

DIPLOMARBEIT

Wintersemester 2002/03

Internetkommunikation



Michael Meyer, Markus Schild

Abstract

Die vorliegende Arbeit beschreibt einen einfachen SIP User Agent für Sprachübertragung. Das Programm funktioniert in einem leistungsstarken Netzwerk, wie z.B. dem Intranet der HSR, hat aber Probleme mit der Sprachübertragung über das Internet, da eine sehr rudimentäre Sprachübertragung gewählt wird. Die Sprachübertragung ist nicht Thema der Arbeit und ist somit nicht weiter behandelt worden.

Es sind die SIP-Methoden ACK, BYE, CANCEL und INVITE implementiert. So kann ein Anruf erzeugt, empfangen und wieder beendet werden. Auf Funktionen wie Weiterleitung oder Konferenzschaltung ist verzichtet worden, da ein einfacher User Agent gefordert wurde. Der User Agent Server reagiert korrekt auf nicht implementierte Methoden wie zum Beispiel OPTIONS, indem er mit einem „Method Not Allowed“ antwortet. Somit kann mit anderen User Agents, wie z.B. einem kommerziellen SIP Hardware Telefon, kommuniziert werden.

Für die Entwicklung wurde MS Visual C++ verwendet. Es ist eine dialogbasierte Applikation entstanden, welche mit Hilfe der Microsoft Foundation Class (MFC) generiert wurde. Für die Analyse und das Design sind die Methoden der objektorientierten Softwareentwicklung zum Einsatz gekommen.

Inhaltsverzeichnis

ABSTRACT	2
INHALTSVERZEICHNIS	3
1. EINLEITUNG	6
1.1. Aufgabenstellung	7
2. EINFÜHRUNG IN SIP	8
2.1. Übersicht	8
2.2. Einfache Verbindung	8
2.3. SIP-Message.....	9
2.3.1. Start-Line	9
2.3.2. Header.....	10
2.3.2.1. General Header	10
2.3.2.2. Entity Header	11
2.3.2.3. Request Header.....	11
2.3.3. Message Body	11
2.3.3.1. SDP-Struktur.....	11
2.3.4. SIP Request.....	12
2.3.5. SIP Response.....	12
2.4. SIP Entities	13
2.4.1. Registrar	13
2.4.2. Proxy	13
2.4.3. Redirect Server	13
3. ANALYSE	14
3.1. Client	15
3.2. Server	15
4. DESIGN	17
4.1. Hauptklassen.....	17
4.1.1. Sequenzdiagramm.....	18
4.2. Client	20
4.2.1. Sequenzdiagramm.....	20
4.3. Server	21
4.3.1. Sequenzdiagramm.....	22

5.	AUDIO	24
5.1.	Anwendung des AudioController	24
5.1.1.	Audio Socket öffnen.....	25
5.1.2.	Versenden und Empfangen von Sprachdaten	25
5.1.3.	Schliessen des Audio Sockets.....	25
5.2.	Implementierung	25
5.2.1.	Sprachcodierung.....	26
5.2.2.	Callbackfunktionen.....	26
6.	SIPUA	27
6.1.	Anwendung	27
6.2.	Implementierung	27
7.	SESSION.....	28
7.1.	Anwendung	28
8.	UA CLIENT	29
8.1.	Anwendung	29
8.2.	Implementierung	29
8.2.1.	Zustandsdiagramm.....	30
8.2.1.1.	INVITE Zustandsdiagramm	30
8.2.1.2.	non INVITE	32
8.2.2.	Auswertung der Responses	33
9.	UASERVER	35
9.1.	Anwendung	35
9.2.	Implementierung	36
9.2.1.	Zustandsdiagramm.....	36
9.2.1.1.	INVITE Server Transaktion	36
9.2.1.2.	Non-Invite Zustandsdiagramm	38
9.2.2.	Auswertung der Requests	39
10.	SIP-MESSAGE.....	42
10.1.	Anwendung	43
10.2.	Implementierung	43
10.2.1.	SIPMessage	44
10.2.2.	StartLine.....	44
10.2.3.	RequestLine.....	44
10.2.4.	StatusLine	44
10.2.5.	MessageHeader.....	44
10.2.6.	MessageBody.....	45
10.2.7.	SDPHeader	45

11.	BEDIENUNGSANLEITUNG	46
11.1.	Schnellstart.....	46
11.2.	Programmübersicht	47
11.2.1.	Verbindung starten.....	47
11.2.2.	Verbindung entgegennehmen oder ablehnen.....	48
11.2.3.	Verbindung beenden	50
11.3.	Meldungen.....	50
12.	TESTEN DES PROGRAMMS	53
12.1.	Testvorgehen	53
12.2.	Ergebnis	53
12.3.	Bemerkungen zu den Tests.....	54
13.	ZUSAMMENFASSUNG	55
ANHANG A: ABKÜRZUNGSVERZEICHNIS		57
ANHANG B: REFERENZEN		58
ANHANG C: ABBILDUNGSVERZEICHNIS		59
ANHANG D: VERZEICHNISSTRUKTUR		60
ANHANG E: STATUS CODE		61
ANHANG F: TESTPROTOKOLL		63

1. Einleitung

Heutzutage steht in vielen Haushalten ein Telefon und ähnlich viele Haushalte haben Zugang zum weltweit umspannenden Internet und meist ist der Internettarif günstiger als der Telefontarif. Wieso sollte man deshalb auf das Kostensparen beim Telefonieren übers Internet verzichten? Die Stolpersteine zur Zeit sind, dass es noch wenige, billige Endgeräte gibt, die ohne einen eingeschalteten PC funktionieren, die zudem noch einfach zu bedienen sind und dass die Leistung des Netzwerkes noch nicht überall den höheren Anforderungen für Internettelefonie gerecht werden.

Ein grosser Unterschied vom herkömmlichen Telefon zum Internettelefon besteht darin, dass bei den bis jetzt gebräuchlichen Telefonen eine Verbindung aufgebaut wird und bestehen bleibt, bis man den Hörer wieder auflegt. Dabei entsteht ein Datenstrom, der ununterbrochen die Sprache beim Sender aufnimmt und über eine geschaltete Verbindung beim Empfänger wieder ausgibt. Selbst Stille wird übertragen und kostet deshalb Leitungsgeld. Wenn man bedenkt, dass nur ca. 35 % eines Gesprächs wirklich Gesprochenes enthält, könnte man eigentlich die Kosten reduzieren.

An diesem Punkt setzt die Internettelefonie an, auch bekannt unter dem Namen Voice over IP (VoIP). Dieses System produziert keinen kontinuierlichen Datenstrom mehr, sondern unterteilt die Konversation in Pakete, die das Internet transportieren kann. Es ist sogar möglich, die stillen Abschnitte einer Konversation nicht und damit weniger Daten zu übertragen. Dies hat zur Folge, dass im Durchschnitt die Datenmenge drastisch sinkt und somit Kosten gespart werden können. Zusätzlich kann die freigewordene Bandbreite für andere Benutzer zur Verfügung gestellt werden.

In dieser Arbeit wird jedoch nicht auf den Audioteil eingegangen. Vielmehr wird beschrieben, wie eine Verbindung aufgebaut und wieder beendet wird und zwar nach dem noch jungen, weltweiten Standard, dem Session Initiation Protocol (SIP). Zwar liegt dieser Standard erst in einer Request For Comment (RFC) vor, doch wird die zur Zeit vorliegende [rfc3261] schon in kommerziellen Produkten als Implementation eingesetzt. SIP ist auch deshalb interessant, weil es in Klartext übermittelt wird und sich deshalb für den Anschauungsunterricht ideal eignet. Das andere Protokoll H.323 hat einen komplexeren Aufbau als SIP und ist für diese Arbeit nicht interessant.

Bei diesem Telefonsystem ist der Anrufer beim Verbindungsaufbau der Client und der Angerufene agiert als Server der Verbindung. Während einer laufenden Verbindung können die Rollen des Clients und des Servers wechseln. Im Weiteren wird jede solche Punkt zu Punkt Verbindung mit einer eindeutigen Nummer versehen, der sogenannten Call-ID. Diese hilft, die SIP-Pakete der richtigen Partnerstation zu übermitteln und die empfangenen Pakete der entsprechenden Verbindung zu zuordnen. Es werden ebenso doppelt erhaltene Nachrichten verworfen, wie solche, die nicht gültig sind oder nicht zur aktuellen Session gehören.

Anhand eines Programms soll der SIP User Agent veranschaulicht werden. Dabei wird gefordert, dass die grafische Oberfläche einfach gestaltet und übersichtlich bleibt und somit keine komplizierte Einführung benötigt, um einen Anruf zu starten.

1.1. Aufgabenstellung

DIPLOMARBEIT

SIP User Agent

Michael Meyer und Markus Schild

Beginn: 21. Oktober 2002

Abgabe: 10. Dezember 2002

Es soll ein SIP User-Agent für einen PC erstellt werden. Dafür müssen die einzelnen Transaktionen für den Client und den Server richtig verstanden und in Software implementiert werden. Das Transportprotokoll sei UDP.

Ziel ist es, ein Gespräch über das Internet mittels zweier PCs führen zu können. Dabei sollen die User Agents so simpel wie möglich gehalten werden. D.h. dass zum Beispiel die Internetadressen bekannt sind und keine Weiterleitungen oder Konferenzschaltungen möglich sein müssen.

Als Grundlagen dienen die im Unterricht betrachteten Übungsprogramme.

Das GUI ist einfach zu halten und soll einfach zu bedienen sein.

Es ist ein Projektplan zu erstellen, welche die Teilziele (Milestones) festlegt.

Während der Arbeit wird ein Laborheft geführt, wo die geleistete Arbeit, sowie allenfalls auftretende Probleme festgehalten werden. Zusätzlich findet alle ein bis zwei Wochen ein Treffen statt.

Ein Bericht soll das Erlernte zusammenfassen.

2. Einführung in SIP

Das Session Initiation Protocol (SIP) ist in der [rfc3261] von der Gruppe MMUSIC (Multiparty Multimedia Session Control) beschrieben, welche eine Gruppe der IETF (The Internet Engineering Task Force) ist.

SIP unterstützt nicht nur Punkt zu Punkt Verbindungen, sondern ermöglicht auch Konferenzschaltungen.

2.1. Übersicht

Bei SIP wird zwischen zwei Nachrichten unterschieden, der Request und der Response Message. Der Initiator einer Verbindung wird SIP-Client genannt und sendet jeweils eine Request Message. Der Angerufene wird SIP-Server bezeichnet und erhält die Requests, die der Client versendet hat und antwortet dem Client mit einer Response Message. Dabei kann der Server eine definitive (final) oder eine provisorische (provisional) Antwort schicken.

Eine Transaktion beinhaltet eine Request- und Response-Nachricht, wobei die Transaktion mit einer final Response abgeschlossen wird. Alle Transaktionen haben ihre eigene CSeq-Nummer mit der Ausnahme der ACK-Message. Obwohl die ACK-Message eine eigene Transaktion ist, hat diese die CSeq-Nummer der Nachricht, die das ACK produziert hat.

SIP ist unabhängig vom Transportprotokoll, d.h. es läuft sowohl mit TCP als auch mit UDP. In dieser Arbeit wird aber ausschliesslich das UDP benützt, weshalb nur auf die SIP-Struktur und -Abläufe mit UDP eingegangen wird. Der Standardport für SIP ist per [rfc3261] 5060.

2.2. Einfache Verbindung

Der SIP-Client sendet ein INVITE an den Server. In dieser Message ist genügend Information enthalten, damit der Server eine Mediaverbindung zum Client erstellen kann. Der Server soll nun die Verbindung akzeptieren und sendet eine OK-Response. Der Client wiederum sendet dann ein ACK, um die Final-Response zu bestätigen.

Ein einfaches Beispiel eines Verbindungsauf- und abbaus ist in Abbildung 1 dargestellt. Das SIP Phone des Users A ist hier während des Verbindungsaufbaus der Client und das SIP Phone des Users B ist der Server.

Auf eine INVITE-Request kann eine Provisional oder eine Final-Response erfolgen. Sobald der Client eine Final-Response erhält, schickt er eine Bestätigung (ACK) und beendet somit die INVITE-Transaktion. Diese (mindestens) drei Nachrichten INVITE, OK, ACK werden auch als Tripple Handshake bezeichnet.

Kann der Server eine angeforderte Verbindung nicht bereitstellen, weil er z.B. den Coder, den der Client ihm mitschickt nicht unterstützt, teilt dies der Server dem Client mit und sendet ev. eine Auswahl der unterstützten Formate zurück. Der Client kann nun einen der angegebenen Coders verwenden oder kann den Anruf neu über einen Transcoding Proxy initialisieren.

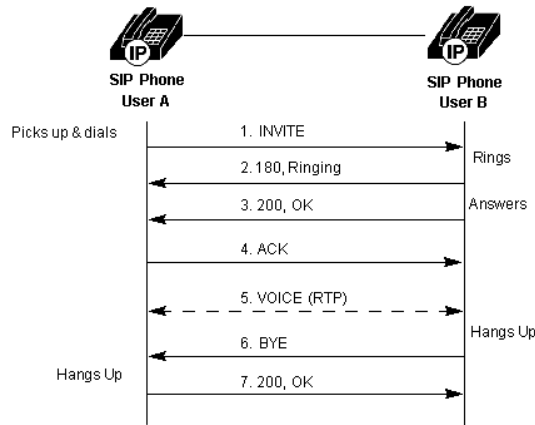


Abbildung 1: SIP Verbindungsauf- und abbau

2.3. SIP-Message

Wie erwähnt gibt es zwei Typen von Nachrichten, eine Request und eine Response. Beide haben eine gleich aufgebaute Struktur (siehe Abbildung 2).

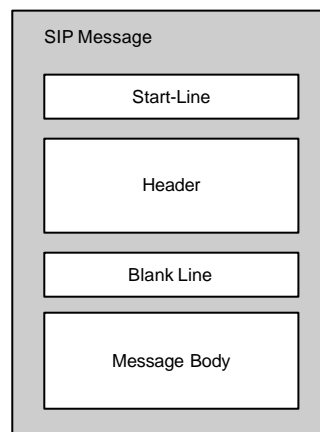


Abbildung 2: Aufbau einer SIP-Message

Die Leerzeile (Blank Line) zwischen dem letzten Header und dem Message Body ist zwingend, auch wenn kein Message Body vorhanden ist.

Jede Zeile, sei das eine Start-Line, eine Headerzeile oder eine Zeile aus dem Message Body, wird mit einem CRLF abgeschlossen

2.3.1. Start-Line

Mit der Start-Line kann unterschieden werden, ob die SIP-Message ein Request oder eine Response ist, denn sie unterscheiden sich in der Syntax.

Per [rfc3261] wird die Start-Line in einer Request Message als Request-Line definiert und die Response Message startet mit einer Status-Line. Das Format für die beiden Start-Lines sieht folgendermassen aus:

Request: method SP request-URI SP SIP-Version CRLF

Response: SIP-Version SP status-code SP reason phrase CRLF

Da die SIP-Version stets das Format `SIP/X.X` hat, kann mit Leichtigkeit festgestellt werden, ob es sich nun um eine Request oder Response handelt, denn es gibt keine Methode, die mit einem 'S' beginnt.

Auf die Methode in der Request, sowie den Status-Code und die Reason Phrase in der Response, werden im entsprechenden Kapitel eingegangen.

2.3.2. Header

Es gibt drei verschiedene Typen der Headers. Gewisse Kopfzeilen sind sowohl in der Request als auch in der Response Message enthalten. Diese Header werden auch General Header genannt. Kopfzeilen, die Bezug auf den Message Body haben, werden im Entity Header berücksichtigt. Die Request-Header enthalten zusätzliche Informationen, die ein Client mit einer Request Message dem Server mitteilen kann.

Das Headerformat lautet:

```
header name : SP header value CRLF
```

Vor und nach dem ':' dürfen beliebig viele Leerzeichen oder Tabulatoren eingefügt werden. Es sind nur die wichtigsten und notwendigen Headers erläutert.

2.3.2.1. General Header

General Headers sind zwingend nötig in der Request und der Response. Die hier aufgeführten finden sich auch in unserem Programm wieder.

Call-ID: Mit der Call-ID wird eine SIP-Verbindung eindeutig identifiziert. Mit Hilfe dieses Headers kann eine Message einer Transaktion zugewiesen werden. Es können Duplikate erkannt werden, die z.B. auf dem Weg durch einen Forking Proxy entstehen.

Die Call-ID ist einmalig und für jeden Anruf neu zu erstellen.

CSeq: Das CSeq Header-Feld besteht aus einer Sequenznummer und dem Methodennamen. Der Anfangswert der CSeq-Nummer ist beliebig und wird bei jedem neuen Request inkrementiert. Es sei denn, dass es eine Wiederholung der Request ist.

Einzige Ausnahmen sind die Methoden ACK und CANCEL, welche die CSeq-Nummer des Request, den sie bestätigen oder abrechnen, übernehmen.

Der Server kopiert die CSeq von der Request in die entsprechende Response.

Contact: Dieses Feld liefert eine URI, dessen Bedeutung abhängig von der Request oder Response ist. Das Feld kann im Weiteren einen Anzeigenamen, URI und Header-Parameter aufweisen.

From: Dieses Feld beinhaltet einen optionalen Anzeigenamen und die Adresse des Erstellers eines Requests. Zusätzlich können Tags angefügt werden.

Bei einer Response wird der From-Header direkt von der Request übernommen und sagt somit nichts über den Sender der Nachricht aus.

- To:** Bezeichnet den beabsichtigten Empfänger eines Requests. Der optionale Tag (meist zufällig gewählt) wird hauptsächlich dann benötigt, wenn die SIP-URI nicht eindeutig ist, z.B. im Falle eines Helpdesks.
- Via:** Mit Hilfe des Via-Headers kann der Weg der Message über mehrere Hops zurückverfolgt werden. So wird sichergestellt, dass eine Response denselben Weg wie der Request, in umgekehrter Reihenfolge, nimmt. Jeder Proxy fügt seine Adresse in diesem Header-Feld an.
Damit die Verbindung auch über einen NAT zurückverfolgbar bleibt, sind spezielle Parameter vorhanden.

2.3.2.2. Entity Header

Zur weiteren Beschreibung des noch folgenden Message Bodys, dienen diese Kopfzeilen. Sie weisen auf die Grösse oder den Typ hin.

Content-Type: Dieser Header beschreibt das Media, welches im Message Body verwendet wird (z.B. SDP, HTML etc.).

Content-Length: Gibt die Anzahl Oktette im Message Body an.

2.3.2.3. Request Header

Diese Header sind nicht zwingend notwendig, verringern aber die Anzahl SIP-Messages, die ausgetauscht werden müssen, um eine gültige Client-Server Verbindung zu erreichen.

- Accept:** Der optionale Header teilt dem Server mit, welche Typen von Medien in der Response akzeptiert werden.
Somit kann der Server von Anfang an den korrekten Typ versenden, sofern er ihn unterstützt, und muss seinerseits nicht auch noch eine Auswahl schicken, die zuerst noch ausgewertet werden müsste.
- Subject:** Der freie Text, der hier angefügt werden kann, sollte Informationen über den laufenden Anruf geben.

2.3.3. Message Body

Im Message Body sind Informationen enthalten, die die Verbindung genauer umschreiben. Hier können verschiedene Protokolltypen zum Einsatz kommen. Unter anderem sind dies SDP, verschlüsselte SDP, HTML etc.

2.3.3.1. SDP-Struktur

In dieser Arbeit wird das Session Description Protocol (SDP) benützt, welches in der [rfc2327] spezifiziert ist. Wie SIP ist auch SDP ein Produkt der MMUSIC Gruppe. Es dient unter anderem dazu, eine Multimedia Session zu beschreiben.

In diesem Protokoll können nähere Angaben über den verwendeten Mediatypen (Video, Audio, Daten etc.), das Transportprotokoll (RTP/UDP/IP, H.320 etc.), das Mediaformat, die Verbindungsadresse, den Port etc. der zu übertragenden Daten gemacht werden. Sie bestehen

wie im Header aus einem Headernamen und einem Headerwert und weisen untenstehendes Format auf:

```
header name = header value CRLF
```

Anders als bei den Headers dürfen hier keine Leerzeichen vor und nach dem '=' eingefügt werden.

2.3.4. SIP Request

SIP Requests werden vom Client zum Server geschickt. Die Methoden unterscheiden die Nachrichten und lösen beim Server entsprechende Responses aus. Die sechs verschiedenen Methoden werden nachfolgend erläutert.

ACK: Der Client bestätigt mit einem ACK, dass er vom Server eine Final-Response (z.B. 200 OK) erhalten hat und beendet so den Tripple Handshake.

BYE: Um einen Anruf zu beenden wird entweder vom Anrufer oder vom Angerufenen ein BYE-Request versendet.

CANCEL: Hat der Server nicht mit einer Final-Response auf eine Request geantwortet, kann der Client den Request mit einem CANCEL abbrechen.

INVITE: INVITE wird benützt, um einen Anruf zu initialisieren.

OPTIONS: Der Client kann mit dieser Methode mehr über die Kapazitäten und die unterstützten Methoden des Servers erfahren.

REGISTER: Clients können ihre aktuelle Position und damit die aktuelle Adresse registrieren lassen. SIP Server, welche REGISTER Requests behandeln können, werden Registrar genannt.

2.3.5. SIP Response

Ein SIP Server antwortet mit einer oder mehreren Response Messages auf einen Request. Mit einer Final-Response, deren Status Code 200-699 ist, wird eine SIP Transaktion beendet. Mit einer provisorischen Response, Status Code 1xx, wird die Transaktion nicht beendet.

In einer Start-Line, genauer Status-Line, wird einem Status Code eine Reason Phrase angefügt, die den Status Code in Worten beschreibt. Jeder Status Code hat seine eigene Reason Phrase.

Der Server kopiert grösstenteils den Header, den er in einer Request erhält, in den Header seiner Response. Falls der Server dem Client noch zusätzliche Informationen schicken will, fügt er den betreffenden Header an.

Es wird zwischen sechs verschiedenen Klassen beim Status Code unterschieden¹. Das erste Zeichen bezeichnet die Kategorie.

Falls vom Client ein spezifischer Status Code Cxx nicht verstanden wird, soll er ihn wie ein C00 Code behandeln. Somit können auch ältere Terminals auf neuere Status Codes richtig reagieren und der User kann anhand der Beschreibung auf das Problem schliessen.

¹ Siehe Anhang E

2.4. SIP Entities

Die verschiedenen SIP Einheiten haben unterschiedliche Einsatzgebiete und verhalten sich dementsprechend anders.

2.4.1. Registrar

Der Registrar akzeptiert als einziger Server die REGISTER Message. Mit Hilfe von Registrars können User, deren IP Adresse wechselt, immer und überall erreicht werden. Der Zweck des Registrars ist, dass er die Verknüpfung zwischen wechselnder IP und fixen SIP Adresse aufrechterhält.

REGISTER Anweisungen müssen regelmässig wiederholt werden, denn sonst wird die Verbindung IP- zu SIP-Adresse aufgelöst (Standardwert ist eine Stunde).

2.4.2. Proxy

Proxies sind gleichzeitig Client und Server in einem, denn sie leiten erhaltene Nachrichten weiter. Sie können die Nachricht mit oder ohne Veränderungen weiter senden und können sogar eine lokal generierte Response versenden.

Da SIP protokollunabhängig ist, besitzen alle Nachrichten einen Via Header. Dieser hilft, eine Schleife beim Routing zu verhindern und hat zur Folge, dass die Response den gleichen Weg zurücklegt wie die Request auf dem Hinweg. Das kann genutzt werden, wenn z.B. der Anruf verrechnet werden soll.

Der Proxy fügt bei Requests seinen Namen im Via Header an, wenn er nicht schon darin enthalten ist und nimmt bei Responses seinen Namen wieder aus der Liste.

Sollen nicht nur Request und Response über denselben Weg geroutet werden, so müssen die Proxies nicht nur einen Eintrag im Via Header vornehmen sondern auch im Record Route Header.

Requests können von einem Forking Proxy dupliziert werden, um an mehrere Endpunkte (Server) verschickt zu werden. Das kann sinnvoll sei, wenn sich eine Person an mehreren Standorten angemeldet hat oder jemand gesucht wird. Wenn ein Server mehrere gleiche Requests erhält, muss er sie alle beantworten.

2.4.3. Redirect Server

Ein Redirect Server antwortet auf ein INVITE Request mit einer 3xx Response. Die 3xx Response gibt dem Client Auskunft, dass der Server nicht hier registriert ist und gibt mögliche alternative Adressen an.

3. Analyse

Zuerst wurde das Problem SIP User Agent in Teilsysteme zerlegt. Es wurde festgestellt, dass sich das Problem grob in die Teile Sprachverbindung und SIP Verbindung aufteilen lässt. Bei der Sprachverbindung handelt es sich um eine bidirektionale Punkt zu Punkt Verbindung. Das heisst, dass jedes Gerät gleichzeitig senden und empfangen kann. Bei der SIP Verbindung handelt es sich um eine Client/Server Verbindung. Die Verbindung wird durch den Client hergestellt, während der Server auf Anfragen des Clients antwortet. Die Sprachverbindung musste nicht weiter untersucht werden, da bereits ein laufendes Programm zur Verfügung gestellt wurde.

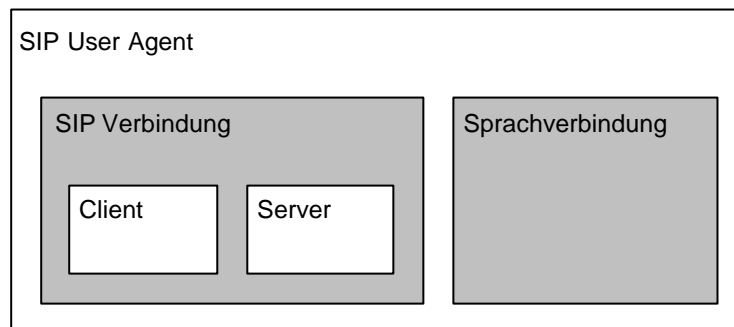


Abbildung 3: Grober Aufbau des SIP User Agent

Für die SIP Verbindung wurde nach möglichen Klassen gesucht:

- User Agent Client: Er ist für die Verbindungsverwaltung zuständig und somit für das Erstellen von Transaktionen. Er versendet Requests und verarbeitet Responses
- User Agent Server: Er ist für die Verbindungsverwaltung zuständig, empfängt die Requests und antwortet mit einer entsprechend Response.
- SIP-Socket: Er ist für das Versenden und Empfangen von SIP-Messages zuständig. Er kann einen UDP- oder TCP-Socket sein. In dieser Arbeit wurde in der Aufgabenstellung¹ UDP als Transportprotokoll gewählt.
- SIP-Message: Sie ist ein Request oder eine Response und enthält Informationen über die aktuelle Transaktion. Sie setzt sich aus den Elementen Startline, Message Headers und Message Body zusammen².
- Session: Sie beschreibt einen Anruf und enthält Informationen betreffend der aktuellen Verbindung.
- Zustand: Client und Server sind in der [rfc3261] als Zustandsmaschinen erklärt. Es ist jeweils immer ein Zustand aktiv. Man unterscheidet zwischen Client und Server Zuständen und zwischen Invite und non-Invite Zuständen. Invite Zustände werden bei Invite Transaktionen eingesetzt, non-Invite bei den restlichen.
Um die Zustände eindeutig beschreiben zu können wurde folgende Notation gewählt: Der erste Buchstabe gibt an, ob es sich um einen Zustand des Clients (C) oder des Servers (S) handelt. Nach dem

¹ Siehe Kapitel 1.1

² [rfc3261], Kapitel 7

Underscore folgt ein INV für Invite oder ein NINV für non-Invite. Anschliessend kommt der Name der Zustands wie er in der [rfc3261] beschrieben ist. Bsp.: C_INVTrying

In einem weiteren Schritt wurde ein Klassendiagramm erstellt. Der Client und der Server wurden im weiteren Verlauf getrennt betrachtet.

3.1. Client

Um einen Anruf zu erstellen, muss der Client eine gültige Session, sowie den INVITE Request erstellen und dem SIP Socket übergeben, damit dieser die Message versenden kann. Danach muss der Zustand C_INVCalling erstellt werden. Der SIP Socket empfängt dann eine Response und leitet diese an den Client weiter, welcher prüft, ob die Message zur aktuellen Transaktion gehört. Wenn ja wird die Message an den Zustand übergeben. Dieser wertet die Nachricht aus und reagiert entsprechend (weitere Message versenden, Request wiederholen, Zustand wechseln etc.). Dies wird so lange durchgeführt, bis die Transaktion abgeschlossen ist. Bei einem non-INVITE wird zu Beginn der Transaktion der Zustand C_NINVTrying aufgerufen. Der Rest entspricht der INVITE Prozedur.

Anhand der SIP Daten wird der Audio Controller angesteuert. Dies kann zum Beispiel die Wahl des Coders betreffen.

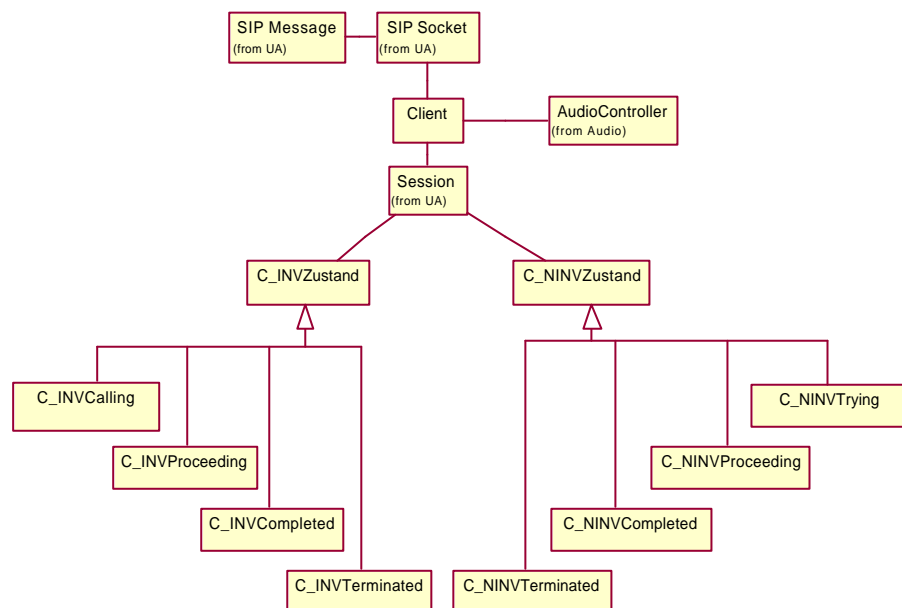


Abbildung 4: Analyse des Clients

3.2. Server

Wenn der SIP Socket einen Request erhält, leitet er diesen an den Server weiter. Handelt es sich um einen INVITE Request, erstellt der Server eine neue Session und eine entsprechende Response. Danach wird dem Zustand S_INVProceeding die erstellte Message übergeben. Daraufhin versendet der Zustand die Message und reagiert entsprechend (weitere Message versenden, Response wiederholen, Zustand wechseln etc.). Wenn weitere Requests folgen, so

überprüft der Server, ob der Request zur aktuellen Session gehört. Trifft dies zu wird darauf reagiert und die Response dem aktuellen Zustand übergeben, andernfalls wird die Message gelöscht. Wird ein non-Invite Request empfangen, wird analog der Invite Prozedur gehandelt, mit dem Unterschied, dass als erster Zustand S_NINVTrying erstellt wird.

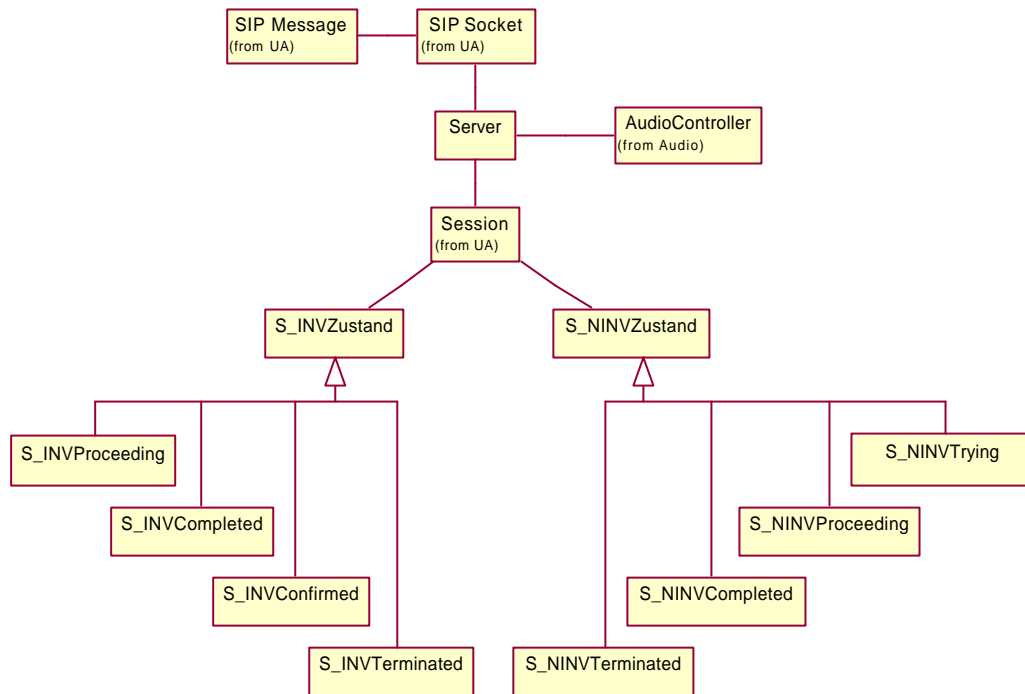


Abbildung 5: Analyse des Servers

4. Design

In der Designphase wurden, von der Analyse ausgehend, die zu implementierenden Klassen bestimmt. Den Klassen wurden ihre Aufgaben und Funktionen zugewiesen und die Schnittstellen mit anderen Klassen vereinbart.

Im Folgenden werden die Hauptklassen und ihr Zusammenspiel vorgestellt und im Weiteren die Funktionsweise des Clients und des Servers erläutert.

4.1. Hauptklassen

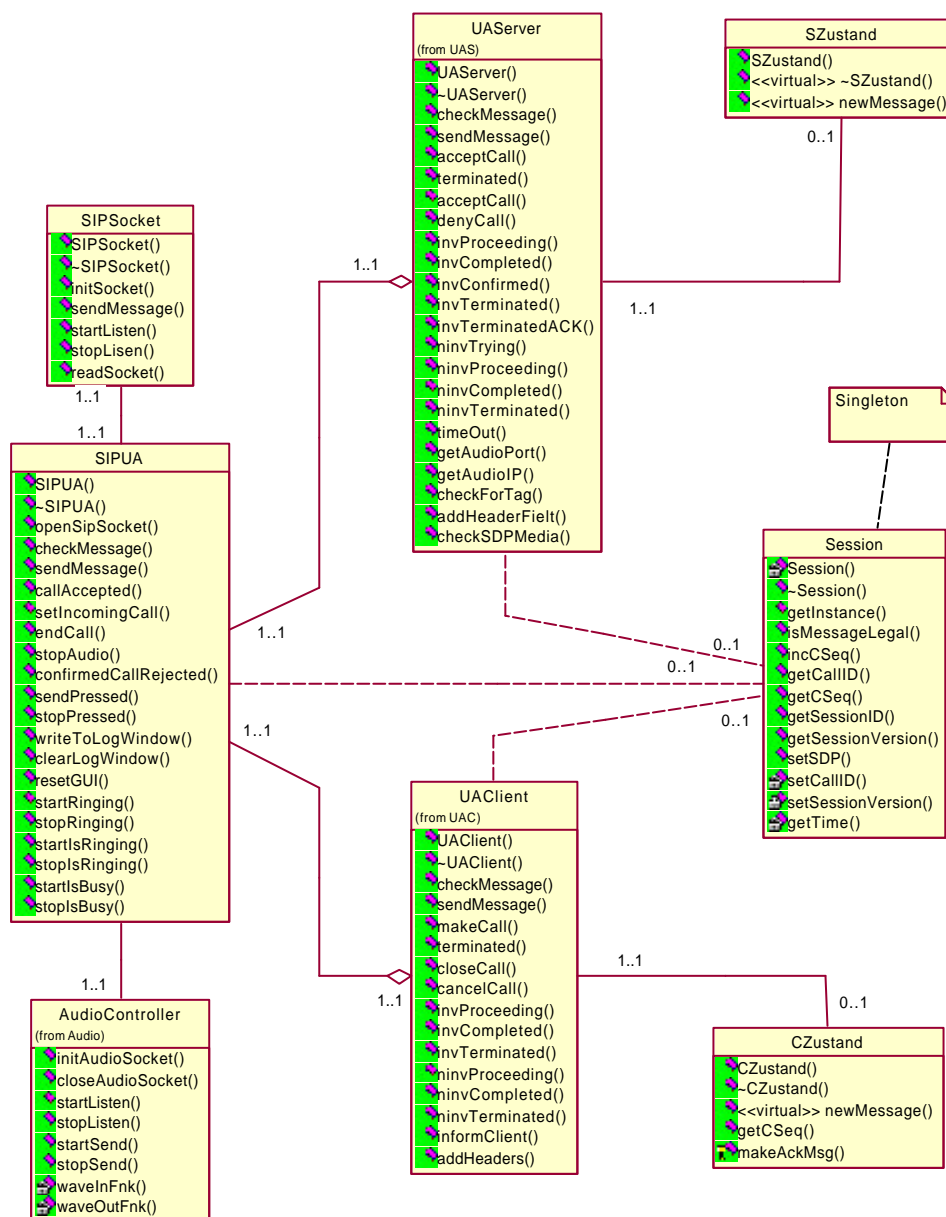


Abbildung 6: Klassendiagramm der Hauptklassen

SIPUA:	Der SIPUA (SIP User Agent) besteht aus UAClient, UAServer, AudioController und SIPSocket. Er entscheidet, ob eine Message zum Client oder zum Server gehört. Er verwaltet die Sprachverbindung mit Hilfe des AudioController und ist die Schnittstelle zum GUI.
UAClient:	Der UAClient erstellt die Requests und verarbeitet die Responses. Er startet die Transaktion.
UAServer:	Der UAServer empfängt die Requests, wertet diese aus und erstellt die entsprechende Response.
AudioController:	Er öffnet und schliesst den Socket für die Sprachverbindung und managt das Audio System.
Session:	Mit ihr wird die aktuelle Verbindung beschrieben. Anhand der Session kann man entscheiden, ob eine Message zur laufenden Verbindung gehört oder nicht.
SIPSocket:	Ein UDP-Socket, der die empfangenen Daten in eine SIPMessage wandelt. Ihm werden SIPMessages zum Versenden übergeben.
CZustand mit seinen Ableitungen:	Mit Hilfe dieser Klassen wird der aktuelle Zustand des Clients beschrieben.
SZustand mit ihren Ableitungen:	Mit Hilfe dieser Klassen wird der aktuelle Zustand des Servers beschrieben.

4.1.1. Sequenzdiagramm

Um einen Anruf zu starten, erstellt der SIPUA eine Session und fordert den UAClient auf, die Invite Transaktion zu starten. Mit Zuhilfenahme der Session erstellt dieser einen INVITE Request, und übergibt die erstellte SIP-Message dem SIPUA. Der SIPUA übergibt die vom UAClient erhaltene SIP-Message dem SIPSocket, welcher diese versendet.

Daraufhin sollte der SIPSocket eine entsprechende Response erhalten, welche er dem SIPUA übergibt. Im Falle eines Requests übergibt er die Message dem UAServer, andernfalls dem UAClient. Der UAClient prüft unter Zuhilfenahme der Session, ob es sich um eine gültige, zur aktuellen Transaktion gehörenden Message handelt. Im Falle eines OK auf das INVITE erzeugt der UAClient eine ACK Message. Diese übergibt er dem SIPUA, welcher wiederum die SIP-Message dem SIPSocket zum Versenden übergibt.

Wenn eine Transaktion abgeschlossen ist, erhöht der UAClient bei der Session die CSeq Nummer und schliesst den aktuellen Zustand.

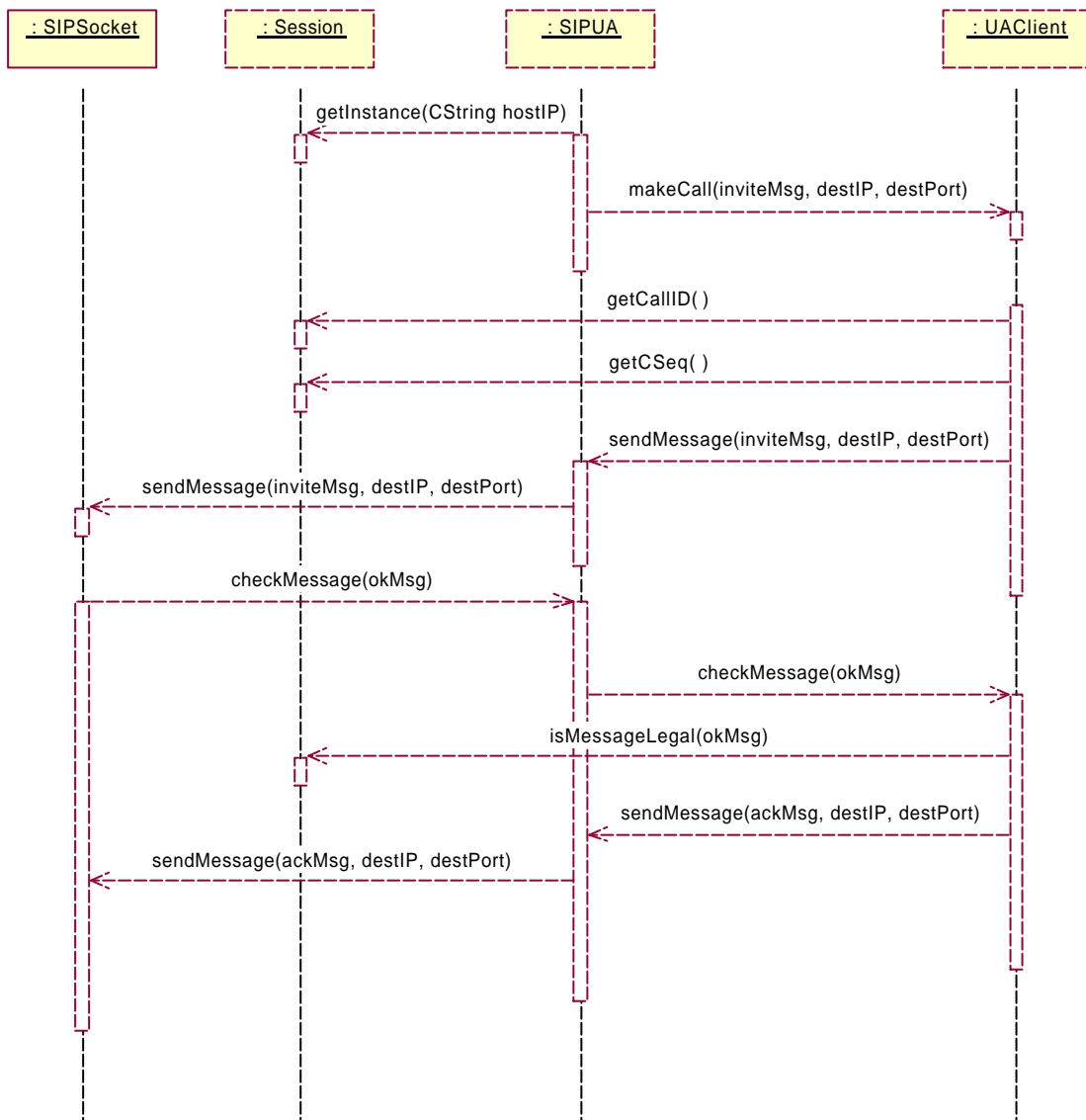


Abbildung 7: Sequenzdiagramm des SIPUA (Invite Transaktion)

4.2. Client

Der Client wird in der Klasse UAClient abgebildet. Der UAClient ist für das Erstellen der Transaktionen zuständig. Zudem erzeugt er die Zustände und kann diese wechseln. Im Gegensatz zur Analyse kennt der UAClient nur sechs anstelle von acht Zuständen. Der Grund liegt darin, dass die Zustände C_INVTerminated und C_NINVTerminated nur dazu da sind, die Transaktion zu beenden und dann geschlossen werden. So können diese beiden Zustände direkt im Zustandswechsel verarbeitet werden.

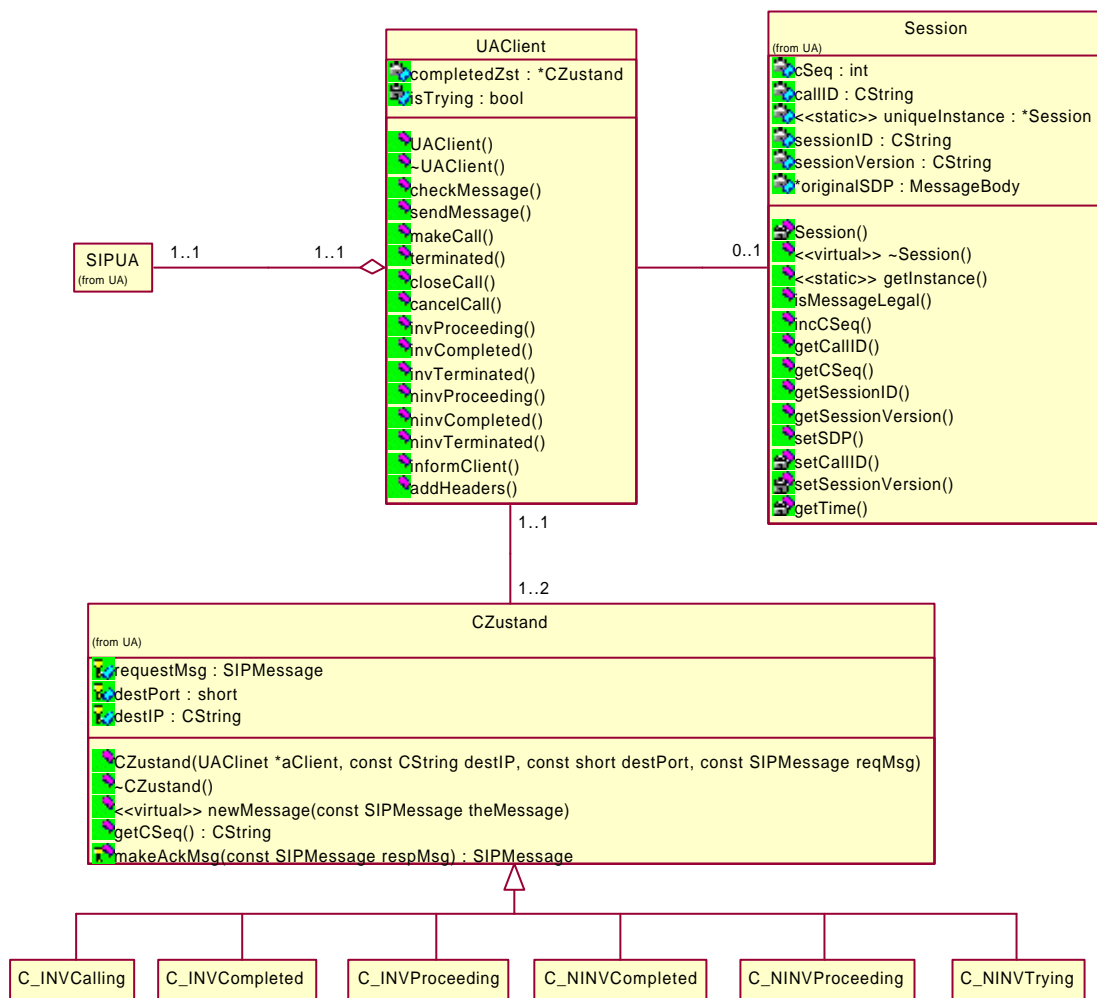


Abbildung 8: Klassendiagramm des UAClient

4.2.1. Sequenzdiagramm

Im folgenden Sequenzdiagramm wird der Ablauf im Client mit den Zuständen und den Zustandswechseln aufgezeigt. Es wurde das Beispiel einer BYE Transaktion gewählt, um das Zusammenspiel von UAClient, SIPUA, Session und den Zuständen aufzuzeigen.

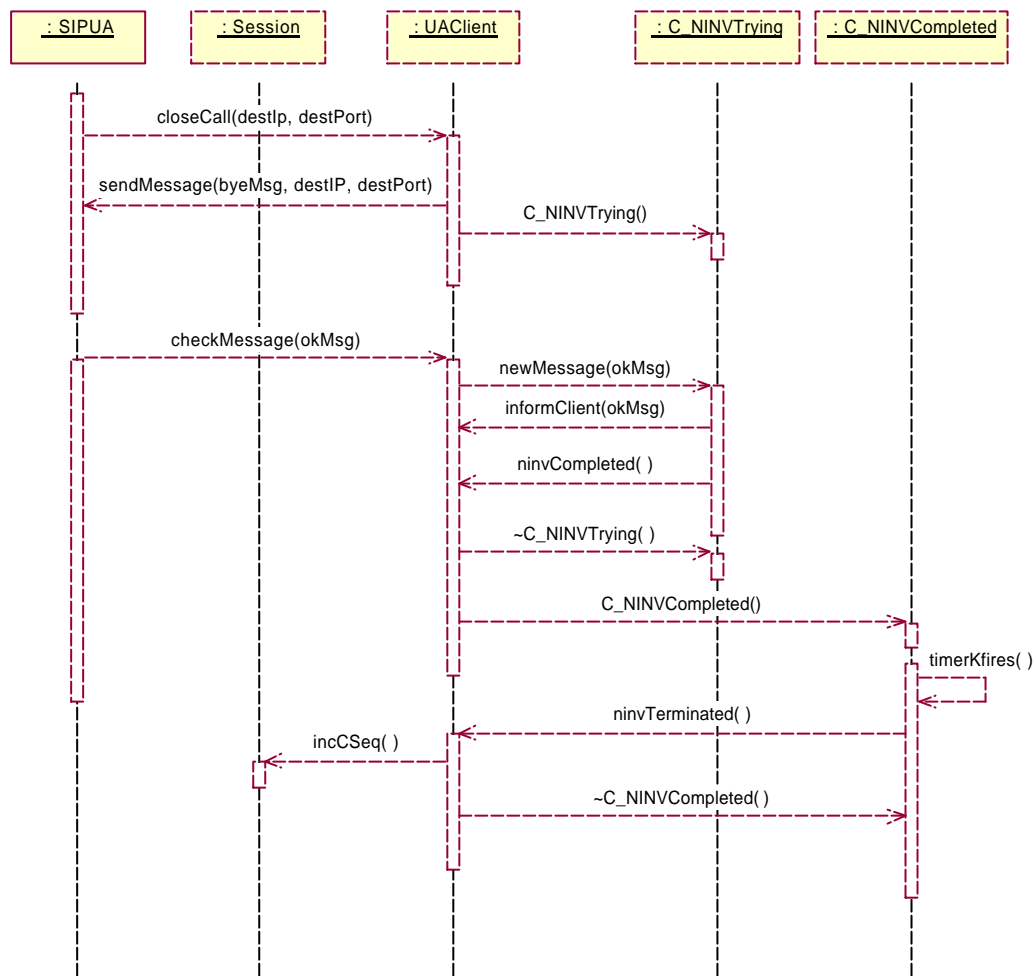


Abbildung 9: Sequenzdiagramm des UAClient (BYE Transaktion)

Der SIPUA ruft beim UAClient die Funktion `closeCall` auf. Der UAClient erstellt ein BYE, übergibt dieses dem SIPUA zum Versenden und öffnet den Zustand `C_NINVTrying`. Der SIPUA übergibt die erhaltene 200 (OK) Message dem UAClient, welcher diese dem aktuellen Zustand übergibt. Der Zustand informiert den UAClient über die empfangene Response und teilt ihm mit, dass er in den `C_NINVCompleted` Zustand wechseln kann. Sobald in diesem Zustand der Timer K abgelaufen ist, wird dem UAClient mitgeteilt, dass er die Transaktion beenden kann. Der UAClient löscht den aktuellen Zustand und erhöht die CSeq-Nummer der Session. Die Transaktion ist somit abgeschlossen.

4.3. Server

Der Server wird in der Klasse `UAServer` abgebildet. Der `UAServer` ist für das Verarbeiten von Requests zuständig. Zudem erzeugt er die Zustände und kann diese wechseln. Im Gegensatz zur Analyse kennt der `UAServer` nur sieben anstelle von acht Zuständen. Zudem wurde ein zusätzlicher Zustand erstellt. Der Grund liegt darin, dass die Zustände `S_INVTerminated` und `C_NINVTerminated` nur dazu da sind, die Transaktion zu beenden und dann geschlossen werden. So können diese beiden Zustände direkt im Zustandswechsel

verarbeitet werden. Der neue Zustand S_INVTerminateACK braucht es, um sicherzustellen, dass der Client auch wirklich ein ACK erhält. Mehr dazu in Kapitel 9.2.1.1.

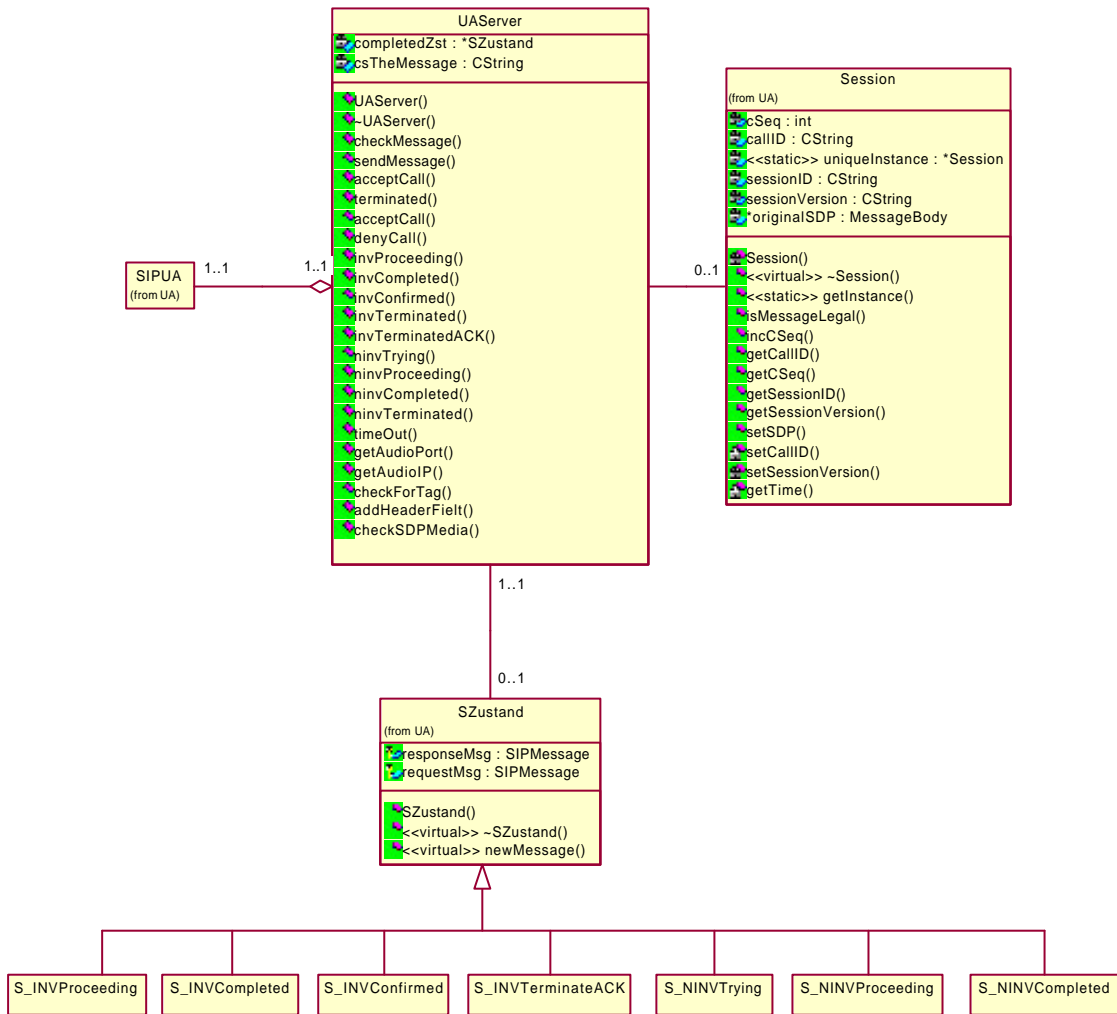


Abbildung 10: Klassendiagramm des UAServer

4.3.1. Sequenzdiagramm

Das abgebildete Sequenzdiagramm zeigt den Ablauf im UAServer anhand des Beispiels eines INVITE Requests und soll das Zusammenspiel zwischen UAServer, SIPUA, Session und den Zuständen zeigen.

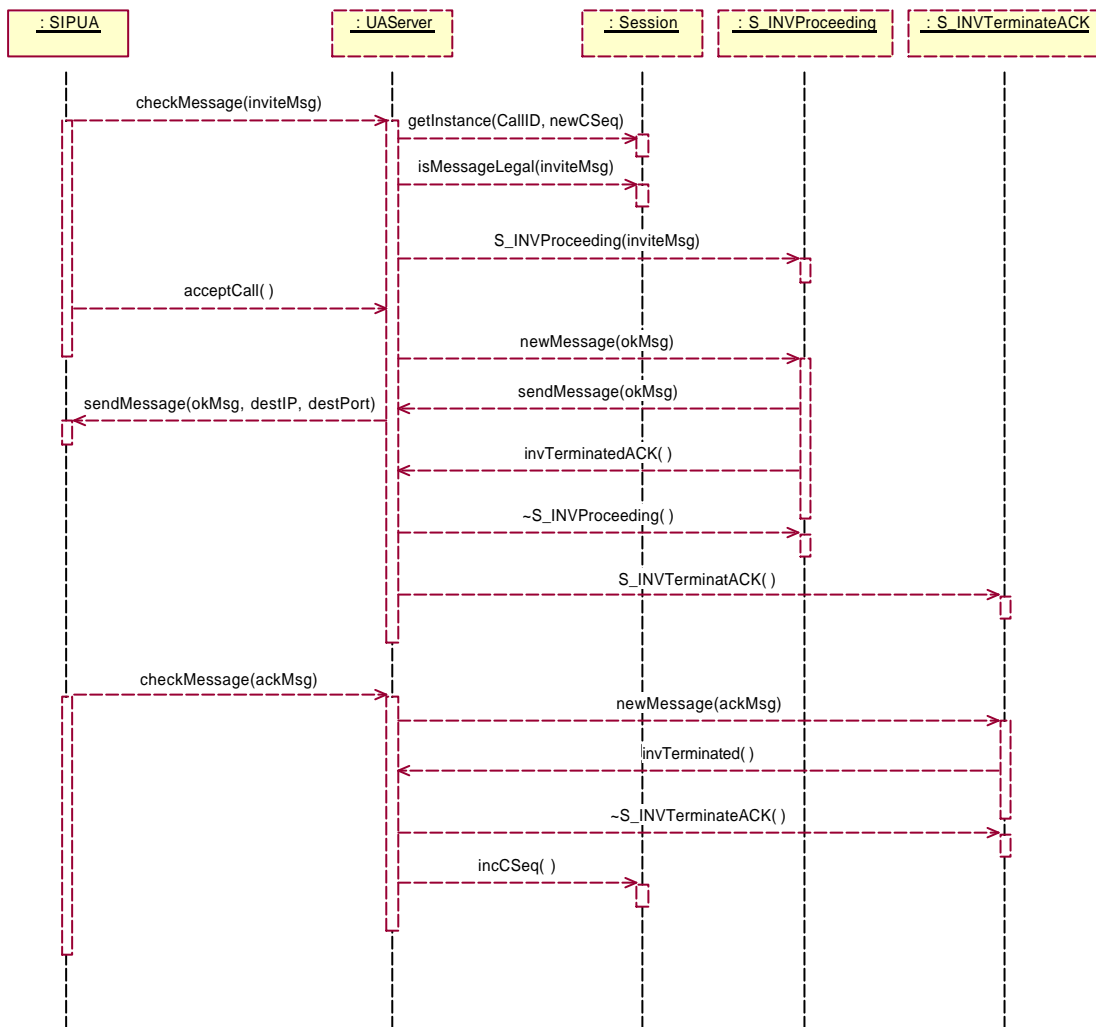


Abbildung 11: Sequenzdiagramm des UAServer (Invite Transaktion)

Der UAServer erhält vom SIPUA eine INVITE Message und erstellt eine neue Session. Er öffnet den Zustand S_INVProceeding und übergibt diesem den INVITE Request. Wenn der Benutzer den Anruf akzeptiert, wird dies dem UAServer mit der Funktion acceptCall mitgeteilt. Der UAServer erstellt eine OK Response und übergibt diese dem aktuellen Zustand. Dieser Zustand wiederum versendet das OK via UAServer und SIPUA und teilt dem UAServer mit, dass er in den S_INVTerminateACK Zustand wechseln kann. Der SIPUA erhält die ACK Message und übergibt sie dem UAServer. Wiederum wird die Message dem aktuellen Zustand übergeben. S_INVTerminateACK teilt dem UAServer mit, dass dieser die Transaktion beenden kann. Der UAServer löscht den aktuellen Zustand und inkrementiert bei der Session die CSeq Nummer.

5. Audio

Der Audioteil der Arbeit wird durch die Klasse `AudioController` beschrieben. Diese Klasse regelt den Audio-Socket, die Sprachcodierung, sowie das Aufnehmen und Abspielen der Sprachdaten. Der `AudioController` enthält diverse Buffer und einen UDP-Socket. Der UDP-Socket wird für das Empfangen und Versenden der Sprachdaten benützt. Es hätte auch die Möglichkeit bestanden je einen Socket für das Versenden und einen Socket für das Empfangen zu erstellen. Dies hätte jedoch keinen Einfluss auf die Datenrate oder ähnliches. Deshalb wurde auf einen zweiten Socket verzichtet, wodurch kein zusätzlicher Port gehandhabt werden muss.

5.1. Anwendung des AudioController

Bevor mit dem `AudioController` Sprachdaten versendet werden können, muss dieser einen Socket öffnen, welcher beim Beenden auch wieder geschlossen werden muss. Wenn der Socket geöffnet ist, kann nach Belieben mit dem Versenden oder Empfangen von Sprachdaten begonnen werden. Wenn der Socket einmal geschlossen worden ist, muss er erneut geöffnet werden um Daten zu versenden oder zu empfangen.

AudioController	
	recordBuffer0[2*SIZE] : char
	recordBuffer1[2*SIZE] : char
	playBuffer0[2*SIZE] : char
	playBuffer1[2*SIZE] : char
	rtpHeader : rtp_hdr_t
	rtpSendBuffer[12+SIZE] : char
	rtpReceiveBuffer[12+SIZE] : char
	wfx : WAVEFORMATEX
	hwi : HWAVEIN
	hwo : HWAVEOUT
	waveHdrRecord[2] : WAVEHDR
	waveHdrPlay[2] : WAVEHDR
	nextRecordBuffer : unsigned char
	nextPlayBuffer : unsigned char
	audioSocket : SOCKET
	audioAddr : SOCKETADDR_IN
	sendAddr : SOCKETADDR_IN
	receiveAddr : SOCKETADDR_IN
	receiveAddrLen : int
	stopLst : int
	stopSnd : int
	initAudioSocket(short portName) : bool
	closeAudioSocket() : bool
	startListen() : bool
	stopListen() : bool
	startSend(CString destIP, short destPort) : bool
	stopSend() : bool
	waveInFnk() : void
	waveOutFnk() : void

Abbildung 12: Klasse `AudioController`

5.1.1. Audio Socket öffnen

Zum Öffnen des Audio Sockets muss die Funktion `initAudioSocket` aufgerufen werden. Ihr muss die Portnummer des Sockets übergeben werden. Es wird nun ein Socket zum Empfangen und Versenden von UDP Daten erzeugt. Der Rückgabewert der Funktion zeigt, ob das Öffnen des Ports erfolgreich war oder nicht. Ist der Rückgabewert `TRUE`, war das Öffnen erfolgreich, ist er `FALSE`, ist es fehlgeschlagen. Grund für ein Fehlschlagen könnte sein, dass der Port bereits anderweitig benutzt wird.

5.1.2. Versenden und Empfangen von Sprachdaten

Wenn das Öffnen des Sockets mit `initAudioSocket` erfolgreich war, kann nach Belieben mit dem Senden und Empfangen der Sprachdaten begonnen werden. Zum Starten der Übertragung muss `startSend` aufgerufen werden. Die Parameter sind die Ziel IP, sowie der Ziel Port. Mit `stopSend` kann die Übertragung wieder beendet werden. Bei beiden Funktionen ist die Rückgabe `TRUE`, wenn der Aufruf erfolgreich war. Ansonsten ist er `FALSE`.

Mit `startListen` können Sprachdaten empfangen werden. Mit `stopListen` wird das Empfangen der Daten beendet. Der Rückgabewert der beiden Funktionen ist `TRUE`, wenn der Aufruf erfolgreich war, respektive `FALSE`, wenn er erfolglos war.

Wenn ein Rückgabeparameter `FALSE` ist, könnte das daran liegen, dass die Funktion bereits vorgängig aufgerufen wurde oder ein Stop aufgerufen wird, ohne dass vorgängig ein Start ausgeführt worden ist.

5.1.3. Schliessen des Audio Sockets

Mit der Funktion `closeAudioSocket` kann der Audio Socket geschlossen werden. Es empfiehlt sich, vor dem Schliessen des Sockets sicherzustellen, dass er nicht mehr in Betrieb ist. Das heisst, dass weder auf Daten gewartet werden muss, noch Daten versendet werden. Zudem muss darauf geachtet werden, dass wenn ein Socket mit `openAudioSocket` geöffnet wurde, dieser mit `closeAudioSocket` vor dem Beenden der `AudioController` Klasse wieder geschlossen wird.

5.2. Implementierung

Als Vorlage für den `AudioController` wurden die im Unterricht betrachteten Programme verwendet. Der Nachteil dieser Programme ist, dass alle Funktionen in der Dialogklasse implementiert wurden und Windows-Messages, welche zum Steuern der Audio Buffer benutzt werden, an die Dialogklasse gesendet werden. Eine solche Anwendung des Audioteils erschwerte eine saubere Aufteilung der Aufgaben in Klassen, da Objekte, die eigentlich keinen direkten Zusammenhang mit der Dialogklasse haben, auf diese zugreifen können müssten. Deshalb wurde eine eigene Klasse entwickelt. Da die Messages, welche im Übungsprogramm verwendet wurden an ein Window gesendet werden müssen, hätte man die Audioklasse von einem Window herleiten müssen. Dies wiederum hätte die Klasse unnötig verkompliziert. Deshalb wurde das Prinzip der Callbackfunktion gewählt.

Für das Buffermanagement wurde das im Unterricht betrachtete Double Buffering System benutzt. Jeweils ein Buffer ist für das Verarbeiten der Daten, der andere für das Zwischenspeichern der Daten zuständig.

5.2.1. Sprachcodierung

Da nicht zu viel Zeit in die Audio Klasse investiert werden sollte, wurde eine einfache Codierung gewählt. Weil das uns zur Verfügung stehende Hardware SIP-Telefon den Coder G.711 μ Law verwendet, war es naheliegend, ebenfalls diesen Coder zu wählen.

Das Code dazu stammt von der Firma Sun Microsystems.

5.2.2. Callbackfunktionen

Da beim Implementieren dieser Funktionen Probleme aufgetaucht sind, werden sie in diesem Abschnitt genauer erläutert.

Die Multimedia Funktion `waveOutOpen` ist für das Abspielen von Audiodaten auf die Soundkarte zuständig. Man hat die Wahl, wie das Audiogerät auf Bufferaktionen reagieren soll. Wegen den unter Kapitel 5.2 genannten Gründen wurde das Prinzip der Callbackfunktion gewählt. Der Funktion `waveOutOpen` wird die Adresse der Callbackfunktion übergeben. Wie die Callbackfunktion aussieht, ist der [msdn] zu entnehmen. Diese statische Funktion wird bei jedem Bufferereignis aufgerufen. Anhand der Message, welche der Callbackfunktion übergeben wird, kann entschieden werden, was für ein Ereignis eingetreten ist. Lautet die Message zum Beispiel `WOM_DONE`, so bedeutet das, dass die Audiodaten abgespielt wurden.

Ein Problem, welches aufgetreten ist, war, dass beim Schliessen des Audiogeräts die Callbackfunktion nochmals mit der Message `WOM_DONE` aufgerufen wurde. Dies obwohl zuvor ein Reset durchgeführt wurde. Die Callbackfunktion ruft nun erneut die `waveOutFnk` auf, welche durch den Zugriff auf den Audio Buffer eine Windows-Message generiert, die wiederum zum Aufruf der Callbackfunktion führt. So entsteht eine Endlosschleife. Dies führt zu einem Absturz des Programms. Mit Hilfe der Hilfsvariabel `stopLst` wird dieses Problem gelöst. `stopLst` muss vor dem Schliessen des Audiogerätes `TRUE` gesetzt werden. Beim Ausführen der Funktion `waveOutFnk` wird zuerst geprüft, ob die Funktion überhaupt ausgeführt werden darf. Ist `stopLst` `TRUE`, wird die Funktion sofort mit `return` abgebrochen.

Derselbe Mechanismus wurde bei den Funktionen zum Aufnehmen von Sprache verwendet.

6. SIPUA

Der SIP User Agent (SIPUA) ist die Schnittstelle zum GUI und somit zum User. Durch ihn können Anrufe gestartet, empfangen, abgelehnt und beendet werden. Er kann auf das Log-Window schreiben und die Buttons aktivieren oder deaktivieren. Er kann das Abspielen der Klingeltöne starten und stoppen und den SIPSocket, den AudioController sowie den UAClient und den UAServer initialisieren. Neben dem Erstellen der Session bei einem ausgehenden Anruf, entscheidet er, ob eine empfangene SIPMessage zum UAClient oder zum UAServer weiterleitet werden muss.

6.1. Anwendung

Das Hauptprogramm ruft neben dem Konstruktor und dem Destruktor drei Memberfunktionen auf.

`openSipSocket`: Mit dieser Funktion wird der Socket für die Übertragungen der SIP-Messages geöffnet. Der Funktion kann die Portnummer übergeben werden. Wird keine Portnummer angegeben, wird der Standardport für SIP gewählt (5060)¹.

`sendPressed`: Wenn der Benutzer auf den „OK“ Button klickt, wird diese Funktion gestartet. Der SIPUA entscheidet dann, ob ein Anruf erstellt oder angenommen wird. Wenn zu diesem Zeitpunkt keine Ziel IP Adresse eingegeben ist, wird der Anrufversuch nach Ablauf des Timers B abgebrochen.

`stopPressed`: Wenn der Benutzer auf den „Abbrechen“ Button klickt, wird diese Funktion gestartet. Der SIPUA entscheidet ob ein Anruf beendet oder abgelehnt wird. Dieser Funktion müssen keine Parameter übergeben werden.

6.2. Implementierung

Damit der SIPUA entscheiden kann, welche Buttons aktiv sein müssen und in welchem Zustand sich die Verbindung befindet (eingehender Anruf, ausgehender Anruf, Verbindung aufgebaut etc.), muss er vom UAClient und UAServer informiert werden. Dies geschieht über einige Hilfsfunktionen, die nicht weiter erläutert werden.

¹ [rfc3261] Kapitel 18.1.1 und 18.2.1

7. Session

In der Session werden die Informationen zur aktuellen Verbindung gespeichert. Dies gilt sowohl für die SIP Verbindung als auch für die Sprachverbindung.

Die Session ist in der Klasse `Session` abgebildet. Eine Besonderheit dieser Klasse ist, dass das Singleton Pattern¹ verwendet wurde.

7.1. Anwendung

Für alle Memberattribute existieren get-Funktionen und zum Attribut `originalSDP` auch eine set-Funktion. Die Anwendung dieser Funktionen ist klar und wird hier nicht weiter beschrieben.

Die `Session` enthält noch einige weitere Funktionen:

- `getInstance`: Mit `getInstance` erhält man die einzige Instanz auf das existierende Objekt der Klasse `Session`. Wenn noch kein Objekt existiert, wird mit diesem Aufruf ein Objekt erzeugt. Dabei muss ein String (vorzugsweise die eigene IP-Adresse) und wenn vorhanden die aktuelle CSeq Nummer übergeben werden. Aus der aktuellen Zeit und dem übergebenen String wird die Call-ID der Session erstellt².
- `incCSeq`: Wenn eine Transaktion abgeschlossen ist, muss mit dem Aufruf dieser Funktion die CSeq-Nummer der Session inkrementiert werden.
- `isMessageLegal`: Dieser Funktion wird eine `SIPMessage` übergeben. Wenn die Message zur aktuellen Transaktion gehört, ist der Rückgabewert der Funktion `TRUE`, ansonsten `FALSE`.

¹ [uml-lehrbuch], Kapitel 7.3

² [rfc3261], Kapitel 8.1.1.4

8. UA Client

Der UA Client ist für das Erstellen von Requests zuständig. Er fügt der SIP-Message, welche er vom User Agent erhält, die nötigen SIP-Header an und übergibt die Message wieder dem User Agent. Weiter regelt er die Zustände. Er öffnet einen Zustand, schliesst diesen auch wieder und ist dafür verantwortlich, dass jeweils nur ein Zustand offen ist und die Wechsel richtig funktionieren.

Der User Agent Client wird mit der Klasse `UAClient` abgebildet.

8.1. Anwendung

Von aussen sind vier Funktionen von Bedeutung. Die Funktionen beziehen sich auf die Art des Anrufs, sprich was der Benutzer mit dem Drücken der „OK“ oder „Abbrechen“ Taste bewirken wollte. Auf die einzelnen Funktionen des `UAClient` wird im Folgenden eingegangen:

`makeCall`: Mit dieser Funktion wird ein neuer Anruf aufgebaut und eine INVITE Message generiert. Zuvor muss vom User Agent eine gültige Session geöffnet worden sein, von welcher der `UAClient` Informationen für den Call-ID und CSeq Header bezieht.

`closeCall`: Ein bestehender Anruf wird beendet und der `UAClient` erstellt eine BYE Message. Zuvor muss eine SIP-Verbindung mit einer Partnerstation aufgebaut worden sein, die dann beendet wird.

`cancelCall`: Ein Anrufversuch wird abgebrochen. Wenn ein Verbindungsversuch vom Benutzer abgebrochen wird, soll diese Funktion aufgerufen werden. Wenn zuvor eine Provisional-Response empfangen wurde, sendet der Client ein CANCEL. Wenn nicht, teilt der `UAClient` dem User Agent mit, dass der Anruf beendet worden ist.

`checkMessage`: Wenn vom User Agent eine Response empfangen wurde, wird die Message mit dieser Funktion an den `UAClient` weitergegeben. Hier wird überprüft, ob es eine Response auf einen aktuellen Request ist. Wenn ja wird er dem aktuellen Zustand übergeben, ist dies nicht der Fall, wird die Response verworfen.

8.2. Implementierung

Neben dem Erstellen von Requests hat der `UAClient` noch zwei weitere Aufgaben: Er muss dafür sorgen, dass Responses richtig ausgewertet werden können und er muss die Zustände, welche in der [rfc3261], Kapitel 17.1 beschrieben sind, regeln.

8.2.1. Zustandsdiagramm

Wie in der [rfc3261] beschrieben, wird der Client (wie auch der Server) mit Hilfe eines Zustandsdiagramms geregelt. Die in der [rfc3261], Kapitel 17.1 beschriebenen Zustandsdiagramme wurden weitestgehend übernommen. Einzig beim Invite Zustand Completed wurde eine grössere Änderung vorgenommen.

8.2.1.1. INVITE Zustandsdiagramm

Das INVITE Zustandsdiagramm hat im Gegensatz zum Diagramm der [rfc3261] eine wesentliche Änderung erfahren. Der Grund dieser Änderung liegt darin, dass nach [rfc3261] zuerst der Timer D abgelaufen sein muss, bevor der Zustand Terminated erreicht wird. Wie aus [rfc3261], Kapitel 17.1.1.2 ersichtlich, dauert dies mindestens 32 Sekunden. Wenn nun eine Verbindung aufgebaut worden ist, muss man mindestens 32 Sekunden warten, bevor man diese Verbindung beenden kann, da jeweils nur ein Zustand aktiv sein darf. Diese Lösung ist unbefriedigend, da der Benutzer zu lange aufgehalten wird.

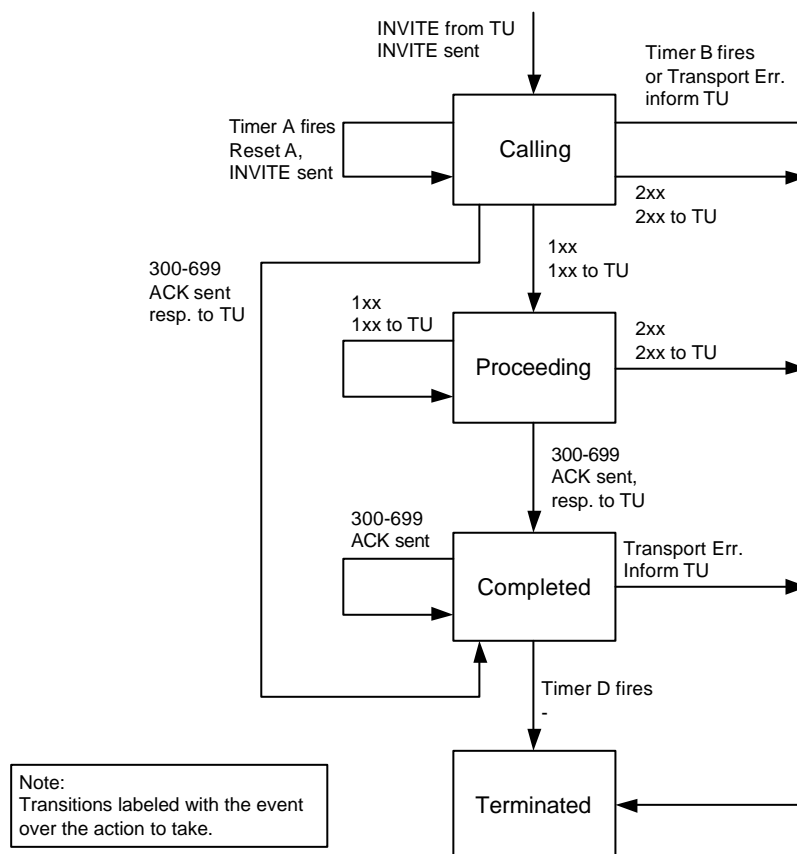


Abbildung 13: Invite Zustandsdiagramm des Clients laut [rfc3261]

Wenn man die Funktion des Zustands Completed betrachtet, stellt man fest, dass sie dazu da ist, auf wiederholte Final-Responses zu Antworten. Final-Responses werden vom Server erneut versendet, wenn dieser kein ACK erhalten hat, das ACK also nicht, oder falsch vom Netzwerk übertragen wurde. Da dies selten der Fall sein wird, kann man davon ausgehen,

dass der Server das erste, beim Zustandswechsel versendete, ACK erhalten hat und seine Transaktion beendet ist. Somit kann eine weitere Transaktion begonnen werden.

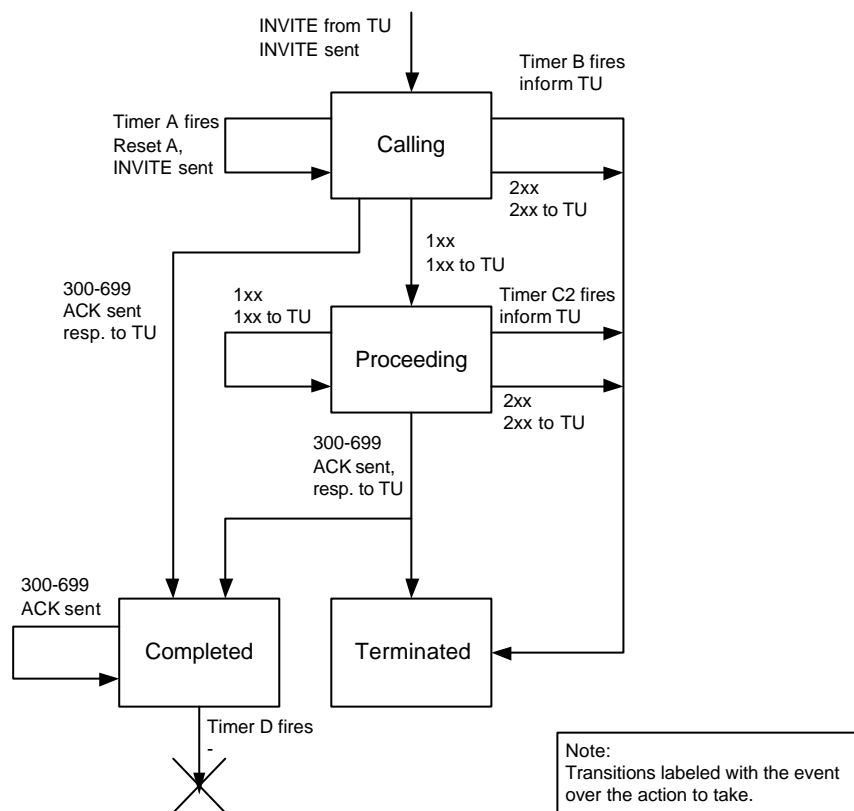


Abbildung 14: Verändertes Invite Zustandsdiagramm des Clients

Das Problem wurde folgendermassen gelöst: Der Client verwaltet neben einem aktuellen auch einen alten Zustand, welcher beim Wechseln in den Completed Zustand erstellt wird. Gleichzeitig wird vom aktuellen Zustand auch in den Terminated Zustand gewechselt. Somit sind kurzfristig zwei Zustände offen. Im Terminated Zustand wird die CSeq Nummer inkrementiert und die Invite Transaktion geschlossen. Wenn nun eine alte Nachricht beim Client eintrifft, wird diese in den Completed Zustand weitergeleitet. Dort wird geprüft, ob es sich um eine Response auf das INVITE bezieht. Ist dies der Fall, wird das ACK erneut versendet und der Timer D neu gestartet. Wenn der Timer D abgelaufen ist, wird der Zustand Completed gelöscht. Wenn der Client erneut in den Completed Zustand kommt, weil z.B. die alte Verbindung kurz nach dem Erstellen wieder beendet wurde und bereits eine neue Verbindung im Aufbau ist, wird der alte Completed Zustand gelöscht und sofort ein neuer Completed Zustand geöffnet. Probleme können keine entstehen, da die Zustandsmaschine nicht nochmals in den Completed Zustand kommen kann, ohne dass die Partnerstation auf ein Request geantwortet hätte, oder die Verbindung nicht sowieso unterbrochen wurde. Somit ist klar, dass der Server das ACK erhalten hat, sonst würde er die neue Message nicht verstehen. Im Gegensatz zum Diagramm der [rfc3261] wurde beim Proceeding Zustand ein weiterer Timer eingebaut. Der Timer hat den Namen C2 bekommen, da er in etwa dieselbe Funktion wie Timer C¹ hat. Timer C2 läuft nach 90 Sekunden ab. Der Angerufene hat somit eineinhalb Minuten Zeit, das Gespräch anzunehmen (dies entspricht auch etwa der Dauer welche die Swisscom benutzt). Wenn er dies nicht tut, wird die Transaktion beendet. Zudem wurde die

¹ Siehe [rfc3261], Kapitel 16.8

Möglichkeit eines Transport Errors ausgeschlossen, da dieser UAClient dies nicht erkennen kann. Diese Möglichkeit ist auch mehr auf TCP als auf UDP bezogen.

8.2.1.2. non INVITE

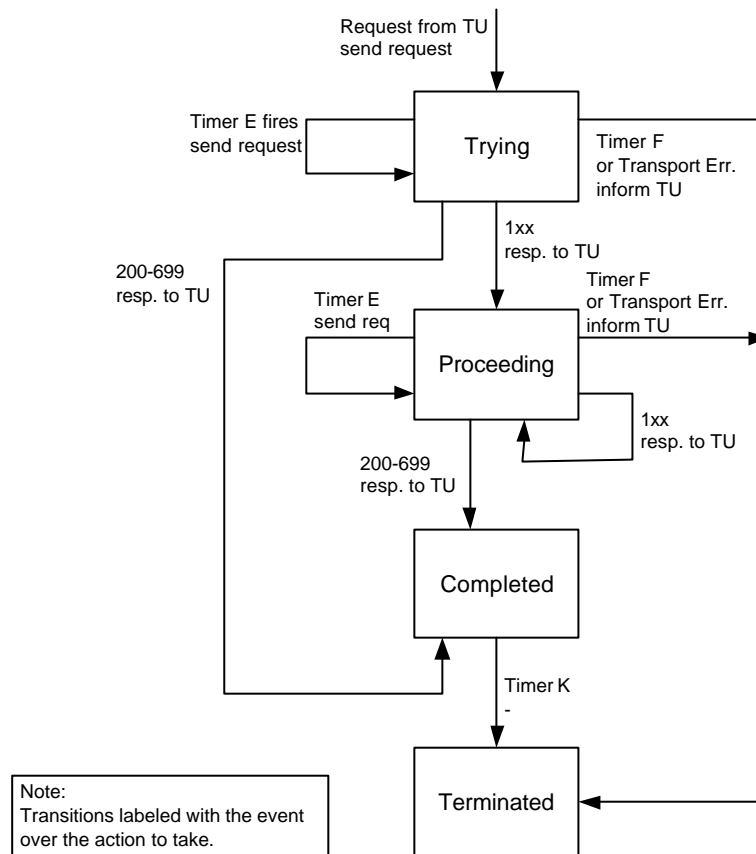


Abbildung 15: Non-Invite Zustandsdiagramm des Clients laut [rfc3261]

Das non-INVITE Zustandsdiagramm wurde in seiner Form belassen, ausser, dass wie beim INVITE Zustandsdiagramm die Transport Error Möglichkeit nicht besteht. Auch hier wurde die Überlegung gemacht, den Completed Zustand anders zuhandhaben. Der Timer K hat allerdings eine andere Funktion als der Timer D. Er sorgt dafür, dass der Zustand solange offen bleibt, bis durchs Netzwerk verspätete oder verdoppelte Messages ankommen und somit im Zustand abgearbeitet werden können. Wenn anstelle des Completed Zustand direkt in den Terminated Zustand gewechselt wird, besteht die Möglichkeit, dass der Socket geschlossen ist. In diesem Falle würden ICMP Pakete versendet, was zu einem erhöhten Traffic führen würde. Deshalb wird der Completed Zustand für fünf Sekunden¹ am Leben erhalten.

Da bei dieser Implementation des Clients der Timer keinen gravierenden Einfluss hat, wird der Zustand im Sinne der [rfc3261] so belassen. Der einzige Nachteil ist, dass beim Ablehnen des Anrufs das Programm für fünf Sekunden angehalten wird. Danach wird der „OK“ Button wieder freigeschaltet.

¹ Siehe [rfc3261], Kapitel 17.1.2.2

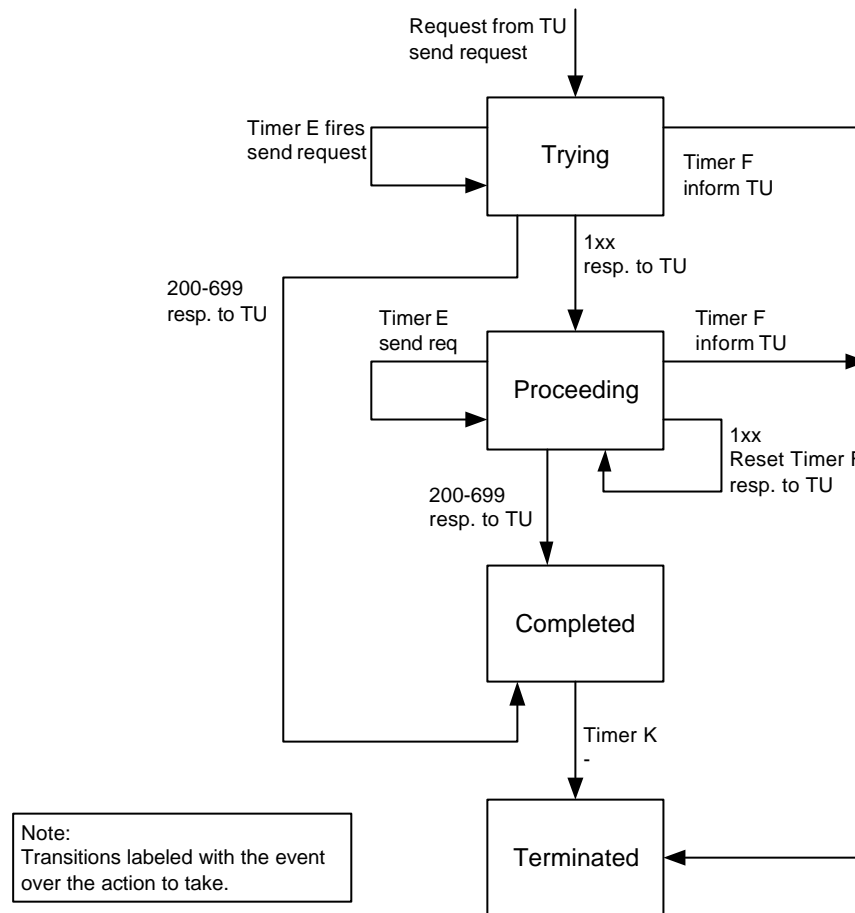


Abbildung 16: Verändertes non-Invite Zustandsdiagramm des Clients

8.2.2. Auswertung der Responses

Wenn der UA Client eine Response erhält, können mehrere Fälle vorkommen. Je nach aktuellem Zustand und Art der Response muss der Client darauf reagieren.

Bei diesem Client werden zwei Arten von Responses unterschieden. Zum einen sind dies die Provisional-Response (1xx), zum anderen die Final-Responses (2xx -6xx). Da das Programm einfach gehalten werden muss, wurden die Fehlermeldungen des Servers nicht weiter untersucht. Wenn als Final-Response kein 2xx kommt, wird der Anruf als abgelehnt betrachtet und es werden keine weiteren Schritte eingeleitet. Wie genau der Client reagiert, wird nachfolgend erklärt.

- 100-199: Es wird der Reason Phrase der Message im Log-Window angezeigt. Ein Spezialfall tritt auf, wenn ein 180 (Ringing) empfangen wurde. Dann wird der Reason Phrase nicht angezeigt, dafür wird das Klingeln für ausgehende Anrufe gestartet.
- 200-299: Hier wird weiter unterschieden: Ausschlaggebend ist, auf welchen Request geantwortet wird:
 - INVITE: Das Klingeln für ausgehende Anrufe wird gestoppt, die CSeq Nummer um eins erhöht und es wird mit dem Versenden von Sprachdaten begonnen.

- BYE: Das Beenden der Verbindung wurde bestätigt. Die Verbindung wird geschlossen und dem User Agent wird mitgeteilt, dass die Session gelöscht werden kann.
- CANCEL: Das CANCEL wurde bestätigt und die Verbindung wird beendet. Es wird ein „CANCELING...“ auf das Log-Window ausgegeben. Theoretisch könnte ein CANCEL auch bei anderen Methoden als bei einem INVITE angewendet werden. Dies ist jedoch nicht möglich, da der Client nur auf INVITE's ein CANCEL versendet, wie das in der [rfc3261], Kapitel 9.1 empfohlen wird.
- 300-699: Der Request wurde nicht akzeptiert. Anhand der Fehlernummer (der Fehlertext kann nach [rfc3261], Kapitel 7.2 frei gewählt werden), könnte weiter unterschieden werden, weshalb ein Fehler aufgetreten ist. Mögliche Gründe sind, dass der Angerufene den Anruf abgewiesen hat (486) oder der Media Typ nicht unterstützt wird (406). Wie bereits erwähnt, wird die Art der Fehlermeldung nicht weiter unterschieden, sondern nur – wie bei der OK Message –, auf welche Methode geantwortet wurde.
- INVITE: Das Klingeln für ausgehende Anrufe wird gestoppt. Die Verbindung wird mit einem CANCEL abgebrochen und es ertönt für etwa fünf Sekunden ein Besetztzeichen. Die andere Möglichkeit wäre gewesen, die Response mit einem ACK zu bestätigen und anschliessend ein BYE zu senden. Anfangs wurde dies auch so gelöst, es traten dann aber Probleme mit den Klingeltönen auf. So wurde die Variante mit dem CANCEL gewählt. Das Beenden des Anrufs gleicht somit demjenigen des BYE.
- BYE: Wie beim INVITE wird hier der Anruf mit einem CANCEL beendet. Es wird für etwa fünf Sekunden ein Besetztzeichen ausgegeben.
- CANCEL: Es wird ein „CANCELING...“ auf das Log-Window angezeigt und die Verbindung wird beendet.

9. UAServer

Einer der Hauptaufgabenbereiche des UAServer beinhaltet das Analysieren der erhaltenen SIP Request Message, die er vom User Agent (UA) erhält. Meist wird dem Benutzer des Programms mitgeteilt, was für ein Request eingegangen ist, indem der Server eine Information im Log-Window anzeigt. Der Benutzer kann nun die Response, die der Server generieren muss, zu einem gewissen Teil mitgestalten, indem er z.B. einen Anruf annimmt oder ablehnt.

Eine weitere Aufgabe des Servers ist es, die Zustände, in denen sich eine Transaktion befindet zu regeln. Er öffnet einen neuen Zustand und schliesst ihn wieder, wenn eine Zustandsänderung erfolgt ist. Die Verantwortung, dass jeweils nur ein Zustand offen ist und der Zustandswechsel korrekt erfolgt, liegt beim Server.

9.1. Anwendung

Die Responses des UAServer hängen vornehmlich von den eingehenden Nachrichten und den Aktionen des Benutzers ab. Diese Antworten werden vom User Agent an den UAServer weitergeleitet indem der UA die entsprechenden Funktionen aufruft. Nachfolgend wird auf diese Funktionen eingegangen:

- checkMessage:** Wird vom User Agent ein Request empfangen, wird die Message dem UAServer mit dieser Funktion übergeben. Nun wird überprüft, ob die Nachricht zu einer aktuellen Transaktion gehört. Trifft dies zu, wird die Response weiter verarbeitet. Das bedeutet, dass je nach Methode des Requests entsprechend gehandelt wird und der betreffenden Zustand geöffnet und in diesen gewechselt wird.
Gehört der Request zu der erst vergangen Transaktion, wird überprüft, ob der Request mit dem vorangegangenen Request übereinstimmt. Ist das der Fall, wird in den entsprechenden Zustand gewechselt.
Falls der Request zu keiner der oben genannten Transaktionen passt, wird der Request verworfen und es wird nicht geantwortet.
- acceptCall:** Mit dieser Funktion teilt der SIPUA dem UAServer mit, dass der Benutzer den eingegangenen Anruf entgegennehmen möchte. Dabei übernimmt er den eingegangenen Request, der dem Benutzer zuvor mitgeteilt hat, dass er angerufen wird. Diesen Request wird in eine Response umgewandelt und mit einem Status Code 200 (OK) versehen.
- denyCall:** Wenn der Benutzer den Anruf ablehnen will, teilt der UA dies dem UAServer mit dieser Funktion mit. Der Server ist nun zuständig, dass eine Response generiert wird, die dem Client und somit dem Anrufer mitteilt, dass die Sprachverbindung nicht gewünscht ist. Dazu versendet der Server ein 486 (Busy Here).

9.2. Implementierung

Neben dem Versenden der Response muss der UAServer die Requests korrekt auswerten und in die entsprechenden Zustände wechseln können. Die Zustandsdiagramme sind in der [rfc3261], Kapitel 17.2 beschrieben.

9.2.1. Zustandsdiagramm

Die in der [rfc3261] beschriebenen Zustandsdiagramme wurden weitestgehend übernommen. Zwei Änderungen sind trotzdem vorgenommen worden. Die eine betrifft die Invite Transaktion die andere die non-Invite Transaktion. Zudem wurde die Möglichkeit eines Transport Errors ausgeschlossen, da dieser UAServer dies nicht erkennen kann. Diese Möglichkeit ist auch mehr auf TCP als auf UDP bezogen.

9.2.1.1. INVITE Server Transaktion

Als erstes soll das Wiederversenden der 200 (OK) Response nach einem INVITE-Request erläutert werden. Das Zustandsdiagramm verlangt nach [rfc3261], Kapitel 17.2.1, dass nach einem Versenden der 200 (OK) Response direkt in den Terminated Zustand gewechselt wird und sobald dies geschehen ist, wird die Transaktion beendet und zerstört.

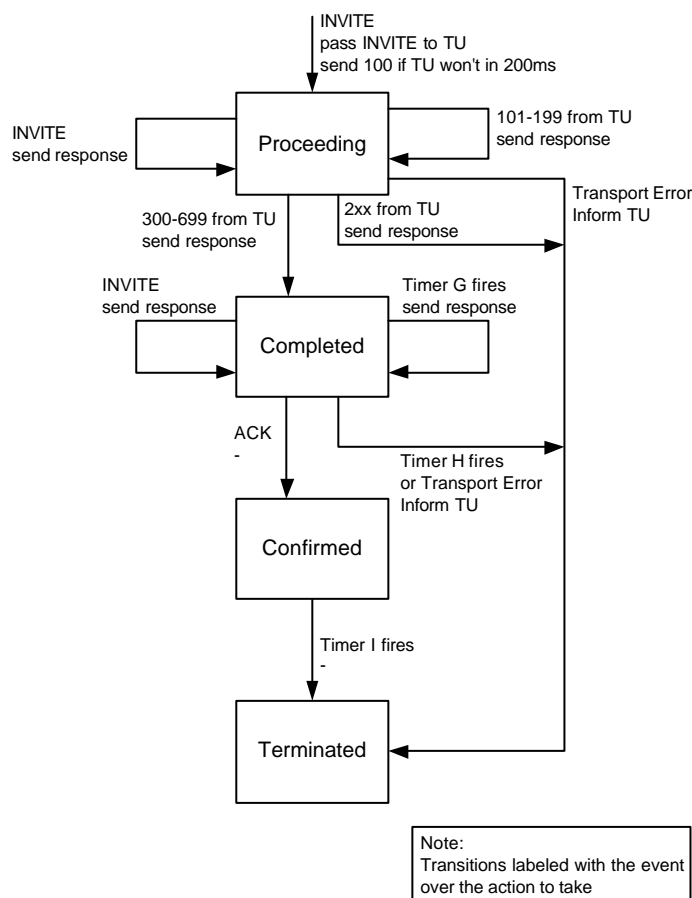


Abbildung 17: Invite Zustandsdiagramm des Servers laut [rfc3261]

Nun wird zusätzlich verlangt, dass das OK solange wieder versendet werden muss, bis entweder ein ACK erhalten wurde oder aber ein Timer abgelaufen ist¹. Dieses Abwarten ist nötig, um den Tripple Handshake zu durchlaufen. Denn schickt der Client kein ACK, so weiss der Server nicht, ob diese Verbindung korrekt etabliert wurde.

Das Abwarten wird erreicht, indem man vom Proceeding Zustand anstatt direkt in den Terminated in den neuen TerminateACK Zustand wechselt. In diesem neu hinzugefügten Zustand TerminateACK wird auf das ACK oder auf das Ablaufen des Timers NoACK gewartet. In der Zwischenzeit wird die 200 (OK) Response jedes Mal versendet, wenn der Timer ACK abgelaufen ist.

Wird ein ACK-Request empfangen, so wechselt der Server in den Terminated Zustand und die Transaktion wird erfolgreich abgeschlossen und beendet.

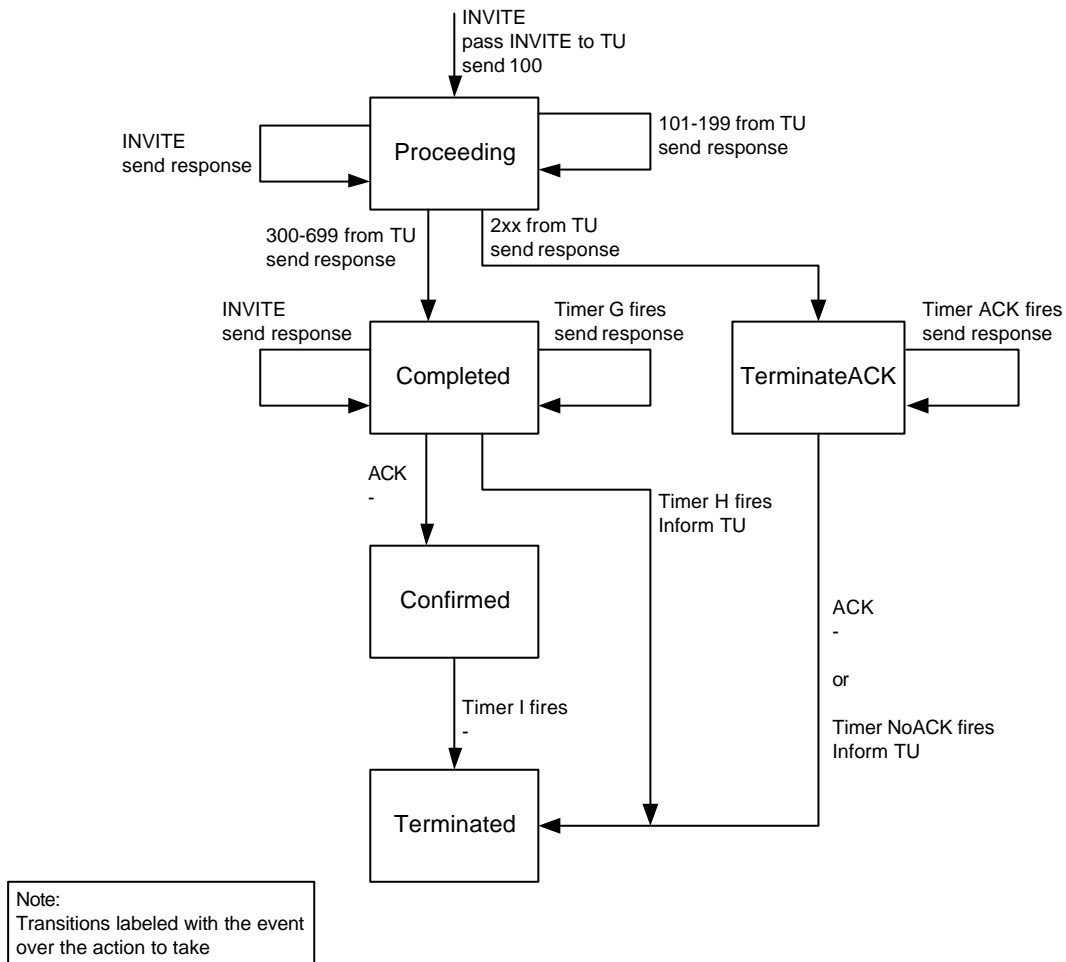


Abbildung 18: Verändertes Invite Zustandsdiagramm des Servers

Läuft der Timer NoACK ab, so wird der Anruf beendet indem der Server dem User Agent mitteilt, dass er den Anruf beenden soll. Der User Agent wird daraufhin den Client bemühen ein BYE zu versenden.

Im Gegensatz zum non-Invite Zustandsdiagramm kann hier auf einen parallel laufenden Completed Zustand zum Terminated Zustand verzichtet werden, denn der Timer I ist wesentlich kürzer als der Timer J. Die Funktion des Timers I ist, die durch das Netzwerk verzögerten oder verdoppelten Nachrichten (Forking Proxy) herauszufiltern.

¹ [rfc3261], Kapitel 13.3.1.4.

9.2.1.2. Non-Invite Zustandsdiagramm

Die zweite Modifizierung betrifft den non-Invite Completed Zustand. Hier wird aus denselben Gründen wie beim Client eine Veränderung vorgenommen. Denn per [rfc3261] wird erst nach Ablauf des Timers J in den Terminated Zustand gewechselt. Dies dauert jedoch mehr als eine halbe Minute. Während dieser Zeit ist das System blockiert, da nur ein einziger Zustand aktiv sein kann. Somit kann kein neuer Anruf gestartet oder entgegengenommen werden. Dass dieses lange Warten für einen Benutzer nicht willkommen ist, versteht sich von selbst.

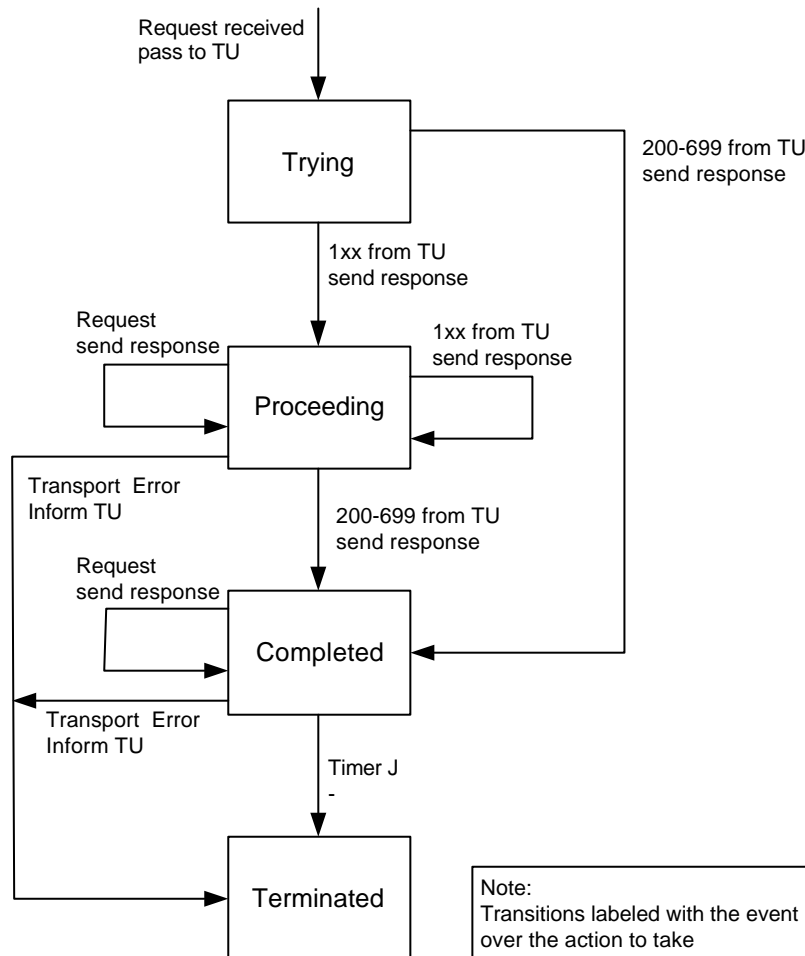


Abbildung 19: Non-Invite Zustandsdiagramm des Servers laut [rfc3261]

Betrachtet man nun den Zustand Completed der non-Invite Transaktion genau, erkennt man, dass dieser Zustand nur dazu da ist, um auf den Request nochmals die Response zu versenden. Einen solchen Request wird vom Client nur wiederholt gesendet, wenn dieser keine Final-Response erhalten hat, die Response also nicht, oder falsch vom Netzwerk übertragen worden ist. Da dies nicht die Regel sein soll, kann davon ausgegangen werden, dass die Final-Response, die beim Zustandswechsel (Trying ? Completed) verschickt worden ist, vom Client empfangen wurde und bei diesem die Transaktion beendet hat. Somit kann eine neue Transaktion gestartet werden.

Das Problem kann folgendermassen umgangen werden: Der Server verwaltet neben einem aktuellen auch einen zusätzlichen Zustand, welcher beim Wechseln in den Completed Zustand erstellt wird. Gleichzeitig wird vom aktuellen Zustand auch in den Terminated Zustand gewechselt. Dort wird die CSeq Nummer inkrementiert und die Transaktion

geschlossen. Trifft nun eine alte Nachricht beim Server ein, wird diese dem Completed Zustand übergeben. Dort wird geprüft, ob es sich um einen Request auf eine Final-Response bezieht. Ist dies der Fall, wird die Response erneut versendet und der Timer J neu gestartet. Läuft der Timer ab, wird der Completed Zustand gelöscht. Kommt der UAServer erneut in den Completed Zustand, weil z.B. eine neue Verbindung, kurz nach dem die alte Verbindung beendet worden ist, schon wieder beendet werden möchte, wird der alte Completed Zustand gelöscht und sofort ein neuer Completed Zustand geöffnet. Da die Zustandsmaschine nicht erneut in den Completed Zustand kommen kann, ohne dass die Gegenstation auf eine Response reagiert hätte oder die Verbindung nicht sowieso unterbrochen wurde, müssen keine Probleme erwartet werden. Somit ist klar, dass der Client die Final-Response erhalten hat, ansonsten würde der Client eine alte Response nicht verstehen. Wird die non-Invite Transaktion durch ein BYE oder ein CANCEL ausgelöst, so wird ausserdem noch der Anruf beendet.

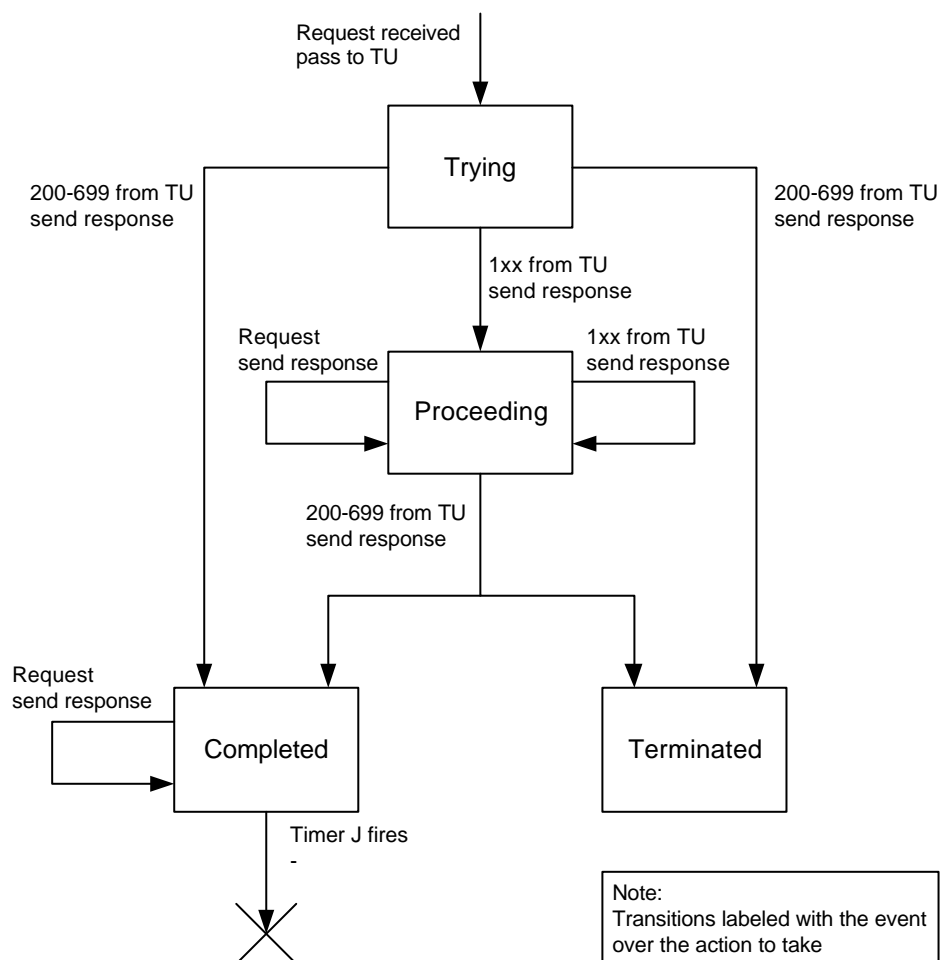


Abbildung 20: Verändertes non-Invite Zustandsdiagramm des Servers

9.2.2. Auswertung der Requests

Wird der erhaltene Request als gültig erkannt, muss sich der UAServer je nach Request und der darin enthaltenen Methode passend verhalten.

Je nach Methode wird unterschiedlich vorgegangen: Die zwei Methoden OPTIONS und REGISTER werden nicht unterstützt. Dies wird dem Client in der Response mitgeteilt, falls er eine solche Request Message geschickt hat. Diese Massnahme ist nötig, damit das Ziel, einen

einfachen SIP User Agent zu implementieren, nicht verfehlt wird. Denn mit diesen Requests, können etliche zusätzliche Informationen und Aktionen ausgetauscht bzw. ausgelöst werden. Die Requests mit den restlichen vier Methoden werden den Zuständen entsprechend verarbeitet.

Ist ein Request mit einer falschen CSeq-Nummer und Call-ID empfangen worden, wird dies mit einem „Server: Keine legale Message.“ erklärt. Stimmt die Call-ID und die CSeq-Nummer mit einer vorangegangenen Nachricht überein, wird sie dem speziellen Completed Zustand zur weiteren Verarbeitung übergeben.

Die Aufgabe jeder Methode wird im Folgenden näher beschrieben.

ACK: Das ACK dient dem Server als Bestätigung, dass der Client bei einem Verbindungsaufbau die Final-Response erhalten und verstanden hat. Erhält der UAServer keine Bestätigung, so wird die Verbindung nach einer gewissen Zeitdauer als nicht korrekt bewertet und beendet. Dem Benutzer wird entweder die Meldung „(ACK) Die Verbindung wird beendet.“ angezeigt, wenn das ACK auf eine 200 (OK) Response fehlt oder „(H) Die Verbindung ist nicht zustande gekommen.“, wenn das ACK auf eine 3xx-6xx Final-Response beim Server nicht angekommen ist.

Ausser bei einer Invite Transaktion hat das ACK keine Wirkung.

BYE: Die Response auf ein BYE-Request ist ein sofortiges 200 (OK). Dieser Überlegung liegt zu Grunde, dass wenn eine Seite das Gespräch beendet, der andere Benutzer wohl kaum weiter ins Leere sprechen will. Zusätzlich wird dem Benutzer das Auflegen erspart und ihm wird „Der Anruf wurde beendet.“ angezeigt. Der Grund, wieso keine Interaktion vom Benutzer abgewartet wird, ist in der Analogie zum herkömmlichen Telefon zu finden. Legt der Gesprächspartner auf, so kann der andere nur noch auflegen, ausser er wartet auf bessere Zeiten. Denn bevor er einen neuen Anruf tätigen oder empfangen kann, muss der Hörer aufgelegt werden.

CANCEL: Erhält der UAServer ein CANCEL in einer Invite Transaktion und er hat noch keine Final-Response zurückgesendet, so wird das CANCEL als Abbruch der Invite Transaktion verstanden und die Verbindung wird abgebrochen, indem aus dem aktuellen Zustand in den non-Invite Trying Zustand gewechselt wird. Ein „Verbindung wird beendet.“ wird dem Benutzer mitgeteilt. Ist der Verbindungsabbruch erfolgt, erscheint noch die Information „Verbindung wurde beendet.“

Ansonsten hat das CANCEL bewusst keinen weiteren Einfluss, da es empfohlen wird¹, nur auf einen INVITE-Request das CANCEL zu beachten.

INVITE: Sobald ein gültiges INVITE empfangen wurde, wird sofort der Proceeding Zustand der Invite-Transaktion geöffnet und eine 100 (Trying) Response versendet. Diese Response ermöglicht dem Client zu erkennen, dass der Anruf bearbeitet wird und dass er nicht noch ein weiteres INVITE versenden muss und damit das Netzwerk zusätzlich belastet. Wird der Mediatype unterstützt, beginnt es beim Angerufenen zu klingeln und auf dem Log-Window angezeigt, von welcher IP Adresse der Anruf initiiert wurde. Dem Benutzer steht es nun frei, den Anruf anzunehmen oder abzulehnen. Die entsprechende Response –

¹ [rfc3261], Kapitel 9.1

entweder 200 (OK) oder 486 (Busy Here) – wird erstellt und der aktuelle Zustand wird, auch entsprechend dieser Response, gewechselt.

Ein INVITE wird einfachheitshalber nur dann verarbeitet, wenn es eine Verbindung anfordert und nicht währenddessen ein laufender Anruf empfangen wird.

OPTIONS: Diese Methode wird nicht unterstützt und es wird dem Client eine 405 (Method Not Allowed) Response gesendet. Damit der Client weiss, welche Methoden der Server versteht, fügt der UAServer der Response noch einen weiteren Header an, den Allow-Header mit den vier unterstützten Methoden als Headerwert.

REGISTER: Auch diese Methode wird nicht unterstützt. Das Vorgehen ist dasselbe wie bei OPTIONS.

Diese Methode würde gebraucht um einen Registrar aufzubauen, der dann die Namen mit der zugehörigen IP Adresse abspeichert. Falls ein Anruf eingeht, der einen der abgelegten Namen oder IP Adressen als Ziel hat, wird dem Anrufer die korrekte Zieladresse mitgeteilt und er kann selber direkt einen neuen Anruf initiieren mit der neu erhaltenen Destination.

10. SIP-Message

Die zu der hier beschriebenen SIP-Message gehörenden Klassen sind die SIPMessage selbst, die StartLine, RequestLine und StatusLine, die MessageHeader und SDPHeader und der MessageBody.

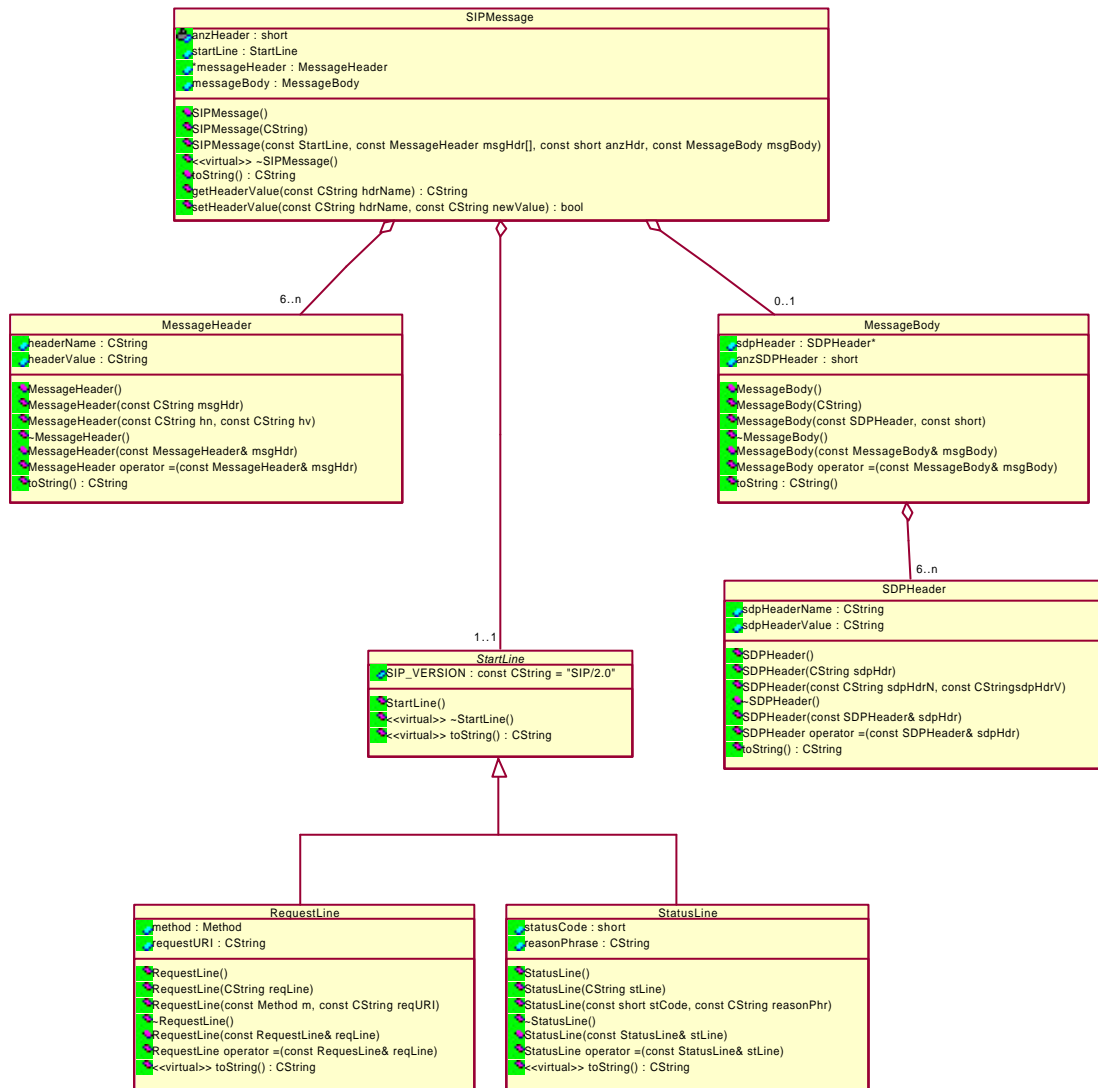


Abbildung 21: Klassendiagramm der SIPMessage

Der im Klassendiagramm dargestellte Aufbau soll noch ein bisschen näher erläutert werden. Die StartLine ist entweder eine RequestLine oder eine StatusLine, je nach dem, ob der Client oder der Server eine Nachricht versendet. Der MessageHeader beinhaltet alle Informationen, die benötigt werden, um die Message z.B. an die gewünschte Destination zu senden, wo sie auch richtig interpretiert werden soll. Die SDPHeader sind im

`MessageBody` enthalten und schließen weiterführende Informationen ein, die über die Endgeräte und die Verbindung zwischen den beiden Endpunkten mehr Auskunft geben. Bis auf `SDPHeader` ist die Namensgebung aus der [rfc3261], Kapitel 7 übernommen worden.

10.1. Anwendung

Der Aufgabenbereich der `SIPMessage` ist das korrekte Initialisieren und Aufbauen einer SIP-Message, die einzelnen Teile der Nachricht exakt herauszufiltern und Dienste bereitzustellen, um die Nachricht je nach Bedarf anzupassen, zu ergänzen oder abzuändern. Die nachfolgend aufgelisteten Funktionen sind in allen Klassen, bis auf die der `StartLine`, enthalten und beabsichtigen den gleichen Zweck.

Kopierkonstruktor: Der Kopierkonstruktor ist nötig, um die jeweiligen Objekte der Klasse korrekt zu initialisieren. Nötig deshalb, weil zum Teil eine dynamische Speicherverwaltung benützt wird.

Zuweisungsoperator: Das Überladen dieses Operators ist sinnvoll, denn es sind nicht nur statische Attribute, die zugewiesen werden. Vielmehr sind die Attribute dieser Klassen dynamisch erstellt worden und es können Komplikationen auftreten, wenn das Vorgehen nicht richtig umgesetzt wird.

Diese Operatorfunktion wurde erst später hinzugefügt, nachdem ein nicht richtiges Funktionieren der Zuweisungen erkannt wurde.

toString: Es können nur in Strings abgepackte Zeichenfolgen versendet werden. Und da die SIP-Message eigentlich aus vielen Einzelteilen besteht, `MessageHeader` mit `headerName` und `headerValue`, `MessageBody` mit `sdpHeader` und dessen `sdpHeaderName` und `sdpHeaderValue` etc. müssen zuerst diese Teilstücke (Membervariablen) klassenintern der [rfc3261] entsprechend zusammengesetzt werden.

10.2. Implementierung

Die Konstruktoren jeder einzelnen Klasse gewährleisten, dass die Informationen, die die Konstruktoren erhalten, den entsprechenden Membervariablen zugewiesen werden und für weitere Verarbeitungen zur Verfügung gestellt werden.

Um die [rfc3261] einzuhalten, aber nicht zu viele Abfragen machen zu müssen, weil die [rfc3261] ev. mehrere ähnliche Notationen zulässt, müssen ein paar Einschränkungen befolgt werden: Es müssen immer CRLFs am Ende einer Zeile vorhanden sein, auch bei der letzten Zeile einer Nachricht. Andernfalls wird die SIP-Message nicht korrekt gelesen. Der eigentlichen SIP-Message vorangestellte Leerzeichen oder CRLFs werden in diesem Fall nicht ignoriert, sondern die ganze Nachricht wird als eine Ungültige verworfen.

Auch werden bei den `HeaderNames` und `-Values` eventuelle Leerzeichen mitgenommen. Einzig beim `MessageHeader` werden vor und nach dem ':' die Leerzeichen gelöscht. Das bedeutet, dass die Headers nicht zu 100 % übernommen werden. Die SIP-Message bleibt trotzdem korrekt.

10.2.1. SIPMessage

Wird eine Nachricht empfangen, so wird der Konstruktor `SIPMessage(CString)` aufgerufen. Gerade dieser Konstruktor der `SIPMessage` muss sorgfältig aufgebaut sein, denn wenn dieser die Nachricht falsch interpretiert und auseinanderflechtet, wird wohl kaum eine Verbindung zu Stande kommen.

Der Konstruktor sucht die einzelnen Bestandteile der SIP-Message heraus. Diese sind die `RequestLine` bzw. `StatusLine`, den `MessageHeader` und den `MessageBody`. Sind die einzelnen Stücke gefunden, werden sie den Konstruktoren der jeweiligen Klasse zur weiteren Analyse übergeben.

Beim Versenden einer Nachricht, sucht die Funktion `toString` die einzelnen Teile wieder zusammen und übergibt dem `SIPUA` einen einzelnen String.

Wird der Standardkonstruktor aufgerufen, werden folgende Membervariablen gesetzt:

Die Anzahl der `MessageHeader` wird auf Null gesetzt, der `MessageBody` wird initialisiert und der Pointer des `MessageHeader` und der `StartLine` wird NULL gesetzt.

10.2.2. StartLine

Die `StartLine` Klasse ist rein virtuell. Die einzige in den abgeleiteten Klassen gleiche Membervariable ist die SIP-Version `SIP_VERSION`, weshalb sie bei der `StartLine` verwaltet wird.

10.2.3. RequestLine

Der Konstruktor dieser Klasse sucht nach einer Methode, die in der [rfc3261] bekannt ist. Wird keine Methode gefunden, wird `UNKNOWN` als Methode gesetzt. Dies ist dann sinnvoll, wenn eine Methode nicht erkannt wird. Somit hat sie trotzdem einen definierten Wert und kann allenfalls abgefangen und ausgewertet werden.

Wird der Standardkonstruktor aufgerufen, werden folgende Membervariablen gesetzt:

Die Methode wird als `UNKNOWN`, die `requestURI` als leerer String initialisiert.

10.2.4. StatusLine

Bei der `StatusLine` werden zwei Teile benötigt. Zum einen ist dies der `statusCode`, zum andern die `reasonPhrase`, die den Status Code in Worten beschreibt.

Wie der `statusCode` interpretiert wird, wird dem `UAClient` und `UAServer` überlassen.

Wird der Standardkonstruktor aufgerufen, werden folgende Membervariablen gesetzt:

Der `statusCode` wird gleich Null gesetzt und die `reasonPhrase` mit einem leeren String initialisiert.

10.2.5. MessageHeader

Der `MessageHeader` beinhaltet alle Headers, die der Client bei einem Request erstellt oder der Server von einem Request übernimmt.

Die Namen und Werte der einzelnen Headers, die der Konstruktor bei einer eingehenden Nachricht extrahieren muss, werden vor und nach dem ':' um die Leerzeichen gekürzt.

Wird der Standardkonstruktor aufgerufen, werden folgende Membervariablen gesetzt:

Beide Variablen, der Name und der Wert, werden mit einem leeren String initialisiert.

10.2.6. **MessageBody**

Der `MessageBody` schliesst die kompletten Headers des SDPs ein.

Wird der Standardkonstruktor aufgerufen, werden folgende Membervariablen gesetzt:

Die Anzahl der `SDPHeader` wird gleich Null gesetzt und der Pointer auf die `SDPHeader` wird mit `NULL` initialisiert.

10.2.7. **SDPHeader**

Bei dieser Klasse wird ein einkommender String in die `SDPHeader` Namen und Werte zuweisen und abgelegt. Diese Headers werden grösstenteils nicht mehr genauer überprüft. Zwei Ausnahmen gibt es jedoch: Die `Media Description (m)` und `Connection Information (c)` werden benötigt um einerseits den unterstützten Coder und den Audioport aus (m) und andererseits die Ziel IP Adresse aus (c) zu extrahieren.

Wird der Standardkonstruktor aufgerufen, werden folgende Membervariablen gesetzt:

Beide Variablen, der Name und der Wert, werden mit einem leeren String initialisiert.

11. Bedienungsanleitung

Das Programm SIP User Agent ist ein einfach zu bedienendes Softtelefon, das die Sprache über das Internet transportiert. Im Vergleich zu einem normalen Telefon ändert sich für den Benutzer wenig. Man muss eine eindeutige Nummer von der Gegenstation kennen und eingeben, den Hörer „abnehmen“ („OK“ Button) um zu telefonieren und wieder „aufzulegen“ („Abbrechen“ Button), um den Anruf zu beenden.

In der folgenden Anleitung wird die Funktionsweise des Telefons näher erklärt. Für die ungeduldigen Benutzer empfiehlt sich der Einstieg über das Kapitel 11.1. Diejenigen, die mehr über den Ablauf des Programms wissen möchten, sind mit dem Kapitel 11.2 gut bedient.

11.1. Schnellstart

Bevor ein Anruf getätigt werden kann, benötigt der Anrufer nur die IP Adresse der Gegenstation und gibt sie im Feld „Ziel IP“ ein. Falls ein anderer Port gewünscht wird, muss der standardmässig gesetzte Wert (5060) überschrieben werden.

Nun muss die Verbindung noch gestartet werden. Dies geschieht, indem man auf den „OK“ Button klickt.

Um das Telefonat zu beenden, wird einzig der „Abbrechen“ Button gedrückt.

Möchte ein einkommender Anruf nicht entgegengenommen werden, kann man durch Anklicken des „Abbrechen“ Buttons dem Anrufer mitteilen, dass man nicht gestört werden will.



Abbildung 22: Benutzeroberfläche beim Programmstart

11.2. Programmübersicht

Die grafische Benutzeroberfläche ist nur mit den nötigsten Elementen versehen. Die Anzahl der Aktionen, die ein Benutzer vornehmen kann, ist deshalb beschränkt, weil die Möglichkeiten des Programms auf ein Minimum reduziert bleiben sollen.

Mit den drei Schaltflächen können dem User Agent Aktionen mitgeteilt werden, die dieser dann entsprechend verarbeitet und falls nötig die entsprechenden SIP-Messages vom UA Client oder UA Server generieren lässt.

Im „Ziel IP, Port“ Feld muss eine gültige IP Adresse der gewünschten Gegenstation eingegeben werden. Die Gültigkeit der IP Adresse wird nur beschränkt geprüft, z.B. auf Eingaben grösser als 255. Wird keine Port Nummer eingegeben, bzw. wird der standardmässig gesetzte Port nicht überschrieben, wird der SIP-Port 5060 übernommen.

Im „Host IP, Port“ Feld werden die am eigenen Rechner gültige IP Adresse und der Port, auf welchem man die Verbindung aufbaut, angezeigt. Diese beiden Felder sind vom Benutzer nicht veränderbar.

Das Log-Window dient dazu, Informationen über die Verbindung oder deren Status anzuzeigen. Das kann zum Beispiel sein, wenn ein Anruf angekündigt, eine Verbindung abgebrochen oder beendet wird.

Die zwei Schaltflächen „OK“ und „Abbrechen“ können vom Benutzer nur zu bestimmten Zeitpunkten ausgewählt werden, währenddessen „Beenden“ zu jedem Zeitpunkt angeklickt werden kann. Programmintern wird deshalb kontrolliert, in welchem Moment dieser Button gedrückt wird. Ist nämlich ein Anruf im Gange, soll das Programm nicht einfach geschlossen, sondern die Verbindung standardgemäss beendet werden. Mit der Einschränkung der „OK“ und „Abbrechen“ Buttons wird erreicht, dass die Zustände der einzelnen User Agents nicht durcheinander geraten und die Signalisierung stets kontrolliert ablaufen kann.

Dem Benutzer wird auditiv zusätzlich signalisiert, ob ein Anruf eingeht, es bei der Gegenstation klingelt oder der Angerufene die Anrufeinladung ablehnt. Die Annäherung der Klänge an das herkömmliche Telefonsystem ist gewollt. Die zwei wav Dateien (ringin.wav und ringout.wav) werden benötigt und müssen daher im gleichen Verzeichnis wie das Programm SIP User Agent vorhanden sein.

Die Meldungen im Log-Window werden erst gelöscht, wenn ein neuer Anruf gestartet wird oder eingeht. Das heisst, dass nach dem Verbindungsabbau die Meldungen noch nachgelesen werden können.

11.2.1. Verbindung starten

Um einen Anruf zu starten, wird eine gültige Ziel IP Adresse eingegeben und falls nötig der Ziel-Port überschrieben. Danach muss nur noch der „OK“ Button gedrückt werden.

Sobald „OK“ gedrückt wurde, wird der Button inaktiv. Dafür wird der „Abbrechen“ Button aktiv gesetzt, um den Anruf abzulehnen oder später zu beenden.

Dem Benutzer wird mitgeteilt, dass nun eine Verbindung aufgebaut wird. Ist diese zu Stande gekommen, wird dies ebenfalls mit einer Meldung im Log-Window angezeigt.

Das „Trying...“ teilt dem Benutzer mit, dass die Gegenstation gefunden wurde und der Anruf dort in Bearbeitung ist.

Wird eine Anrufeinladung empfangen währenddem man eine Verbindung aufbauen möchte, muss zuerst dieser eingehende Anruf abgelehnt werden, bevor man die eigene Verbindung starten kann. Würde man den „OK“ Button drücken, wird der Anruf angenommen.

11.2.2. Verbindung entgegennehmen oder ablehnen

Eine Verbindung kann dann angenommen oder abgelehnt werden, wenn ein Anruf eingeht. Dies wird mit der Meldung „x.x.x.x ruft an.“ visuell angekündigt. Akustisch ertönt ein Klingeln. Zudem ist nicht nur der „OK“ sondern auch der „Abbrechen“ Button aktiv.



Abbildung 23: Anzeige bei einem eingehenden Anruf

Jetzt kann durch Drücken des „OK“ Buttons der Anruf entgegengenommen werden. Sobald dieser Button gedrückt wurde, wird er inaktiv, die Sprachverbindung besteht und das Klingeln verstummt. Auf dem Log-Window erscheint „Der Anruf wurde angenommen.“.

Es ist nun aber auch möglich, den „Abbrechen“ Button auszuwählen. Wird diese Wahl getroffen, wird der anrufenden Partei mitgeteilt, dass der Anruf nicht entgegengenommen wurde und wird beendet. Es bleiben nun beide Tasten solange inaktiv bis entweder die Gegenstation den Verbindungsabbau bestätigt hat oder ein Timer (H) abgelaufen ist. Meist wird aber die Gegenstation auf das Ablehnen hin schnell eine Antwort schicken und somit sind die Tasten wieder frei, um einen neuen Anruf zu starten oder anzunehmen.

Der Benutzer, der den Anruf initiiert hat, erhält die Meldung „Der Anruf wurde abgelehnt.“. Daraufhin wird der Anruf ordnungsgemäss abgemeldet. Ist die Verbindung korrekt abgebaut worden, wird ihm dies mit „Der Anruf wurde beendet.“ mitgeteilt und der Benutzer kann einen neuen Anruf tätigen oder empfangen. Die Tasten sind wieder frei, um einen neuen Anruf zu starten oder anzunehmen.

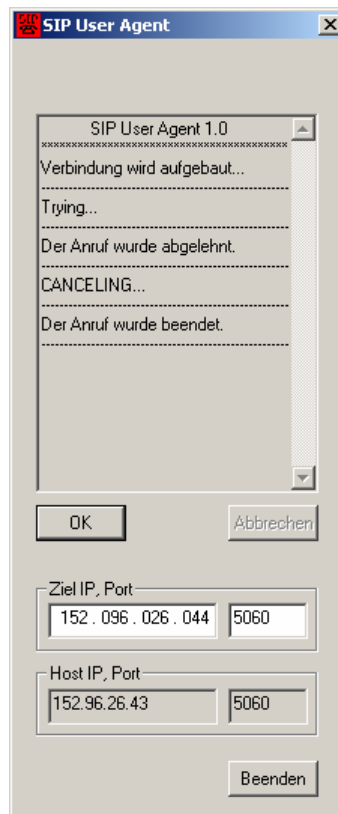


Abbildung 24: Anzeige nach einem abgelehnten Anruf

Wird der Anruf abgelehnt, wechselt das normale Klingeln beim Anrufenden zu einem Besetztzeichen und verstummt, sobald der Anruf beendet wurde. Dies wird durch das Ablehnen automatisch übernommen.

11.2.3. Verbindung beenden

Das Beenden eines Anrufs kann von beiden Seiten erfolgen. Wer zuerst auflegt, indem er auf „Abbrechen“ drückt, erhält zusätzlich zur Meldung „Der Anruf wurde beendet.“ vorgängig die Nachricht „Der Anruf wird beendet...“.

Bei diesem Screenshot hat der Anrufer den Anruf erfolgreich beendet. Nun kann wieder eine neue Verbindung aufgebaut oder ein Anruf entgegengenommen werden.

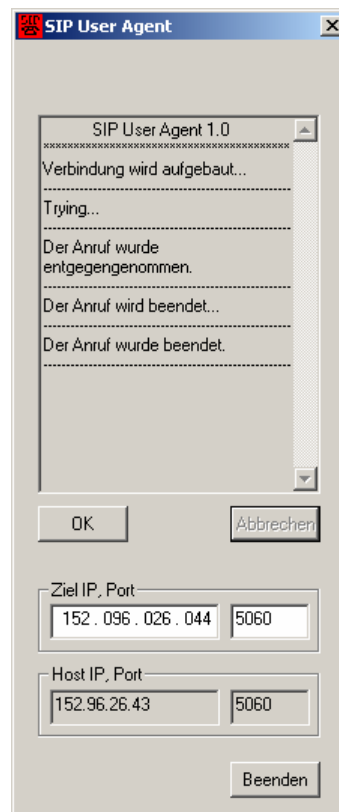


Abbildung 25: Anzeige nach einem beendeten Gespräch

11.3. Meldungen

Die folgenden alphabetisch aufgelisteten Meldungen können im Log-Window erscheinen. Die angeführte Erklärung soll erläutern, warum es zu der Meldung kam.

x.x.x.x ruft an.

Zeigt beim Angerufenen an, dass ein Anruf eingeht und von der gültigen IP Adresse x.x.x.x initiiert worden ist.

CANCELING...

Die Verbindung wird abgebrochen. Diese Meldung erscheint beim Initiant eines Verbindungsaufbaus, wenn:

- er die Verbindung abbrechen möchte, bevor die Gegenstation den Anruf definitiv angenommen oder abgelehnt hat. Das heisst, dass die Gegenstation schon mit einer Provisional- nicht aber mit einer Final-Response geantwortet hat. Das Telefon klingelt auf beiden Seiten entsprechend.

- die Gegenstation den Anruf nicht entgegennehmen möchte und deshalb ablehnt.

Der Anruf wird beendet...

Kündigt einen Verbindungsabbau an und zwar demjenigen, der den Anruf durch Drücken des „Abbrechen“ Buttons beenden möchte.

Der Anruf wurde abgelehnt.

Erscheint beim Anrufer, wenn auf der anderen Seite der Anruf abgelehnt wurde. Bestätigt dem Angerufenen, dass er den Anruf nicht angenommen, sondern abgelehnt hat.

Der Anruf wurde beendet.

Auf beiden Seiten wird dadurch mitgeteilt, dass der Anruf erfolgreich beendet wurde oder wenn:

- der Timer ACK abgelaufen ist. (Siehe Timer ACK)
- der Timer B abgelaufen ist. (Siehe Timer B)
- der Timer H abgelaufen ist. (Siehe Timer H)
- kein INVITE sondern eine andere Methode nach Programmstart empfangen wird, wird diese Meldung im Log-Window angezeigt. Der Grund dafür ist, dass der Request nicht bearbeitet werden kann und die Verbindung somit abgebrochen und beendet wird. Der Server antwortet dem Client mit einer 481 (Call Leg/Transaction Does Not Exist) Response.

Der Anruf wurde entgegengenommen.

Auf beiden Seiten wird damit angezeigt, dass der Angerufene der Anrufeinladung nachgekommen und somit der Anruf zustande gekommen ist.

Server: Keine legale Message.

Hat der Server eine Nachricht erhalten, die in keiner Weise zu einer laufenden Verbindung gültig ist, d.h. sind die CSeq und Call-ID nicht korrekt bzw. identisch mit einer laufenden Verbindung, wird diese Nachricht angezeigt.

Server: Keine Request-Message.

Wird serverseitig eine Message empfangen, die zwar im Bezug auf CSeq und Call-ID gültig ist, aber keine der sechs Methoden¹ in der Request-Line hat, wird obige Meldung ausgegeben.

SIP Socket geöffnet.

Der SIP Socket konnte erfolgreich geöffnet werden.

SIP Socket konnte nicht geöffnet werden.

Konnte der SIP Socket nicht erfolgreich geöffnet werden, ist eine der häufigsten Ursachen, dass der Port (5060) schon durch ein anderes Programm belegt ist.

Trying...

Dass die Gegenstation erfolgreich eingeladen wurde und dass die Anrufeinladung in Bearbeitung ist, wird mit dieser Meldung dem Anrufer bestätigt.

¹ Siehe Kapitel 2.3.4

Verbindung wird aufgebaut...

Wird ein neuer Anruf initiiert, so wird dies dem Anrufer mitgeteilt. Diese Meldung sagt nichts über den Fortschritt des Verbindungsaufbaus aus.

(ACK) Die Verbindung wird beendet.

Der Timer ACK ist serverseitig abgelaufen. Der Server hat kein ACK vom Client bekommen, nachdem der Server ein OK auf eine Anrufeinladung versendet hat. Der Anruf wird anschliessend beendet.

(B) Es konnte keine Verbindung hergestellt werden.

Der Timer B ist clientseitig abgelaufen. Der Client hat auf sein INVITE keine Antwort vom Server erhalten. Der Anruf wird anschliessend beendet.

(C2) Der Anruf wurde nicht entgegen genommen.

Der Timer C2 ist clientseitig abgelaufen. Bei der Ziel-Station wurde der Anruf weder angenommen noch abgelehnt, d.h. es hat ins Leere geklingelt. Der Anruf wird anschliessend beendet.

(F) Verbindung wurde unterbrochen.

Der Timer F ist clientseitig abgelaufen. Der Client hat auf ein non-Invite Request höchstens eine Provisional-Response empfangen, aber sicher keine Final-Response. Der Anruf wird anschliessend beendet.

(H) Die Verbindung ist nicht zustande gekommen.

Der Timer H ist serverseitig abgelaufen. Der Server hat kein ACK bekommen, nachdem er den Anruf abgelehnt hat. Der Anruf wird anschliessend beendet.

Schlimmer Ausnahmefehler: SESSION TERMINATED.

Ein schlimmer Ausnahmefehler tritt auf, wenn der Softstate Timer abgelaufen ist. Dieser Timer garantiert, dass das Programm in keinem Zustand hängen bleibt, auch wenn sich die Gegenstation unfreundlich – sprich gar nicht – abgemeldet hat.

Dies kann serverseitig zB. im Zustand Proceeding der non-Invite Transaktion geschehen, wenn das Telefon klingelt, ohne dass ein Benutzer den Anruf ablehnt oder beantwortet und sich der Client nicht mit einem CANCEL verabschiedet hat.

12. Testen des Programms

Ziel der Tests war es, neben der Funktionalität auch die Korrektheit des Programmablaufs zu überprüfen. Das heisst, ob das implementierte Protokoll wirklich SIP konform arbeitet. Weiter wurde geprüft, inwiefern das Programm Fehlmanipulationen und Angriffen standhalten kann. Das Testprotokoll ist dem Anhang F zugefügt.

12.1. Testvorgehen

Das vorliegende Testprotokoll wurde zuerst in der Theorie erarbeitet und dann am Programm getestet. Das Vorgehen war folgendermassen: Nach dem Erstellen des Protokolls wurden die Tests der Reihe nach durchgeführt. Wenn ein Testergebnis nicht mit dem erwarteten Resultat übereinstimmte, wurde abgeklärt, ob die Überlegung beim Erstellen des Protokolls falsch war, der Standard nicht richtig implementiert wurde oder ein Programmierungsfehler vorlag. Bei einem Programmierfehler oder nicht korrekter Anwendung des Standards wurde das Programm überarbeitet und alle Tests nochmals der Reihe nach durchgeführt.

Die Tests wurden in drei Kategorien unterteilt. In der ersten Kategorie (Test-Nr. 1-5) wurde getestet ob das Programm beim korrekten Anwenden richtig funktioniert. Es muss einen Anruf korrekt gestartet, angenommen, abgelehnt und beendet werden können. In der zweiten Kategorie (Test-Nr. 6-16) wurden die Timer der Zustände getestet. Es wurde geprüft, ob beim Ablauf der Timer auch die richtige Reaktion erfolgt. In der dritten Kategorie (Test-Nr. 17-28) wurde getestet, inwiefern unsachgemässe Bedienung, Angriffsversuche oder Zufälligkeiten wie gleichzeitiges Anrufen oder ähnliches, zu Fehlern im Programm oder Protokoll führen.

Während in den ersten beiden Kategorien das Bestehen der Tests unerlässlich ist, geht es in der dritten Kategorie eher darum, das Fehlverhalten zu Dokumentieren, als einen korrekten Programmablauf zu gewährleisten.

12.2. Ergebnis

Bei den ersten Testversuchen wurden noch einige Fehler im Programm gefunden. So wurde zum Beispiel anfangs beim Abbruch eines Anrufes immer ein CANCEL versendet, auch wenn noch keine provisional Response vom Server empfangen wurde¹. Ein anderes Problem war, dass SDP Daten wie der Session Name oder der Session Identifier nicht vom INVITE übernommen wurde, sondern immer eine eigene Session erstellt wurde. Diese Fehler konnten allerdings behoben werden.

Somit wendet das vorliegende Programm SIP korrekt an. Es läuft auch dann noch stabil, wenn falsche SIP-Messages empfangen wurden. Die implementierten Timer, inklusive dem Softstate Timer, funktionieren. Dadurch kann das Programm beim unkorrekten Beenden einer Transaktion immer wieder in den Ausgangszustand gelangen. Es ist für die Sprachübertragung in leistungsstarken Netzwerken geeignet, stösst aber bei langsameren Netzwerken wie manchmal beim Internet an seine Grenzen was die Sprachübertragung betrifft. Jedoch wird auch hier die Verbindung korrekt aufgebaut und auch wieder beendet.

¹ [rfc3261], Kapitel 9.1

12.3. Bemerkungen zu den Tests

Das Testprotokoll ist im Anhang F zu finden. Die Nummern beziehen sich auf den Entsprechenden Test in diesem Protokoll.

Zum Test-Nr. 24 muss noch eine kurze Erläuterung angefügt werden. Der Fehler wird dadurch erklärt, dass der Client sich noch im Completed Zustand der Invite Transaktion befindet, gleichzeitig aber schon der Terminated Zustand die Transaktion abgeschlossen hat¹. Weil die Transaktion nun schon abgeschlossen ist, wird vom Client ein BYE anstatt ein CANCEL verschickt.

¹ Siehe Kapitel 8.2.1.1

13. Zusammenfassung

Der SIP User Agent ist ein Programm, welches SIP (Session Initiation Protocol) benutzt, um eine Sprachverbindung über ein internetfähiges Netzwerk aufzubauen. Ist eine Verbindung zustande gekommen, kann zwischen den Endgeräten normal telefoniert werden.

Es soll anhand einer einfachen Implementation in MS Visual C++ gezeigt werden, wie SIP aufgebaut ist und wie die zwei Stationen miteinander kommunizieren. In der Abbildung 26 ist ein Verbindungsaufbau mit den SIP-Messages INVITE, RINGING, OK und ACK dargestellt, sowie das Beenden des Anrufs mit BYE und OK.

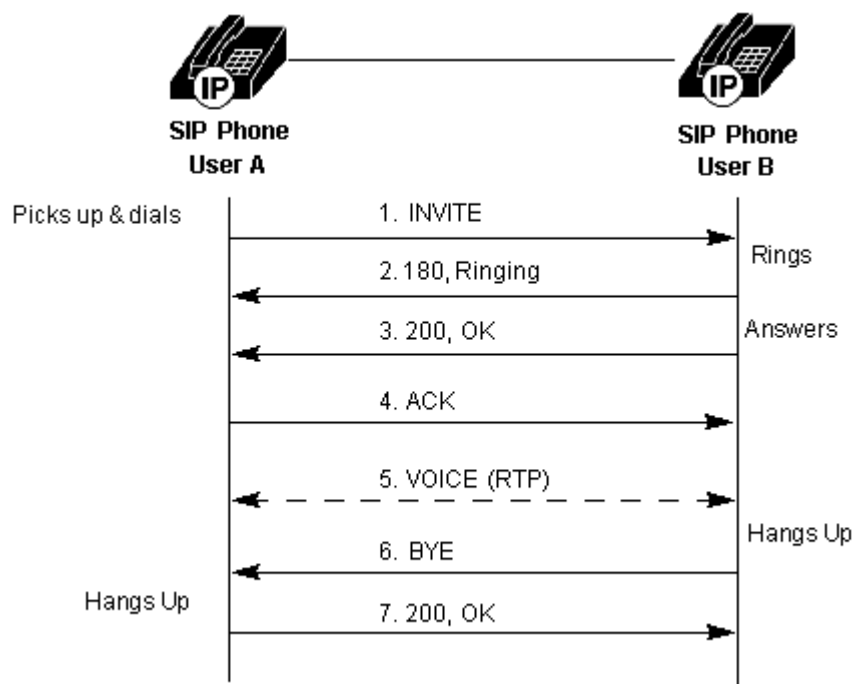


Abbildung 26: SIP Verbindungsauf- und abbau

SIP eignet sich sehr gut für den Anschauungsunterricht, denn die Nachrichten, die SIP zwischen den Telefonen über das Netzwerk austauscht sind in Klartext und leicht verständlich.

Von den sechs Methoden (ACK, BYE, CANCEL, INVITE, OPTIONS und REGISTER) die SIP definiert, werden in diesem Fall OPTIONS und REGISTER nicht unterstützt. Mit ein Grund ist, dass das Programm übersichtlich und möglichst einfach bleiben soll. Ein weiterer, dass für eine umfassendere Lösung zu wenig Zeit vorhanden war.

Für die Sprachcodierung wird G.711 μ Law benützt. Dieser Coder ist mitverantwortlich, dass die Sprachübertragung rudimentär bleibt und auch eine gewisse Bandbreite des Netzwerks voraussetzt. Somit wird der Spass des Telefonierens über das Internet je nach Netzwerk getrübt oder erhellt.

Die einfache und übersichtliche Grafikoberfläche lässt eine unkomplizierte Bedienung zu und mit wenigen Eingaben kann ein Anruf initiiert werden. Im Log-Window werden verschiedene Meldungen über den Zustand der Verbindung angezeigt.



Abbildung 27: Grafische Benutzeroberfläche

Das Programm SIP User Agent ist lauffähig und kann auch mit einem in Hardware implementierten SIP-Telefon kommunizieren.

Während der Arbeit mit der Audio Klasse wurde viel Zeit mit dem Problem der Callbackfunktion verloren. Dies hat es uns verunmöglicht, mehr als nur einen einfachen Verbindungsaufbau zu programmieren. Ansonsten wurden die von uns gesetzten Milestones erreicht. Allerdings mit einer Verschiebung um die Zeit, welche die Audio Klasse mehr beansprucht hatte.

Möglichkeiten für die Erweiterung des Programms sind die Implementierung der Methoden REGISTER und OPTIONS. Die Audio Klasse könnte verbessert werden, indem eine Voice Activity Detection (VAD), ein dynamisches Buffermanagement und weitere Coder hinzugefügt werden.

Die grafische Benutzeroberfläche könnte noch um eine Menuleiste erweitert werden, mit deren Hilfe man gewisse Variablen, wie z.B. die UDP-Paketgrösse und der Audioport, dynamisch verwalten könnte. Da zum Teil lange Wartezeiten vorkommen können, wäre eine Anzeige, die dem Benutzer mitteilt, ob noch ein Timer aktiv ist, hilfreich.

Mit geringem Aufwand, ist es auch möglich, das bestehende Programm in einen Registrar umzuschreiben.

Anhang A: Abkürzungsverzeichnis

CRLF:	Carriage Return Line Feed
GUI:	Graphical User Interface
IETF:	The Internet Engineering Task Force
IP:	Internet Protocol
MMUSIC:	Multiparty Multimedia Session Control
NAT:	Net Address Translation
OOA:	Objektorientierte Analyse
OOD:	Objektorientiertes Design
RFC:	Request For Comment
RTP:	Real time Transport Protocol
SDP:	Session Description Protocol
SIP:	Session Initiation Protocol
SP:	Single Space
TCP:	Transmission Control Protocol
UDP:	User Datagram Protocol
URI:	Universal Resource Identifier
URL:	Universal Resource Locator
VoIP:	Voice over IP

Anhang B: Referenzen

Referenz- und Nachschlagewerke:

- [rfc3261] Rosenberg, J. et. al, „SIP: Session Initiation Protocol“, RFC 3261, Juni 2002.
- [rfc2327] Handley, M. und V. Jacobson, „SDP: Session Description Protocol“, RFC 2327, April 1998
- [msdn] Microsoft Corporation, „MSDN Library Visual Studio 6.0 release“
- [uml-lehrbuch] Balzert, H. „Lehrbuch der Objektmodellierung, Analyse und Entwurf“, Spektrum Akademischer Verlag, Berlin, 1999
- [iptelephony] Hersent, O. et al, „IP Telephony, Packet-based multimedia communications systems“, Addison Wesley, 2002
- [cnwui] Comer, D. E. „Computernetzwerke und Internets“, Pearson Studium, München, 2000
- [c++einf] Breyman, U. „C++ Eine Einführung“, Hanser, München, 1997
- [c++prog] Strasser, T. „C++ Programmieren mit Stil“, dpunkt, Heidelberg, 1997
- [c++inside] Kruglinski, D. et al, „Inside Visual C++ 6.0“, Microsoft Press, Unterschleissheim, 1998

Anhang C: Abbildungsverzeichnis

Abbildung 1: SIP Verbindungsauf- und abbau.....	9
Abbildung 2: Aufbau einer SIP-Message	9
Abbildung 3: Grober Aufbau des SIP User Agent	14
Abbildung 4: Analyse des Clients	15
Abbildung 5: Analyse des Servers.....	16
Abbildung 6: Klassendiagramm der Hauptklassen.....	17
Abbildung 7: Sequenzdiagramm des SIPUA (Invite Transaktion)	19
Abbildung 8: Klassendiagramm des UAClient	20
Abbildung 9: Sequenzdiagramm des UAClient (BYE Transaktion).....	21
Abbildung 10: Klassendiagramm des UAServer.....	22
Abbildung 11: Sequenzdiagramm des UAServer (Invite Transaktion).....	23
Abbildung 12: Klasse AudioController	24
Abbildung 13: Invite Zustandsdiagramm des Clients laut [rfc3261]	30
Abbildung 14: Verändertes Invite Zustandsdiagramm des Clients	31
Abbildung 15: Non-Invite Zustandsdiagramm des Clients laut [rfc3261]	32
Abbildung 16: Verändertes non-Invite Zustandsdiagramm des Clients	33
Abbildung 17: Invite Zustandsdiagramm des Servers laut [rfc3261].....	36
Abbildung 18: Verändertes Invite Zustandsdiagramm des Servers	37
Abbildung 19: Non-Invite Zustandsdiagramm des Servers laut [rfc3261]	38
Abbildung 20: Verändertes non-Invite Zustandsdiagramm des Servers	39
Abbildung 21: Klassendiagramm der SIPMessage	42
Abbildung 22: Benutzeroberfläche beim Programmstart	46
Abbildung 23: Anzeige bei einem eingehenden Anruf	48
Abbildung 24: Anzeige nach einem abgelehnten Anruf.....	49
Abbildung 25: Anzeige nach einem beendeten Gespräch.....	50
Abbildung 26: SIP Verbindungsauf- und abbau.....	55
Abbildung 27: Grafische Benutzeroberfläche	56

Anhang D: Verzeichnisstruktur

Stammverzeichnis

DA 2002 - SIP User Agent.pdf	Dokumentation im pdf-Format
Anhang F - Testprotokoll.pdf	Testprotokoll im pdf-Format
Readme.txt	Verzeichnisstruktur im txt-Format

Unterverzeichnisse

\Bericht	DA 2002 - SIP User Agent.doc Zeitplan.xls	Zentraldokument im doc-Format Zeitplan des Projektes im xls-Format
\Bericht\Filialdokumente	*.doc	Filialdokumente im doc-Format
\Bilder	*.*	Diverses Bildmaterial
\Code	*.*	MS Visual C++ 6.0 Projekt
\OOA	SIP UserAgent OOA.mdl	Objektorientierte Analyse im mdl-Format
\OOD	SIP UserAgent OOD.mdl	Objektorientiertes Design im mdl-Format
\Programm	SIP User Agent.exe *.wav	Ausführbares Programm Klingeltöne des SIP User Agents
\Protokolle	*.doc	Sitzungsprotokolle im doc-Format
\RFC	SDP - rfc2327.pdf SIP - rfc3261.pdf	RFC des SDP im pdf-Format RFC des SIP im pdf-Format
\SIP Messenger	Messenger.exe SIP_Messenger.doc	SIP-Message Generator Beschreib des Messengers

Anhang E: Status Code

Status Code Reason Phrase

Informational

100	Trying
101	Switching Protocols
180	Ringing
181	Call Is Being Forwarded
182	Queued

Success

200	OK
201	Created
202	Accepted
203	Non-Authoritative Information
204	No Content
205	Reset Content
206	Partial Content

Redirection

300	Multiple Choices
301	Moved Permanently
302	Found
303	See Other
304	Not Modified
305	Use Proxy
307	Temporary Redirect

Client Error

400	Bad Request
401	Unauthorized
402	Payment Required
403	Forbidden
404	Not Found
405	Method Not Allowed
406	Not Acceptable
407	Proxy Authentication Required
408	Request Time-out
409	Conflict
410	Gone
411	Length Required
412	Precondition Failed
413	Request Entity Too Large
414	Request-URI Too Large

415	Unsupported Media Type
416	Requested range not satisfiable
417	Expectation Failed

Server Error

500	Internal Server Error
501	Not Implemented
502	Bad Gateway
503	Service Unavailable
504	Gateway Time-out
505	HTTP Version not supported

Global Failure

600	Busy Everywhere
603	Decline
604	Does Not Exist Anywhere
606	Not Acceptable

Anhang F: Testprotokoll

Nachfolgend ist das in Kapitel 12 erwähnte Testprotokoll angefügt.