiosignale entsprechend anpasst.

4.4 Entwicklung

Bei Übernahme des Projekts existieren bereits einige Klassen aus einer abgeschlossenen Masterarbeit:

- VideoBridge
- Background
- FrameColor
- Participant
- ScreenResolution

VideoBridge und Participant müssen jedoch stark angepasst werden.

Als erstes wird der Audiomixer entwickelt. Dieser ist für das akustische Mischen der Audiosignale der Konferenzteilnehmer zuständig.

4.4.1 Vorüberlegungen

Audioanteil

Der Audiomixer hat die Aufgabe, die Audioströme aller Konferenzteilnehmer zu empfangen, zu mischen und zu den Teilnehmern zurückzusenden. Der Audiomixer kann nur uncodierte Audiodaten mischen. Deswegen müssen mehrere Schritte durchlaufen werden (Siehe Abbildung 4.5).



Abbildung 4.5: Prozess Audiomixer

Das Signal muss empfangen, depakettiert und decodiert werden, bevor es mit anderen Audiosignalen gemischt werden kann. Danach muss es wieder codiert und paketiert werden, damit es anschließend versendet werden kann.

Nicht ganz trivial ist das eigentliche Mischen der Audiosignale. Während alle Teilnehmer das gleiche Videosignal erhalten können (die Videos aller Teilnehmer inklusive sich selbst) wäre es sehr störend, wenn die Teilnehmer sich selbst, um einige Millisekunden verzögert, hören müssten, während sie sprechen.

Es soll daher vermieden werden, dass die Teilnehmer sich selbst hören können. Jeder Teilnehmer soll nur die gemischten Audiosignale der jeweils anderen Teilnehmer empfangen, nicht jedoch sein eigenes Audiosignal. Dies ist in Abbildung 4.6 schematisch dargestellt. Teilnehmer A empfängt nur die Audiosignale der Teilnehmer B, C und D, Teilnehmer B nur die Audiosignale der Teilnehmer A, C und D usw.

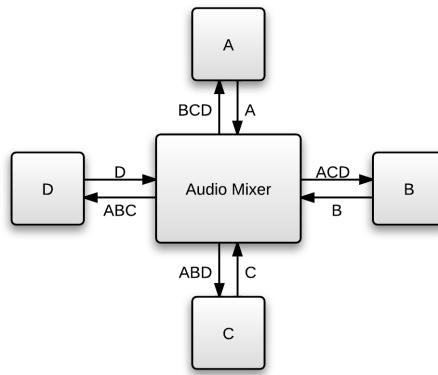


Abbildung 4.6: Mischen der Audiosignale

Es ist deswegen sinnvoll, wenn es für jeden Teilnehmer einen eigenen **liveadder** (Siehe Abschnitt 2.5.2) für das Mischen des Audiosignals gibt.

Beim Hinzufügen eines neuen Teilnehmers muss das Audiosignal, das dieser sendet, an die Liveadder aller anderen Teilnehmer, ausser seinem eigenen, „angeschlossen“ werden. Ebenso werden die Audiosignale, die alle anderen Teilnehmer senden, an den Liveadder des neuen Teilnehmers „angeschlossen“.

In GStreamer-Terminologie bedeutet das, dass sich die Pipeline verzweigt: Das depakettierte und decodierte Audiosignal wird durch ein Tee-Element (Siehe Abschnitt 2.5.2) verzweigt und in die **liveadder** der anderen Teilnehmer eingespeist, die Audiosignale aller übrigen Teilnehmer werden in den Liveadder des neuen Teilnehmers eingespeist.

Um zu testen, ob diese Arbeitsweise grundsätzlich wie gewünscht funktioniert, sollen mehrere verschiedene Testtöne unterschiedlicher Frequenz versendet, empfangen, entsprechend gemischt

und die Wellenformen der resultierenden Audiosignale grafisch (Siehe Abschnitt 2.5.2) dargestellt werden. Dazu wird die später in der eigentlichen Anwendung zu implementierende Pipeline in der Kommandozeile getestet.

Zunächst auftauchende Fehler sind einer fehlerhaften Syntax zuzuschreiben. Diese Fehler können mit Hilfe der GStreamer-Mailingliste [GSTR02] schnell gelöst werden. Im weiteren Verlauf ist durch die Ausgabe als Wellenform ein resultierendes Audiosignal zu erkennen, das darauf hindeutet, dass die Audiosignale nicht akustisch gemischt werden, sondern Abschnitte der einzelnen, unveränderten Originalsignale schlicht nacheinander versendet werden. Das lässt darauf schließen, dass der `liveadder` nicht richtig arbeitet. Durch die grafische Ausgabe der Pipeline (Siehe Abschnitt 4.3.1) wird als Ursache eine nicht korrekte Verknüpfung der Elemente der Pipeline ausgemacht und entsprechend korrigiert.

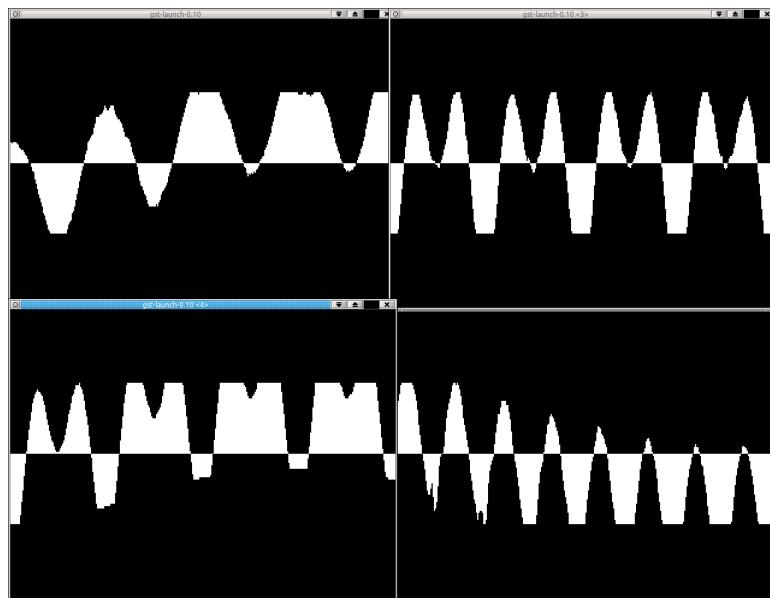


Abbildung 4.7: 4 Clients - 4 verschiedene Audiosignale

Wie in Bild 4.7 deutlich zu sehen ist, zeigen die 4 Fenster unterschiedliche Audioströme, nämlich jeweils das gemischte Signal dreier unterschiedlicher Sinustöne. Die folgende Pipeline beinhaltet im Gegensatz zu dem Screenshot nur 3 Quellen, da der Screenshot von einem anderen, jedoch prinzipiell gleichen, Test stammt.

Es folgt eine Beschreibung der gesamten Pipeline in Form des entsprechenden Kommandozeilenbefehls.

```

1 GST_DEBUG_DUMP_DOT_DIR=/tmp/ gst-launch --gst-debug-level=2
2 gstrtpbin name=rtpbin latency=2 sync=false

```

In den ersten beiden Zeilen wird das GStreamer Programm mit einigen Optionen zur Fehlersuche aufgerufen sowie eine RTP-Bin mit dem Namen `rtpbin` erstellt.

```

3 udpsrc name=udpsrc0 caps="application/x-rtp, media=audio,
4   clock-rate=8000, payload=0, encoding-name=PCM" port=5000 !
5   rtpbin.recv_rtp_sink_0

```

In diesem Teil wird eine UDP-Source mit dem Namen `udpsrc0` erstellt und entsprechende Capabilities mit übergeben. Deren Ausgang wird durch das Ausrufezeichen mit einem fest definierten Eingang (`recv_rtp_sink_0`) der zuvor definierten RTP-Bin verbunden.

```

6 rtpbin. !
7   rtppcmudepay !
8   mulawdec !
9   tee name=t0 !
10  queue max-size-buffers=150 !
11  liveadder name=mix1 !
12  mulawenc !
13  rtppcmupay !
14  udpsink name=udpsink1 host=192.168.0.101 port=5050

```

In diesem Abschnitt wird ein Ausgang der RTP-Bin mit Elementen zum Depaketieren und Decodieren verbunden, das resultierende Signal mit einem Tee-Element namens `t0` aufgeteilt. Eines der aufgeteilten Signale wird an ein Queue-Element angeschlossen und dann an einen `liveadder` mit Namen `mix1` angeschlossen. Das Ausgangssignal von diesem `liveadder` wird dann wieder codiert und paketiert und mit dem `udpsink1` an den Socket 192.168.0.1:5050 verschickt.

```

15 t0. !
16   queue max-size-buffers=150 !
17   liveadder name=mix2 !
18   mulawenc !
19   rtppcmupay !
20   udpsink name=udpsink2 host=192.168.0.112 port=5050

```

Das Ausgangselement in diesem Abschnitt ist das Tee-Element, dass in Zeile 9 definiert wurde. Eines dessen Source-Pads wird an das Sink-Pad eines neuen `liveadder mix2` angeschlossen. An dessen Source-Pad wiederum folgen Codieren, Paketieren und an den Socket 192.168.0.112:5050 Versenden mit `mulawenc`, `rtppcmupay` und `udpsink`.

```

21 rtpbin. !
22   rtppcmudepay !
23   mulawdec !
24   tee name=t1 !
25   queue max-size-buffers=150 !
26   liveadder name=mix0 !
27   mulawenc !
28   rtppcmupay !
29   udpsink name=udpsink0 host=192.168.0.101 port=5051

```

Ein neuer Ausgang (Source-Pad) der RTP-Bin führt unter anderem über ein neues Tee-Element **t1** und einen neuen **liveadder mix0**, um am Ende über ein neues UDP-Sink-Element **udpsink0** an den Socket 192.168.0.101:5051 verschickt zu werden.

```
30 t1. !
31   queue max-size-buffers=150 !
32   mix2.
```

In diesem Abschnitt wird ein Ausgang des in Zeile 24 definierten Tee-Elements über ein Queue-Element an den ebenfalls bereits definierten **liveadder** mit Namen **mix2** (Zeile 17) angeschlossen.

```
33 rtpbin. !
34   rtppcmudepay !
35   mulawdec !
36   tee name=t2 !
37   queue max-size-buffers=150 !
38   mix0.
```

Der Ausgang der RTP-Bin wird depaketiert, dekodiert durch ein neues Tee-Element **t2** aufgeteilt und einer der Ausgänge des Tee-Elements über ein Queue-Element mit dem **liveadder** namens **mix0** verbunden.

```
39 t2. !
40   queue max-size-buffers=150 !
41   mix1.
```

Ein weiteres Source-Pad von **t2** wird über ein Queue-Element in den in Zeile 11 definierten **liveadder** mit Bezeichnung **mix1** eingespeist.

Auch hier wurden wieder Zeilenumbrüche und Einrückungen aus Gründen der Lesbarkeit eingefügt, alle Pipelines befinden sich auf der beigelegten CD.

Ein Abschnitt der Pipeline für einen Teilnehmer ist in Abbildung 4.12 schematisch dargestellt.

Die Pipeline funktioniert beim Erstellen nicht auf Anhieb. Hier kommt hauptsächlich die „Trial and Error“-Methode zum Einsatz, unter Zuhilfenahme der gebotenen Debugging-Möglichkeiten, sprich: Ausgabe der Pipeline als DOT-Datei und anschließendes Konvertieren in eine Grafik (Siehe Abbildung 4.8) sowie die Ausgabe von Debugging-Meldungen (Siehe Abbildung 4.9). Eine spätere Version der Pipeline funktioniert, enthält jedoch zu viele Source-Pads, da beim Entfernen und wieder Hinzufügen von RTP-Quellen die alten Pads nicht entfernt werden. Beim Konsultieren der Hilfe mit **gst-inspect** fällt der Parameter **autoremove** ins Auge, der Pads nach einer gewissen Zeit der Inaktivität automatisch entfernt. Dieser Parameter wird im Programm verwendet und auf **true** für die RTP-Bin gesetzt.

Hier kommt ein bisweilen nachteiliger Aspekt von Open Source-Software zum Tragen. Da diese oft schlecht dokumentiert ist, ist der Anwender oder Entwickler auf Internetforen, Mailinglisten und das angesprochene „Trial and Error“ angewiesen. Dieser Aspekt ist je gravierender, desto spezieller die Software und damit desto kleiner die Anzahl an involvierten Personen ist.

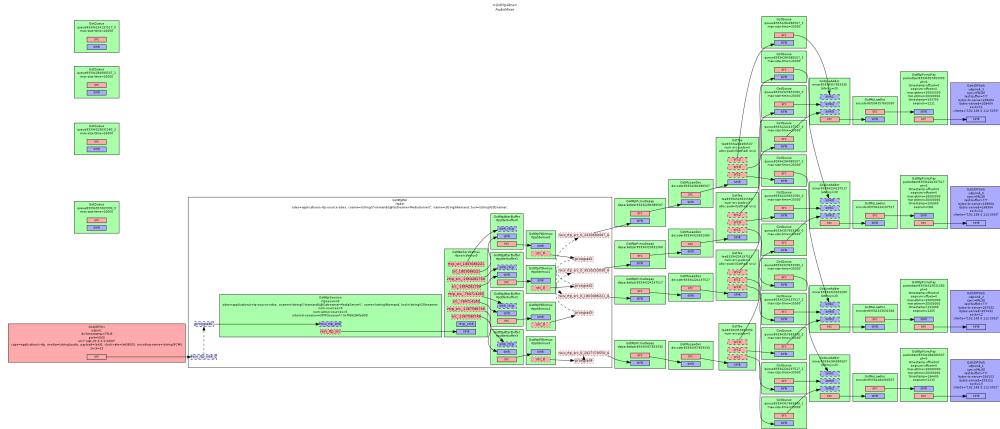


Abbildung 4.8: GStreamer Pipeline Grafik

Abbildung 4.9: GStreamer Debugging Stufe 3 Bildschirmausgabe

Videoanteil

Der Verarbeitungsprozess im Videoanteil des Mediaservers ist prinzipiell der Gleiche wie im Audioanteil (Siehe Abbildung 4.5). Hier erhalten alle Teilnehmer jedoch das gleiche Signal zurück, was die Verarbeitung etwas vereinfacht. Darüber hinaus ist die entsprechende Klasse in Grundzügen bereits vorhanden.

Aufbau des Programms

Gemäß dem objektorientierten Ansatz wird die logische Struktur weitestgehend in Klassen abgebildet. Als Hauptklasse, also die Klasse, die die `main()`-Funktion enthält, dient die Klasse `MediaServer`. Diese beinhaltet eine Klasse `Focus`, die die SIP-Signalisierung übernimmt. Ebenso beinhaltet die Klasse `MediaServer` beliebig viele Konferenzräume, in Form einer Liste von Instanzen der Klasse `ConferenceRoom`.

Die Klasse `ConferenceRoom` enthält dabei jeweils eine Instanz der Klassen `AudioMixer` und `VideoBridge` sowie maximal N `ConferenceMember`-Klassen, wobei N durch eine Variable eingestellt werden kann. Die Klasse `ConferenceMember` wiederum enthält die Klassen `ParticipantAudio` und `ParticipantVideo`, die den Audio- und den Videoanteil eines Teilnehmers repräsentieren.

Diese Struktur ist in Abbildung 4.10 schematisch dargestellt.

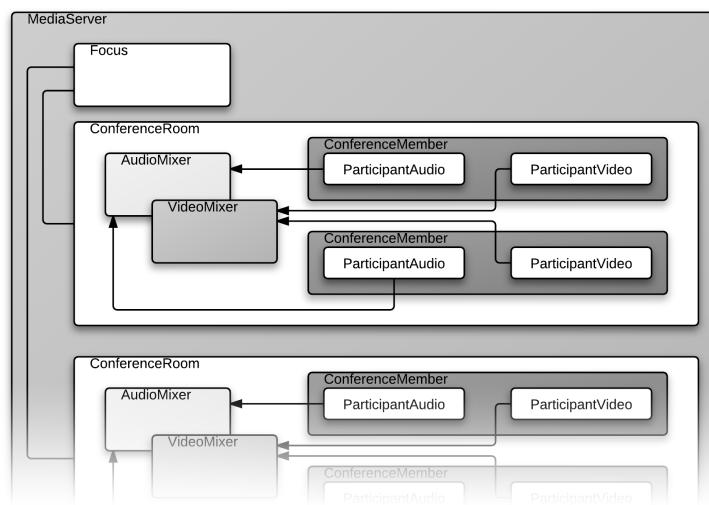


Abbildung 4.10: Klassenstruktur des Mediaservers

4.4.2 Klassen

Im folgenden Abschnitt werden Teile einiger Klassen erläutert. Der gesamte Quelltext befindet sich auf der beigelegten CD.

Participant

Der `Participant` ist die Basisklasse für den Medienanteil eines einzelnen Konferenzteilnehmers. Von dieser Klasse erben die beiden Klassen `ParticipantAudio` und `ParticipantVideo`, die den Audio- und Video-Anteil der Teilnehmerdaten darstellen (Siehe Abbildung 4.11).

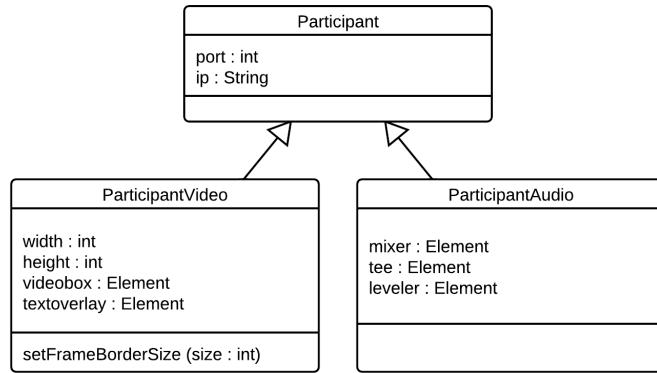


Abbildung 4.11: Participant UML Klassendiagramm

Anmerkung: Das Diagramm enthält aus Gründen der Übersichtlichkeit nur einige der Methoden und Felder der tatsächlichen Klasse.

ParticipantAudio

Die Klasse **ParticipantAudio** enthält all die GStreamer Elemente, die zur Verarbeitung der Audiodaten eines Teilnehmers nötig sind, in Form einer Liste. Diese Liste lässt sich über die Funktion `getElements()` von außerhalb der Klasse abrufen. Tritt ein neuer Teilnehmer der Konferenz bei, werden in der Klasse **AudioMixer** die Elemente so verknüpft, dass dieser nur die anderen Teilnehmer und nicht sich selbst hören kann. Dies ist in Abbildung 4.12 dargestellt.

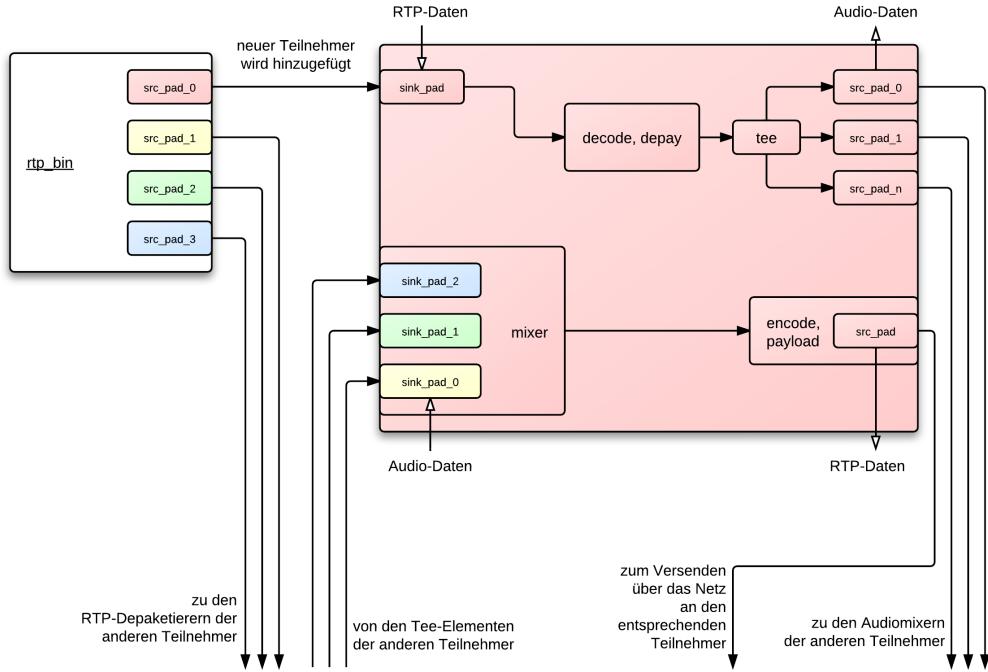


Abbildung 4.12: Funktionsdiagramm Participant

ParticipantVideo

`ParticipantVideo` ist die Klasse, die all die GStreamer-Elemente für die Verarbeitung des Videosignals enthält. Wie bei `ParticipantAudio` können diese Elemente über eine Funktion `getElements()` von außerhalb der Klasse abgerufen werden. Wenn ein neuer Teilnehmer der Konferenz beitritt, verknüpft die Klasse `VideoBridge` diese Elemente entsprechend, so dass eine vollständige Pipeline entsteht. Die Videosignale aller Teilnehmer werden über- und nebeneinander dargestellt, die jeweiligen Namen der Teilnehmer aus der SIP-INVITE-Anfrage über dem Bild jedes Teilnehmers eingeblendet und jedem Teilnehmer das gleiche Video zurückgesendet.

MediaProcessor

Während die Klassen `AudioMixer` und `VideoBridge` zunächst nebeneinander entwickelt werden, ergeben sich immer mehr Gemeinsamkeiten und dementsprechend Redundanzen, weswegen eine Klasse `MediaProcessor` eingefügt wird, von der beide einige Eigenschaften erben sollen. So haben beide Klassen einen Port, auf dem diese die Mediendaten empfangen sollen. Beide haben darüber hinaus eine GStreamer Pipeline, eine maximale Anzahl an Teilnehmern und auch gleiche Methoden, wie etwa zum Hinzufügen oder Entfernen von Teilnehmern (Siehe Abbildung 4.13). Die Klasse `MediaProcessor` implementiert ebenfalls eine Methode, zum Feuern eines Events, wenn die Initialisierung abgeschlossen ist. Dieses Event nutzen `AudioMixer` und `VideoBridge`, um der Klasse `MediaServer` mitzuteilen, dass ihre Initialisierung abgeschlossen ist. Diese läuft in separaten Threads und dauert länger als die Initialisierung der `MediaServer`-Klasse, was auf den Start

von GStreamer zurückzuführen ist.

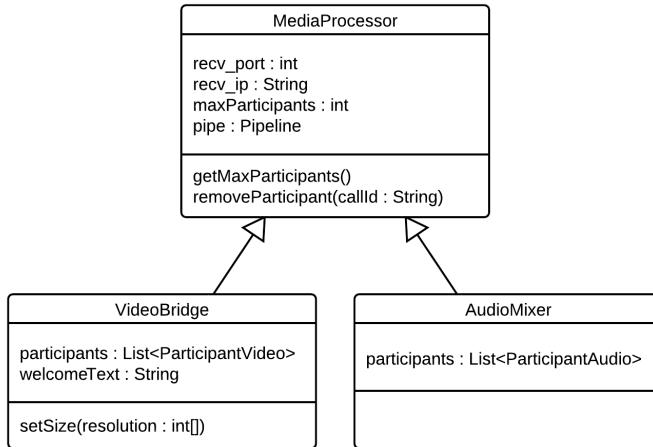


Abbildung 4.13: MediaProcessor UML Klassendiagramm

Anmerkung: Das Diagramm enthält aus Gründen der Übersichtlichkeit nur einige der Methoden und Felder der tatsächlichen Klasse.

Methode getSsrcFromPadName Die SSRC im Header eines RTP Paketes dient der eindeutigen Identifikation einer Quelle. Der Name eines Source-Pads, das von der GStreamer RTP-Bin automatisch erstellt wird, sobald ein neuer RTP-Stream empfangen wird, enthält diese SSRC. Die Benennung des Source-Pads erfolgt durch GStreamer nach dem folgenden Schema:

```
recv_rtp_src_x_y_z
```

Dabei sind `x` = Session-Nummer, `y` = SSRC und `z` = Payloadtype-Nummer. Die SSRC soll in der jeweiligen Instanz der Participant-Klasse (Audio und Video) gespeichert werden, um den Teilnehmer jederzeit identifizieren zu können. Dazu extrahiert die Funktion `getSsrcFromPadName` die SSRC mit String-Funktionen aus dem Pad-Namen.

```

1 protected String getSsrcFromPadName(String padname){
2     String returnValue;
3     int beginIndex=padname.indexOf("_", 13)+1;
4     int endIndex = padname.indexOf("_", 15);
  
```

Die Variable `returnValue` wird erstellt, um später den Rückgabewert aufzunehmen. Daraufhin werden 2 Integer-Variablen erstellt und mit Werten belegt. Die erste Variable, `beginIndex`, mit der Position des ersten Vorkommens eines Unterstrichs, gezählt ab dem 13. Zeichen. Danach wird eine 1 addiert.

Die Zeichenkette „`recv_rtp_src_`“, mit der der Name des Pads beginnt, hat 13 Zeichen. Anschließend folgt per Definition eine Session-Nummer, deren Stellenanzahl unbekannt ist, danach

wieder ein Unterstrich und darauf folgt die SSRC, die es zu extrahieren gilt. Der erste Unterstrich *nach* dem 13. Zeichen (daher die addierte 1 am Ende) ist also der Unterstrich *vor* der SSRC.

Die Variable `endIndex` wird im Anschluss mit dem ersten Vorkommen eines Unterstrichs ab dem 15. Zeichen belegt, was dem ersten Unterstrich *nach* der SSRC entspricht.

```

5   try {
6     returnValue = padname.substring(beginIndex, endIndex);
7   } catch (StringIndexOutOfBoundsException e) {
8     returnValue = null;
9   }
10  return returnValue;
11 }
```

Nun wird die Variable `returnValue` mit dem belegt, was zwischen den vorher belegten Variablen `beginIndex` und `endIndex` liegt, also der SSRC. Sollte es zu einer `StringIndexOutOfBoundsException` kommen, wenn etwa der Name des Pads eine Länge kleiner als 13 Zeichen hat, wird der Rückgabewert mit `null` belegt.

Am Ende wird `returnValue` zurückgegeben.

AudioMixer

An dieser Stelle muss zunächst eine begriffliche Unterscheidung vorgenommen werden. Die Klasse innerhalb der Mediaserver-Applikation, die das Mischen der Audiosignale organisiert und entsprechende Funktionen bereitstellt, ist die Klasse `AudioMixer`. Eine Instanz dieser Klasse existiert einmal in jeder Instanz der Klasse `ConferenceRoom`.

Das eigentliche Mischen der Audiosignale wird jedoch vom GStreamer-Plugin `liveadder` durchgeführt. Instanzen dieses Elementes sind in einer Instanz der Klasse `ParticipantAudio` zur Laufzeit je einmal vorhanden.

Die Klasse `AudioMixer` ist als organisatorische Einheit zu verstehen, die bei Hinzufügen eines neuen Teilnehmers den Audiopfad entsprechend verknüpft.

Werden von der Klasse `AudioMixer` RTP-Daten eines neuen Teilnehmers empfangen, löst dies eine ganze Reihe an Aktionen aus. Hier dargestellt ist der Listener für das Event `PAD_ADDED` der RTP-Bin, das ausgelöst wird, wenn ein neues Pad hinzugefügt wird. Ein neues Pad wird unter anderem dann automatisch hinzugefügt, wenn die RTP-Bin die RTP-Daten eines neuen Teilnehmers empfängt. Der Quelltext wurde gekürzt, um die Lesbarkeit zu verbessern.

```

1 rtpbin.connect(new Element.PAD_ADDED() {
2   @Override
3   public void padAdded(Element element, Pad pad) {
```

Die Funktion `padAdded` wird mit dem Schlüsselwort `@Override` überschrieben. Diese Funktion wird ausgelöst, wenn im Element `rtpbin` das `PAD_ADDED`-Ereignis ausgelöst wird, also immer dann, wenn der RTP-Bin ein neues Pad hinzugefügt wird.

```

4   Element[] elements;
5   ListIterator<ParticipantAudio> it = audioParticipants.listIterator();
6   ParticipantAudio pa = null;
7   ParticipantAudio p = null;

```

Hier werden die Variablen erstellt und belegt. Elements ist ein Array, das später die GStreamer-Elemente enthalten soll, der `ListIterator` `it` dient dem Durchlaufen der Liste von Participant-Audio-Instanzen. Die Variablen `p` und `pa` werden benutzt, um temporär einen ParticipantAudio zu speichern.

```

8   while(it.hasNext()){
9     pa = it.next();
10    if(!(pa.getCallId()==null)){
11      if(!(pa.getElements()[0].getStaticPad("sink").isLinked())){
12        p = pa;
13      }
14    }
15  }

```

Mit einer while-Schleife wird die ParticipantAudio-Liste durchlaufen. Jedes Listenelement wird geprüft, ob dessen Call-ID nicht null ist. Ist das der Fall, wird außerdem geprüft, ob das Sink-Pad des ersten GStreamer-Elements dieses Listenelements (nämlich der Depaketierer) bereits verknüpft ist. Ist das nicht der Fall, wird dieser ParticipantAudio vorgemerkt.

Dadurch wird der für die Verbindung vorgesehene ParticipantAudio (Call-ID gesetzt, aber noch nicht in die Pipeline eingebunden) gefunden.

```

16   p.getUdpsink().set("clients", p.socket());
17   try{
18     elements = p.getElements();
19   }

```

Die Methode `getUdpsink()` gibt den UDPSink des ParticipantAudio zurück. Diesem wird über seine `set`-Methode der Socket zugewiesen, an den der UDPSink später seinen RTP-Stream senden soll. Dieser Socket wurde zuvor per SIP gesetzt. Anschließend wird die Liste der GStreamer-Elemente noch der zuvor definierten Liste `elements` zugewiesen, um die weitere Verwendung zu vereinfachen.

```

20   if (pad.getName().indexOf("recv_rtp_src")>-1){
21     linkOk = pad.link(elements[0].getStaticPad("sink"));
22     return;
23   }

```

Hier wird nun schließlich der Name des neu hinzugefügten Pads geprüft, um festzustellen, was für eine Art von Pad es ist (Die RTP-Bin verfügt über eine ganze Reihe möglicher Pads). Ist das Pad ein Source-Pad für empfangene RTP-Pakete (`recv_rtp_src`), wird es mit dem Sink-Pad des Depaketierers des vorgemerkten ParticipantAudio verknüpft.

```

24     if (pad.getName().indexOf("send_rtp_sink")>-1){
25         linkOk = elements[6].getStaticPad("src").link(pad);
26         return;
27     }
28 }
29 });

```

Hier wird nochmals der Name des Pads geprüft. Ist das Pad ein Sink-Pad zum Senden von RTP-Paketen (`send_rtp_sink`), wird es mit dem Source-Pad des Paketierers des ParticipantAudio verknüpft.

Zwischentest Nach Fertigstellung des AudioMixers wird dieser mit dem Ekiga Softphone getestet. Der bis zu diesem Zeitpunkt nur rudimentär implementierte Focus kann lediglich auf SIP-Requests antworten, interagiert jedoch noch nicht mit dem AudioMixer. Deswegen wird der SDP-Body, den der Focus an seine OK-Antwort anhängt, manuell erstellt. Port und Medienbeschreibung werden so hinterlegt, dass der AudioMixer bereits an der richtigen Adresse lauscht. Der so durchgeführte Test verläuft erfolgreich. Im Screenshot Abbildung 4.14 zeigt Ekiga den verwendeten Codec (PCMU) an.

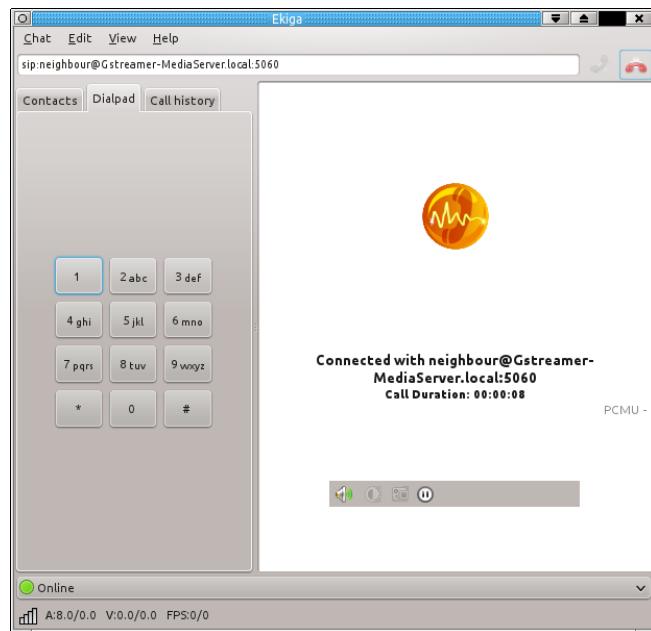


Abbildung 4.14: Ekiga Softphone

Focus

Der sogenannte Focus ist der Teil der Konferenzlösung, der für die SIP-Kommunikation zuständig ist und dementsprechend AudioMixer und VideoBridge steuert. Als Basis für den Focus dient

ein Beispiel aus der JAIN-SIP Bibliothek, das bereits Funktionen für die Reaktion auf die verschiedenen Anfragen enthält und entsprechend angepasst wird. Der anfängliche Versuch, den Focus komplett selbst zu entwickeln, stellt sich als extrem aufwändig heraus und wird nach 2 Wochen verworfen.

Funktionsbeschreibung Focus Der Focus muss nur auf die Anfragen INVITE, BYE, ACK und CANCEL reagieren. Erhält der Focus ein INVITE, versucht er, einen Teilnehmer zur Konferenz hinzuzufügen. Ein BYE entfernt den Teilnehmer aus der Konferenz. ACK ist die letztmalige Bestätigung in einem 3-Wege-Handshake und CANCEL bricht die aktuelle Anfrage ab.

INVITE Erhält der Focus eine INVITE-Anfrage, sendet er dem Teilnehmer sofort ein 100 TRYING zurück. Als nächstes extrahiert er aus der SIP URI, an die die Anfrage gestellt wurde (TO-Header), den Namen des Konferenzraums. Wird ein Anzeigename mit übergeben, wird dieser verwendet, andernfalls der User-Anteil der URI. Anschließend wird der Raumname an eine Funktion übergeben, die prüft, ob ein Konferenzraum mit diesem Namen existiert. Das heißt, ob eine Instanz der Klasse ConferenceRoom existiert, deren Feld `roomName` den gleichen Wert wie der extrahierte Name hat. Ist das der Fall, wird dieser Raum zurückgegeben. Andernfalls wird eine neue Instanz der Klasse ConferenceRoom erstellt, das Feld `roomName` entsprechend gesetzt und dieser Raum dann zurück gegeben. Der Teilnehmer kann so bestimmen, welchem Konferenzraum er beitreten möchte. Als nächstes wird geprüft, ob dieser Raum Kapazitäten frei hat. Ist das der Fall, wird dem Raum ein neuer Teilnehmer hinzugefügt, indem dessen IP-Adresse und Port (aus dem SDP-Body der INVITE-Nachricht) in dem Raum vorgemerkt werden. Als letztes erhält der Teilnehmer ein 200 OK zurück. Dieser Ablauf ist in Abbildung 4.15 dargestellt.

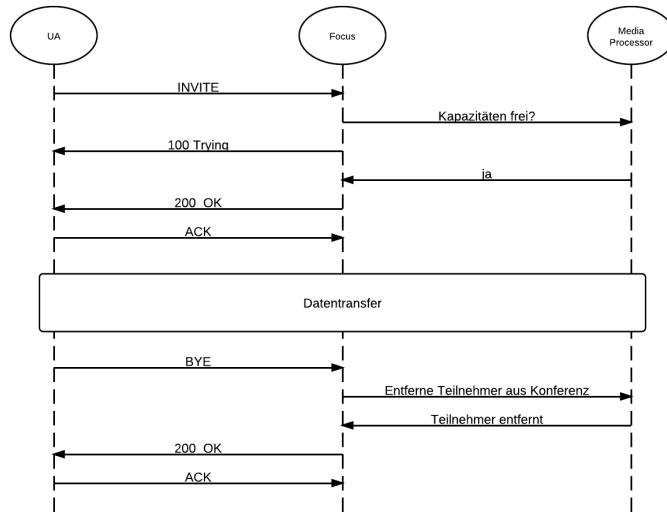


Abbildung 4.15: Verbindungsaufbau erfolgreich, anschließend Verbindungsabbau

Sind keine Kapazitäten frei, wird dem Teilnehmer ein 486 BUSY HERE zurück gesendet, was in Abbildung 4.16 dargestellt ist.

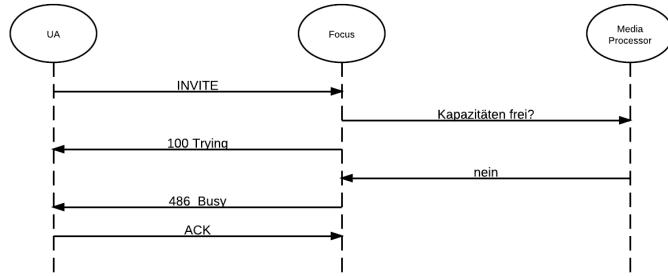


Abbildung 4.16: Verbindungsauftbau nicht erfolgreich

Im Quelltext wird diese Funktionalität in der Methode `processInvite` abgebildet. Diese Methode stammt aus einer Beispielanwendung der JAIN-SIP-Bibliothek, wurde aber stark angepasst. Der abgebildete Quelltext wird aus Gründen der Lesbarkeit um weniger wichtige Stellen (wie etwa Bildschirmausgaben und einige Kommentare) gekürzt.

```

1 public void processInvite(RequestEvent requestEvent,
                           ServerTransaction serverTransaction) {
2     SessionDescription sessionDescription;
3
4     SipProvider sipProvider = (SipProvider) requestEvent.getSource();
5     Request request = requestEvent.getRequest();
  
```

Hier wird die Methode deklariert und aus dem ihr übergebenen Request-Event das Request-Objekt sowie der SIP-Provider extrahiert. Ebenso wird das `SessionDescription`-Objekt erstellt.

```

7     try {
8         Response response = messageFactory.createResponse(Response.TRYING, request);
9         ServerTransaction st = requestEvent.getServerTransaction();
10        st.sendResponse(response);
  
```

Mit einem `try`-Block umgeben, wird als Erstes eine 100-TRYING-Antwort auf die INVITE-Nachricht erstellt und dann innerhalb der entsprechenden Transaktion versendet.

```

11     String roomName = null;
12     roomName = getRoomNamefromToHeader(request.getHeader(ToHeader.NAME));
13     ConferenceRoom room = MediaServer.getRoomByName(roomName);
14
15     if (room.slotsAvailable()){
16         String sdpRequestContent = new String(request.getRawContent());
17         SessionDescription sdpRequestSessionDescription =
18             sdpFactory.createSessionDescription(sdpRequestContent);
19
20         int audioRemotePort =
21             getMediaDescriptionPort(sdpRequestSessionDescription, "audio");
22         int videoRemotePort =
23             getMediaDescriptionPort(sdpRequestSessionDescription, "video");
24
25         String remoteIp =
26             sdpRequestSessionDescription.getConnection().getAddress();
27
28         room.addParticipant(remoteIp,
29             audioRemotePort,
30             videoRemotePort,
31             request.getHeader(CallIdHeader.NAME).toString(),
32             getParticipantNameByFromHeader(request.getHeader(FromHeader.NAME)));
33
34         String sdpResponseContent = createSdpResponse();
35         byte[] sdpResponseContentBytes = sdpResponseContent.getBytes();

```

In diesem Block wird zunächst ein Konferenzraum bestimmt. Der Name des gewünschten Raumes wird mit der Funktion `getRoomNamefromToHeader` aus der angerufenen SIP URI (aus dem To-Header der INVITE-Nachricht) extrahiert. Danach wird dieser Raum zurückgegeben, falls er existiert oder erst erstellt und dann zurückgegeben, falls er nicht existiert. Das ist in Funktion `getRoomByName` der Klasse MediaServer abgebildet.

Wenn in dem zurückgegebenem Raum Plätze frei sind (Funktion `slotsAvailable`), werden aus dem SDP-Body der INVITE-Nachricht die Ports, auf dem der Teilnehmer Audio und Video zu senden beabsichtigt, extrahiert (Funktion `getMediaDescriptionPort`). Anschließend wird noch die IP-Adresse des Teilnehmers extrahiert.

Daraufhin wird die Funktion `addParticipant` des Raumes, also der entsprechenden Instanz der Klasse `ConferenceRoom`, aufgerufen und dieser die IP, die Ports für Audio und Video, die Call-ID und den Namen des Teilnehmers aus dem From-Header der INVITE-Nachricht übergeben.

```

36    // C-TYPE
37    ContentTypeHeader contentTypeHeader = null;
38    contentTypeHeader =
39        headerFactory.createContentTypeHeader("application", "sdp");
40
41    this.okResponse = messageFactory.createResponse(Response.OK, request);
42    Address address = addressFactory.createAddress(SIP_URI);
43
44    // ALLOW
45    AllowHeader allowHeader =
46        headerFactory.createAllowHeader("INVITE,BYE,ACK,CANCEL");
47    okResponse.addHeader(allowHeader);
48
49    // CONTACT
50    ContactHeader contactHeader = headerFactory.createContactHeader(address);
51    System.out.println("contactHeader: " + contactHeader);
52    okResponse.addHeader(contactHeader);
53
54    // TO
55    ToHeader toHeader = (ToHeader) okResponse.getHeader(ToHeader.NAME);
56    toHeader.setTag("Conference-Server");
57
58    okResponse.setContent(sdpResponseContentBytes, contentTypeHeader);
59
60    this.inviteTid = st;
61    this.inviteRequest = request;
62    this.sendInviteOK();

```

An dieser Stelle wird der Header erstellt und die notwendigen Header-Felder gesetzt. Dieser Abschnitt wird nahezu unverändert übernommen. Der ALLOW-Header wird jedoch angepasst, um den UACs mitzuteilen, welche Requests der Server akzeptiert.

ALLOW-Header Beim Testen werden von den eingesetzten UACs an den Server immer wieder Requests gesendet, für die der Server keine Handlungsanweisungen besitzt und auch nicht benötigt. Um diese Anfragen zu unterbinden, wird dem UAC in der ersten Antwort auf eine von ihm gestellte Anfrage eine ALLOW-Header mitgeschickt. Dieser enthält alle Requests, die der Server zu beantworten im Stande ist. Im Falle des Mediaservers sind das lediglich INVITE, BYE, ACK und CANCEL. Der UAC speichert diese Informationen und sendet fortan nur noch entsprechende Anfragen an den Server.

```

63 } else {
64
65     this.okResponse =
66         messageFactory.createResponse(Response.BUSY_HERE, request);
67     ToHeader toHeader = (ToHeader) okResponse.getHeader(ToHeader.NAME);
68     toHeader.setTag("Conference-Server");
69
70     this.inviteTid = st;
71     this.inviteRequest = request;
72     this.sendInviteOK();
73 }
74 }
75 }
```

Hier schließlich ist das **else**-Statement der anfänglichen **if**-Abfrage (ob im angeforderten Konferenzraum Plätze frei sind). Sind keine Plätze frei, wird dem Teilnehmer eine BUSY HERE Antwort zurückgeschickt.

Verbinden von nicht-authentifizierten Clients In der bisherigen Planung wird einem Umstand nicht Rechnung getragen: Was, wenn ein Teilnehmer von sich aus RTP-Mediendaten an den Server schickt, ohne sich vorher per SIP „angemeldet“ zu haben? Bisher würde der Teilnehmer trotzdem zur Konferenz hinzugefügt, da MediaProcessor immer auf eingehende RTP-Ströme lauscht.

Diese Möglichkeit ist zwar überaus unwahrscheinlich, da der Teilnehmer die exakten Verbindungsdaten, also Port, IP-Adresse und Codec, kennen müsste, aber nicht gänzlich ausgeschlossen.

Es wird zunächst versucht, den RTP-Verkehr so zu filtern, dass RTP-Pakete an die lokale Adresse 127.0.0.1 umgeleitet werden, wenn diese von einem zuvor per SIP authentifizierten Teilnehmer stammen. GStreamer wird so konfiguriert, dass es nur auf der lokalen Adresse 127.0.0.1 lauscht.

Dieser Versuch wird allerdings nach ersten Tests verworfen. Die Untersuchung jedes einzelnen Pakets mittels Java DatagramSocket stellt sich als so langsam heraus, dass der UAC des Teilnehmers die Verbindung abbricht.

Eine andere Variante über Java Non-Blocking IO stellt sich als zu ambitioniert heraus, um sie in der begrenzten Zeit zu implementieren.

Um dieses Problem zu adressieren, sei hier auf das SIP-Netzelement Session Border Controller verwiesen, das in Abschnitt 2.1.3 behandelt wird.

Zwischentest Nach Fertigstellung des Focus wird seine ordnungsgemäße Funktion mit dem Programm `sendip` getestet. Es wird dazu eine SIP INVITE Nachricht in Form einer Textdatei erstellt und an den Server gesendet. Die gesamte Kommunikation wird mit Wireshark überwacht. Der Kommandozeilenbefehl dazu lautet folgendermaßen:

```
sudo sendip -v -f ~/Dropbox/Studium/Diplomarbeit/DOC/invite.txt
192.168.0.91 -p ipv4 -is 192.168.0.112 -p udp -us 5050 -ud 5050
```

Die referenzierte Datei invite.txt enthält folgenden Text:

```
1 INVITE sip:41215500309@192.168.1.91 SIP/2.0
2 Via: SIP/2.0/UDP 192.168.1.9;branch=z9hG4bKfae8cb69f547b8cb
3 Max-Forwards: 70
4 To: <sip:41215500309@192.168.1.91>
5 From: <sip:41215500311@192.168.1.112>;tag=102
6 User-Agent: UDP Packet Sender
7 Call-ID: 070403-200101@192.168.1.9
8 CSeq: 5000 INVITE
9 Contact: <sip:41215500311@192.168.1.9>
10 Content-Type: application/sdp
11 Content-Length: 0
```

Als Reaktion soll der Focus zunächst nur den Empfang des INVITE-Paketes erkennen und mit einem 100 TRYING antworten, was auf Anhieb funktioniert. Anschließend wird die Integration des Focus mit dem Mediaprocessor geschrieben. Auch diese wird nach Fertigstellung getestet.

Nach einigen missglückten Versuchen, die jedoch allesamt ihre Ursache in Fehlern im SDP-Body der SIP-INVITE hatten, klappt der 3-Wege-Handshake. Dies ist in Abbildung 4.17 dargestellt.

Filter: sip rtp	Expression...	Clear	Apply			
.	Time	Source	Destination	Protocol	Length	Info
22 3.568254	192.168.0.50	192.168.0.91		SIP/SDP		1187 Request: INVITE sip:conf1@192.168.0.50, with session description
23 3.586412	192.168.0.91	192.168.0.50		SIP		371 Status: 100 Trying
25 4.287725	192.168.0.91	192.168.0.50		SIP/SDP		684 Status: 200 OK, with session description
30 4.295041	192.168.0.50	192.168.0.91		SIP		517 Request: ACK sip:conf1@192.168.0.91
31 4.302124	192.168.0.91	192.168.0.50		H264		56 PT=H264, SSRC=0x5E6B606B, Seq=20163, Time=28571
32 4.302167	192.168.0.91	192.168.0.50		H264		61 PT=H264, SSRC=0x5E6B606B, Seq=20164, Time=28571
33 4.302193	192.168.0.91	192.168.0.50		H264		62 PT=H264, SSRC=0x5E6B606B, Seq=20165, Time=28571

Abbildung 4.17: 3-Wege Handshake in Wireshark

Wie deutlich zu erkennen ist, antwortet der Server mit der IP-Adresse 192.168.0.91 auf den INVITE-Request vom UAC (Bosch Communicator) auf dem Client mit der IP-Adresse 192.168.0.50 zunächst mit der provisorischen Response 100 TRYING und erst dann mit der endgültigen 200 OK Antwort. Deren Empfang wird vom Bosch Communicator umgehend mit einem ACK quittiert. Hier ist ebenfalls gut zu erkennen, dass die INVITE-Anfrage des Clients und die OK-Antwort des Servers jeweils einen SDP-Body haben, mit dem die Medien-Informationen ausgetauscht und ausgehandelt werden. Direkt nach der erfolgreichen Initiierung fangen Client und Server an, RTP-Pakete auszutauschen. Hier im Bild sind noch 3 H.264-Pakete vom Server an den Client zu erkennen.

ConferenceMember

Diese Klasse ist in erster Linie organisatorischer Natur. Sie repräsentiert einen Konferenzteilnehmer und enthält jeweils eine Instanz von ParticipantAudio und ParticipantVideo, die im Konstruktor der Klasse erzeugt werden.

VideoBridge

Die Klasse VideoBridge war schon vorhanden, muss allerdings stark angepasst werden. So werden etwa dem Codierer einige Parameter übergeben, um die Geschwindigkeit zu erhöhen und ein Event ausgelöst, wenn die Initialisierung des Klasse abgeschlossen ist.

Methode start Im Folgenden wird ein Teil der `start()`-Methode der Klasse VideoBridge beschrieben. Hier ist gut zu erkennen, wie die Erstellung einer GStreamer-Pipeline im Programm erfolgt. Außerdem enthält die Methode einen Abschnitt zum Laden von Werten aus einer Config-Datei.

```

1 // get configuration from Config File
2 PropertiesConfiguration config = new PropertiesConfiguration();
3 config.load(PROPERTIES_FILE_NAME);
4
5 RECV_PORT=config.getInt("VIDEO_RECV_PORT");
6 WELCOMETEXT=config.getString("WELCOMETEXT");
7 RECV_IP = config.getString("IP_ADDR");

```

In diesem Abschnitt wird die Konfigurationsdatei, deren Name in der Variable `PROPERTIES_FILE_NAME` gespeichert ist, geladen. Im Anschluss werden Empfangs-Port (`RECV_PORT`), Willkommenstext (`WELCOMETEXT`) und Empfangs-IP-Adresse (`IP_ADDR`) aus dieser extrahiert und den entsprechenden Variablen zugewiesen.

```

8 // create the Pipeline
9 pipe = new Pipeline("VideoForwarder");
10
11 // create elements for sending out the data
12 mixer = ElementFactory.make("videomixer", "mixer");
13 mixer.set("background", "1");
14
15 encoder = ElementFactory.make("x264enc", "encoder");
16 encoder.set("tune", 4);
17 encoder.set("bitrate", 4096);
18 encoder.set("byte-stream", true);
19 encoder.set("profile", 1);
20 encoder.set("speed-preset", 1);
21 encoder.set("psy-tune", 4);

```

Hier wird die Pipeline erstellt. Dabei wird dieser der Name „VideoForwarder“ übergeben. Im Anschluss werden die GStreamer Elemente `videomixer` und `x264enc` erstellt und konfiguriert.

Die Funktion `ElementFactory.make` erwartet dabei 2 Strings, wobei der erste String die Bezeichnung des Elements und der zweite String der frei definierbare Name der Instanz des Elements ist.

Dem Codierer werden hier noch einige Parameter übergeben, die einen maßgeblichen Einfluss auf die Video-Performance haben. Der Parameter `tune` betrifft nicht-psychovisuelles Tuning, die möglichen Werte reichen von 0 (kein Tuning) bis 4 (latenzfrei). Da einer geringen Latenz gegenüber der Videoqualität in diesem Fall der Vorzug gegeben wird, ist hier die 4 gewählt worden.

Die Bitrate wird mit 4096 kbit pro Sekunde festgelegt. Das entspricht 4 Mbit pro Sekunde, was eine für ein 100 Mbit LAN angemessene Geschwindigkeit darstellt.

Als H.264-Profil wird über den Parameter `profile` Baseline mit einer 1 eingestellt.

Beim Profil für die Geschwindigkeit (`speed-preset`) wird ebenfalls die schnellstmögliche Variante gewählt. Die Varianten reichen hier von unter anderem 0 (kein Preset) über 3 (sehr schnell) und 7 (langsam) bis zu 10 („Placebo“). [GSTR03]

```

22 packetizer = ElementFactory.make("rtph264pay", "packetizer");
23 packetizer.set("pt", 123);
24 packetizer.set("scan-mode", 1);

25
26 multiudpsink = ElementFactory.make("udpsink", "sink");
27 multiudpsink.set("sync", false);

28
29 // create the elements for the background
30 background = ElementFactory.make("videotestsrc", "Source");
31 background.set("pattern", Background.BLACK);
32 background.set("is-live", "true");

33
34 backgroundCapsfilter = ElementFactory.make("capsfilter", "backgroundcaps");
35 backgroundCapsfilter.setCaps(Caps.fromString("video/x-raw-yuv, width=" +
36     WIDTH + ", height=" + HEIGHT));
37 textoverlay = ElementFactory.make("textoverlay", "textoverlay");
38 textoverlay.set("halign", "center");
39 textoverlay.set("valignment", 4);
40 textoverlay.set("text", WELCOMETEXT);
41 textoverlay.set("shaded-background", "false");

42
43 timeoverlay = ElementFactory.make("timeoverlay", "timeoverlay");
44 timeoverlay.set("halign", "right");
45 timeoverlay.set("valignment", 4);

46
47 backgroundQueue = ElementFactory.make("queue", "background");
48 backgroundQueue.set("leaky", 2);
49 backgroundQueue.set("max-size-time", QUEUETIME * 1000);

```

In diesem Abschnitt wird der Paketierer erstellt und ihm mit dem Parameter `pt` der Payload-Typ übergeben. Im Anschluss wird der Hintergrund für das Videosignal erstellt und konfiguriert.

Dieser Teil war so schon vorhanden.

```

50 // create the elements for the participants
51 ListIterator<ParticipantVideo> it1 = videoParticipants.listIterator();
52 while(it1.hasNext()){
53     ParticipantVideo pv = it1.next();
54     pv.setQueueSize(QUEUETIME);
55     pv.setFrameBorderSize(2);
56     pv.setFrameColor(FrameColor.BLACK);
57     pv.init(true);
58 }
```

Hier werden alle Instanzen der ParticipantVideo-Klasse durchlaufen, verschiedene Parameter gesetzt und am Ende die `init`-Methode aufgerufen, um die Klasse zu initialisieren.

```

59 // create the receiver for RTP
60 rtpbin = (RTPBin) ElementFactory.make("gstrtpbin", "rtpbin");
61 rtpbin.set("autoremove", true);

62 udpsrc = ElementFactory.make("udpsrc", "udpsrc");
63 udpsrc.set("uri", "udp://" + RECV_IP + ":" + RECV_PORT);
64 udpsrc.set("caps", new Caps(CAPABILITY));
65 // link the udpsrc with rtpbin
66 pipe.addMany(rtpbin, udpsrc);
```

In diesem Teil wird die RTP-Bin erstellt und deren `autoremove`-Eigenschaft auf „wahr“ gesetzt, damit Pads automatisch entfernt werden, deren Quellen über einen gewissen Zeitraum keine Pakete gesendet haben (Timeout). Ausserdem wird noch eine `udpsrc` erstellt und beide Elemente anschließend zur Pipeline hinzugefügt, indem sie mit der Methode `addMany` an diese übergeben werden.

```

68 Pad sinkpad = rtpbin.getRequestPad("recv_rtp_sink_0");
69 Pad srcpad = udpsrc.getStaticPad("src");
70 srcpad.link(sinkpad);

71 // link the background
72 pipe.addMany(background, backgroundCapsfilter, textoverlay, timeoverlay
               backgroundQueue);
73 Element.linkMany(background, backgroundCapsfilter, textoverlay, timeoverlay,
                  backgroundQueue);
```

In Zeile 68 wird von der RTP-Bin ein Sink-Pad für RTP-Daten angefordert, also das Pad, auf dem RTP-Daten eingehen sollen. Von der UDP-Source wird ein Source-Pad angefordert. Beide Pads werden über die `link()`-Funktion des Source-Pads miteinander verknüpft (Zeile 70). Im Anschluss daran werden die Elemente für den Hintergrund der Pipeline hinzugefügt und untereinander verlinkt.

```

77 pipe.addMany(mixer, encoder, packetizer, multiudpsink);
78 Element.linkMany(mixer, encoder, packetizer, multiudpsink);

```

Hier werden der Videomixer und die Elemente für das Paketieren, Codieren und Versenden der Pipeline hinzugefügt und verlinkt.

```

79 ListIterator<ParticipantVideo> it2 = videoParticipants.listIterator();
80 while(it2.hasNext()){
81     ParticipantVideo pv = it2.next();
82     pipe.addMany(pv.getElements());
83 }
84 ListIterator<ParticipantVideo> it3 = videoParticipants.listIterator();
85 while(it3.hasNext()){
86     ParticipantVideo pv = it3.next();
87     Element.linkMany(pv.getElements());
88 }

```

Die einzelnen Elemente der Instanzen von Klasse ParticipantVideo werden hier, in 2 Schleifen, zunächst der Pipeline hinzugefügt und danach verlinkt.

```

89 Pad backgroundSrcPad = backgroundQueue.getStaticPad("src");
90 Pad mixerSink0Pad = mixer.getRequestPad("sink_" + mixer.getPads().size());
91
92 mixerSink0Pad.set("zorder", 0);
93 mixerSink0Pad.set("alpha", 1);
94 backgroundSrcPad.link(mixerSink0Pad);
95
96 GstDebugUtils.gstDebugBinToDotFile(pipe, 5, "vidPipeBeforeAddPrtcpnt");

```

In diesem, letzten Abschnitt wird der Hintergrund als erstes Signal in den Videomixer eingespeist. Danach wird eine DOT-Datei erstellt, wie es in der Kommandozeile auch möglich ist (Siehe Abschnitt 4.3.1).

Zwischentest Beim nun folgenden Zwischentest wird auf der Client-Seite ein Video empfangen, jedoch mit einem Fehler (Siehe Abbildung 4.18). Die Ursache wird zunächst beim Codierer gesucht und durch das Ändern verschiedener Parameter des Codierers zu lösen versucht. Dies ist jedoch nicht von Erfolg gekrönt. Eine Anpassung der Auflösung bringt dann den gewünschten Erfolg. Nur die Auflösung 704 x 576 stellt sich als funktionstüchtig heraus.

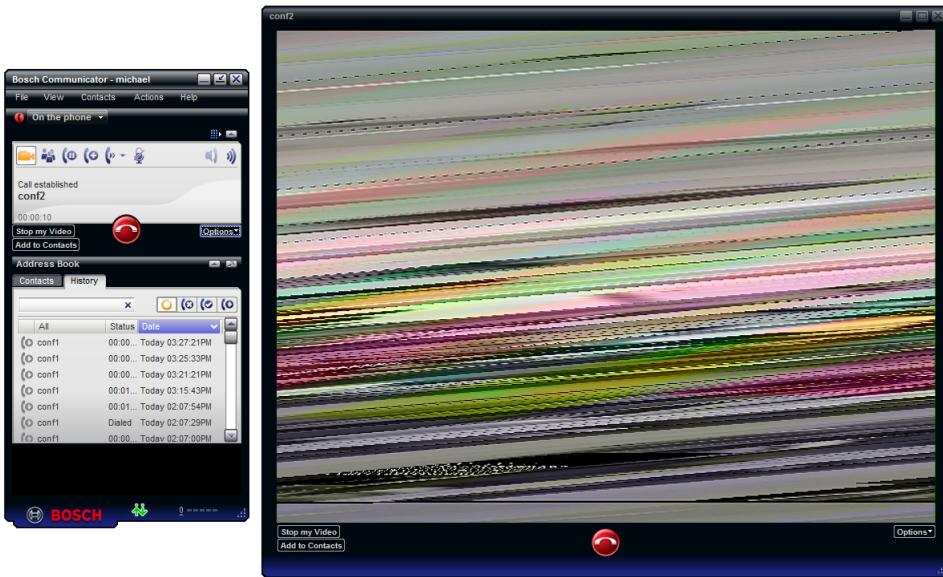


Abbildung 4.18: Fehlerhafte Videodarstellung

ScreenResolution

ScreenResolution stellt ein Interface bereit, um auf verschiedene, genormte Bildschirmauflösungen komfortabel zugreifen zu können. Diese Klasse war bereits vorhanden und wird unverändert übernommen. Die einzige, als funktionierend getestete Auflösung hat die Bezeichnung `_4CIF_4SIF_625` (704×576), in diesem Interface durch einen eindimensionalen Integer-Array 2 Werten (Breite und Höhe) abgebildet.

ConferenceRoom

Zunächst beschränkt sich die Planung auf einen einzelnen Konferenzraum. Im Laufe der Entwicklung stellt sich jedoch heraus, dass in der Organisation der Klassen ein Objekt „Konferenzraum“ sinnvoll ist, da diesem andere Objekte am sinnvollsten zugeordnet werden können. So gehört zu jedem Konferenzraum etwa eine Liste der Teilnehmer sowie die Funktion `slotsAvailable()`, um zu erfahren, ob in dieser Instanz des Konferenzraumes noch Plätze frei sind.

Wird die Klasse instanziert, werden so viele Instanzen von ConferenceMember instanziert, wie es durch die Variable `MAX_MEMBERS` vorgegeben ist.

Instantiierung Die Initialisierung von GStreamer mit der Funktion `Gst.init` terminiert den entsprechenden Thread, das Programm „hängt“. Dies ist erwartetes Verhalten, da die Kontrolle an GStreamer übergeben werden muss, damit dieser entsprechend seiner Vorgaben am Netzwerk lauschen kann. Es werden jedoch weitere Anweisungen im Programm nicht ausgeführt. Die Aufrufe von GStreamer müssen also in separaten Threads platziert werden, damit das Hauptprogramm fortgesetzt wird.

```

1 Runnable rAudio = new Runnable() {
2     public void run() {
3         while (true) {
4             audiomixer = new AudioMixer();
5             audiomixer.init(PARTICIPANTS);
6         }
7     }
8 };
9 Runnable rVideo = new Runnable() {
10    public void run() {
11        while (true) {
12            videobridge = new VideoBridge();
13            videobridge.start();
14        }
15    }
16 };

```

Hier werden Instanzen der Klasse Runnable erstellt und deren Methode `run()` überschrieben (Siehe Abschnitt 2.8.5). Die Methode enthält eine Schleife, die nichts anderes macht, als immer wieder die Initialisierung der beiden MediaProcessor-Varianten zu starten. In der Initialisierung wird GStreamer aufgerufen, dessen Aufruf dann den Thread terminiert. Aus Java-Sicht bleibt die Methode also am Ende stehen, die Schleife wird nur einmal durchlaufen. Die Kontrolle wird am Ende an GStreamer übergeben.

```

17 Thread thr1 = new Thread(rAudio);
18 Thread thr2 = new Thread(rVideo);
19 thr1.start();
20 thr2.start();

```

Mit der Instantiierung der Klasse Thread und dem Übergeben der zuvor erstellten Runnable-Instanzen werden 2 Threads erstellt und mit deren Methode `start()` gestartet.

Methode addParticipant Bei Instantiierung eines neuen Konferenzraumes (Klasse ConferenceRoom) werden zwar bereits ebenfalls alle Instanzen der Participant-Klassen für Audio und Video mit erstellt, jedoch nicht mit Werten befüllt. Dies geschieht, sobald der Server eine INVITE-Nachricht erhält und alle sonstigen, für den Beitritt des Teilnehmers notwendigen, Bedingungen erfüllt sind. Das Übergeben der Werte an die Participant-Instanzen ist in der Funktion `addParticipant` realisiert.

```

1 public void addParticipant(String remoteIp, int audioRemotePort,
2     int videoRemotePort, String callId, String participantName){
3
4     ListIterator<ParticipantAudio> itA =
5         this.getAudioParticipants().listIterator();
6
7     while(itA.hasNext()){
8         ParticipantAudio pa = itA.next();
9
10        if(pa.getCallId()==null){
11            pa.setCallId(callId);
12            pa.setIp(remoteIp);
13            pa.setPort(audioRemotePort);
14            break;
15        }
16    }

```

Im ersten Block wird die Liste aller ParticipantAudio-Instanzen durchlaufen und jeweils geprüft, ob die Call-ID gesetzt ist. Bei der ersten Instanz, bei der dies nicht der Fall ist, werden Call-ID, IP-Adresse und der entfernte Port für Audio gesetzt. Danach wird die Schleife mit dem `break`-Schlüsselwort verlassen.

```

17     ListIterator<ParticipantVideo> itV =
18         this.getVideoParticipants().listIterator();
19     while(itV.hasNext()){
20         ParticipantVideo pv = itV.next();
21
22         if(pv.getCallId()==null){
23             pv.setCallId(callId);
24             pv.setIp(remoteIp);
25             pv.setPort(videoRemotePort);
26             pv.setName(participantName);
27             videoBridge.addParticipant(remoteIp, videoRemotePort);
28             break;
29         }
30     }
31 }

```

Im zweiten Block geschieht quasi das Gleiche, nur für die entsprechende ParticipantVideo-Instanz. Als Besonderheit wird hier neben Call-ID, IP-Adresse und Port noch der Name des Teilnehmers aus SIP gesetzt, damit dieser auf dem Videosignal des Teilnehmers eingeblendet werden kann.

MediaServer

Die Klasse MediaServer ist die Hauptklasse, also die Klasse mit der `main()`-Methode. Innerhalb dieser Methode werden lediglich 2 Befehle ausgeführt. Die Instantiierung von MediaServer und

der Aufruf der `init()`-Methode der Instanz:

```

1 public static void main(String[] args) throws PeerUnavailableException,
2         TransportNotSupportedException, ObjectInUseException,
3         InvalidArgumentException, TooManyListenersException,
4         IOException, ConfigurationException{
5
6     MediaServer mediaserver = new MediaServer();
7     mediaserver.init();
8 }
```

init-Methode In der `init()`-Methode wird, falls die Config-Datei nicht existiert, diese erzeugt und einige Standardwerte in diese geschrieben, damit der Mediaserver arbeitet, falls die Config-Datei gelöscht wurde. Existiert die Datei, wird diese nicht verändert. Als Nächstes wird Focus instantiiert und initialisiert, GStreamer initialisiert und als Letztes wird eine Meldung ausgegeben, dass der MediaServer bereit ist, Anrufe entgegenzunehmen.

Methode `getRoomByName` Die Methode `getRoomByName` soll eine Instanz der Klasse ConferenceRoom zurückgeben, die den Namen hat, der ihr als Parameter übergeben wird. Gibt es einen entsprechenden Raum, soll dieser andernfalls ein neu erstellter Raum mit diesem Namen zurückgegeben werden. Die angerufene SIP URI definiert also den Konferenzraum, dem der Teilnehmer beizutreten wünscht.

```

1 public static ConferenceRoom getRoomByName(String name)
2         throws NullPointerException {
3     ConferenceRoom room = null;
4     ListIterator<ConferenceRoom> itc =
5             MediaServer.getConferenceRooms().listIterator();
6     while(itc.hasNext()){
7         ConferenceRoom element = itc.next();
8         if(element.getRoomName().equals(name)){
9             room=element;
10        }
11    }
```

Hier wird zunächst eine Variable vom Typ ConferenceRoom, namens `room` ohne Inhalt erstellt. Anschließend werden alle im Mediaserver existierenden Konferenzräume (Instanzen der Klasse ConferenceRoom) durchlaufen und jeweils geprüft, ob ihr Name dem als Parameter übergebenen Namen entspricht.

Ist das der Fall, wird die zu Beginn erstellte Variable `room` mit diesem Raum belegt.

```

12 if (room==null){
13     room = MediaServer.addRoom(name);
14 }
15 return room;
16 }
```

Wird kein Raum mit diesem Namen gefunden, wird eine neue Instanz der Klasse ConferenceRoom mit diesem Namen erstellt und die Variable `room` damit belegt. Danach wird die Variable `room` zurückgegeben.

4.4.3 Erstmals etablierte Videokonferenz

Bei dem nun durchgeföhrten Test lässt sich eine Videokonferenz etablieren. Das Audiosignal jedoch wird nicht wie gewünscht empfangen. Einer der Teilnehmer hört sich selbst und der andere Teilnehmer hört nichts. Eine falsch umbenannte Zählvariable in der Klasse AudioMixer wird als Verursacher ausfindig gemacht.

4.4.4 Entfernen von Teilnehmern

Das Aufbauen einer Konferenz funktioniert, es können mehrere Teilnehmer der Konferenz beitreten, ihr Name wird unter ihrem Video angezeigt, jeder hört die anderen Teilnehmer. Das Entfernen von Teilnehmern führt jedoch zu einem Problem. Beendet einer der Teilnehmer die Sitzung und sein Videosignal stoppt dementsprechend, sehen auch alle anderen Teilnehmer kein Video mehr. Die Ursache hierfür liegt im GStreamer Element `videomixer` begründet. Das Ende des Videosignals an einem Eingang lässt das resultierende Gesamtvideosignal ebenfalls enden, da `videomixer` keine Synchronisation der Frames zur Laufzeit unterstützt, was über einen IRC-Chat mit den Entwicklern herausgefunden wird [IRC01].

Zur Lösung dieses Problems wird versucht, das Ende des Signals zu erkennen, bevor es das Element `videomixer` erreicht und die Verbindung vorher zu trennen. Ein Stoppen der Pipeline, Entfernen der Quelle und anschließendes Wiederstarten der Pipeline ist nicht erfolgreich. Im IRC-Chat mit den Entwicklern wird die Verwendung des Elements `videomixer2` nahe gelegt, das eine relativ neue, jedoch noch als instabil angesehene Entwicklung (Beta-Status) ist. Dieser lässt sich jedoch nicht wie erhofft durch einfaches Austauschen implementieren. Eine rechtzeitige Implementierung gelingt daher leider nicht. Das Audiosignal wird nach dem Entfernen eines Teilnehmers aufrecht erhalten, da dies in einer separaten GStreamer-Instanz verarbeitet wird.

4.4.5 Gesamtfunktionstest

Der zum Schluss durchgeföhrte Test mit Bosch Communicator auf mehreren Clients funktioniert, bis auf das beschriebene Entfernen von Teilnehmern.



Abbildung 4.19: Funktionierende Konferenz mit Bosch Communicator

Kapitel 5

Fazit und Ausblick

5.1 Fazit

5.1.1 Erfüllung der Anforderungsanalyse

In der Anforderungsanalyse werden folgende, funktionale und nicht-funktionale Anforderungen gestellt:

Anfrage des Clients und Initialisierung der Verbindung per SIP Diese Anforderung wird vollständig erfüllt.

Hinzufügen des Teilnehmers in die GStreamer Pipeline Diese Anforderung wird ebenfalls vollständig erfüllt.

Abbau einer Verbindung Der Abbau der Verbindung per SIP ist implementiert und die entsprechende Signalisierung ist funktionstüchtig. Das Beenden des Audioanteils eines Teilnehmers funktioniert ebenfalls. Das Entfernen des Videosignals kann leider nicht funktionstüchtig implementiert werden. Dies ist auf mehrere Ursachen zurück zu führen:

- Verwendung nur rudimentär dokumentierter Open Source-Software
- Entwicklungsstatus des GStreamer Videomixers

Anforderungen an Performance Nach Einstellen der entsprechenden Parameter des H.264-Codecs wird eine hervorragende Performance erreicht. Ob diese Performance in Nichtlaborbedingungen aufrecht erhalten werden kann, wenn etwa die RTP-Ströme über einen SBC geleitet werden, kann nicht getestet werden.

Qualitätsanforderungen Das Produkt läuft im Test prinzipiell stabil. Sporadisch treten jedoch Segmentation Fault Fehler auf, die die gesamte Software abstürzen lassen. Diese Fehler sind den GStreamer-Bindings zuzuschreiben. Darüber hinaus werden beim Decodieren Fehlermeldungen („Too many slices, increase MAX_SLICES and recompile“) ausgegeben. Dies ist ein bekannter Fehler [FFMPG]. Gleiches gilt für eine Warnung bezüglich einer angeblichen Schleife in der RTP-Bin, die GStreamer ausgibt, sobald ein zweiter Teilnehmer der Konferenz hinzugefügt wird („GStreamer-WARNING **: loop detected in the graph of bin play!!“)[GSTR04]. Auf all diese Fehler kann nur bedingt und im Rahmen dieser Diplomarbeit leider überhaupt kein Einfluss genommen werden.

Design Wie in den Anforderungen verlangt, sind im Quelltext ausreichend Kommentare eingefügt. Alle Kommentare sind in englischer Sprache verfasst.

Wünsche Die Möglichkeit, die Anwendung ohne Änderungen des Quelltextes konfigurieren zu können, wurde über eine Konfigurationsdatei realisiert.

5.1.2 Persönliche Erfahrungen

Die wohl grundlegendste Erkenntnis ist, dass nicht alles umgesetzt werden kann, was umzusetzen gewünscht wird. Ein perfektes Produkt zu erstellen, ist nur mit unbegrenzten Ressourcen möglich. Jedoch etwas mit einem begrenzten Budget (in diesem Fall ein begrenztes Zeitbudget) umzusetzen, erfordert Entscheidungen: Welche Funktionen sind essentiell, welche optional? Oftmals sind diese Entscheidungen schwer, doch müssen getroffen werden, wenn es darum geht, gewünschte Funktionen wegzulassen, um der Stabilität den Vorrang zu geben oder gar den geforderten Mindestumfang überhaupt in der vorgegebenen Zeit umsetzen zu können.

5.2 Ausblick

Als weitere Entwicklung gibt es zahlreiche Möglichkeiten:

5.2.1 Interne Verbesserungen

Diese Verbesserungen sind für den Anwender nicht direkt ersichtlich, verbessern jedoch die Kompatibilität und Funktion des Mediaservers.

Mehrere Codecs

Der Mediaserver sollte mehrere Codecs für Audio und Video unterstützen. In der vorliegenden Version werden lediglich PCMU und H.264 angeboten. In einer realen Umgebung gibt es jedoch eine Vielzahl an möglichen Codecs. Daher sollten alle auf einem System installierten Codecs auch verwendet werden können. So wird die Wahrscheinlichkeit erhöht, dass die unterschiedlichsten Teilnehmer Mediendaten austauschen können. Das schließt die Verbindung von Teilnehmern mit verschiedenen Codecs in einer derselben Konferenz explizit mit ein.

Dynamisches SDP

In direktem Zusammenhang mit dem vorangegangenem Punkt steht die dynamische Erstellung des SDP-Bodys entsprechend des vorhandenen Codecs und der zugehörigen Parameter. So können auch mehrere Profile für einen Codec hinterlegt werden, je nachdem in welcher Netzwerkumgebung die Anwendung zum Einsatz kommt. Hierfür muss eine Funktion entwickelt werden, die entsprechend der Vorgaben der Konfigurationsdatei die Medieninformationen im SDP erstellt. Darin sollten ebenfalls die für RTP verwendeten Ports dynamisch vergeben werden.

Bessere Konfiguration

Die Konfiguration über eine Textdatei ist hier nur sehr rudimentär realisiert. Die Textdatei sollte alle relevanten Variablen enthalten und diese sollten im Programm überall mittels einer entsprechenden Klasse geladen werden. Vorstellbar ist etwa ein Laden aller Variablen der Konfigurationsdatei zum Zeitpunkt des Starts des Mediaservers. So werden wiederholte Dateioperationen zur Laufzeit vermieden, was die Performance verbessert.

Darüber hinaus ist eine Bearbeitung der Datei über ein grafisches Web-Interface vorstellbar, inklusive der Möglichkeit, den Mediaserver im Anschluss neu zu starten.

Nutzung von RTCP

Zum RTP-Standard gehört der regelmäßige Versand von RTCP-Paketen. Die Nutzung dieser Pakete ist jedoch nicht obligatorisch, weswegen nicht alle UACs diese Zusatzinformationen verwenden. Für den Fall, dass diese Pakete empfangen werden, sollte der Mediaserver die darin enthaltenen Informationen auch verarbeiten. Der enthaltene CNAME, aus dem die IP-Adresse des Senders extrahiert werden kann, ist von großem Nutzen, da UDP diese Informationen nicht enthält. Dies ist insbesondere von Interesse, wenn der Teilnehmer mit einem Mobilgerät arbeitet und seinen Standort mit dazugehöriger IP-Adresse verändert.

Logging

Das Programm enthält bereits in Grundzügen Funktionen für das Logging, also die Ausgabe und Speicherung von Informationen, Warnungen und Fehlermeldungen des Programms während der Laufzeit. Diese Informationen sollten in Dateien gespeichert werden, idealerweise getrennt nach Informationen und Fehlern/Warnungen. Dies würde die Fehlersuche im Nachhinein erleichtern. Derzeit müssen die Meldungen direkt am Bildschirm verfolgt werden. Die Tiefe der Meldungen sollte einstellbar sein.

Nutzung von CSRC

Da die Teilnehmer ein gemischtes Audiosignal zurückhalten, könnte unter Umständen von der im RTP-Standard [RFC3550] vorgesehenen CSRC Gebrauch gemacht werden, um einzelne Teilnehmer beim Empfänger identifizieren und ggf. stumm schalten zu können. Dazu dürfen die Audiosignale jedoch nicht akustisch gemischt übertragen werden, da eine anschließende Trennung so nicht möglich ist. Entsprechende Möglichkeiten wären zu prüfen.

5.2.2 Weitere Features

Diese Features geben den Konferenzteilnehmern mehr Handlungsspielraum.

Teilnehmer hinzufügen

Für eine Audio-/Videokonferenz generell wäre es sinnvoll, wenn ein Teilnehmer von sich aus weitere Teilnehmer hinzufügen kann. So wäre es möglich, dass der Assistent oder die Assistentin der Geschäftsführung einer Firma im Vorfeld alle Teilnehmer anruft und der Konferenz hinzufügt, so dass bereits alle Teilnehmer in der Konferenz sind, wenn die Geschäftsführung beitritt.

Diese Funktion ließe sich eventuell mit der REFER-Anfrage realisieren.

Möglichkeit für Screencast im Client

Für Seminare oder Fernunterricht wäre die Möglichkeit nützlich, wenn ein Teilnehmer anstatt seines eigenen Videos seine Bildschirmausgabe übertragen würde. Dies ist eine Funktion des Clients und kann im Mediaserver so nicht realisiert werden. Würde der Mediaserver ein entsprechendes Video empfangen, würde es in der derzeitigen Version schon übertragen. Interessant wäre in dem Fall allerdings, wenn die Videobridge eine externe Konfiguration erlauben würde, so dass das Videosignal des Lehrers oder Seminarleiters größer als die Videos der anderen Teilnehmer angezeigt werden könnte.

Transfer zu anderem Konferenzraum

Sinnvoll wäre darüber hinaus die Möglichkeit, als Teilnehmer zu einem anderen Konferenzraum wechseln zu können. Die kann über ein Re-INVITE umgesetzt werden.

Abbildungsverzeichnis

1.1	Vision der Zukunft aus dem Jahr 1922	6
2.1	OSI-Referenzmodell	8
2.2	INVITE-Nachricht mit SDP-Body	9
2.3	RTP-Header	13
2.4	UDP-Header	15
2.5	Digitalisierung eines kontinuierlichen Signals	20
2.6	Digitalisierung eines Videosignals	22
2.7	Sequentielle Programmierung	23
3.1	Unterschied zwischen einem Videomixer und einer Videobridge	30
3.2	Umgebung Konferenzlösung	32
3.3	Use Cases Mediaserver	33
3.4	Schematischer Aufbau des Mediaservers	36
4.1	Testumgebung GStreamer per Kommandozeile	39
4.2	Ausgabe eines Testbildes mit videotestsrc	41
4.3	Pipeline zum Mischen zweier, über das Netz empfangener Audioströme	44
4.4	Gemischte Sinustöne	45
4.5	Prozess Audiomixer	46
4.6	Mischen der Audiosignale	46
4.7	4 Clients - 4 verschiedene Audiosignale	47
4.8	GStreamer Pipeline Grafik	50
4.9	GStreamer Debugging Stufe 3 Bildschirmausgabe	50
4.10	Klassenstruktur des Mediaservers	51
4.11	Participant UML Klassendiagramm	52
4.12	Funktionsdiagramm Participant	53
4.13	MediaProcessor UML Klassendiagramm	54
4.14	Ekiga Softphone	57
4.15	Verbindungsauftbau erfolgreich, anschließend Verbindungsabbau	58
4.16	Verbindungsauftbau nicht erfolgreich	59
4.17	3-Wege Handshake in Wireshark	63
4.18	Fehlerhafte Videodarstellung	68
4.19	Funktionierende Konferenz mit Bosch Communicator	73

Literaturverzeichnis

- [GSTR04] “Archiv der GStreamer Mailingliste”. <http://lists.freedesktop.org/archives/gstreamer-devel/2008-April/017567.html>. [Online; zugegriffen am 14.7.2012]
- [LINU01] “grep”. Linux integrierte Hilfe
- [GSTR03] “gst-inspect x264enc”. GStreamer integrierte Hilfe
- [GSTR02] “GStreamer Developer Mailingliste”. <http://lists.freedesktop.org/mailman/listinfo/gstreamer-devel>. [Mailingliste; abonniert am 06.02.2012]
- [IRC01] “GStreamer IRC Chat”. <http://irc.lc/freenode/gstreamer/>. [Online; zugegriffen mehrfach im Mai 2012]
- [GAJI01] “The Long History Of The Video-Phone”. <http://gajitz.com/formerly-futuristic-the-long-history-of-the-video-phone/>. [Online; zugegriffen am 17.07.2012]
- [FFMPG] “Ticket System von FFmpeg”. <http://ffmpeg.org/trac/ffmpeg/ticket/273>. [Online; zugegriffen am 14.07.2012]
- [IEEE830] “IEEE Guide to Software Requirements Specifications”, 1984
- [CISC06] “Waveform Coding Techniques”. http://www.cisco.com/application/pdf/paws/8123/waveform_coding.pdf, 2006. [Online; zugegriffen am 25.06.2012]
- [GSTR01] Boulton, Richard John; Walthinsen, Erik; Baker, Steve; Johnson, Leif; Bultje, Ronald S.; Kost, Stefan; Müller, Tim-Philipp: “GStreamer Plugin Writer’s Guide”. <http://gstreamer.freedesktop.org/data/doc/gstreamer/head/pwg/html/index.html>. [Online; zugegriffen am 28.06.2012]
- [RFC4566] Handley, M.; Jacobson, V.; Perkins, C.: “SDP: Session Description Protocol”. RFC 4566 (Standards Track), 2006
- [GOLL01] Joachim Goll and Cornelia Weiß and Frank Müller: *Java als erste Programmiersprache*. Teubner, 2001. ISBN 3-519-22642-1
- [KERN84] Kernighan, Brian W.; Pike, Rob: *The Unix Programming Environment*. Prentice Hall, Inc., 1984. ISBN 0-13-937681-X (Taschenbuch), 0-13-937699-2 (gebundene Ausgabe)
- [GANC95] Mike Gancarz: *The UNIX Philosophy*. Digital Press, 1995. ISBN 1-55558-123-4
- [PERE08] Perea, Rogelio Martínez: *Internet Multimedia Communications Using SIP*. Morgan Kaufmann Publishers, 2008. ISBN 978-0123743008

- [TUDO95] P.N. Tudor: "MPEG-2 Video Compression". Electronics & Communication Engineering Journal, 1995
- [RFC768] Postel, J.: "User Datagram Protocol". RFC 768 (Standard), 1980
- [GNU01] Richard Stallman: "Linux und das GNU Projekt". <http://www.gnu.org/gnu/linux-and-gnu>. [Online; zugegriffen am 26.06.2012]
- [RFC3261] Rosenberg, J.; Schulzrinne, H.; Camarillo, G.; Johnston, A.; Peterson, J.; Sparks, R.; Handley, M.; Schooler, E.: "SIP: Session Initiation Protocol". RFC 3261 (Proposed Standard), 2002. Updated by RFCs 3265, 3853, 4320, 4916, 5393, 5621, 5626, 5630, 5922, 5954, 6026, 6141
- [SCHM08] Schmidt, Ulrich: *Digitale Film- und Videotechnik*. Carl Hanser Verlag, 2008. ISBN 978-3446412507
- [RFC3551] Schulzrinne, H.; Casner, S.: "RTP A/V Profile". RFC 3551 (Proposed Standard), 2003
- [RFC3550] Schulzrinne, H.; Casner, S.; Frederick, R.; Jacobson, V.: "RTP: A Transport Protocol for Real-Time Applications". RFC 3550 (Standard), 2003. Updated by RFC 5506
- [TANN03] Tannenbaum, Andrew S.: *Moderne Betriebssysteme*. Pearson Studium, 2003. ISBN 3-8273-7019-1
- [TRIC09] Trick, Ulrich; Weber, Frank: *SIP, TCP/IP und Telekommunikationsnetze*. Oldenbourg, 2009. ISBN 978-3-486-59000-5
- [ULLE06] Ullenboom, Christian: *Java ist auch eine Insel*. Galileo Computing, 2006. ISBN 978-3836211468



Dieses Dokument steht unter der CC-BY-SA 3.0 Lizenz.

Abbildung 1.1 alle Rechte vorbehalten Frank Rudolph Paul; Erschienen in Radio For All, Artikel „The Future of Radio“ von Hugo Gernsback, 1922, J.B. Lippincott Co., Philadelphia & London

Abbildung 2.5 lizenziert mit CC-BY-SA 2.0 von Keenan Tims

Abbildung 3.1 unter Verwendung von Bildern lizenziert mit CC-BY-SA 2.0 von Kevin Saff und Rebecca Partington



Audio- und Videoverarbeitung mit GStreamer Framework



(Fast) alle Diagramme erstellt mit Lucidchart