SIP/H.323 Gateway

Løsning basert på abstraksjon

Rune Schumann

Hovedfagsoppgave
1. november 1999

UNIVERSITETET I OSLO
Institutt for informatikk



Til min kjære hustru Marianne

Forord

Husk at din egen beslutning om å lykkes er viktigere enn noe annet.

— Abraham Lincoln

Dette er en hovedoppgave til cand.scient.-graden innen fagfeltet kommunikasjonssystemer ved Institutt for Informatikk. Denne rapporten er en del av det arbeidet som ble utført med og lage en gatewayløsning mellom H.323 og SIP. Oppgaven begynte høsten 1998 og ble avsluttet i november 1999.

Jeg ønsker å takke alle som har hjulpet meg under arbeidet med hovedoppgaven. Den jeg vil spesielt takke er veilederen min Lill Kristiansen. Hun har bidratt med et vell av ideer for hva som kan inngå i oppgaven, og bidratt med konstruktiv kritikk underveis i arbeidet. I tillegg har hun gjort det mulig for meg å få tilgang til Ericssons lokaler og utstyr.

Uten Ericssons vilje til et samarbeid ville ikke denne oppgaven vært mulig. Derfor vil jeg takke de på Ericsson som jeg har forstyrret i sin travle hverdag, og spesielt takke Frode Sivertsen for sin store velvilje og tålmodighet ovenfor plagsomme hovedfagsstudenter.

Utover dette vil jeg takke mine medstudenter ved IFI. Abid Ali Teepo, Per Thomas Jahr, Knut-Olav Traa, Eirik Maus, Eirik Torske og Torstein Strøm er jeg stor takk skyldig for nyttige innspill, korrekturlesning og tilførsel av nye krefter når motivasjonen har vært på bunn. Ellers vil jeg takke Steinar Sørlie for korrekturlesning. Siden han er del av det lille IP-telefoni miljøet på IFI, har hans tilbakemeldinger på rapporten betydd mye for meg.

Jo Herstad vil jeg også rette en spesiell takk til for å komme med svært nyttige tips og innspill på slutten av rapportskrivingen.

Andreas Eide er en av mine gode venner som jeg vil takke for meget nyttig korrekturlesning. Hans bakgrunn fra Tandberg med billedtelefoni kombinert med svært gode norskkunnskaper har ført til mange nyttige innspill både faglig som språkmessig.

Sist, men ikke minst, vil jeg takke min kjære kone Marianne. Hun har utvist en tålmodighet utenfor fatteevne ovenfor sin studerende mann i de seks og et halvt årene etterutdanningen har tatt.

Oslo, november 1999

Rune Schumann

Sammendrag

SIP og H.323 er to standarder for IP-telefoni fra henholdsvis IETF og ITU. Disse to standardene er ikke kompatible med hverandre, men har motsvarende funksjonalitet.

Dette er utgangspunktet for denne oppgavens hovedmål; å fremstille en gatewayløsning mellom disse to protokollene på et konseptuelt plan. Denne løsningen baserer seg på å abstrahere funksjonaliteten i SIP og H.323 til et felles språk. Dette letter konverteringen fra den ene protokollen til den andre.

For å få fram en slik løsning tas det utgangspunkt i et sett med brukstilfeller (Use Cases) som modelleres i UML. Realiserbarheten i modellen blir så testet i en implementasjonsfase.

Oppgaven presenterer to modeller av gatewayen; en som kun konverterer signalering, og en utvidet modell som i tillegg støtter mediatranslasjon. I den utvidete modellen har jeg basert meg på en løsning tuftet på distribuerte mediagatewayer.

Resultatene jeg har kommet frem til blir diskutert i slutten oppgaven, men viser i korthet at løsningen er realiserbar på det konseptuelle planet.

I tillegg til selve gatewayløsningen presentert i denne oppgaven, beskrives noen forskjellige aspekter rundt VoIP og telekommunikasjon.

Innhold

1 Innledning **1**

1.1 Problemstilling 3

1.1.1 Presisering og avgrensning av problemstillingen 4

1.2 Metoder 5

1.3 Viktige resultater 5

1.4 Rapportens oppbygning 6

1.5 Valg av skriftlig stil i rapporten 7

2 Voice Over IP **9**

2.1 En bit telekommunikasjonshistorikk 10

2.2 Voice over IP kontra Voice over Internett 11

2.3 Motivasjon for VoIP 12

2.4 Plassering av intelligensen 14

2.5 *Legacy* – Arven som ingen vil ha 16

2.6 "No such thing as a free lunch" – Fakturering 16

2.7 Oppsummering og konklusjon 18

3 H.323 **21**

3.1 Bakgrunn og motivasjon 21

3.2 Oppbygning og virkemåte 22

3.2.1 Funksjoner for kommunikasjon- og kallkontroll 24

3.2.2 Mediatyper 25

3.3 Bestanddeler og enheter 25

3.3.1 Terminal 26

3.3.2 Gateway 27

3.3.3 Gatekeeper 28

3.3.4 Multipoint Controller 29

3.3.5 Multipoint Processor 30

3.3.6 Multipoint Control Unit 30
3.4 Eksempler på oppsett av H.323 samtaler 30
3.4.1 Basic Call anrop uten GK 31
3.4.2 Basic Call anrop rutet gjennom GK 31
3.5 Oppsummering og konklusjon 32

4 Session Initation Protocol 35
4.1 Bakgrunn og motivasjon 35
4.2 Oppbygning og virkemåte 36
4.3 Bestanddeler og enheter 37
4.3.1 User Agent Client 38
4.3.2 User Agent Server 38
4.3.3 Registrar 39
4.3.4 Redirect Server 39
4.3.5 Proxy Server 39
4.4 SIP eksempler 39
4.4.1 Anrop med Proxy Server 39
4.4.2 Anrop med omdirigering av mottaker 40
4.5 Oppsummering og evaluering 41

5 Session Description Protocol 43
5.1 Bakgrunn og motivasjon 43
5.2 Beskrivelse av SDP 44
5.2.1 Konklusjon SDP 45

6 SIP/H.323 gateway 47
6.1 Access Toolkit 48
6.1.1 Motivasjon 48
6.1.2 Oppbygning 49
6.1.3 Design og virkemåte 50
6.1.4 Eksempel: H.323 anrop i ATK 54
6.2 Modell av SIP/H.323 gateway 56
6.2.1 Motivasjon 56
6.2.2 Forutsetninger 56
6.2.3 Modellens brukstilfeller (Use Cases) 58
6.2.4 Klassediagram 60
6.2.5 Initiering 62
6.2.6 Innkommende H.323 anrop 64

6.2.7 Innkommende SIP anrop 66
6.2.8 Avslutte samtale fra H.323 68
6.2.9 Avslutte samtale fra SIP 70
6.2.10 Forandre mediaformater underveis i en samtale 72
6.3 Oppsummering og konklusjon 72

7 Implementasjon av gatewaymodellen **77**
7.1 Forutsetninger, mål og strategi 77
7.2 Implementasjonsfasen 78
7.2.1 Implementasjon av grunnleggende komponenter 78
7.2.2 Implementasjon av SIP inn i ATK 80
7.3 Problemer underveis 81
7.4 Gjenstående arbeid 82
7.5 Oppsummering og konklusjon 82

8 Media Gateway Control Protocol **85**
8.1 Bakgrunn og motivasjon 85
8.2 Forbindelsesmodell (Connection Model) 86
8.3 Omtalte anvendelsesområder 87
8.3.1 Trunking Gateway 88
8.3.2 Access Gateway 88
8.3.3 Residential Gateway 88
8.3.4 Network Access Server 88
8.3.5 Business Gateway 88
8.4 Eksempel MGCP 89
8.5 Utvidete anvendelsesområder 89
8.5.1 Formidling av ønskede mediatyper i MGCP 90
8.5.2 Multiparty samtaler med ulike mediatyper i MGCP . . 90
8.5.3 Forhandlingsteknikker for mediatyper 91
8.5.4 Betydning av effektivitet i MGCP 92
8.5.5 Konklusjon bruksområde 92
8.6 Lokalisering av brukere i MGCP 93

9 Media Device Control Protocol **95**
9.1 Oppbyggning og arkitektur 95
9.1.1 Kontrollmodell MDCP 96
9.2 Omtalte anvendelsesområder 98
9.2.1 Trunking Gateway 98
9.2.2 Access Gateway 99
9.2.3 Residential Gateway 100
9.2.4 Network Access Server 100
9.2.5 Multipoint Processors 100
9.2.6 Interaktive Voice Respons System 101
9.2.7 Transcoding Devices 101
9.3 MDCP anvendt som SIP/H.323 Gateway 101

10 Media Gateway Control **103**
10.1 Bakgrunn og motivasjon 104
10.2 Forbindelsesmodell (Connection Model) 104
10.2.1 Terminations 104
10.2.2 Context 105
10.2.3 Eksempler på Context/Termination 105
10.3 MEGACOs transaksjonsmekanisme 109
10.4 Anvendelsesområder 109
10.5 Ting under utarbeidelse 110
10.6 MEGACO anvendt som SIP/H.323 gateway 111
10.7 Konklusjon MEGACO 111

11 Utvidelse av gatewaymodellen **113**
11.1 Presentasjon av konseptet 113
11.2 Utvidelse av modellen 114
11.3 Et konkret brukstilfelle 116
11.4 Evaluering av løsningen 118

12 Oppsummering og diskusjon **119**
12.1 Oppsummering av oppgaven og rapporten 119
12.2 Evaluering og diskusjon av løsningen 121
12.3 Videre arbeid 123

Bibliografi **125**

Vedlegg **129**

A Forkortelser **131**

B Kildekode **133**

B.1 Basiskomponenter 133

B.1.1 UDPClient.java 133

B.1.2 UDPServer.java 135

B.1.3 UDPClientTest.java 137

B.1.4 UDPServerTest.java 138

B.1.5 SIPMessage.java 139

B.1.6 SIPMethodType.java 140

B.1.7 SIPParser.java 141

B.1.8 Parsertest.java 145

B.2 SIP implementasjon i ATK 147

B.2.1 SIPTerminalDescriptor.java 147

B.2.2 SIPTerminal.java 150

B.2.3 SIPMediaDescriptor.java 155

B.2.4 SIPCall.java 158

B.2.5 SIPOrigCall.java 164

B.2.6 SIPTermCall.java 165

Figurer

2.1 Internetts infrastruktur vs globalt telekommunikasjonsnett . 12
2.2 Eksempel *Toll Bypass* 13
2.3 Tradisjonell fakturering i telesystemet 17

3.1 H.323 Zone 23
3.2 Sammenkobling H.323 komponenter 25
3.3 Komponenter i en H.323 terminal 26
3.4 Basic Call anrop uten GK 31
3.5 Basic Call anrop rutet gjennom GK 33

4.1 INVITE eksempel 37
4.2 Meldingsflyt SIP klient og server i Proxy Mode 38
4.3 Anrop med Proxy Server 40
4.4 Anrop med omdirigering av mottaker 41

6.1 ATK lagdeling 49
6.2 ATK klassediagram 50
6.3 Aktuelle Use Case SIP/H.323 gateway 59
6.4 Klassediagram SIP/H.323 gateway 61
6.5 Initiering SIP/H.323 gateway 63
6.6 Sekvensdiagram innkommende H.323 anrop 65
6.7 Sekvensdiagram innkommende SIP anrop 67
6.8 Sekvensdiagram for avsluttet samtale fra H.323 terminal . . 69
6.9 Sekvensdiagram for avsluttet samtale fra SIP terminal 71
6.10 Felles løsningsrom SIP og H.323 73

8.1 MGCP Connection Model 87
8.2 Eksempel MGCP VoIP til PSTN 89

9.1 MDCP objekt arkitektur 97

9.2 MDCP Trunk Gateway 99
9.3 MDCP Access Gateway 99
9.4 MDCP Residential Gateway 100
9.5 MDCP Network Access Server 101
9.6 MDCP CODEC translasjon 102

10.1 MEGACO eksempel Connection Model 106
10.2 MEGACO Context relatert til fysiske endepunkter 107
10.3 T3 i Null Context, T1 og T2 aktive 108
10.4 T2 i Null Context, T1 og T3 aktive 108
10.5 Modell for MEGACOs transaksjoner 109
10.6 Eksempel MEGACO transaksjon 110

11.1 Klassediagram utvidet gatewaymodell 115
11.2 Use case Find Mediagateway 116
11.3 Sekvensdiagram for å finne en mediagateway 117
11.4 Signalering og mediastrømmer i SIP/H.323 GW 117

Kapittel 1

Innledning

> «Det var flink Tigergutt, du kommer akkurat tidsnok til å ta styrkemedisinen din.» Og hun helte opp en skje for ham.
> «Jeg *har* alt tatt min,» ropte Ru stolt.
> Og Tigergutt svelget sin og sa:
> «Det har jeg også.»
> Så skumpet og dyttet de til hverandre for moro skyld, og Tigergutt veltet uheldigvis en stol eller to, og Ru veltet uheldigvis en med vilje.
>
> — *A. A. Milne: Huset på Bjørnehjørnet*

HOVEDTEMA i denne oppgaven er å se på hvordan en kan lage en gatewayløsning mellom to protokoller for IP-telefoni; *SIP* og *H.323*. Denne gatewayen blir et sted hvor protokollene kan eksistere sammen og kommunisere med hverandre. Protokollene som sådan har oppstått som et resultat av den rivende utviklingen i Internettverdenen det siste 10-året. I dette ligger både en teknologisk utvikling og den enorme utbredelsen Internett har fått.

Norsk Gallups Intertrackundersøkelse[1] fra september 1999 gir et godt bilde på utbredelsen av Internett i Norge. Denne undersøkelsen viser at antall som har tilgang til Internett har økt fra 822 000 i 1996 til 1,9 millioner i september 1999. Hobbes' Internet Timeline[2] har tilsvarende tall for antall vertsmaskiner på Internett i samme periode. Her er utviklingen gått fra 9,5 millioner maskiner i januar 1996 til 56,2 millioner maskiner i juli 1999. Hvis en antar at det er sted mellom 5 og 10 brukere i gjennomsnitt per maskin vil det i dag være et sted mellom 280 og 562 millioner brukere tilknyttet Internett. Den eksplosive veksten har ført til at Internett har utviklet seg enormt. Fra å være en kommunikasjonsløsning for universiteter og forskere preget av lave båndbredder og karakterbaserte grensesnitt, til å bli en sterk defacto standard med grafiske brukergrensesnitt og applikasjoner som kan overføre data, lyd, bilder og video. En forutsetning for denne veksten har vært tilgang til økt båndbredde og et enkelt grensesnitt til Internett gjennom innføringen av World Wide Web.

[1] Se: http://www.gallup.no/menu/internett/resultater/intertrack/intertracarkiv.htm
[2] Se: http://info.isoc.org/guest/zakon/Internet/History/HIT.html

Den økte båndbredden og utbredelsen har ført til nye typer Internettapplikasjoner i tillegg til de tradisjonelle som e-post og nyhetsgrupper. En av disse applikasjonsområdene er *multimediatelefoni*. Dette er et vidt begrep som omfatter alt fra en enkel telefonsamtale mellom to parter, til en flerparts multimediakonferanse med overføring av lyd, stillbilder, applikasjonsdeling og video.

Populariteten til Internett har i tillegg medført at internettprotokollene anvendes i større og større grad som kommunikasjonsbærere. Dette er protokoller som IP, TCP og UDP. Disse protokollene er i dag svært utbredt, ikke bare i WAN kommunikasjon, men er også ofte førstevalget i LAN kommunikasjon. Dette resulterer i at de samme protokollene brukes om du skal ha en multimediakonferanse med noen på andre siden av jorda, eller om samtalepartneren befinner seg i kontoret ved siden av. Denne typen multimediakonferanse havner innenfor gruppen *multimediatelefoni over IP*.

Multimediatelefoni over IP har grovt sett to hovedaktører; Internettverdenen med IETF (Internet Engineering Task Force) [Sec94] og telekommunikasjonsverdenen med ITU (International Telecommunication Union) [ITU98a]. Disse aktørene har tilnærmet seg multimediatelefoni med forskjellig utgangspunkt. IETFs utgangspunkt er å etablere multimediakommunikasjon mellom eksisterende datamaskiner klargjort for multimedia. ITUs utgangspunkt har vært å tilby allmen kommunikasjon på tvers av avstander. Dette fordrer rimelig og enkelt brukerutstyr. Kompleksiteten har da blitt liggende i store og dyre sentraler.

Et annet moment for hvilken retning standardene tar i henholdsvis IETF og ITU, er hvilke motiv som ligger bak utviklingen. En av drivkreftene bak IETFs arbeid er teknologiforsking fra universiteter. Dette medfører at det ikke er en allment fremtredende tankegang at noen skal tjene penger på det arbeidet som gjøres. Dette medfører at det drives fram standarder som ofte er meget gode funksjonelt, mens andre aspekter ikke nødvendigvis er et tema. Et eksempel på dette hvordan en kan tjene penger på å tilby god båndbredde og tjenestekvalitet. ITU derimot drives av en stor andel kommersielle krefter. Dette fører til en tankegang om at en skal betale for de ressursene man bruker samt den tjenestekvaliteten man forlanger og blir tilbudt. En stor annen drivkraft bak ITUs standarder er *interoperabilitet*. Alle skal som et minimum kunne ha en vanlig samtale med hverandre uavhengig av hva slags utstyr som er hver i ende eller hvem som har laget utstyret.

Disse til dels motstridene synene påvirker standardene og hva som skal stå i sentrum for arbeidet med utviklingen.

Selv om standardiseringsarbeidet i leirene nevnt over har forskjellige drivkrefter, lever de ikke i hver sin verden upåvirket av hverandre. Dette vises i arbeidet med multimediatelefoni over IP. Her innser IETF at hvis ingen kan ta betalt for tjenester blir det heller ikke bygd ut nødvendig båndbredde. På den andre siden ser ITU at IETF har enkle og elegante løsninger på noen problemstillinger, og valgt å implementere noen av disse i sine standarder.

Det største skillet mellom ITUs og IETFs løsninger for multimediatelefoni er i dag protokollene som sørger for lokalisering, oppkobling, signalering og

utveksling av terminalegenskaper.

For denne funksjonaliteten har IETF utviklet *SIP* [HSSR99a]. Tilsvarende har ITU utviklet sin standard *H.323* [ITU99].

Status idag er at begge sett med standarder eksisterer side om side. Markedet har enda ikke gitt sin endelige dom over hvilken standard som er best egnet. Det må understrekes at den standarden markedet velger ikke behøver å være den teknisk beste løsningen, men kan like godt være den som får størst utbredelse.

Så lenge markedet ikke har bestemt seg vil det eksistere løsninger basert på de to forskjellige standardene side om side. Dette utløser et behov for at utstyr fra ulike miljøer kan snakke med hverandre på tvers av standarder. For å muliggjøre dette må det eksistere en *gateway* mellom SIP og H.323. Denne gatewayen vil fungere som en tolk mellom protokollene, og dermed muliggjøre kommunikasjon mellom endepunkter med forskjellige VoIP protokoller.

Denne oppgaven handler om å lage en slik gateway, og tar utgangspunkt i at begge protokollene er laget for IP-telefoni. Funksjonelt sett gjør de stort sett det samme. Dette er årsaken til at jeg vil se på hvordan denne funksjonaliteten kan generaliseres til et felles språk for protokollene, og benytte dette til å lage en gateway mellom SIP og H.323.

1.1 Problemstilling

De innledende studier av SIP og H.323 viser at disse protokollene grovt sett har meget lik funksjonalitet. Utfra dette dukket ideen opp om å lage et grensesnitt hvor protokollenes funksjonalitet var abstrahert. Selve implementasjonen av protokollen vil være skjult og funksjonaliteten presenteres til omverdenen via et felles grensesnitt.

Jeg har utarbeidet følgende problemstilling som en ledetråd i dette arbeidet:

> *Er det mulig å realisere en gateway mellom SIP og H.323 basert på abstraksjon av funksjonaliteten i de respektive protokollene?*

Hvorvidt det er mulig å *realisere* en slik løsning er et sentralt spørsmål i denne oppgaven. I dette legger jeg både om det er mulig å komme frem til en modell av systemet, og om det er sannsynlig å implementere modellen.

En *gateway* vil fungere som en tolk mellom to forskjellige språk i form av protokoller, og gjøre det mulig å kommunisere på tvers av disse språkbarrierene.

Abstraksjon innebærer å sammenfatte enkeltfenomener under et begrep ved å se bort fra avvikende detaljer. Det er dette jeg vil benytte meg av for å skape en felles uttrykksform mellom SIP og H.323.

Med *funksjonaliteten* i protokollene SIP og H.323 mener jeg hva de i realiteten kan gjøre. Jeg ser at SIP og H.323 har mange likheter, men uttrykker

det på forskjellig måte. Jeg vil derfor se om det lar seg gjøre å få en felles uttrykksform ved å abstrahere denne funksjonaliteten, og bruke dette i utviklingen av en SIP/H.323 gateway.

1.1.1 Presisering og avgrensning av problemstillingen

Tanken om å skape et felles grensesnitt ved abstraksjon utløser andre spørsmål som må drøftes. Dette er temaer som:

- Hvordan oversette signaleringen mellom H.323 og SIP?
- Hvordan skal eventuelle mediatranslasjoner foretas?
- Forsvinner noe av funksjonaliteten i de respektive protokollene når de abstraheres?

Problemstillingen og temaene som er skissert over søkes løst gjennom å lage en modell. Denne modellen berører de to hjørnesteinene i telefoni; signalering og media. SIP og H.323 er protokoller for signalering, og derfor er det oversettingen av signaleringen modellen primært berører. Jeg har ønsket å bruke *Access Toolkit* (ATK) [Siv98] fra Ericsson som en grunnpilar for modellen. Dette APIet spesifiserer et rammeverk for å lage endepunkter i gitte telefoniprotokoller, og tilbyr et felles grensesnitt mot den spesifikke protokollimplementasjonen. H.323 er allerede implementert fra Ericssons side, og en del oppgaven er å implementere SIP inn i ATK.

Når det gjelder media overføres dette ved hjelp av egne protokoller. Eksempler på dette er G.711 for audio og H.263 for video. I den grad partene i en SIP/H.323 sesjon ikke har felles mediaformat må det en mediatranslasjon til. Dette er en oppgave som krever stor prosessorkraft, og ivaretas best av hardwareenheter for å konvertere i sanntid. Modellen som jeg beskriver i denne rapporten tar ikke for seg selve mediatranslasjonen, men jeg vil se på problematikken ved å benytte seg av frittstående mediagatewayer kontrollert av en egen protokoll.

Vi ønsker primært å se på om min idé lar seg gjennomføre på det konseptuelle planet. Derfor vil oppgaven begrenses til å omfatte et sett valgte brukstilfeller som er essensielle. Disse er:

- Innkommende anrop fra en SIP terminal.
- Innkommende anrop fra en H.323 terminal.
- Terminering av anrop fra en SIP terminal.
- Terminering av anrop fra en H.323 terminal.

At jeg vil se på det konseptuelle i dette innebærer at jeg velger å operere med en ideell verden hvor det aldri forekommer feil. Dette fører til at feilsituasjoner, som er et stort område innen telekommunikasjon, i liten grad blir berørt i denne oppgaven. Videre konsentrer jeg meg om *basic call*. Altså standard anrop fra A til B. Dette medfører at jeg ikke involverer noen form

for tjenster i modellen. Eksempler på dette er klassiske tjenester som viderekobling og samtale venter, eller mer avanserte IN-aktige tjenester.

Dette påvirker også den implementasjonen jeg ønsker å utføre. Jeg har for eksempel ikke som mål å implementere en SIP parser med all nødvendig funksjonalitet. For å vise det konseptuelle er det tilstrekkelig i dette tilfellet å kunne håndtere en forhåndsbestemt SIP melding, og relatere denne til tilsvarende H.323 melding (eller omvendt).

1.2 Metoder

Metodene som er anvendes i denne oppgaven har vært litteraturstudier, modellering i UML og implementasjon av en del av modellen.

Diagrammene som illustrerer modellen bruker UML-notasjon. Det forutsettes at leseren er kjent med UML-notasjonen som er brukt i disse diagrammene.

Litteraturstudiene omfatter standardene for de protokollene som oppgaven berører. I tillegg til dette artikler om IP-telefoni, og hvordan dette har konsekvenser for det eksisterende telekommunikasjonsnettet. IP-telefoni er et nytt område med svært mange aspekter, og derfor finnes det lite etablert litteratur innen fagfeltet. Informasjonssøk på Internett har dermed blitt en sentral kilde for litteratur. I tillegg til dette har også i tidsskrifter som IEEE Computing og Dr. Dobbs vært kilder for relevant informasjon. Bruk av denne typen kilder gjør at det har vært viktig å være kritisk til kildene, og å få informasjon fra flere steder.

All modellering er utført i UML med Rational Rose versjon 98i som modelleringsverktøy. Vi har ikke brukt alle aspekter i UML i modelleringen, men brukt det jeg synes har vært nyttig. Kort fortalt omfatter dette Use Case-, sekvens- og klassediagrammer.

Implementasjonen er utført i Java JDK 1.2 med Borland JBuilder 3.0 som utviklingsmiljø. Den primære årsaken til disse valgene er at Java er et språk jeg følte meg fortrolige med. I tillegg er Borland JBuilder 3.0 en del av utviklingsmiljøet hos Ericsson AS hvor implementasjonen ble utført. I implementasjonsfasen brukte jeg en metode hvor jeg først utviklet de basale komponentene, deretter utviklet jeg min del av rammeverket i ATK. Dette førte til en metode som gikk *nedenfra og opp* i første fase. Deretter i neste fase implementerte jeg *ovenfra og ned*. Således kan en si at den metoden jeg valgte var styrt av fasen utviklingen var i.

Implementasjonen hadde som hovedmål å avdekke hvor vidt modellen var realiserbar, og avdekke eventuelle svakheter.

1.3 Viktige resultater

I denne oppgaven har jeg funnet at en gatewayløsning mellom SIP og H.323 basert på å abstrahere protokollfunksjonaliteten med all sannsynlighet kan

realiseres. Både den modellen jeg har laget på grunnlag av brukstilfeller, og det jeg implementerte tyder på dette.

Jeg diskuterer i hvilket anvendelsesområde vår SIP/H.323 gateway er mest egnet, og kommer fram til at løsningen hører hjemme på domenenivå[3]. For mer sentrale løsninger preget av et høyt antall samtidige anrop vil jeg argumentere for at en ikke benytter seg av abstraksjon. I disse sammenhengene vil antakelig en todelt løsning være mer egnet. Med det menes at det oversettes direkte mellom protokollene uten mellomledd.

Videre skisserer vi en idé om hvordan en kan løse CODEC translasjon ved å benytte seg av en protokoll for distribuerte mediagatewayer. Selv om denne teknologien i dag ikke er implementert av leverandører av mediagatewayer, er konseptet så interessant at det burde undersøkes nærmere.

Mot slutten av oppgaven diskuterer jeg hvilke valg som er gjort underveis, og evaluerer det arbeidet og resultatet jeg har kommet fram til.

1.4 Rapportens oppbygning

Denne rapporten er satt sammen på en slik måte at leseren skal få nødvendig bakgrunnskunnskap for å forstå de resultatene som legges frem. Derfor er det først en del med bakgrunnsstoff, så en modell av gatewayen og tilhørende implementasjon. Etter dette følger en ny del med bakgrunnsstoff for å presentere det som ligger til grunn for delen som beskriver hvordan modellen kan utvides. Til slutt i rapporten er et kapittel som summerer opp og diskuterer arbeidet som er gjort.

Innholdet i kapitlene i rapporten er:

Voice over IP Dette kapittelet beskriver begrepet og gir en kort historikk. I tillegg diskuteres forskjellige aspekter rundt introduksjonen av VoIP som kjerneteknologi. Kapittelet trekker frem disse motivene som taler for at pakkebaserte nett kan bli den nye kjerneteknologien innen telekommunikasjon:

- Rimelige telefonsamtaler.
- Lette utviklingen av tjenester relatert til telefonnettet.
- Vedlikehold av et pakkebasert nett koster vesentlig mindre enn vedlikehold av det tradisjonelle linjesvitsjede nettet.
- Nye muligheter for tjenesteinovasjon og integrasjon med andre typer datatjenster.

H.323 I hovedsak beskrives her bakgrunn og motivasjon for protokollen og hvilke enheter standarden omfatter. Kapittelet evaluerer protokollen til en viss grad.

[3] I denne sammenheng mener vi domener med størrelsesorden en stor bedrift (for eksempel Statoil).

Session Initiation Protocol Motparten til protokollen over er tema i dette kapittelet. Presentasjonen av SIP har samme oppbygging som kapittelet om H.323; motivasjon og oppbygging, bestanddeler og en konklusjon.

Session Description Protocol Dette er egentlig en underliggende standard til SIP, men siden SDP brukes i MGCP og til dels MEGACO, har jeg valgt å skille den ut i et eget kapittel. SDP blir beskrevet uten å gå alt for mye i dybden.

SIP/H.323 gateway I dette kapittelet presenteres den modellen jeg har kommet frem til som et resultat av vårt arbeide. I tillegg til modellen beskrives *Access Toolkit* (ATK) som er en sentral enhet i modellen.

Implementasjon av gatewaymodellen Resultatet av det implementasjonsarbeidet jeg utførte blir beskrevet i dette kapittelet. Dette inkluderer selve implementasjonsarbeidet, hva jeg kom frem til som et resultat av dette, og videre arbeid.

Media Gateway Control Protocol Dette kapittelet er det første av tre som danner grunnlaget for utvidelsen av gatewaymodellen for å takle mediakonvertering. MGCP kommer fra IETF og er beregnet på distribuerte mediagatewayer.

Media Device Control Protocol Konseptuelt sett beskriver dette kapittelet en lik protokoll som MGCP, men MDCP har sin opprinnelse i ITU.

Media Gateway Control Det er denne protokollen som jeg baserer meg på i utvidelsen av modellen. Denne protokollen representerer en sammenslåing av MGCP og MDCP. IETF og ITU vil videreutvikle denne standarden videre sammen.

Utvidelse av gatewaymodellen Dette kapittelet presenterer en modell for hvordan vår gatewayløsning kan utvides til å takle mediatranslasjoner basert på å bruke en protokoll for distribuerte mediagatewayer. Kapittelet beskriver hvordan denne utvidelsen kan gjøres med minimale endringer på den opprinnelige modellen.

Oppsummering og diskusjon I dette kapittelet oppsummeres hele oppgaven. Videre betraktes oppgaven ut fra et større perspektiv, og til slutt skisseres det hvordan arbeidet med modellen kan videreføres.

Forøvrig inneholder rapporten bibliografi, forkortelser og kildekode.

1.5 Valg av skriftlig stil i rapporten

I en rapport som denne, hvor tele- og datakommunikasjon er et tema, vil det være et stort antall engelske begreper og forkortelser. Etter en nøye

vurdering har jeg valgt å ikke oversette de fleste av disse begrepene. Hovedårsaken til dette er at begrepene er så innarbeidet i fagfeltet at oversettelser kan virke fremmedgjørende. Jeg har valgt å oversette noen begreper der hvor det er naturlig, og en ikke risikerer misforståelser.

Jeg har valgt å bruke "jeg"-form i innledningen av rapporten, mens resten av rapporten bruker "vi"-form. Dette valget er gjort for å understreke at arbeidet som er utført har vært et samarbeidsprosjekt mellom meg som student og min veileder.

Kapittel 2

Voice Over IP

«God dag, Tussi,» sa Brumm.
«Dette er Tigergutt.»
«Hva er?» spurte Tussi.
«Dette,» sa Brumm og Nøff i munnen på hverandre. Og Tigergutt smilte sitt aller lykkeligste smil og sa ingenting.
Tussi gikk rundt Tigergutt, først den ene veien og så den andre veien.
«Hva sa dere det var?» spurte han.
«Tigergutt,» sa Brumm.
«Å,» sa Tussi.
«Han er akkurat kommet,» forklarte Nøff.
«Å,» sa Tussi enda en gang.
Tussi stod enda en stund og tenkte, så sa han:
«Når skal han gå igjen?»

— *A. A. Milne: Huset på Bjørnehjørnet*

VOICE OVER IP (VoIP) eller IP-telefoni er en teknologi som er kommet for å bli, og tegner til å kunne bli det som kan erstatte tradisjonell teknologi innen telekommunikasjon for å oversende lyd og bilder. Å bruke IP baserte nett til dette begynner å få større og større utbredelse. Drivkreftene bak dette er i hovedsak i to mektige interessegrupper; *Internet Engineering Task Force* (IETF) og *International Telecommunication Union* (ITU). Hver av disse gruppene har noe ulike motiver for VoIP, og er prinsippryttere av litt forskjellige paradigmer.

Dette kapittelet er et forsøk på å skildre hvilken utvikling og motivasjon som ligger bak VoIP. Videre gis det en beskrivelse av hvilket potensiale som ligger i denne teknologien både for rask utvikling og markedspenetrering av nye tjenester.

VoIP er et stort felt som strekker seg alt fra hvordan man skal ivareta tjenestekvalitet til utvikling av e-handelstjenester. Dette nedslagsfeltet er stoff nok til flere bøker. Derfor vil dette kapittelet konsentrere seg mest om å gi en overordnet forståelse av området.

Begrepet VoIP brukes i sammenhenger utover å *bare* være overføring av lyd. Det anvendes også i sammenhenger hvor det overføres flere mediastrømmer samtidig (for eksempel lyd og bilde). En betegnelse som noen ganger er brukt på dette er *Multimedia over IP* (MMoIP). I dette kapittelet vil VoIP, i tråd med andre kilder, bli brukt *både* om lyd- og multimediasesjoner.

2.1 En bit telekommunikasjonshistorikk

Telefonen må kunne kalles selve basiskomponenten innen telekommunikasjonstjenester. Den har, som [Sch99] skriver, gjennomgått dramatiske endringer omtrent hver tiende år. I 1950 årene gjorde introduksjonen av transatlantiske coax-kabler det mulig ringe internasjonale samtaler uten å gå via en manuell sentral. I 1960 årene ble lydkvaliteten på telefonsamtaler dramatisk forbedret som en følge av at digital overføring og svitsjing ble innført. Utviklingen fortsatte i 1970 årene med programmerbare svitsjer. Dette resulterte i tonesignalering og lokale tjenester som "samtale venter". 1980 årene markerte et nytt teknologiløft ved å skille media (f.eks. tale) og signalering i to forskjellige kanaler. Media fikk separate kanaler mens signaleringen delte en pakkebasert kanal. Slike systemer kalles *Out-Of-Band Common Channel Signaling Systems*. Et eksempel på et slikt system er *Signaling System Number 7* (SS7), som gjorde det mulig å lage tjenester som for eksempel 800-nummere.

Disse endringene markerer overgangen fra analoge overføringer til digitale linjesvitsjete overføringer og pakkebasert signalering.

Internettelefoni markerer i 1990 årene det foreløpig siste steget i utviklingen mot en infrastruktur bestående av kun pakkebaserte nett.

IP-telefoni, eller VoIP, har røtter som strekker seg så langt tilbake som tidlig på 1970 tallet. Det første eksperimentet på pakkebasert overføring av tale ble utført i følge [Sch99] august 1974. Da ble det demonstrert sanntidsoverføring av pakkebasert tale mellom University of Californias Information Sciences Institute og Massachusetts Institute of Technologys Lincoln Laboratories (MIT). Utviklingen av IP-telefoni gikk relativt sakte fram til 1991-1992 da det ble utført eksperimenter med pakkebasert audio på *DARPA Research Testbed network* (DARTnet), og de første IETF møtene ble sendt i multicast over Mbone. Noe av årsaken til at denne utviklingen gikk så sakte er at det var først i begynnelsen av 1990 årene lydkort ble så rimelig at de ble tilgjengelig for folk flest.

I 1995 lanserte Vocaltec en av de første applikasjonene innen IP-telefoni som baserte seg på PCer. Etter dette kom det gatewayer mellom de Internett baserte løsningene og det tradisjonelle telefonisystemet. Fokus ble etterhvert noe endret fra å bruke PCer som endeutstyr til at endeutstyret var vanlige telefoner. Slike konfigurasjoner hadde som hovedhensikt å gi rimelige telefonsamtaler over lange distanser.

2.2 Voice over IP kontra Voice over Internett

Ser man på det behovet *Voice over IP* i hovedsak prøver å dekke er det behovet for allmen, tradisjonell telekommunikasjon. Med dette menes behovet for å ha en toveis kommunikasjon med medmennesker i sanntid.

Når man bruker begrepet *Voice over IP* er det viktig å gjøre en distinksjon; om begrepet brukes på å sende audio- og videopakker i IP-format, eller om man bruker infrastrukturen i Internettet til transportør av audio- og videopakker. Brukes begrepet i den siste betydningen, stiller behovet nevnt over store krav til den eksisterende infrastrukturen i Internettet.

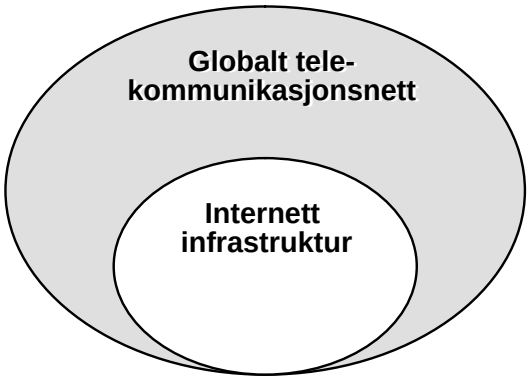
For det første skaper ordet *sanntid* en del hodebry. I følge [PD96] kan det ikke ta mer enn 300 millisekunder fra den ene parten har sagt noe til svaret er kommet tilbake . Dette er krav som det allmene Internettet aldri har blitt designet for. Her eksisterer det i dag en teknologi som behandler alle typer data likt. Det vil med andre ord si at en telefonsamtale vil konkurrere på lik linje med en e-post og www trafikk om båndbredde. Dette til tross for at forsinkelsen er mye mer kritisk for telefonsamtalen enn e-posten. Resultatet blir at det overlates til tilfeldighetene om det er tilfredsstillende kvalitet på telefonsamtalen, og ikke hvor viktig det er at dataene kommer fram i rett tid til mottakeren.

Et annet moment er den mengden av data som genereres av multimedia kommunikasjon. Audiokommunikasjon har relativt beskjedne krav for å kunne gi en akseptabel lydkvalitet, mens videobilder krever mer ressurser. Siden rimelige videokameraer for PC har blitt svært utbredt er det vanlig at samtalen suppleres av videobilder. Dette gjør at det forlanges mye større båndbredde. Som eksempler på dette kan det nevnes at en akseptabel lydkvalitet kan oppnås på en båndbredde på 4800 bit/sekund, mens video bør ha 64 000 bit/sekund eller mer. For sluttbrukerne innebærer ikke dette noe veldig stort problem siden svært mange bruker ISDN linjer for å koble seg til sin lokale Internettleverandør. Problemene vil i første rekke oppstå hos Internettleverandørene hvis VoIP får den utbredelse som er spådd og den eksisterende oppkoblingsteknologien fortsatt benyttes. Et eksempel for å illustrere dette er å se for seg 1000 samtidige VoIP brukere hos samme Internettleverandør. Brukerene stiller små krav til kvalitet og greier seg med 64000 bit/sekund hver til tale og video. Kravet båndbredden da 64 megabit/sekund. Selv med disse forsiktige anslagene summeres båndbreddekravet opp til en anselig mengde. Kravet til båndbredde vil også øke etter hvor mange man skal motta lyd og bilde fra, og hvor god lyd- og billedkvalitet som er påkrevet. Dette fører til at den datamengden som skal håndteres vil øke dramatisk for Internettleverandører etter hvert som bruk av VoIP blir mer utbredt. Det er stor sannsynlighet for at mange av disse leverandørene vil møte taket med tanke på båndbredde i deres stamnett.

Dette er i høyeste grad et tema i de ulike arbeidsgruppene i IETF. Det arbeides med løsninger for å kunne ha ulik prioritet på datapakker, men problemet med at båndbredde er en knapp ressurs er antagelig et vedvarende problem. Erfaring viser at Internettbrukere bruker mer dataintensive tjenester etter hvert som båndbreddekapasiteten øker.

En løsning på båndbreddeknappheten er å ta betalt for de dataene som

sendes over. En mulig modell er å oversende data med høy prioritet kostet mye, mens data med lav prioritet kostet lite. En konsekvens av dette er at data med høy prioritet går foran lavprioritets data gjennom nettet.



Figur 2.1: Internetts infrastruktur vs globalt telekommunikasjonsnett

En mer radikal løsning for å skaffe mer båndbredde til veie er å se på det faktum som figur 2.1 viser. Nemlig at Internetts infrastruktur i realiteten er en delmengde av det globale telekommunikasjonsnettet. Dette fordrer at Internett og telekommunikasjonsnettet smelter sammen. Da ville den til en hver tid ledige kapasiteten i nettet være en fellesressurs som alle kunne ha nytte av. Realiseringen av en slik løsning ligger sannsynligvis en godt stykke fram i tid. Det er investert for mange penger i dagens telekommunikasjonsnett til at det vil skje en revolusjon over natten i form av en overgang til et rent pakkebasert nett. Antagelig vil det også være krefter i Internettsamfunnet som motsetter seg en slik sammensmelting. Tanken på at Internett skal kommersialiseres ytterligere vil nok falle fremtredene personer i dette miljøet tungt for brystet.

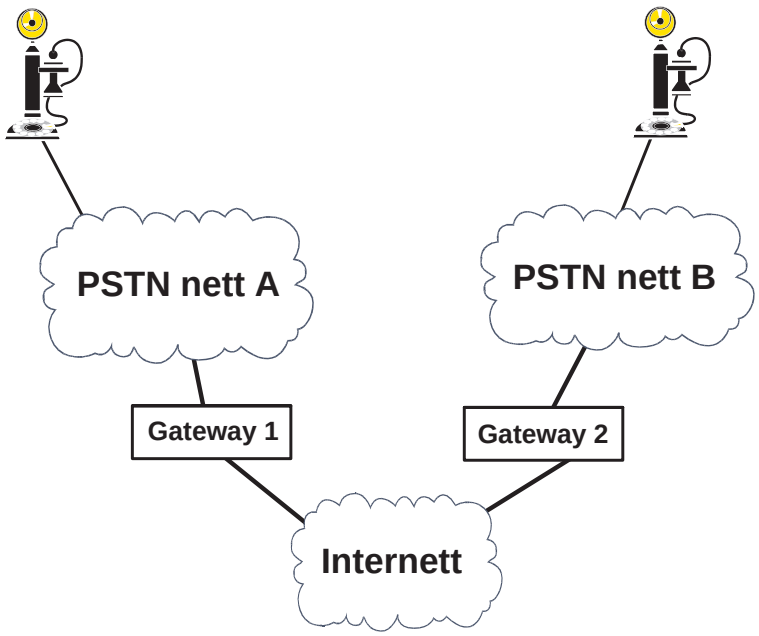
Til tross for dette er det klare fordeler for at nettene smelter sammen, og VoIP kan være det momentet som kan markere starten på en slik prosess.

2.3 Motivasjon for VoIP

Som nevnt før er drivkreftene bak VoIP i hovedsak IETF og ITU. Disse organisasjonene er tuftet på vidt forskjellige ideologier. IETF på forskning knyttet til universitetsmiljøene, og at den beste løsningen er den som overlever. ITU på den andre siden at det skal utvikles standardiserte løsninger før noe implementeres. Dette skal sikre interoperabilitet, men gjør ofte at spesifikasjonen blir store og detaljerte . Til tross for dette har både IETF og ITU mange sammenfallende interesser for at telefoni skal eksistere over et pakkebasert nett.

Disse interessene vil bli beskrevet nedenfor, men først er det riktig å nevne den opprinnelige motivasjonen for IP-telefoni; *å gi rimelige telefonsamtaler*. Dette er, og vil være, en drivkraft for utbredelsen av IP-telefoni.

Argumentet om at VoIP gir rimelige telefonsamtaler må settes i en litt større sammenheng enn kostnaden ved selve telefonsamtalen. For å få til dette er det nødvendig å presisere motivasjonen for de gruppene av brukere som vil anvende VoIP. Den første gruppen er personer som ikke er tilknyttet Internett, men ønsker å bruke VoIP for å spare penger. Denne gruppen vil typisk ringe til et 800 nummer som gir tilgang til IP nettet via en *gateway*. Et eksempel på dette er vist på figur 2.2. Her kommuniserer to telefoner tilknyttet det tradisjonelle telenettet sammen via Internett ved hjelp av to gatewayer. Dette markedet betegnes som *Toll Bypass markedet*. Den andre gruppen er brukere som allerede er tilknyttet Internett og ønsker å benytte IP-telefoni i tillegg til andre Internettjenester som websurfing og e-post.



Figur 2.2: Eksempel *Toll Bypass*

Ser en på gruppen som benytter seg av Internett for Toll Bypass trafikk, vil dette markedet sannsynligvis bli mindre. Årsaken til dette er at både Telia og Telenor tilbyr lokaltakst på samtaler over hele Norge i helger og etter klokken 17. Denne utviklingen vil i løpet av få år trolig utvides til å omfatte telefonering på dagtid. Dette gjør at potensiale for å spare penger ved Toll Bypass trafikk vil begrense seg til å gjelde samtaletrafikk på tvers av landegrensene.

For den andre gruppen er det flere momenter som gjør seg gjeldende. Det første er hvordan man er tilknyttet Internet. Hvis dette skjer via et modem og telenettet, er det allerede opprettet et telefonsamband mellom brukeren og Internettleverandøren. Hvis det da skal være lønnsomt å bruke VoIP kontra vanlig telefoni må Internettforbindelsen være av en type som belastes med en fast avgift, eller at telefoneringen er preget av utenlandstrafikk. Selv om det isolert sett kan være besparelser ved bruk av IP-telefoni,

er det ikke sikkert at dette regnestykket går i pluss hvis kostnaden ved anskaffelsen av nødvendig datautstyr og Internettabonnement tas med i betraktningen.

Dette kan summeres opp til at VoIP først gir rimelig telefonsamtaler ved utenlandstrafikk, og når Internettilknyttningen blir belastet med en fast avgift uavhengig av oppkoblingstid og datamengde.

Et annet motiv som på sikt kan drive utbredelsen av IP-telefoni er ønsket om å forenkle kommunikasjonsnettet. Det er flere grunner til ønsket om å forenkle telekommunikasjonsnettet til et altomfattende pakkenett. En tungtveiende årsak er, som [deC99] påpeker, at vedlikeholdskostnadene ved et linjesvitsjet nett er det dobbelte av et pakkebasert nett.

Å skille transport fra tjenester er et annet moment som vil forenkle nettet. I følge [PPY+99] forenkler en slik løsning både design av nettet og andre nettverkskomponenter, fordi utvikling av pakketransport og tjenester kan foregå uavhengig av hverandre. Som et eksempel på dette påpekes det at pakkerutere ofte er optimerte hardwareimplementasjoner mens tjenester som regel er implementert på vanlige datamaskiner.

Dette forenkler ikke bare nettet, men gjør det i tillegg mye lettere å implementere nye tjenester fordi denne utviklingen kan skje uavhengig av evolusjonen i transportnettet. Av nye tjenester vil det være naturlig at kjent funksjonalitet som for eksempel "viderekobling", "sette på vent" og "samtale venter" først blir implementert. Deretter er det sannsynlig at det dukker opp tjenester som integrerer VoIP tettere mot andre Internettjenester. Eksempler på dette kan være viderekobling til e-post ved ubesvarte kall og oppslag på websider for å finne mottakerens alternative lokaliseringer. Som en konsekvens av at endeutstyret i disse miljøene både har stor prosesseringskraft og er utstyrt for å håndtere multimedia, kan det ligge sterke innslag av multimediateknologi implementert i nye tjenester. Tenkte tjenester kan for eksempel være fjernundervisning og telemedisin.

2.4 Plassering av intelligensen

Når man har sørget for å skille transport og tjenester fra hverandre er det naturlig å vurdere hvor intelligensen i nettet skal være. Med dette menes det hvor tjenestelogikken skal være lokalisert. Et alternativ er at alt ligger på innsiden av telekommunikasjonsnettet. Det er dette utgangspunktet som ligger til grunn for ITU anbefalingen *Intelligent Network* (IN) [ITU93b]. I IN er transport og tjenester skilt, og filosofien er at man skal kunne bygge opp nye tjenester ved å sette sammen et vilkårlig sett med gjenbrukbare tjenester og nettverksuavhengige byggeklosser. Disse komponentene kan ses på som ord i et språk som det kan dannes meningsfulle setninger av. Fordelen er at endeutstyret kan være enkelt siden kompleksiteten blir liggende i nettet. Det viser seg imidlertid at antallet tjenester kan bli så stort at det kan bli problemer med å administrere dem. Jan A. Audestad gir et eksempel på dette i [Aud99, side 27]. Han poengterer at antall byggeklosser i et IN system lett kan komme opp i 100 eller mer. Da

vil antall mulige kombinasjoner lett overstige $2^{10} \approx 10^{30}$. Audestad understreker at selv om mange av disse kombinasjonene ikke vil gi meningsfulle tjenester, vil det enda være igjen et så stort antall kombinasjoner at det ikke er mulig å administrere dem. For å yte IN rettferdighet må det sies at meningen med IN ikke var å redusere kompleksiteten i nettet, men å øke fleksibiliteten i design av tjenester.

Et annet alternativ er en fordeling hvor all tjenestelogikk ligger i endepunktene og nettet kun har som oppgave å rute og transportere datapakker. Dette vil holde kompleksiteten på et lavt nivå slik at administrasjons- og vedlikeholdskostnadene i telekommunikasjonsnettet blir holdt på et minimum. På den annen side vil det kreve mer av endeutstyret. I utgangspunktet er dette ikke noe umulighet hvis majoriteten av endeutstyr i realiteten er PCer. Brukere av IP-telefoni er i høy grad i besittelse av en PC som de bruker til VoIP, så slik sett ville ikke en slik fordeling mellom nett og endeutstyr være umulig. Det som derimot ville skape problemer er at alle tjenester en bruker måtte finne det for godt å bruke, må ligge lokalt hos brukeren. Dette vil føre til en meget høy grad av duplisering av tjenester, og at tjenester som krever sikkerhet lett kan bli utsatt for svindel. Netthandel og telebank som har store krav til sikkerhet og autentisering, kan være svært vanskelig å realisere i en modell hvor alle tjenester må ligge lokalt hos abonnenten.

En modell som virker mer tiltalende er modellen [PPY+99] presenterer. I denne modellen foreslås en tredeling av kommunikasjonsnettet:

- Et transportnett som transporterer og ruter datapakker.
- Servere hvor tjenestene ligger.
- Endeutstyr med ulik grad av prosesseringskraft.

I en slik modell vil servere være endepunkter sett fra transportnettet, mens de vil være en del av nettet når det gjelder tilbud av tjenester. Denne modellen vil ikke være ulik hvordan Internett er bygd opp; selve transportnettet bestående av rutere, gatewayer og topologiske forbindelser og tjenestene som web og e-post på egne servere.

I [PPY+99] gis det fire begrunnelser for en slik modell. For det første er det behovet for å administrere og fordele transportressurser, registrere brukte ressurser, fakturering og forhindring av svindel og bedrageri. Slike behov ivaretas trolig best av dedikerte servere i nettet. Den andre begrunnelsen er behovet for å ta vare på tilstanden i nettet selv om endepunkter for tiden ikke er tilknyttet eller tilgjengelige. Dette er tilstander som for eksempel brukes for å vite når samtaler skal omdirigeres. Et eksempel på dette er viderekobling når nummeret er opptatt. Som den tredje begrunnelsen gis behovet for spesielle tjenester tilknyttet nødnummer. En av disse tjenestene kan være å rute samtalen til brukerens nærmeste alarmsentral. Det siste argumentet [PPY+99] bruker for denne fordelingen er at ved å lagre implementasjoner av tjenester på et relativt lite antall servere i steden for millioner av endepunkter, vil det lette oppdatering med nye versjoner og øke hastigheten på utviklingen av nye tjenester.

2.5 *Legacy* – Arven som ingen vil ha

For at en overgang fra det eksisterende telekommunikasjonssystemet til et nytt pakkebasert system skal være vellykket er man nødt til å ivareta behovet til de brukerene som fortsatt har gamle konvensjonelle telefonapparater. Dette går under begrepet *Legacy*[1]. Med dette menes at en er forpliktet til å sørge for bakoverkompabilitet med gammelt utstyr.

Hvis man betrakter IP-telefoni og det tradisjonelle PSTN systemet som to forskjellige rør med forskjellige former, det ene rundt mens det andre er firkantet, vil det være behov for et spesielt skjøtestykke for å forbinde disse rørene. Disse skjøtestykkene vil opptre i form av *gatewayer* som sørger for at VoIP nettet og PSTN nettet kan kommunisere smertefritt sammen. En kan tenke seg en gradvis utvikling. Først vil gatewayene være bindeledd mellom større systemer fra hvert teknologidomene. Etterhvert som telefonnettet omdannes til IP teknologi, vil gatewayene rykke nærmere endeutstyret. Til slutt kan en tenke seg at de som fortsatt har gamle PSTN telefoner kobler disse til en "svart" boks som i realiteten er en liten gateway.

Denne fremgangsmåten vil sikre en gradvis overgang fra det eksisterende systemet til et pakkebasert telefonsystem uten at brukerne blir unødig berørt.

2.6 "No such thing as a free lunch" – Fakturering

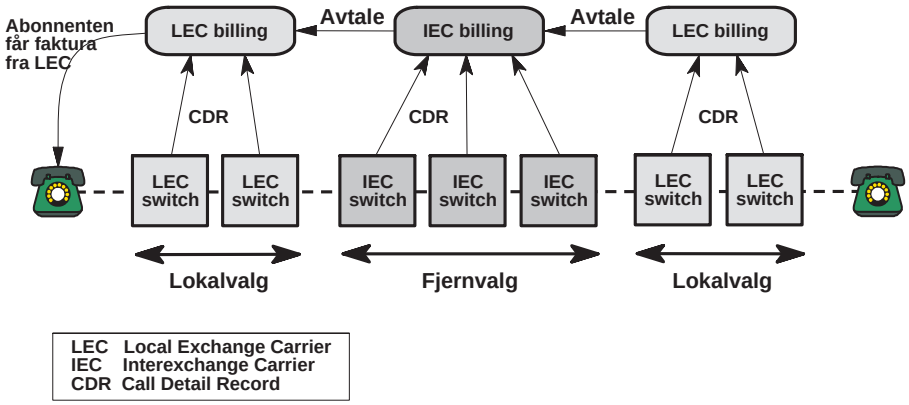
For at det skal være interessant å bygge ut en internasjonal infrastruktur for IP-telefoni vil det være nødvendig at det er kommersielle interesser involvert i finansieringen. At en slik krafttak som en slik utbygging tilsier skal være finansiert av ideelle organisasjoner vil ikke være realiserbart hvis man skal operere med en overskuelig tidshorisont. Med andre ord må det være mulig å tjene penger på den trafikken som blir generert i nettet for å sikre at det er interessant å være med på å finansiere et slikt løft.

Tradisjonelt sett har inntektene til teleoperatørene blitt generert som en følge av samtaletrafikk. Figur 2.3 på neste side viser hvordan abonnenten blir fakturert for samtaler. Den lokale sentralen (LEC) abonnenten er tilknyttet vil sammen med eventuelle sentraler for fjernvalg (IEC), generere detaljerte journaler (CDR) for samtalen. En CDR vil inneholde blant annet samtalens starttid, lengde, nummeret det ringes fra og til. Samtalens CDR blir så oversendt til abonnentes faktureringssenter (LEC billing), som samler opp dataene og sender ut fakturaer med jevne mellomrom.

Hele grunnlaget for fakturering av samtaler på tvers av landegrenser er basert på avtaler om samtrafikk mellom de ulike landenes teleoperatører. Utgangspunktet for slike avtaler og den betalingsmodellen som ligger i dem, er målt og forventet trafikk mellom landene.

Modellen med et avtaleverk mellom teleoperatører for å sikre interoperabilitet vil nok fortsatt bestå i et pakkebasert nett. Men hva og hvor mye man

[1] På norsk omtales dette begrepet som "installert base".



Figur 2.3: Tradisjonell fakturering i telesystemet

skal betale for ulike tjenester er et annet spørsmål. Et eksempel er om man skal ta mer betalt for en telefonsamtale som har lengere varighet enn en filoverføring, men har generert mindre data. Dagens modell med å basere seg på en tidsakse for hvor mye man skal betale kan bli utilstrekkelig. En annen løsning kan være å ta betalt for typen data som sendes over transportnettet, slik at det er dyrere å overføre data for telefoni enn filoverføring. Denne løsningstypen vil stride mot ønsket om å forenkle transportnettet. Transportnettet må da kunne se forskjell på data som sendes i nettet, og klassifisere de etter gitte kategorier. Dette gjør at nettet blir belastet med mye ekstra funksjonalitet utover det å transportere og rute pakker. Igjen vil dette fordyre vedlikehold og administrasjon av nettet.

En enklere betalingsmodell er å ta betalt for den tjenestekvaliteten som forlanges tilbudt av nettet med et sett med enkle parametere. En framgangsmetode kan være en lav fastavgift som dekker datatrafikk uten krav til kvalitet[2]. E-post vil typisk være tjeneste som ikke krever mer enn dette. Utover denne fastavgiften betaler man for det man forlanger av nettet. Det kan være en garantert båndbredde for en sesjon eller at dataene kommer fram i løpet av en gitt tid. En slik modell for betaling for bruk av transportnettet er foreslått i [PPY+99, side 88]. Modellen vil holde administrasjonskostnadene ved fakturering nede, samtidig som den kan sikre en rettferdig betaling for transportkapasiteten som anvendes.

Utover de inntektene som er relatert til bruk av transportnettet ligger det et stort potensiale for inntekter i tjenester relatert til IP-telefoni. Et eksempel på dette er en tjeneste som tilbyr voicemail. Slike og andre mer avanserte tjenester vil det være sannsynlig at tilbys fra leverandører som har spesialisert seg på å utvikle slike tjenester. Dette fordrer at det er mulig at de kan tjene penger på sine tjenester uten for mye kostnader knyttet til det å få kunden til å betale. Bruk av en form for kredittkort vil løse denne problemstillingen, men krever til gjengjeld at det finnes tilstrekkelig sikkerhet rundt systemene. Denne typen systemer krever at det på et eller annet

[2] også kalt *Best Effort*.

tidspunkt må opprettes et tillitsforhold mellom kunden og tjenesteleverandøren. Hvis dette er en nødvendighet for enhver bruk av slike tjenester vil man komme opp i et utall av slike relasjoner. En løsning på denne problematikken kan være å bruke en *tiltrodd tredjepart* som går god for kunden og leverandøren, og fungerer som en mellommann for transaksjonen. Organisasjoner som kan ha en slik rolle må ha meget stor integritet for at de skal kunne bli tiltrodd en slik oppgave. En aktør som kan være aktuell er Det Norske Veritas.

De modellene som er skissert over vil tilfredsstille både den typiske abonnementssituasjonen, og det markedet hvor man betaler for den eller de tjeneste som brukes. En fordel med slike modeller er at man betaler for det man forbruker av nettressurser og tjenester utover den lave fastavgiften man betaler for tilknytting til transportnettet.

2.7 Oppsummering og konklusjon

VoIP eller IP-telefoni har sterke drivkrefter bak seg i form av IETF og ITU. Dette i tillegg til den store interessen for konseptet, tilsier at telefoni basert på IP teknologi vil få en viktig rolle i fremtiden.

Selv om flere betegner VoIP som et paradigmeskifte innen telekommunikasjon, må ikke det føre til den misforståelsen at alle ønsker dette teknologiskiftet. Det finnes store aktører i telekommunikasjonsmarkedet som tjener gode penger på tradisjonelt utstyr. Disse aktørene kan ha forskjellige motiver for å ikke ønske velkommen et raskt teknologiskifte. Et av dem kan være tanken på å miste milliardinntekter.

Dagens systemer fungerer svært godt både når det gjelder tilgjengelighet og håndtering av feilsituasjoner. Tanken på å kaste vrak på disse systemene til fordel for teknologi som i høy grad blir basert på PC, kan virke skremmende for mange innen telekommunikasjonsverdenen. Dette kan også være et motiv for å ville holde igjen utbredelsen av IP-telefoni.

Til tross for dette har IP-telefoni et stort potensiale. Ikke minst med tanke på en forenkling av transportnettet, med sitt markerte skille mellom nett og tjenester og en lav terskel for å lage tjenester tilknyttet teknologien.

En forenkling av det eksisterende telekommunikasjonsnettet vil virke svært tiltalende på mange teleoperatører. En av hovedårsakene til dette er store økonomiske gevinster i form av reduserte kostnader relatert til mindre administrasjon og vedlikehold av nettet.

Det markante skillet mellom nett og tjenester åpner markedet for mange nye aktører når det gjelder utvikling og leveranser av tjenester. Det er grunn til å tro at i fremtiden vil IP-telefoni skape store muligheter innen tjenesteutvikling også for små bedrifter, og ikke bare være forbeholdt store selskaper som Alcatel, Lucent, Nortel og Ericsson.

VoIP står ovenfor store utfordringer når det gjelder å erstatte dagens eksisterende og velfungerende telekommunikasjonssystemer. Noen av dem imøtekommes med forslag som kan vareta temaer som sikkerhet og fakturering. I [PPY+99, side 89] ramses det opp forslag til eksisterende protokoller

som kan løse disse problemene. Når det gjelder sikkerhet og autentisering kan det være nødvendig at systemer basert på IP-telefoni pålegges strengere regler enn nåværende telesystemer. Årsaken til dette er at dagens systemer til dels er sentraliserte og proprietære. Noe som i seg selv gir et visst nivå av sikkerhet. Dette mister man ved å gå for desentraliserte og åpne løsninger. Det er grunn til å tro at det tapte må kompenseres med et ekstra nivå av sikkerhet i forhold til tradisjonelle telekommunikasjonssystemer.

Et tankekors er dog at løsningene som er skissert i liten grad er implementerte løsninger som er satt i drift. Riktignok eksisterer løsningene som enkeltstående systemer, men det er fortsatt et åpent spørsmål om hvor vidt løsningene fungerer godt nok sammen for IP telefoni, og om de skalerer til å dekke nasjonale og internasjonale trafikkmengder.

Fremtiden vil vise om det er IP-telefoni som blir drivkraften til en konvergering av Internett og den tradisjonelle telekommunikasjonsverdenen. Det blir interessant å se om disse miljøene forenes, eller om dataene fra de respektive systemene lever hver sine liv og passerer hverandre som skip i natten.

Kapittel 3

H.323

> «Kunne du fly opp til brevkassen med Nøff på ryggen?» spurte han.
> «Nei,» sa Nøff fort. «Det kunne han ikke.»
> Ugla begynte på en forklaring om Nødvendige Ryggmuskler. Han hadde forklart dette til Brumm og Kristoffer Robin en gang før, og siden den gang hadde han ivrig ventet på en anledning til å gjøre det igjen, fordi det er noe du lett kan forklare minst to ganger før noen begriper hva du snakker om.
>
> — *A. A. Milne: Huset på Bjørnehjørnet*

INTEROPERABILITET er en egenskap *International Telecommunication Union* (ITU) setter svært høyt. Dette fører til at spesifikasjoner blir svært detaljerte og derav omfangsrike. På grunn av dette kreves det ofte en betydelig innsats for å trenge inn i ITU Spesifikasjoner. H.323 [ITU99] for pakkebaserte multimedia kommunikasjonssystemer er ikke noe unntak fra dette. Standarden krever både et omfattende studie av spesifikasjonen til H.323, og i tillegg kreves det kunnskaper om de underliggende standardene.

Et annet moment som utilsiktet gjør arbeidet med å sette seg inn i H.323 vanskeligere, er at kildematerialet ikke er blitt fullstendig oppdatert etter hvert som standarden har utviklet seg. Et eksempel på dette finnes i [Dat98, side 1]. Her omtales i første kapittel H.323 som en standard for kommunikasjon over IP inkludert Internet. I neste kapittel nevnes H.323 som en anbefalt standard for multimediakommunikasjon over LAN uten tjenestekvalitet.

3.1 Bakgrunn og motivasjon

H.323 er en videreføring av ITUs arbeid med audiovisuell kommunikasjon, og bygger på tidligere standarder for koding av lyd, bilde og telesignaler i data- og telenett. H.323 fungerer som en "paraply" for et sett av standarder for multimediakommunikasjon over pakkebaserte nett. I utgangspunktet

var H.323 versjon 1 (1996) tiltenkt å brukes for multimediakommunikasjon i et lokalnett (LAN). Denne tankegangen kommer tydelig fram i hvordan H.323 versjon 1 bruker multicast til lokalisering av tjenesteytere.Et eksempel på tjenesteyter kan være en *Gatekeeper* (GK) (se side 28). Andre ting å påpeke i versjon 1 var en utstrakt bruk av TCP og en veldig stegvis tankegang. Et eksempel på dette er at det først ble satt opp et kall, deretter startet forhandlingen om media. Disse momentene kunne føre til et høyt antall rundturer[1].

Momentene nevnt over ble tatt til etterretning, og i versjon 2 (1998) av H.323 ble disse problemene adressert. Det ble spesifisert en *Fast Start* mekanisme som muliggjør overføring av mediapreferanser i den initielle fasen av kalloppsettet. En annen nyhet var muligheten for å tunnelere H.245 (side 24) signalering i TCP kanalen for Q.931 (side 24) signalering. Dette ble gjort for å angripe problematikken med mange TCP forbindelser.

Noe av den opprinnelige LAN tankegangen er fortsatt en del av H.323 versjon 2. Dette kommer til uttrykk i sonebegrepet i H.323 (se figur 3.1 på neste side fra [Dat98]). I en *H.323 Zone* er det en Gatekeeper som har ansvaret for lokalisering av brukere. Metoden versjon 2 av H.323 bruker for lokalisering av brukere i andre soner er fortsatt spesifisert på en slik måte at den vil skalere svært dårlig. Dette er et problem som diskutereres blant annet i Tiphon [TIP99]. Tiphon er etablert av ETSI [ETS99] som et forum for diskusjon og etablering av standarder innefor VoIP. Et av innleggene, [KHN99], tar for seg denne problematikken ved å foreslå en oppdeling av Gatekeeper. Dette løser langt på vei problemet med å operere på tvers av flere H.323 Zones. Oppdelingen av Gatekeeper er foreslått slik:

u-GK (User Gatekeeper) er bruker/abonnent orientert.

SiteKeeper er en *Access Site* i tillegg til å håndtere nødvendige VoIP og MMoIP ressurser.

Denne oppdelingen, som ikke er ulik oppdelingen i GSM mobilnettet, skiller nett fra brukere og gjør det dermed lettere å kommunisere på tvers av flere Gatekeepere.

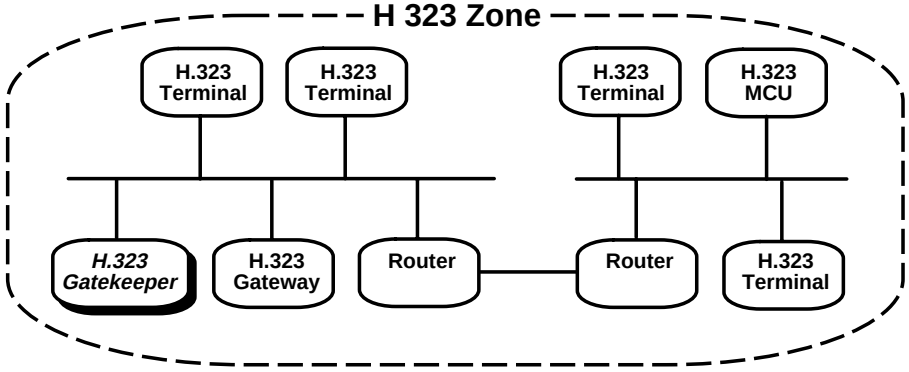
H.323 er en standard som er under stadig utvikling, og draftet til versjon 3 utgitt. Den offisielle versjonen er planlagt å være ferdig i år 2000.

3.2 Oppbygning og virkemåte

Som nevnt over kan H.323 betraktes som en “paraply” for andre ITU standarder. Disse standardene kan grovt deles inn i følgende områder:

- Funksjoner for kommunikasjon- og anropskontroll.
- Mediatyper:

[1] Å sende data fra A til B og tilbake til A. Bør holdes på et minimum fordi dette kan ta uforholdsmessig lang tid hvis avstandene er lange.



Figur 3.1: H.323 Zone

  - Data.
  - Audio.
  - Video.
- Tilleggstjenester.
- Sikkerhet.

Den første delen av listen omfatter *informasjonsstrømmene* som H.323 entitetene bruker for å kommunisere. [ITU99] klassifiserer disse informasjonsstrømmene i video, audio, data, kommunikasjonskontroll og kallkontroll.

Disse informasjonsstrømmene skal formateres og sendes over det pakkebaserte nettet i henhold til standarden H.225.0 [ITU98b]. Denne standarden inkluderer RTP, RTCP, RAS og kallsignalering (Q.931). RTP og RTCP brukes for innpakking av data og synkronisering av det underliggende pakkenettet. Figur 3.3 på side 26 viser hvordan dette deles inn. H.225.0 brukes altså både som en del av systemkontrollen *og* et H.225.0 lag for innpakking og synkronisering. Verdt å merke seg er at H.245 ikke er en del av H.225.0.

En ting som kan virke forvirrende ved H.225.0 er at begrepene “Q.931” og “H.225.0 Call Control Messages” brukes om hverandre. For å presisere dette så bruker H.225.0 bare et *subsett* av Q.931 meldingene. Dette betyr at “H.225.0 Call Control Messages” er Q.931 meldinger, men ikke *alle* meldingene i Q.931 [ITU93a] standarden er spesifisert. Årsaken til dette er at Q.931 opprinnelig er spesifisert for ISDN. Til tross for dette brukes begrepene “Q.931” og “H.225.0 Call Control Messages” om hverandre innen H.323, og i det følgende skal det forstås at disse begrepene er ensbetydende.

Når det gjelder tilleggstjenester og sikkerhet i H.323 er dette forhold som tas hånd om av de respektive standardene H.450 og H.235. Med tilleggstjenester menes det tjenester utover vanlige samtaler. Eksempler på dette er viderekobling og sette samtaler på vent.

3.2.1 Funksjoner for kommunikasjon- og kallkontroll

Som [Dat98, side 10] nevner, kan selve funksjonaliteten for kallkontroll kalles hjertet i et H.323 system. Denne funksjonaliteten inkluderer signalering for kalloppsett, utveksling av mediapreferanser og egenskaper, og i tillegg, all nødvendig funksjonalitet for å opprette og kontrollere mediastrømmer.

Den overordnede systemkontrollen er satt sammen av tre ulike signaleringsfunksjoner:

- Kanal for Q.931 kallsignalering.
- Kanal for H.245 kontroll.
- Kanal for *Registration Admission Status* (RAS).

Q.931 — kallkontroll

Q.931 brukes i kanalen for kallsignalering til å etablere, ta ned og utføre annen nødvendig signalering for en forbindelse mellom to H.323 entiteter.

Denne kanalen er det første som settes opp mellom to entiteter, og er pålitelig i likhet med H.245 kanalen. Fra H.323 versjon 2 er det mulig å tunnelere H.245 meldinger gjennom Q.931 kanalen. Dette gjør at antall TCP forbindelser reduseres og dermed tiden det tar for å sette opp et kall.

H.245 — kommunikasjonskontroll

Kanalen for H.245 kontroll er en pålitelig kanal (dvs. en TCP forbindelse) som overfører meldinger angående hvordan H.323 enheten opereres. Dette inkluderer utveksling av mediakapabiliteter, åpning og lukking av logiske kanaler, meldinger for flytkontroll og andre generelle kommandoer og indikasjoner.

Utveksling av mediakapabiliteter og preferanser er en av de fundamentale egenskapene i H.245. I dette ligger det blant annet mulighet for å formidle et ønske om forskjellig media for sending og mottaking. I tillegg til dette er det innebygget en mekanisme for å formidle både preferanser og mediakapabiliteter som logiske uttrykk. Dette gir H.245 en stor styrke når det gjelder forhandling om media. Som en følge av denne egenskapen reduseres antall rundturer mellom partene under mediaforhandling.

Antallet H.245 kanaler som er nødvendig per kall er begrenset til en.

Registration Admission Status

Registration Admission Status (RAS) er en signaleringsfunksjon som utfører prosedyrer for registrering, tillatelser, endring av båndbredde, statusmeldinger og nedkobling av forbindelser mellom H.323 endeutstyr og Gatekeepere (se 3.3.3 på side 28). Signaleringskanalen for RAS er uavhengig

av kanalene for kallsignalering og H.245 kontroll. I H.323 konfigurasjoner som har en Gatekeeper åpnes en kanal for RAS signalering mellom Gatekeeper og endepunktet.

RAS brukes bare hvis det eksisterer en Gatekeeper i H.323 konfigurasjonen.

3.2.2 Mediatyper

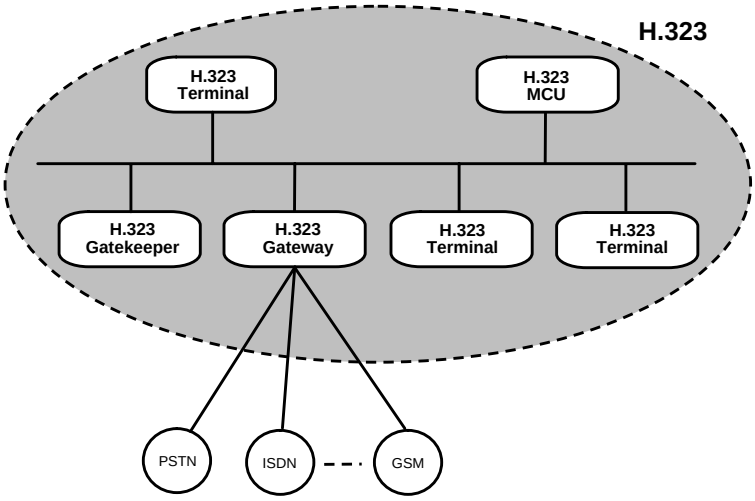
H.323 støtter mange mediatyper for audio, video og data[2]. Det legges visse krav for å sikre interoperabilitet. I dette ligger det at alle H.323 terminaler *må* støtte tale som et minimum. Og støtter man tale, må alle kunne håndtere tale CODEC'en G.711. I tillegg kan terminaler støtte andre typer audio CODEC'er.

På denne måten fortsetter det for andre mediatyper; terminalene *kan* ha video, og i så fall med mulighet for å kode og dekode H.261 QCIF som et minimum. I likhet med video er data valgfritt, men hvis det støttes må alle kunne tolke formatet T.120.

3.3 Bestanddeler og enheter

H.323 definerer fire hovedkomponenter; *Terminal*, *Gateway*, *Gatekeeper* og *Multipoint Control Unit* (MCU).

Figur 3.2 viser de forskjellige komponentene i H.323 og hvordan de hører sammen. Komponentene beskrives mer utførlig under.



Figur 3.2: Sammenkobling H.323 komponenter

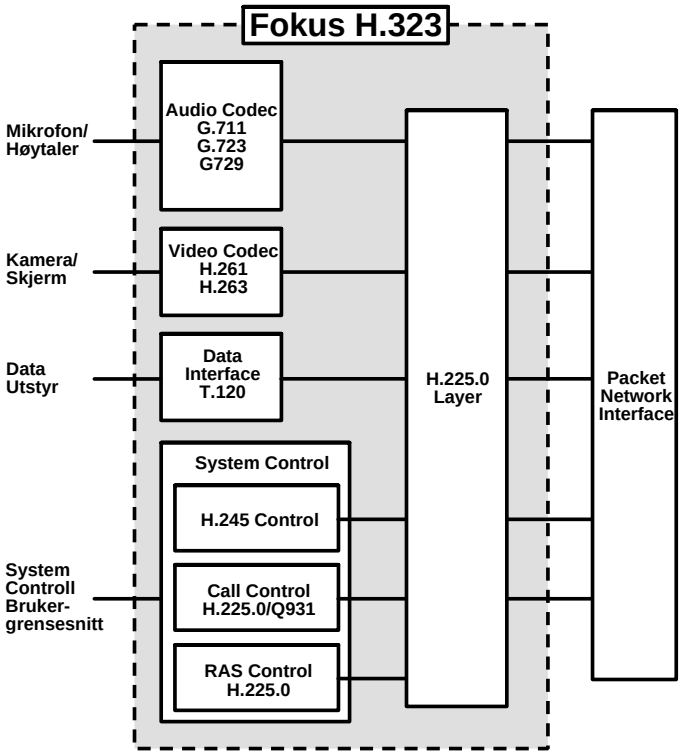
[2]I H.323 brukes dette begrepet om protokoller for datautveksling og applikasjonsdeling. Et eksempel kan være T.120 som ofte er brukt i *Whiteboard* applikasjoner.

3.3.1 Terminal

En H.323 terminal er, i følge [ITU99], et klientendepunkt som støtter toveis sanntidskommunikasjon i pakkebaserte nett. På grunn av denne vide definisjonen kan en H.323 terminal anvendes i mange forskjellige typer nett.

Som nevnt tidligere må en H.323 terminal støtte tale, mens støtte for video- og datakommunikasjon er valgfritt. Det er i tillegg nødvendig at terminalene støtter H.245. Ut over dette er er det nødvendig med Q.931 for kallsignalering og RAS for å kommunisere med en eventuell Gatekeeper.

Valgfrie komponenter i en H.323 terminal er video CODEC, datakanal (til overføring av f.eks. T.120) og MCU (3.3.6 på side 30).



Figur 3.3: Komponenter i en H.323 terminal

Figur 3.3 fra [ITU99, side 14], viser en anvendelse av komponenter i en H.323 terminal og hvilke enheter H.323 spesifikasjonen berører:

Video CODEC koder video fra en kilde (f.eks et kamera) for transmisjon, og dekoder det mottatte videoformatet for å videresende det til en skjerm.

Audio CODEC koder audio fra en kilde (f.eks en mikrofon) for transmisjon, og dekoder den mottatte audiokoden for så å videresende det til en høytaler.

Datakanal støtter telematikkapplikasjoner som applikasjonsdeling, filoverføring, databaseaksess, o.l. Den standardiserte applikasjonen for *audiografiske konferanser*[3] er T.120, men andre protokoller og applikasjoner kan bli brukt.

System Control tilbyr signalering for å kontrollere H.323 terminalen. System Control gir mulighet for kallkontroll, utveksling av mediakapabiliteter, signalering av kommandoer og indikasjoner samt meldinger for å åpne logiske kanaler og beskrive hva de inneholder.

H.225 Layer formaterer video-, audio-, data- og kontrollstrømmer til meldinger for overføring på nettet. Ved mottak har dette laget ansvar for å reformatere meldingene til sin opprinnelige form. Laget har i tillegg ansvar for logisk innramming, sekvensnummerering, feilfinning og feilretting.

Det bemerkes at applikasjonene eller protokollene som ønskes brukt i datakanalen er utenfor H.323 spesifikasjonen.

3.3.2 Gateway

I den definisjonen av *Gateway* (GW) som [ITU99] bruker slås det fast at de primære oppgavene er å:

- oversette mellom transmisjonsformater (pakkestørrelser etc).
- oversette mellom kommunikasjonsprosedyrer.
- kunne foreta oppsett og nedtagning av kall på begge sider av Gatewayen.

I tillegg til dette *kan* en GW være ansvarlig for oversetting mellom forskjellige mediaformater, men hovedoppgaven er å oversette signalering og kommunikasjonsprosedyrer.

I H.323 spesifikasjonen brukes begrepet GW i sammenheng med å kunne oversette kall mellom pakkenettet på den ene siden og det linjesvitsjede nettet på den andre siden, eller omvendt. Gatewayen er altså tiltenkt en rolle som mellomledd mellom tradisjonelle telefoner og multimediaterminaler i H.323 systemet.

En H.323 Gateway opptrer med to ansikter i forhold til nettene den står mellom. På H.323 siden oppfattes den som en annen H.323 terminal eller som en MCU (se på side 30). På SCN siden vil Gatewayen oppfattes som en SCN terminal eller en MCU.

Det anbefales at H.323 systemer som inneholder en Gateway også består av en Gatekeeper. Årsaken til dette er at Gatekeeperen sørger for å oversette mellom telefonnummeret og terminalens IP adresse.

[3]Konferanser mellom parter hvor for eksempel en applikasjon deles samtidig mens man deler en elektronisk skissebok. I tillegg til dette kan lyd og bilde av konferansedeltakerene overføres.

Selv om [ITU99] bruker begrepet GW i en H.323/SCN kontekst, er det ikke noe konseptuelt i veien for at en GW kan brukes for å forbinde terminaler som bruker forskjellige multimediaprotokoller. Et eksempel på dette kan være en GW brukt som bindeledd mellom H.323 terminaler og terminaler som anvender SIP (se kapittel 4 på side 35).

3.3.3 Gatekeeper

En *Gatekeeper* (GK) er en valgfri komponent i et H.323 system. Gatekeeperen opptrer som et knutepunkt innenfor sin H.323 Zone (se figur 3.1 på side 23), og har som oppgave å tilby tjenester for kallkontroll til H.323 endepunkter. Endepunktene er logisk skilt fra Gatekeeperen som egne enheter, men en Gatekeeper kan fysisk være en del av en terminal, MCU, Gateway eller en annen H.323 enhet.

Hvis det eksisterer en Gatekeeper som en del av et H.323 system *skal* den tilby følgende tjenester:

Oversette adresser Oversette *alias* adresser til transport adresser (f.eks. IP adresser).

Adgangskontroll Gatekeeperen skal autorisere aksess til nettverket. Dette kan være basert på kallautorisering, båndbredde e.l.

Båndbreddekontroll Støtte for forespørsler, tildeling og avvisning av krav om båndbredde.

Zone Managment Administrasjon av H.323 Zone ved å tilby funksjonene ovenfor til terminaler, MCUer og Gatewayer som er registret hos GK.

I tillegg til funksjonene over, kan en GK tilby følgende tilleggstjenester:

Kallkontroll signalering En GK kan velge enten å rute kallsignaleringen via seg selv, eller at signaleringen går direkte mellom endepunktene.

Autorisering av kall GK kan avvise kall etter gitte regler satt av GK administratoren.

Båndbreddeadministrasjon Kontroll av antall samtidige H.323 terminaler som kan få tilgang til nettet. En terminal kan bli avvist på grunn av at antall samtidige terminal sesjoner blir overskredet. Hvilke kriterier som brukes for å bestemme om båndbredde er tilgjengelig er ikke en del av H.323 spesifikasjonen. Dette vil typisk være leverandøravhengig.

Kalladministrasjon GK kan for eksempel vedlikeholde en liste av aktive kall for å holde rede på hvilke terminaler som er opptatt. Et annet eksempel er å forsyne funksjoner for båndbreddekontroll eller faktureringssystem med nødvendig informasjon om kallene (antall audiosesjoner, varighet, etc).

Katalogtjenester Funksjonalitet for lokalisering av for eksempel terminaler og Gatekeepere. Dette er et felt som er under utarbeiding.

Når det gjelder funksjonaliteten for båndbreddekontroll og administrasjon, vil dette i hovedsak basere seg på at GK administratoren har spesifisert en øvre grense for samtidige kall i den aktuelle H.323 Zonen. Denne metoden kan være noe statisk fordi den ikke tar hensyn til hvor mye data som sendes i den enkelte konferanse og at sesjoner ikke bruker hele sin tildelte kapasitet. Dette kan resultere i at en ny forbindelse blir avvist selv om det faktisk er ledig båndbredde. En slik situasjon kan oppstå når de eksisterende forbindelsene utveksler data med lav båndbreddeintensitet. Et annet eksempel på dette er at det i starten av en konferanse kreves båndbredde for audio og video, og at videooverføringen avsluttes senere i sesjonen.

Den valgfrie egenskapen med å kontrollere kallsignalering gjennom GK kan føre til at H.323 kall rutes mer effektiv. Dette gjelder særlig i flerpartskonferanser. Da har GK en mulighet for å samordne signaleringen og dermed unngå redundans. I tillegg til dette er det mulig å logge samtaletrafikk for eksempel for å ta betalt for bruk av nettet når signaleringen går gjennom Gatekeeperen.

Hvis H.323 kall rutes gjennom Gatekeeper kan den brukes til å videresende data til terminaler plassert i andre H.323 Zoner. Mekanismen for lokalisering av terminaler på andre Gatekeepere er foreløpig under utarbeiding, men man kan tenke seg en tjeneste som ikke er ulik katalogtjenestene LDAP eller DNS.

H.323 entitetene som benytter seg av tjenestene som Gatekeeperen tilbyr bruker RAS signalering for å kommunisere med GK. I RAS er disse tjenestene definert som adresseoversetting, adgangskontroll, administrasjon av båndbredde og administrasjon av den respektive H.323 Zone.

3.3.4 Multipoint Controller

I konferanser med tre eller flere endepunkter vil en *Multipoint Controller* (MC) ha som hovedoppgave å yte funksjonalitet for å kontrollere denne typen konferanser. Nærmere spesifisert opptrer MC som et knutepunkt for terminalene i sesjonen når det gjelder utveksling av mediakapabiliteter og master/slave determinering.

En MC er ikke en egen fysisk enhet og er samlokalisert i andre H.323 entiteter. Av disse entitetene nevner [ITU99]:

- Gatekeeper.
- Gateway.
- Terminal.
- Multipoint Control Unit.

3.3.5 Multipoint Processor

I en multipartykonferanse vil det være mange mediastrømmer som sendes fra de ulike deltagerene til de andre i konferansen. For å effektivisere oversendelsen av mediastrømmer kan det anvendes en *Multipoint Processor* (MP). Denne H.323 enheten har som oppgaver å *svitsje* og *mikse* mediastrømmer. Med svitsjing menes det i denne sammenheng å velge hvilken mediastrøm som skal være kilden for den utgående mediastrømmen. Som et eksempel på dette kan en tenke seg at MP velger audiostrømmen til den deltageren som snakker, og sender den til alle de andre deltagerene. Miksing av mediastrømmer går ut på å sette sammen flere strømmer til et mindre antall. Dette kan for eksempel være å sette sammen video strømmene fra fire deltagere til en videostrøm hvor hver av deltagerene vises som fire mindre videovinduer.

I likhet med MC er en MP ikke en egen fysisk enhet. De H.323 enhetene en MP kan være en del av er i [ITU99, figur 6] beskrevet som:

- Gatekeeper.
- Gateway.
- Multipoint Control Unit.

3.3.6 Multipoint Control Unit

En *Multipoint Control Unit* (MCU) er et H.323 endepunkt som yter støtte til H.323 konferanser hvor tre eller flere endepunkter er involvert.

I motsetning til MC og MP er en MCU et kallbart endepunkt som må bestå av en MC og null eller flere MPer. Det kan sies at en MCU er et samlebegrep og lokaliseringspunkt for MC og MP.

Arbeidsfordelingen mellom MC og MP er at MC tar seg av H.245 forhandlinger mellom alle terminalene for å finne felles egenskaper for audio-, data- og videoprosessering. I tillegg til dette kontrollerer MC konferanseressursene ved å bestemme om noen av audio- eller videostrømmene skal være i multicast. MP derimot, mikser, svitsjer og prosesserer audio-, video- og datastrømmer.

3.4 Eksempler på oppsett av H.323 samtaler

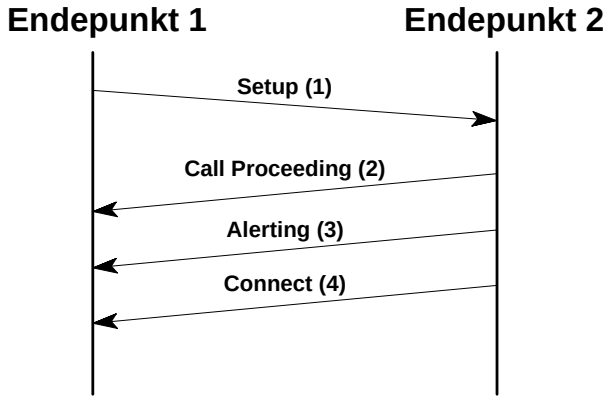
I denne delen blir det vist eksempler på oppsett av H.323 sesjoner. Eksemplene er vist med sekvensdiagrammer kombinert med beskrivende tekst. Eksemplene er hentet fra [ITU99], og er ment å gi en grunnleggende forståelse av gangen i H.323 anrop. Eksemplene tar for seg et *Basic Call* anrop. Det vil si et enkelt anrop med to deltakere; den første anroper den andre, som deretter svarer.

3.4.1 Basic Call anrop uten GK

I eksempelet som er vist i figur 3.4, eksisterer det ingen Gatekeeper i H.323 systemet, og de to endepunktene kommuniserer direkte med hverandre. Gangen i et Basic Call anrop vil være som følgende:

1. Endepunkt 1 sender en *Setup* til den *velkjente adressen*[4] til Endepunkt 2.
2. Endepunkt 2 varsler at anropet pågår med å sende *Call Proceeding* tilbake til Endepunkt 1.
3. Endepunkt 2 sender *Alerting* som en indikasjon på at det "ringer".
4. Oppsettsfasen av anropet avsluttes med at Endepunkt 2 sender *Connect* til Endepunkt 1. Denne meldingen inneholder transportadressen for kanalen som skal brukes til H.245 signalering.

Hvis endepunktene har benyttet seg av *Fast Start* opsjonen i H.323, er det nå klart til å begynne selve sesjonen med å utveksle data. Det kan være lyd, bilde eller overføring av data. Brukes ikke denne muligheten går endepunktene nå inn i en fase hvor det forhandles om hvilke media som skal anvendes.



Figur 3.4: Basic Call anrop uten GK

3.4.2 Basic Call anrop rutet gjennom GK

I konfigurasjonen i dette eksempelet eksisterer det en Gatekeeper som endepunktene allerede har registrert seg hos. I eksempelet er det valgt å la all kallsignalering gå via Gatekeeper.

Det bemerkes at figur 3.5 på side 33 viser at begge endepunktene må be Gatekeeper om tilgang til nettet (ARQ – Admission Request) før de kan begynne å signalere. Dette behøves ikke hvis Gatekeeper har gitt endepunktene tillatelse på forhånd (Pregranted ARQ).

[4]Med dette menes det at adressen er blitt gjort kjent på forhånd på en eller annen måte.

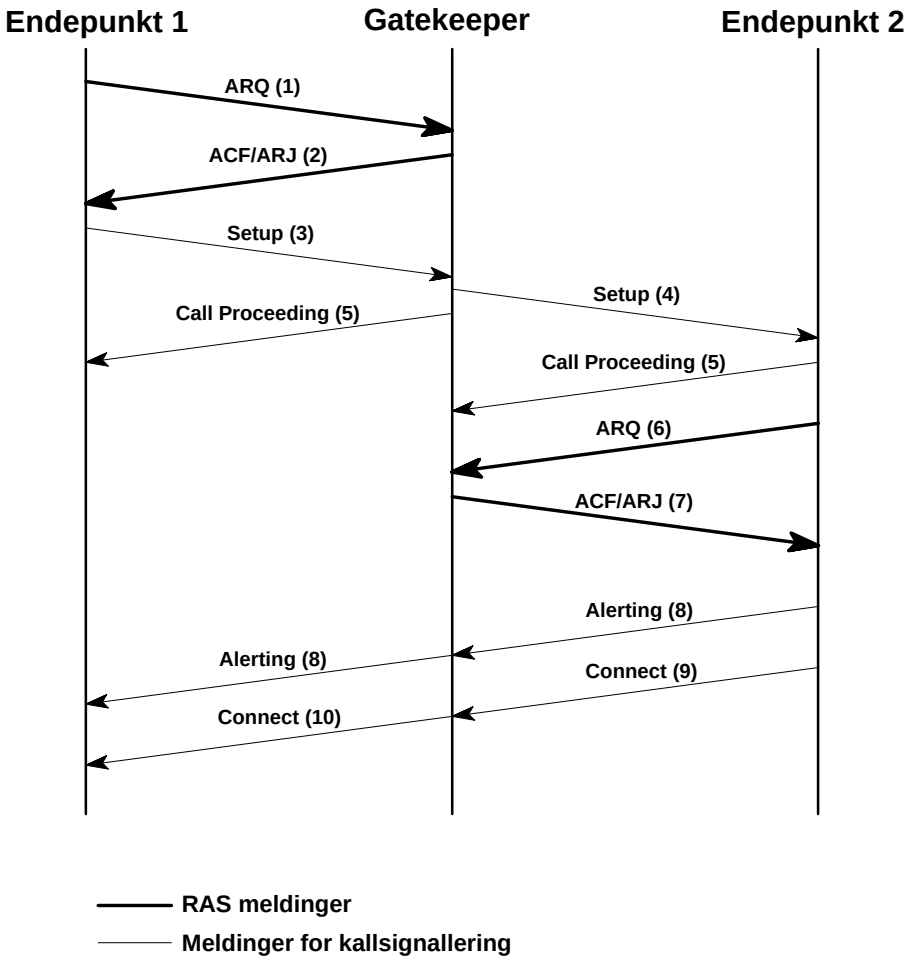
1. Endepunkt 1 ber Gatekeeper om nødvendig tilgang for et anrop til Endepunkt 2 med *ARQ*.
2. Gatekeeper returner et positivt svar med *ACF* (Admission Confered), eller eventuelt at avslag med *ARJ* (Admission Rejected). Hvis Gatekeeper svarer endepunktet med ACF, returneres det samtidig i ACF meldingen hvilken adresse endepunktet skal bruke for kallsignalering mot Gatekeeper.
3. Endepunkt 1 sender så *Setup* til Gatekeeper som
4. videresender meldingen til Endepunkt 2.
5. Gatekeeper indikerer til Endepunkt 1 at anropet er i gang ved å sende *Call Proceeding*. Tilsvarende indikasjon sendes fra Endepunkt 2 til Gatekeeper.
6. Hvis Endepunkt 2 vil motta anropet, ber det om tilgang fra Gatekeeper.
7. Endepunkt 2 får enten tilgang eller avslag med henholdsvis ACF og ARJ.
8. Endepunkt 2 varsler Gatekeeper om at det "ringer" med *Alerting*. Denne meldingen videresendes fra Gatekeeper til Endepunkt 1.
9. Anropsfasen avsluttes med at Endepunkt 2 sender *Connect* til Gatekeeper som
10. videresender meldingen til Endepunkt 1.

Selv om all signalering i dette tilfellet går via Gatekeeper, vil mediastrømmene gå fra endepunkt til endepunkt. Signalering og media er altså skilt. Et unntak fra dette er i multipartysesjoner hvor media mikses eller svitsjes ved hjelp av f.eks en MCU.

3.5 Oppsummering og konklusjon

H.323 er omfattende, noe som dette kapittelet skulle indikere. Årsaken til dette er et ønske om å detaljspesifisere standardene for å sikre interoperabilitet. Dette er også en av H.323s sterke sider på grunn av sin rigiditet i spesifikasjonen.

Omfanget av H.323 gjør det til en stor oppgave å implementere hele standarden, men at hele standarden implementeres på en gang vil sjelden være tilfelle. Siden H.323 fungerer som en paraply for andre standarder er grensesnittene mot de ulike standardene veldefinerte. Dette gjør at det finnes mange biblioteker på markedet som støtter de ulike delstandardene i H.323. Som et eksempel på dette kan nevnes ASN.1 kompilatorer for å kode og dekode det binære meldingsformatet. Implementasjoner av H.323 kan dermed i stor utstrekning lages ved hjelp av komponentbaserte prinsipper.



Figur 3.5: Basic Call anrop rutet gjennom GK

En annen motivasjon er å tilby et minste felles løsningsrom for mediaformater. Det kommer blant annet til uttrykk i at alle H.323 terminaler må som et minimum støtte G.711 audio. Dermed har man sikret at alle i det minste skal kunne sende og motta audio. Baksiden av medaljen er at det ikke er så enkelt å prøve ut en ny CODEC i H.323. En CODEC må i utgangspunktet være registrert som en ITU CODEC før den kan anvendes i et H.323 system. Dog finnes det muligheter for å sende formater som ikke er standardisert, men da må det i H.245 meldingen spesifiseres et standard format i tillegg.

En annen diskusjon er hvem dette er et problem for. I sluttbrukermarkedet vil det i de fleste tilfeller brukes CODEC'er som allerede er godt etablerte og har et solid fotfeste i markedet. Dette gjør at problemet stort sett gjør seg gjeldene i utviklingsmiljøer som har et behov for å teste ut nye CODEC'er og hos svært innovative brukere. Det må nevnes at utviklingsmiljøer er lukkede testmiljøer, og derfor er det ikke så kritisk om en her følger H.323

standarden.

H.323 som standard har utviklet seg svært mye fra versjon 1 som var beregnet for LAN, til versjon 3 som har generalisert dette begrepet til å gjelde pakkebaserte nett. Versjon 1 av H.323 hadde en god del mangler som gjorde protokollen lite egnet i WAN sammenhenger. Disse manglene og problemstillingene har blitt adressert og stort sett løst etter hvert som standarden har utviklet seg gjennom versjon 2 og 3. Det står fortsatt igjen problemer som må løses hvis H.323 skal kunne skalere til store sammenhenger. Dette er temaer som lokalisering ved hjelp av katalogtjenester, og bruker- og tjenestemobilitet.

En ting som kan bli et problem i H.323 er at den skal være bakoverkompatibel. Dette kan føre til utfordringer implementasjonsmessig hvis et versjon 1 system skal kunne samspille med et versjon 3 system. Løsningen kan kanskje være å kalle H.323 versjon 1 for en slags prematurversjon som det ikke stilles krav om bakoverkompabilitet til.

Til tross for momentene som er nevnt over må H.323 kalles et modent system, og standarden har rukket å få en stor utbredelse i markedet. For å nevne noen eksempler på dette baserer både Telenor Nextels Interfontjeneste og Microsofts NetMeeting seg på H.323. Om H.323 fortsatt kommer til å være markedsleder er avhengig av om protokollen klarer å skalere i stor nok grad, om det er lett å utvikle tjenester i protokollen og hvilken prispolitikk leverandører av H.323 utstyr legger seg på.

Kapittel 4

Session Initation Protocol

> Mens han plukket tornene ut av munnen, forklarte Tigergutt at tigergutter ikke likte tistler.
> «Hvorfor da ødelegge et feilfritt eksemplar?» spurte Tussi.
> «Men du sa ...» begynte Brumm, «du *sa* jo at Tigergutter likte alt unntagen honning og eikenøtter.»
> «Og tistler,» sa Tigergutt, som nå løp i ring rundt seg selv med tungen ut av munnen.
>
> — *A. A. Milne: Huset på Bjørnehjørnet*

DET finnes et utall aspekter vedrørende multimediakommunikasjon. Tjenestekvalitet, medieformat, signaleringsmetode, mediaforhandling og lokalisering for å nevne et lite utvalg. I motsetning til H.323, forsøker ikke *Session Initiation Protocol* (SIP) [HSSR99b] å definere alle aspekter rundt multimediakommunikasjon, men fokuserer kun på signalering. Dette gjør at SIP ikke blir så stor i omfang, og kan dokumenteres på rundt 150 sider.

Teksten og figurene i dette kapittelet er basert på tilsvarende figurer i [HSSR99b], [HSSR99a] og [deC99]. Materialet i disse referansene er oversatt og sammenstilt sammen med eget materiale.

4.1 Bakgrunn og motivasjon

SIP ble unnfanget på Columbia Univeristy og senere sendt inn som et *draft*[1] til IETF. SIP fikk status som *RFC*[2] mars 1999, og dette kan betraktes som en versjon 1.0 av protokollen.

Utgangspunktet for SIP er et annet enn H.323. SIP er først og fremst et resultat av at programvare beregnet for videokonferanser over Internet har

[1] Alle forslag til nye protokoller i IETF må sendes inn som et *draft* til gjennomsyn og kommentarer fra de som er interessert. Kan betraktes som et førsteutkast av protokollen.

[2] Når et forslag har vært minimum seks måneder som draft og det eksisterer to implementasjoner, kan forslaget oppgraderes til RFC.

eksistert i flere år, men med utilstrekkelig funksjonalitet. Denne funksjonaliteten har SIP utvidet med mulighet for å "ringe" personer. Det vil si at en ikke trenger å avtale på forhånd når man skal snakke sammen. I tillegg definerer SIP en mekanisme for å lokalisere samtalepartnere.

Et av hovedmålene har vært å lage en enkel protokoll som er lett å implementere. Med tanke på den utbredelsen World Wide Web har fått falt valget på en utvidelse av HTTP. Dette gjør at de som er kjent med HTTP raskt vil kjenne seg igjen i oppbyggingen av SIP meldinger.

Som en konsekvens av at et av målene er enkelhet, er det ikke i dagens versjon av SIP mekanismer som kan håndtere reservasjon av båndbredde i nettet. I utgangspunktet overlater SIP dette til andre protokoller i IETF familien som for eksempel RSVP. Til tross for dette er reservasjoner et tema i miljøet som arbeider med SIP, og det utarbeides mekanismer som kan utfylle en slik rolle. Et eksempel på på et slikt forslag er beskrevet i [MRM+99].

4.2 Oppbygning og virkemåte

Som i HTTP er meldingsformatet i SIP tekstbasert. Noe av argumentasjonen for dette er at feilsøking i protokollen blir enkelt. Det tapet som ligger i parsering av tekstmeldinger kontra en binær protokoll (f.eks. ASN.1) betraktes som uvesentlig.

Siden SIP ikke definerer enhver potensiell protokoll brukt i multimediakommunikasjon, må den kombineres med andre IETF protokoller for å få til en komplett løsning. Dette kan for eksempel være å bruke avspillings- og opptaksmulighetene som ligger i *Real Time Streaming Protocol* (RTSP), med den hensikt å lage en *Voicemail* løsning med SIP.

Siden en mulitimediakonferense kan bestå av parter med ulike egenskaper må SIP ha en måte å formidle hvilke medier og medieformater som skal benyttes for lyd, bilde og data. Til dette formålet anvender SIP *Session Description Protocol* (SDP) (se side 43).

SIP har i likhet med HTTP en klient-tjener arkitektur. Klienter kaller metoder på tjenere, og tjenerne svarer med en bekreftelse. I SIP er det tre kjernemetoder:

INVITE forespør en deltaker om å slutte seg til en samtale eller konferanse.

BYE terminerer en forbindelse mellom to brukere i en konferanse (som å legge på røret).

ACK bekrefter at en INVITE er mottatt.

I tillegg inkluderer SIP og følgende metoder:

OPTIONS brukes til valgfrie tjenester. Et eksempel på bruk kan være å sende en OPTIONS melding for å finne ut hva slags mediatyper som mottakeren støtter, uten sette opp et kall.

CANCEL brukes for å kansellere et spesifikt metodekall.

REGISTER registrerer en adresse til for eksempel en klient i en SIP tjener.

Figur 4.1 viser et eksempel på en INVITE melding sakset fra [HSSR99b]. Den nederste delen etter den blanke linjen, er SDP meldingen. De forskjellige feltnavnene i SDP meldingen er protokollversjon (`v`), eier og sesjonsid (`o`), sesjonsnavn (`s`), informasjon om forbindelsen (`c`) og til slutt mediatypen og transportadressen (`m`). For `m` feltet i dette tilfellet er det audio som skal anvendes, og det skal mottas på portnummer 3456 med overføringsformat RTP/AVP type 0.

```
INVITE sip:watson@boston.bell-tel.com SIP/2.0
Via: SIP/2.0/UDP kton.bell-tel.com
From: A. Bell <sip:a.g.bell@bell-tel.com>
To: T. Watson <sip:watson@bell-tel.com>
Call-ID: 3298420296@kton.bell-tel.com
CSeq: 1 INVITE
Subject: Mr. Watson, come here.
Content-Type: application/sdp
Content-Length: ...

v=0
o=bell 53655765 2353687637 IN IP4 128.3.4.5
s=Mr. Watson, come here.
c=IN IP4 kton.bell-tel.com
m=audio 3456 RTP/AVP 0
```

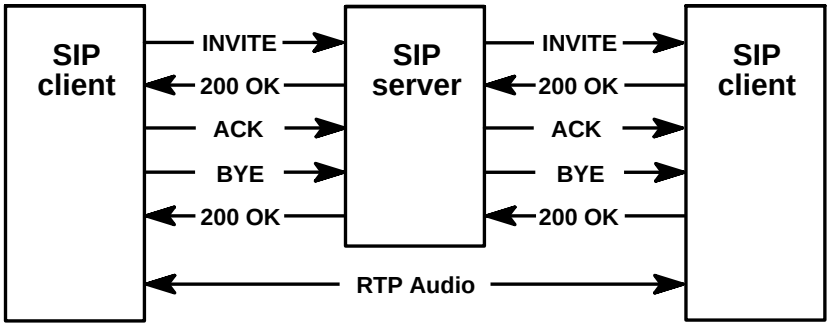
Figur 4.1: INVITE eksempel

En typisk meldingsflyt mellom to SIP klienter og en SIP server i *Proxy Mode* er illustrert i figur 4.2 på neste side. Hvert anrop og svar er bygd opp av *headers* og en mulig kropp. Hvis en SIP parser kommer over en uforståelig header, hoppes den over og fortsetter med det den kan tolke. Et unntak er hvis det ukjente elementet er en del av REQUIRE headeren, da må det returneres en feilmelding. Alle headere må kunne tolkes, ellers må det returneres med feilkode 420 (Bad Extension) sammen med en liste over opsjonene som ikke er forstått i Unsupported headeren.

I omtrent alle meldinger må CALL-ID headeren være med. Dette er en unik streng som assosierer meldingen til de partene (eller ressursene) som er med i kallet.

4.3 Bestanddeler og enheter

SIP deler i hovedsak inn enhetene i systemet i klienter eller tjenere. Disse kan være av følgende type:



Figur 4.2: Meldingsflyt SIP klient og server i Proxy Mode

- Klient som består av følgende:
  - User Agent Client (UAC)
  - User Agent Server (UAS)
- Tjenere som kan være av disse typene:
  - Registrar
  - Redirect Server
  - Proxy Server

Typene er ikke gjensidig utelukkende. En server kan f.eks både være en UAS og en Redirect Server.

En servertype som nevnes ofte i SIP sammenheng er *Location Server*. Denne serveren er ikke spesifisert i SIP, men betraktes som en tjeneste som SIP kan benytte seg av. En Location Server kan f.eks være en DNS eller en LDAP server.

4.3.1 User Agent Client

Dette er klientapplikasjonen som initierer SIP forespørsler. Eksempler på forspørsel kan være å gjøre et anrop til andre deltagere, eller registrere seg hos Registrar eller en Proxy server.

4.3.2 User Agent Server

Denne enheten er en del av klienten. Den har ansvaret for å kontakte brukeren når det kommer innkommende SIP forespørsler og returnere et svar på vegne av brukeren. Svaret som gis er at forespørselen enten er akseptert, avvist eller omdirigert.

4.3.3 Registrar

En Registrar er en server som aksepterer `REGISTER` anrop. Dette er forespørsler som gis når en SIP klient ønsker å meddele sin eksistens. En Registrar er typisk lokalisert sammen med en Proxy eller Redirect server. I tillegg kan en Registrar tilby tjenester for lokalisering av SIP klienter.

4.3.4 Redirect Server

En Redirect server er en server som aksepterer SIP forespørsler, relaterer adressene til null eller flere nye adresser og deretter returnerer disse adressene til klienten. En Redirect server initierer ikke egne SIP forespørsler til forskjell fra en Proxy server. Den aksepterer heller ikke anrop til forskjell fra en User Agent server.

4.3.5 Proxy Server

Proxy er en type server som står som en skillevegg mellom to SIP miljøer. En Proxy vil fungere både som en server og klient i den hensikt å gjøre SIP forespørsler på vegne av andre SIP klienter. Forespørsler blir prosessert internt eller sendt videre til andre servere. En proxy interpreterer og skriver om forespørsler før de videresendes. Det bemerkes ar forespørslene bare skrives om hvis det er nødvendig.

4.4 SIP eksempler

Eksemplene som er beskrevet nedenfor er hentet fra [HSSR99a], og tar for seg invitasjon av en part til en to-parts konferanse. I det ene tilfelle er mottakeren bak en Proxy Server, og det andre eksempelet skal illustrere lokalisering av motparten ved hjelp av en Redirect Server.

4.4.1 Anrop med Proxy Server

I dette eksempelet illustrert i figur 4.3 på neste side vises et anrop mot en part som befinner seg bak en “brannmur” i form av en Proxy Server.

Oppsettet av anropet vil ha følgende steg relatert til figur 4.3 på neste side.

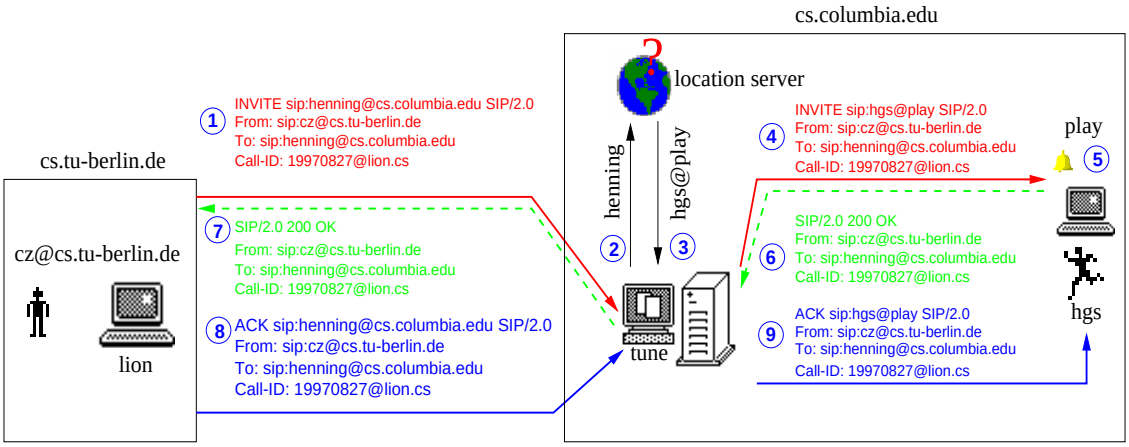
1. Proxy Serveren aksepterer den innkommende `INVITE`,
2. kontakter Location Server med alle delene av adressen,
3. og får tilbake en mer presis lokalisering av mottakeren.
4. Proxy Serveren sender så en `INVITE` til den nye adressen (eller adressene) som ble returnert av Location Serveren.
5. User Agent varsler brukeren (med f.eks. ringesignal),

6. og returnerer en bekreftelse til Proxy Server om at kallet er vellykket.

7. Proxy Serveren videresender denne bekreftelsen til den som anropte brukeren. Legg merke til at den "lokale" adressen er erstattet med den adressen anroperen brukte.

8. Mottakeren av bekreftelsen om at anropet var vellykket, sender en ACK tilbake til Proxy Serveren som en bekreftelse på dette.

9. Proxy Serveren mottar denne og videresender den til mottakeren.



Figur 4.3: Anrop med Proxy Server hentet fra [HSSR99a]

4.4.2 Anrop med omdirigering av mottaker

En bruker i SIP kan flytte seg mellom et antall forskjellige steder. For at brukeren skal kunne nås trengs det en tjeneste som kan sørge for at brukere kan lokaliseres på det stedet de for tiden måtte befinne seg.

Eksempelet vist i figur 4.4 på neste side, tar for seg et slikt scenario. Meldingsstrømmen vist på figuren vil være som følger:

1. Redirect serveren aksepterer innkommende `INVITE`,

2. kontakter Location Server med alle delene av adressen,

3. og får tilbake en mer presis lokalisering av mottakeren.

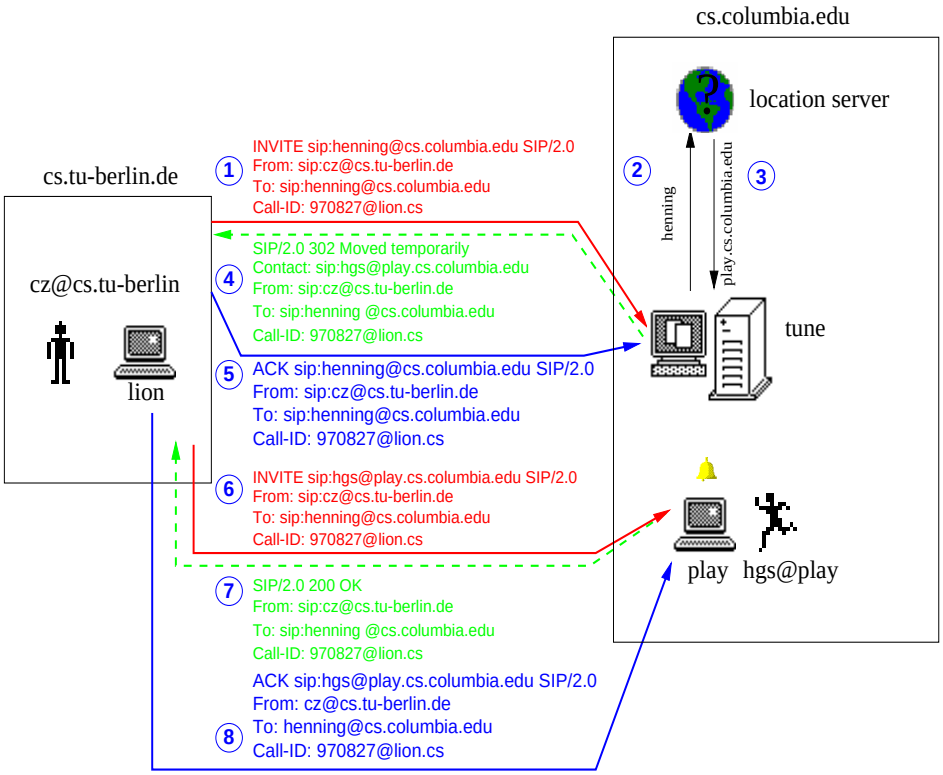
4. Isteden for at Redirect server kontakter den nye adressen selv, returneres den til anroperen.

5. Dette bekreftes med at anroperen sender en ACK til Redirect serveren.

6. Anroperen sender nå avgårde en ny `INVITE` med den nye adressen.

7. Mottakeren bekrefter at anropet har vært vellykket ved å sende bekreftelsen `200 OK` tilbake.

8. Meldingsstrømmen fullføres ved at anroperen sender en ACK til mottakeren.



Figur 4.4: Anrop med omdirigering av mottaker hentet fra [HSSR99a]

4.5 Oppsummering og evaluering

En av SIPs fordeler er at den er enkel når man sammenligner den med protokoller som H.323 (se kapittel 3). Siden SIP er tekstbasert og består av cirka 40 headere vil det være en rimelig enkel oppgave å lage et program for å parsere SIP meldinger.

En SIP parser skal i følge spesifikasjonen overse headere i meldingen den ikke har kjennskap til. Dette gjør at det er svært enkelt å legge til elementer i SIP eller lage nye versjoner, uten at det behøver å gi problemer med bakoverkompabilitet. Det kan være mulig at dette ikke er en tilstrekkelig fremgangsmåte for fremtiden. Årsaken til dette er at SIP, slik den eksisterer i dag, er i den første offisielle versjonen (jf. utgivelse av [HSSR99b]). Det betyr at man ikke har fått testet ut potensielle problemer ved håndtering av forskjellige SIP versjoner eksisterende side om side.

I [deC99, side 36] påpekes det at SIP arkitekturen er som et sett med legoklosser. Som et eksempel på dette skisseres et scenario hvor de bruker SIPs katalogegenskaper for å finne en bruker, som deretter kan omdirigere deg til en H.323 server for å gjøre selve anropet.

SDP (se kapittel 5) er den protokollen SIP har valgt for å formidle mediakapabilitet. SDP er i dagens utgave ikke beregnet for bruk i situasjoner hvor det er ønskelig å forhandle om media. Dette gjør at SIP har en begrensning i sammenhenger hvor utvalget av mediaformater er svært stort, og det er vanskelig å finne frem til en felles CODEC.

SIP servere har mulighet til å operere i to tilstander; *Statefull* og *Stateless*. Hvis serveren opererer i den siste tilstanden, kan serveren skalere veldig bra med hensyn på hvor mange kall som kan prosesseres. Dette kommer av at serveren ikke trenger å holde rede på tilstanden på de respektive kallene.

Imidlertid kan en SIP server (f.eks. en Proxy server) sannsynligvis ikke operere i en Stateless tilstand hele tiden. Dette kan være tilfeller hvor det er ønskelig med en mekanisme for å gjøre reservasjoner av for eksempel båndbredde. Da må serveren både overvåke at den respektive sesjonen får tildelt det den har bedt om, og samtidig passe på at den ikke overstiger den faktiske reservasjonen. For å overvåke slike ting må det være tilstander som må vedlikeholdes. Dermed krever dette at servere opererer i en Statefull tilstand.

Nye tjenester kan lett implementeres i SIP ved hjelp av REQUIRE headeren. Dette kan for eksempel være en tjeneste for fakturering av samtaler. En ting som må bemerkes er at denne typen tjenester vil være leverandørspesifikke og i mange tilfeller proprietære.

Som en konklusjon på dette kapittelet kan en si at SIP har et stort potensiale, men at det gjenstår å se implementasjoner og bruk i stor skala. Først da vil SIP vise hva den er god for.

Kapittel 5

Session Description Protocol

Så skrev Ugla . . . Og dette er hva han skrev:

GLATU LATU EREME IDAGEN
EDAGEN

Brumm så beundrende på Uglas bokstaver.
«Det står bare: 'Gratulerer med dagen',» sa Ugla.
«Men jammen er det langt nok,» sa Brumm imponert.
«Vel, det jeg sier er naturligvis mer: Gratulerer så hjertelig med dagen – og vennlig hilsen Brumm, og du har rett i at det trengs en god del blyant for å skrive slike ting.»
«Javel, ja,» sa Brumm.

— *A. A. Milne: Ole Brumm*

DET kan være vanskelig å gi meningsfulle beskrivelser hvis vokabularet er lite eller passer dårlig i den sammenhengen det brukes i. Dette kan lett bli tilfellet for protokoller som blir tatt i bruk i anvendelsesområder som de i utgangspunktet ikke var laget for. Slik er det for *Session Description Protocol* (SDP) [HJ98]. SDP ble i utgangspunktet laget for å beskrive multimedia *multicastsesjoner*[1] på MBONE, mens den idag brukes i like stor grad i multimedia *unicast*[2] sesjoner.

5.1 Bakgrunn og motivasjon

SDP er en tekstbasert protokoll som er ment for å beskrive multimediastrømmer i en multimediasesjon. Dette skal skje på en slik måte at en

[1] Data sendes fra *en* sender til *mange* mottakere.
[2] Data sendes fra *en* sender til *en* mottaker.

mottaker av en *Session Description* kan knytte seg til og delta i den aktuelle multimediasesjonen. Situasjoner hvor protokollen er tenkt brukt kan være annonsering av sesjoner, invitasjon til sesjoner, eller en annen form for initiering av en multimediasesjon.

SDP var opprinnelig basert på protokollen som ble brukt av *session directory* (sd) fra Lawrence Berkeley Labs, men har blitt utvidet og generalisert betydelig siden den gang. I dag er SDP en del av den protokollserien som arbeidsgruppen *Multiparty Multimedia Session Control* (MMUSIC) i IETF har ansvaret for.

5.2 Beskrivelse av SDP

SDP er *bare* et format for å beskrive multimediasesjoner, og inkluderer ingen transportprotokoll. Hensikten med SDP blir da å bruke den som en del av en annen protokoll. Dette kan være protokoller som *Session Announcement Protocol* (SAP), *Session Initiation Protocol* (SIP), *Real Time Streaming Protocol* (RTSP), etc.

En multimediasesjon i SDP er definert som et sett av multimediastrømmer som eksisterer over en viss tid. Mediastrømmene kan være mange-til-mange.

I en multicastsesjon tjener SDP to hensikter; den ene er å formidle sesjonens eksistens, og den andre er å gi tilstrekkelig informasjon til å kunne knytte seg til og delta i sesjonen.

Når det gjelder unicastsesjoner er det mest aktuelle bruksområdet å kunne knytte seg til og delta i sesjonen.

For å kunne realisere dette inkluderer SDP følgende informasjon:

- Navnet og hensikten med sesjonen.
- Hvilke tider sesjonen er aktiv.
- Media som skal anvendes i sesjonen.
- Hvordan media skal mottas (adresser, portnummer, format, . . .).

I [HJ98] nevnes det at i tillegg kan være ønskelig å vite hvilken båndbredden som skal brukes, og informasjon om personen som er ansvarlig for sesjonen.

Det er mange egenskaper ved forskjellige mediatyper som er nødvendig for å kunne behandle mediet på en riktig måte. Dette kan for eksempel være å velge riktig media CODEC, eller å vite på hvilken IP adresse og portnummer mediastrømmen skal mottas. Av slike egenskaper har SIP de følgende attributter inkludert:

- Type media (audio, video, etc).
- Transportprotokollen (RTP/UDP/IP, H.320, etc).

- Mediaformatet (H.261 video, MPEG video, G.711 audio, etc).

I tillegg er følgende støttet for IP multicastsesjoner:

- Multicast adresse for media.
- Portnummer for media.

Egenskapene under er myntet på unicastsesjoner:

- Adressen for media.
- Portnummer for kontaktadresse.

Semantikken til denne adressen og portnummeret er avhengig av hvilken media- og transportprotokoll som skal brukes. Som standard definerer SDP dette til adresse og portnummer hvor mottakeren sender og mottar data. Noen protokoller vil benytte dette til å etablerer en kontrollkanal for media (f.eks H.245).

Når det gjelder å annonsere hvilke mediatyper som skal anvendes i en sesjon har [HJ98] definert følgende typer:

- audio.
- video.
- application.
- data.
- control.

Forskjellen på "application" og "data" er at den første er en mediastrøm for f.eks. applikasjonsdeling, mens den andre beskriver en overføring av data (f.eks. filoverføring).

I [HJ98] understrekes det at denne listen kan utvides etterhvert som SDP brukes i nye sammenhenger. Dette kan enkelt gjøres ved å registrere den nye typen i *InterNet Assigned Numbers Authority* (IANA).

SDP er ikke beregnet for forhandling om mediakapabiliteter mellom parter i en multimediasesjon. Dette definerer [HJ98, side 2] til å være utenfor SDPs fokus.

5.2.1 Konklusjon SDP

SDP er har flere gode egenskaper. En av disse er at den er lett å utvide med nye mediatyper samtidig som den er ukomplisert.

Formatet til SDP er tekstbasert. Dette gjør at det blir enklere å sirkle inn problemet ved feilsituasjoner, samt at man ikke må ha en komplisert parser for å tolke hva som sendes. Noen vil argumentere for at en protokoll på et

binært format vil være mer effektiv, men dette er neppe relevant med de små datamengdene det vil være snakk om i en SDP melding.

En av de åpenbare manglene ved SDP er muligheten for å kunne utveksle beskrivelser av mediakapabiliteter som logiske uttrykk. Dette kan skape situasjoner med mange rundturer. Dette vil være tilfelle der hvor det er mange mediaformater å velge mellom, og endeutstyret har forskjellige preferanser med hensyn til ønsket mediaformat.

Som en konklusjon på dette kan det sies at SDP er en protokoll som er relativt lett å implementere, og er mest egnet i sammenhenger hvor det er få alternativer når det gjelder mediaformater. Det skal også fremheves at SDP vil være en svært anvendelig protokoll i situasjoner hvor nye mediaformater skal prøves ut, og det stilles krav om at disse skal annonseres i en sesjonsbeskrivelse.

Kapittel 6

SIP/H.323 gateway

> «Joda,» sa han til slutt, «det er et meget hyggelig hus, og dersom ens eget hus blåser ned, *må* en jo flytte til et annet sted, ikke sant, Nøff? Hva ville *du* gjøre dersom huset *ditt* blåste ned?»
> Før Nøff fikk tenk seg om, hadde Brumm svart for ham.
> «Han ville komme og bo sammen med meg,» sa Brumm. «Ville du ikke, Nøff?»
> Nøff klemte labben hans.
> «Takk skal du ha, Brumm!» sa han. «Det ville jeg svært gjerne.»
>
> — *A. A. Milne: Huset på Bjørnehjørnet*

Å lage en gateway mellom to protokoller handler mye om å kunne få protokollene til å eksistere harmonisk sammen under et tak. SIP og H.323 burde ha mange forutsetninger for dette siden funksjonaliteten grovt sett er svært lik. Dermed vil det være mulig å finne motsvarende funksjoner i SIP og H.323. Oppsett av samtaler er for eksempel rimelig like. Dette forutsetter riktignok visse betingelser. En av disse er at *fast-start* opsjonen i H.323 benyttes.

I dette kapittelet presenteres en modell av en gatewayløsning for konvertering mellom SIP og H.323. Modellen tar primært for seg problematikken rundt hvordan konvertering av signaleringen skal løses. Feilsituasjoner er et stort område som ikke vil bli berørt av modellen. Modellen forutsetter med andre ord en enkel og ideel verden. Allikevel vil dette være tilstrekkelig for å vise hvordan en gateway mellom H.323 og SIP vil fungere på det konseptuelle planet.

I dette arbeidet ble APIet *Access Toolkit* (ATK) fra Ericsson benyttet. En av årsakene til dette valget er at dette APIet allerede hadde H.323 protokollen implementert. SIP protokollen måtte derimot implementeres. En del av dette kapittelet vies til å beskrive dette APIet, og som en del av denne beskrivelsen presenteres det et klassediagram av ATK. En del av ATK som mangler i denne beskrivelsen er hvordan det underliggende *eventsystemet*

er bygd opp. Dette skyldes dels at det ikke var dokumentert, og dels at tiden rant ut på slutten slik at det ikke ble tid til å trace hvordan og hvor de ulike modulene ble kalt.

6.1 Access Toolkit

Access Toolkit (ATK) er et javabasert API som skal lette arbeidet med å lage endepunkter i et IP-telefonimiljø, og ATK er en hjørnesten i den gatewayløsningen som er vist senere i dette kapittelet. Dette gjør at designet av gatewayen i stor grad blir påvirket av de føringene ATK gir. Det er derfor viktig å presentere APIets grunnleggende oppbygning. Kilden til denne beskrivelsen er henholdsvis ATKs User Guide [Siv98], reverse-engineered kildekode og egne erfaringer med ATK.

Som nevnt i innledningen til kapittelet er ikke det underliggende eventsystemet beskrevet. I meget korte trekk kan det sies at det består av et sett med tråder[1] i ATK prosessen som overvåker hendelser i systemet. For å effektivisere API'et allokeres tråder fra en *Thread pool*. I Threadpoolen er et gitt antall tråder instansiert når ATK initialiseres. Dette gjør at man får en gevinst i form av spart tid fordi trådene er instansiert på forhånd. Årsaken til at man velger å bruke tråder kontra prosesser er at *kontekstskifter* er svært kostbart mellom prosesser. Med kontekstskifter menes å holde rede på tilstanden til den respektive prosess eller tråd, og å bytte på hvilken tråd eller prosess som for tiden eksekverer på prosessoren. I tillegg til dette er kommunikasjon mellom tråder i samme prosess svært enkel. Dette er et resultat av at trådene i en prosess deler *heap*. Dermed kan trådene nå hverandres offentlige variabler og metoder. Dette kan så utnyttes til å forenkle kommunikasjonen mellom trådene, men kan gi problemer med synkronsering.

6.1.1 Motivasjon

H.323 er ingen enkel materie å håndtere med sitt vell av underliggende protokoller. Kapittel 3 på side 21 skulle gi et godt inntrykk av dette. Dette er motivasjonen bak å lage et API som ATK. Altså å forenkle tilgangen til protokollen gjennom å abstrahere funksjonaliteten i H.323.

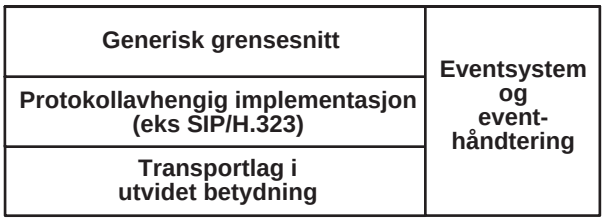
Paradigmet som ligger til grunn for ATK kan man si er å tilby en abstrahert protokollfunksjonalitet gjennom et modulært, objektorientert grensesnitt.

Målet er å kunne anvende den funksjonaliteten H.323 omfatter uten å måtte håndtere meldinger eller elementer som er protokollspesifikke. Samtidig tilbyr ATK tilgang til protokollimplementasjonen ved behov.

[1] En tråd kan best beskrives som en "lettvektsprosess". En tråd må eksekveres som en del av en prosess. En prosess kan altså ha mange tråder, og disse trådene har hver sin *stack*, men deler minneområde for variabler.

6.1.2 Oppbygning

Figur 6.1 viser hvordan ATK er lagvis delt opp. I bunnen er et lag som vi har valgt å kalle "Transportlag i utvidet betydning". Årsaken til dette er at det fungerer som et samlebegrep for de underliggende protokollene som H.323 og SIP er avhengig av. Eksempler på dette er Q.931, UDP, TCP, IP, RAS, RTP osv. Dette laget tilbyr i vid forstand det som selve IP-telefoni protokollen krever av underliggende infrastruktur.



Figur 6.1: ATK lagdeling

Over dette laget ligger selve implementasjonen av den respektive protokollen for IP-telefoni. I dette nivået implementeres den funksjonaliteten som tilbys av interfacene. Hvis det trengs protokollspesifikk funksjonalitet i tillegg til den generiske som interfacet tilbyr, vil det implementeres på dette nivået.

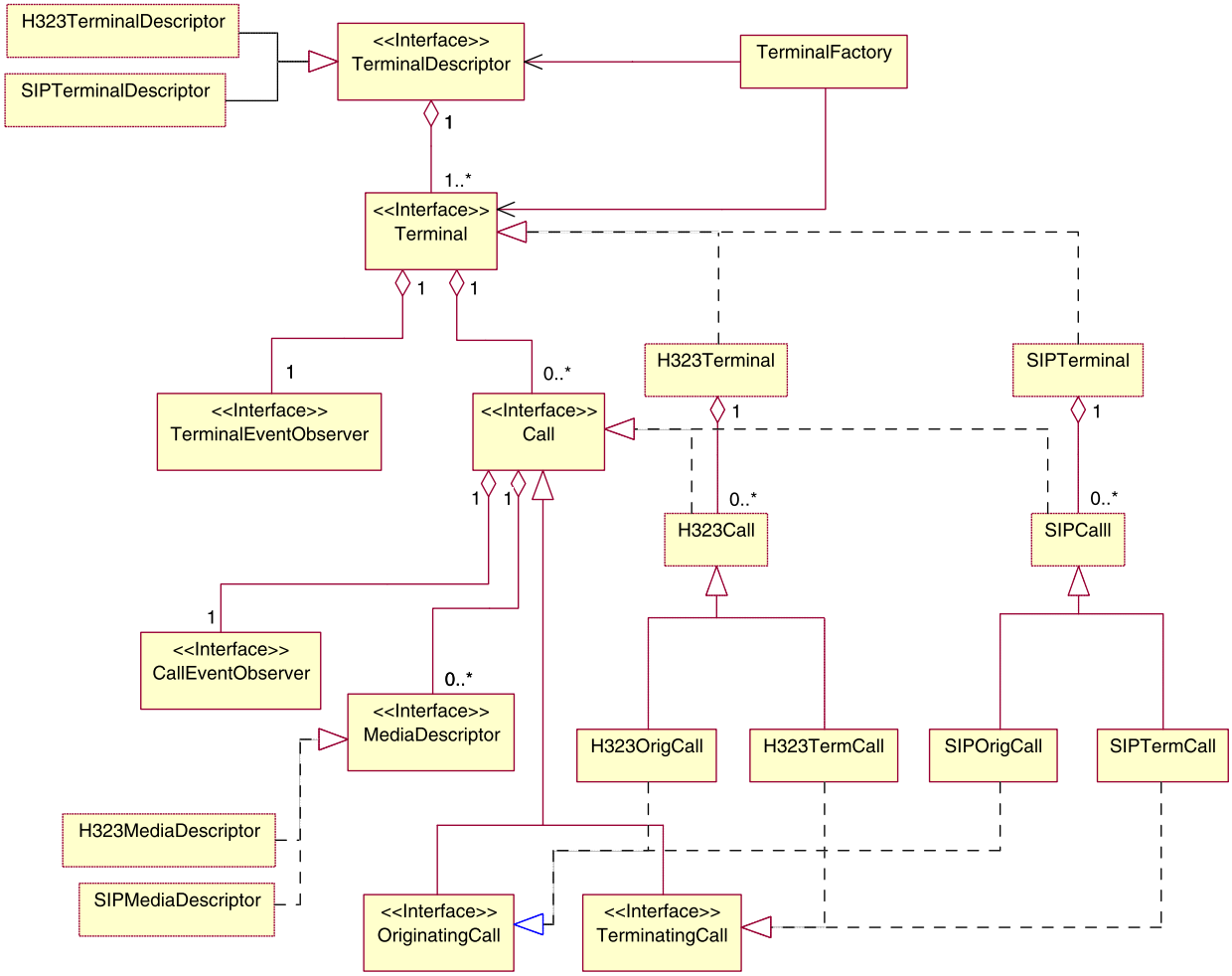
Øverst i figuren ligger det generiske grensesnittet som brukeren primært kommer i kontakt med. Dette grensesnittet uttrykker funksjonaliteten til H.323 og SIP som abstraherte metoder. Alle metoder på dette nivået *må* implementeres av det respektive protokollaget. Derfor er det meget viktig at de spesifiserte funksjonene ikke er spesifikke for den enkelte protokollen.

På siden av disse tre lagene ligger et eventsystem. Dette er i større eller mindre grad en del av lagene beskrevet over. Eventsystemet realiserer et asynkront meldingssystem. Med det menes det at hendelser ikke behøver å oppstå i noen spesiell rekkefølge. Eksempler på slike hendelser kan være oppsett av samtale, innkommende samtale og avslutt samtale. Eventsystemet vil sørge for å varsle hendelser i systemet ved å kalle gitte *Callback prosedyrer*[2]. Et eksempel på en slik prosedyre er `incommingCall`. Denne prosedyren er så implementert i protokollaget slik at reaksjonene på hendelsen er i henhold til hvordan det er beskrevet i SIP eller H.323.

[2]Prosedyrer som er definert av ATK, men implementeres unikt for hver protokoll.

6.1.3 Design og virkemåte

I det designet som er valgt for ATK kommer det fram at det er lagt vekt på modularitet for å tilby brukeren et abstrakt grensesnitt mot protokollene. Klassediagrammet i figur 6.2 viser dette. Det viser en grunnstruktur i form av interfacer, og hvordan sammenhengen er mellom interfacet og de respektive protokollklassene.



Figur 6.2: ATK klassediagram

Tanken bak en slik designløsning er at man kun skal angi en gang hvilken protokolltype som skal anvendes. Dette gjøres ved hjelp av klassen *TerminalFactory*. Denne klassen har en sentral rolle når det gjelder å instansiere riktig protokollimplementasjon. Spesifiseringen av ønsket protokoll gjøres når man lager en *TerminalDescriptor*. Dette er en klasse som inneholder egenskaper som er felles for en viss type terminaler. Dette kan være informasjon om terminaltype, leverandør av terminaltypen og hva som er fo-

retrukket audio og video CODEC. Et kall for å lage en TerminalDescriptor for H.323 vil typisk se slik ut:

```
TerminalDescriptor td =
      TermnalFactory.createTerminalDescriptor(ImplType.H323,
                                              TerminalType.TERMINAL,
                                              vendor);
```

TerminalFactory vil så returnere et objekt av type H.323 TerminalDescriptor. Dette objektet blir så brukt som en mal for å skape H.323 terminaler:

```
Terminal t = TerminalFactory.createTerminal(td, alias);
```

Objektet td, som er en TerminalDescriptor, sendes med som parameter i dette kallet sammen med et aliasnavn for terminalen. TerminalFactory returnerer nå et Terminalobjekt med H.323 implementasjon på grunnlag av hvilken type TerminalDecriptor som ble sendt med `createTerminal` funksjonen. I dette tilfellet ble det sendt med en TerminalDescriptor av type H.323, og dermed ble det skapt et H323 Terminalobjekt.

Motivasjonen for å bruke *Factorys* for å skape objekter er primært å garantere at riktig objekttyper blir instansiert. I dette tilfellet dreier dette seg om hvilken protokolltype objektet relaterer seg til.

Ved hjelp av TerminalFactory har nå brukeren fått tilgang til et Terminalobjekt med valgt protokollimplementasjon. Alle metoder som kalles på Terminalobjektet kalles nå i realiteten på den funksjonaliteten som er en del av den implementerte protokollstakken. I dag begrenser dette seg til H.323 og SIP, men konseptuelt sett er det ingen forskjell for andre telefoniprotokoller. En forutsetning er at protokollens funksjonalitet kan abstraheres til de samme metodene som ATKs interface tilbyr.

Terminalobjektet tilbyr i følge [Siv98] funksjonalitet for å oppdage og registrere seg hos tilgjengelige *Domain Controllers*. I tillegg inneholder Terminalobjektet metoder for å initiere utgående samtaler.

Begrepet Domain Controller fortjener en ytterligere forklaring. I denne betydningen menes det en enhet som har en sentral kontrollfunksjonalitet i domenet Terminalobjektet forholder seg til. I en H.323 sone vil denne enheten være en Gatekeeper. Tilsvarende rolle i et SIP domene vil ivaretas av en SIP server. Dette kan være en SIP Registrar eller SIP Proxy. Det må understrekes at en Domain Controller vil ha vidt forskjellig funksjonalitet i et H.323 og SIP miljø.

Funksjonaliteten som Terminal tilbyr kan i korte trekk summeres opp til å være bekjentgjøring av sin eksistens for omverdenen, og å initiere utgående kall.

En samtale i ATK er modellert i henhold til *Half Call modellen*. Dette innebærer at et kall består av to deler; *Originating Call* som har initiert kallet, og *Terminating Call* som mottar kallet. Hver av disse halvdelene kalles for

en *Call Leg*. Denne modellen kommer klart fram av figur 6.2 på side 50 hvor klassen *Call* er generalisert i klassene *OriginatingCall* og *TerminatingCall*.

Superklassen Call har funksjonalitet som er felles for subklassene. Vel å merke er dette funksjonalitet som går utover selve oppsettet av eller mottakingen av et anrop. Det er med andre ord en forutsetning at kallet allerede eksisterer. Slik sett dreier funksjonaliteten seg om det å administrere selve kallet. Nærmere spesifisert er dette:

- Å avslutte kall.
- Å signalere tilstanden til kallet.
- Å administrere hva slags media som skal anvendes i kallet.

Callklassen vil aldri instansieres, men vil tjene som en basisklasse med funksjonalitet som arves ned i de spesialiserte klassene OriginatingCall og TerminatingCall.

OriginatingCall instansieres ved å utføre følgende kall på et Terminalobjekt t:

```
OriginatingCall origCall  = t.createCall(destination, bandwith);
```

Ved å instansiere OriginatingCall objektet gjennom en metode i Terminal sikres det at objektet blir for riktig protokolltype. Dette er en konsekvens av diskusjonen over hvordan Terminal objektet blir skapt fra et Factory.

Metodene i OriginatingCall er de følgende:

```
ResultValue placeCall()

void setMediaWaitForConnect(boolean wait)
```

Metoden `placeCall` forsøker å etablere et forbindelse med den mottakeren som ble oppgitt når objektet ble instansiert. Returverdien fra metoden gir tilbakemelding hvordan dette gikk.

Den andre metoden kan diskuteres om virkelig hører hjemme i et generalisert interface. Dette er en funksjon som er myntet på *Fast Start* opsjonen i H.323. Hvis denne funksjonen kalles med **false** som parameter vil det ikke bli utvekslet mediakapabiliteter før selve forbindelsen er satt. Først da vil det være en forhandlingsfase hvor endepunktene blir enig om mediaformatet. Spørsmålet blir da om denne metoden kan brukes i andre protokollimplementasjoner enn H.323. Med tanke på hvordan en SIP INVITE fungerer vil det alltid sendes med en mediabeskrivelse. Derfor har i utgangspunktet ikke denne funksjonen noen stor nytteverdi i en SIP implementasjon. Den kan imidlertid forsvares hvis man tenker å bruke tjenester som ikke anvender mediastrømmer. Et eksempel på en slik tjeneste er Instant Messaging [DRS99]. Dette er en tjeneste som har som mål å oversende en melding uten å forvente svar. Hvis dette dreier seg om å overlevere en enkel

tekstmelding, kan dette gjøres som en del av selve SIP meldingen. I tillegg til dette kan en fremtidig reservasjonsmekanisme i SIP ha behov for å sende meldinger uten mediakapabiliteter. Disse to momentene kan forsvare å ikke oversende mediabeskrivelse med en INVITE i SIP.

I motsetning til OriginatingCall instansieres objekter av klassen TerminatingCall implisitt. Dette gjøres gjennom ATKs eventsystem som oppdager et innkommende anrop. ATK vil så skape et TerminatingCall objekt og varsle TerminalEventObserver gjennom å kalle metoden `incommmingCall` i den respektive implementasjonen.

Disse metodene er spesifiert i TerminatingCall:

```
ResultValue proceeding()

ResultValue alerting()

ResultValue connect()

boolean willMediaWaitForConnect()
```

Den første metoden, `proceeding`, kalles etter at et innkommende anrop er mottatt og akseptert. Metoden skal varsle den andre parten om at kallet prosesseres. Metoden `alerting` kalles etter at brukeren er "varslet" med ringetone. Til slutt i anropet kalles `connect` for å indikere at nå er det klart for å begynne selve samtalen. Disse metodene er altså ment for å gi en tilbakemelding til *anroperen* om kallforløpet og hvilken tilstand kallet er i.

Når det gjelder den siste metoden, `willMediaWaitForConnect`, er den som `setMediaWaitForConnect` motivert av Fast Start opsjonen i H.323. Derfor henvises det til diskusjonen over når det gjelder denne funksjonen.

Den siste klassen i ATK som er essensiell for å få en forståelse av virkemåten er *CallEventObserver*. Som det går ut fra figur 6.2 på side 50 har denne klassen en en-til-en relasjon til Call klassen og har essensielle oppgaver knyttet til det enkelte anrop. Oppgavene går ut på å videreformidle hendelser og tilstandsendringer som er sendt til Callobjektet til brukeren. En ting som er verdt å merke seg er at dette interfacet må implementeres av den som bruker ATK. På denne måten kan det være opp til API brukeren å bestemme hva som skal skje ved de enkelte hendelsene. Et eksempel på dette kan være hvordan brukeren skal varsles om innkommende anrop. Et alternativ kan være å gi en tradisjonell ringetone, et annet kan være å ha et blinkende icon på skjermen for å varsle den innkommende samtalen.

De metodene som er spesifisert i dette interfacet og som må implementeres av ATK brukeren er disse:

```
void callProceeding()

void callAlerted()

void callConnected()

void callClosed()
```

```
void callCancelled(Reason reason)

void callStatus(CallStatus status)

void callProgress(ProgressIndicator pi)
```

Disse metodene er tilstrekkelig for å gi sluttbrukeren informasjon om samtalens forløp. Normaltilstanden er dekket opp og feilsituasjoner kan signaleres til brukeren på et grovkornet nivå. Metoden `callCancelled` er tiltenkt unormale situasjoner i anropsforløpet. Metoden gir tilbakemelding om feilsituasjonen på et relativt lavt detaljnivå, men burde være tilstrekkelig for å gi sluttbrukeren relevant informasjon.

6.1.4 Eksempel: H.323 anrop i ATK

Dette kodeeksempelet er ment for å gi en grov forståelse av hvordan et H.323 anrop vil bli foretatt ved hjelp av ATK. Kodeeksempelet er ikke den komplette kildekoden, men fragmenter for å gi en konseptuell forståelse av hvilke metoder som involveres i et anrop. Dette innebærer at mindre viktigere elementer som deklarering av variabler og lignende, ikke er tatt med i koden som er vist. Dette kodeeksemplet er sakset fra [Siv98], og er gjengitt med tillatelse fra Ericsson AS.

Det første som må gjøres er å bygge opp en TerminalDecriptor. Denne forsynes med data om protokolltype, terminaltype, terminalleverandøren og standard domenekontroller (f.eks en Gatekeeper).

Kodefragmentet under viser hvordan denne informasjonen bygges opp og sendes som parametere til `createTerminalDescriptor`. Kallet på denne metoden resulterer i at det instansieres en TerminalDescriptor for en H.323 terminal. I tillegg er denne TerminalDescriptoren utstyrt med informasjon om terminalleverandøren og hvilken Gatekeeper som er foretrukket som standard.

```
2  //We must now build a terminal descriptor for the terminals we
   //are going to use.
4
   TerminalVendor vendor = new TerminalVendor("ATKXMPL", "VerXX", 0,0,0);
6
   // This will be the default domain controller address
8  TransportAddress dfltdcnaddr = new TransportAddress(DCN_HOSTNAME,1719);

10 TerminalDescriptor td =
       TerminalFactory.createTerminalDescriptor(ImplType.H323,
12                                               TerminalType.TERMINAL,
                                                 vendor,
14                                               dfltdcnaddr);
```

Neste steg er å skape et Terminalobjekt ved å bruke en TerminalDescriptor som mal. Legg merke til at det ikke er nødvendig å sende med hvilken protokolltype som skal anvendes. Denne informasjonen finner TerminalFactory i TerminalDescriptor objektet. I tillegg forsynes `createTerminal` med et aliasnavn på terminalen og et TerminalEventObserver objekt.

```
20 // we use the terminal descriptor when we create the Terminal.

22 // Alias for this terminal
   Alias alias = {new Alias("54321", AliasType.E164)};
24

26 // We add ourselfs as an TerminalEventobserver in this example
   t = TerminalFactory.createTerminal(td, alias, this);
```

I dette eksemplet er terminalen avhengig av å registrere seg hos en Gatekeeper. Dette gjøres ved å kalle `register` på Terminal objektet t. Resultatet av dette metodekallet blir så testet, og meddelt omverdenen.

```
   // we'll now let the terminal register itself
30
   ResultValue r = t.register();
32
   if ( r == ResultValue.SUCCESS )
34 System.out.println("Terminal registered successfully.");
   else
36     throw new Exception("Terminal did NOT register successfully: " + r);
```

Det neste kodefragmentet tar for seg selve anropet. Metoden `createCall` må forsynes med et alias på mottakeren. Dette aliaset bygges opp først, med en mottakeradresse og adressetype. Denne mottakeradressen vil sendes til Gatekeeper som igjen relaterer den til den virkelige adressen. De andre parameterene til `createCall` er ønsket båndbredde og et callEventObserver.

```
38 // now we can start making a call

40 // Alias we want to call
   Alias dstalias = new Alias(dst, AliasType.E164);
42

44 System.out.println("Trying to setup call to " + dstalias + " ... ");

46 // OriginatingCall origCall is class member

48 origCall = t.createCall(dstalias, 64, this);
```

6.2 Modell av SIP/H.323 gateway

Denne delen vil presentere den modellen av en SIP/H.323 gateway som vi har kommet frem til. Motivasjon og forutsetninger for modellen vil bli gått gjennom under i detalj. Dette kan grovt sett summeres opp til å være; *å lage en gateway ved å emulere to endepunkter og abstrahere protokollspesifikk funksjonalitet*.

Videre vil kapittelet ta for seg konkrete brukstilfeller for gatewayen. Disse brukstilfellene beskrives detaljert gjennom sekvensdiagrammer og supplerende pseudokode.

6.2.1 Motivasjon

Ved studier av VoIP protokollene SIP og H.323 er det slående hvor lik funksjonalitet de har på et overordnet nivå. Protokollene er grovt sett laget for å gjøre de sammen oppgavene; å sette opp en samtale mellom A og B med et eller flere mediaformater. Dette førte til ideen om at protokollenes funksjonalitet kunne abstraheres til et felles sett av metoder. Som et hjelpemiddel i dette arbeidet ble Ericssons *Access ToolKit* (ATK) anvendt. Dette APIet var tuftet på noe av den samme grunnleggende tankegangen som vi ville anvende i en gatewayløsning. ATK hadde på den andre siden aldri vært tiltenkt å brukes i gatewayløsninger, men for å lette realiseringen av endepunkter. Dette førte til designbeslutningen om å la gatewayen opptre som et endepunkt for henholdsvis SIP og H.323 miljøene.

Disse ideene og føringene er det som ligger til grunn for modellen som blir presentert nedenfor.

6.2.2 Forutsetninger

I modellen ligger det en rekke forutsetninger. Noen av disse blir implisitte, mens andre er eksplisitte. En av forutsetningene som skaper implisitte forutsetninger er at modellen i utgangspunktet er designet med tanken om at den skal implementeres i Java. Årsaken til dette ligger mest i at ATK er implementert i Java. Dette sammen med en del erfaring med Javaprogrammering gjorde at Java ble valgt som programmeringsspråk.

ATK er nevnt over som en forutsetning for modellen. Grensesnittet det tilbyr påvirker som sådan hvordan designet av gatewayen ser ut.

Andre valg er at vi for enkelthets skyld forutsatte at *Fast Start* opsjonen i H.323 ble brukt. Grunnen til dette er at dette forenkler samspillet med SIP en god del. Hvilket media som skal anvendes vil da signaliseres ved anrop. Dette passer godt med hvordan SIPs INVITE melding er bygd opp. Valget med å bruke Fast Start resulterer i at gatewayen ikke kan fungere med eldre versjoner av H.323. Denne muligheten kom først med fra versjon 2 av H.323.

Modellen er først og fremst ment for å beskrive en konseptuell løsning, og vil dermed kutte noen hjørner. En av disse tingene er at den ikke tar høyde

for noen translasjon av mediaformat. Det vil med andre ord si at gatewayen tar for seg *kun* signalering.

Feilsituasjoner er et annet tema som er stort og viktig. Dette er heller ikke en del av det konseptet vi vil vise. Dermed overlates dette til fremtidige utvidelser av modellen.

Et annet i moment er at sekvensdiagrammene i modellen viser at signaleringen går via en H.323 Gatekeeper. Dette påvirker rekkefølgen i sekvensdiagrammet noe, men vil ikke påvirke modellen på utover dette.

Kapittelet fortsetter neste side ...

6.2.3 Modellens brukstilfeller (Use Cases)

De Use Case (brukstilfeller) som modellen legger til grunn er vist på figur 6.3 på neste side. Disse avspeiler at modellen i første rekke er tiltenkt standard anrop fra enten en SIP eller H.323 terminal. Det enkelte Use Case blir presentert i detalj senere i kapittelet i form av sekvensdiagrammer. Dette omfatter følgende Use Case:

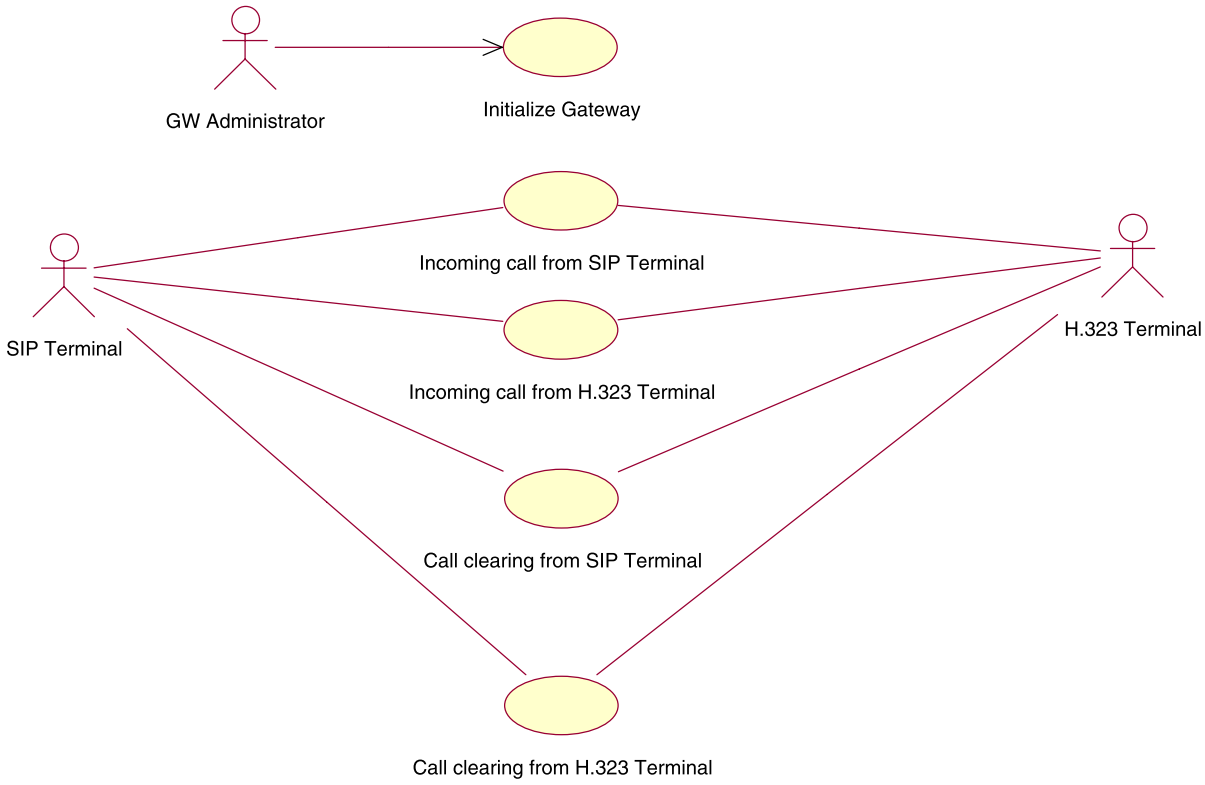
- Initialize Gateway.
- Incomming Call from SIP Terminal.
- Incomming Call from H.323 Terminal.
- Call Clearing from SIP Terminal.
- Call Clearing from H.323 Terminal.

Det første Use Caset tar for seg å bringe gatewayen i en tilstand hvor det er mulig å ta i mot enten et SIP eller H.323 anrop. En tenkt administrator av gatewayen antas å sette i gang denne prosessen. Denne administratoren kan like godt være et annet program som et menneske.

De to neste Use Casene tar for seg å håndtere et innkommende anrop fra henholdsvis en SIP eller H.323 Terminal. Dette inkluderer nødvendig interaksjon for å bringe signalering over til det respektive endepunktet på andre siden av gatewayen.

De to siste punktene på listen over tar for seg terminering av anropet. De tilfellene som det er sett på er at samtalen avsluttes fra SIP terminalen, og at den avsluttes fra H.323 terminalen.

Terminering av anrop kan skje i andre tilfeller enn de som er beskrevet over. Det kan være at selve gatewayen velger å ta ned samtalen eller at det oppstår en feilsituasjon som ender med at anropet blir terminert. I denne modellen blir ikke disse tilfellene berørt, og er momenter som bør ses på i fremtidige versjoner.



Figur 6.3: Aktuelle Use Case SIP/H.323 gateway

6.2.4 Klassediagram

Gatewayens klassediagram som er vist i figur 6.4 på neste side er bygget opp med klassen *Gateway* som en sentral komponent. ATK APIets klassehierarki former i tillegg mye av det underliggende designet. I denne modellen er det utelatt detaljer i ATKs klassehierarki for å bevare oversikten. Det komplette klassehierarkiet for ATK er allerede vist i figur 6.2 på side 50.

Den overordnede enheten i modellen er Gatewayklassen. Denne klassen har kontroll på tilstandene i hvert "terminalben". Gatewayklassen har også ansvaret for å foreta nødvendige metodekall som respons på en tilstandsendring. Et eksempel på dette er å sette opp et H.323 anrop som resultat på at det kommer et innkommende SIP anrop til gatewayen. Selve Gatewayklassen fungerer med andre ord som en bro mellom de to terminalene generert av ATK APIet.

For å kunne utføre denne oppgaven må Gatewayklassen ha følgende metoder:

```
initialize()
displayResults()
incommingSIPCall()
proceedingH323Call()
alertingH323Call()
connectedH323Call()
closedSIPCall()
closedH323Call()
incommingSIPCall()
proceedingSIPCall()
alertingSIPCall()
connectedSIPCall()
Gateway()
```

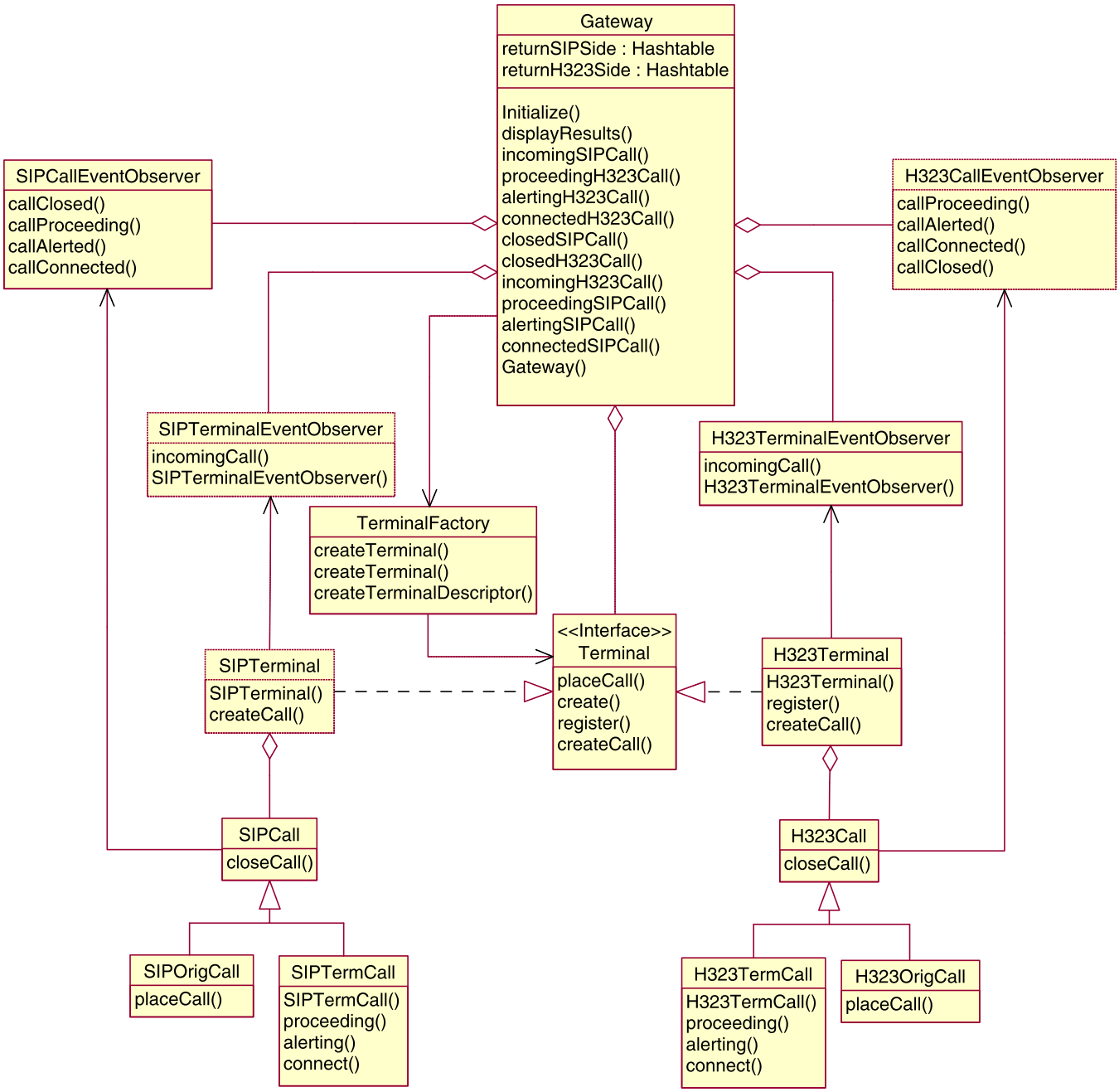
Noen av disse metodene er valgt å ha et navn som er unikt for hendelsen som skjer i den respektive protokollen. Årsaken til dette er at hvilken respons den enkelte hendelse skal resultere i kan være forskjellig for hver protokoll. ATK gir en mulighet til å aksessere den protokollspesifikke delen av implementasjonen. Dermed kan slike metodekall legges inn i metodene i Gatewayklassen.

Metodene over beskrives ytterligere i sekvensdiagrammene hvor de blir anvendt.

I tillegg til metodene står to attributter i en særstilling:

```
returnSIPSide
returnH323Side
```

Disse attributtene tar vare på assosiasjonen mellom SIP siden av gatewayen og H.323 siden. Attributtene er en Java Hashtable, og returner det motsvarende callobjektet når det gis et callobjekt som parameter.



Figur 6.4: Klassediagram SIP/H.323 gateway

6.2.5 Initiering

Dette brukstilfellet som er vist i figur 6.5 på neste side har som mål å få gatewayen i tilstand hvor den er klar til å ta imot anrop enten fra en SIP eller H.323 terminal.

Det første som skjer etter at Gatewayobjektet er instansiert, er at en gatewayadministrator setter igang hele prosessen ved å kalle metoden `initialize()`. Dette vil sette igang en serie av metodekall.

Gatewayadministratoren kan like godt være et annet program som en person. Dermed kan SIP/H.323 gatewayen være en del av et større system som anvender gatewayen som en av mange underliggende komponenter.

Første del av metodekallene er det som trengs for å instansiere en SIP terminal. Dette gjøres som beskrevet i avsnitt 6.1.3 på side 50. Altså først ved hjelp av *TerminalFactory* lage en *TerminalDescriptor*, for så sende denne som parameter til TerminalFactory metoden `createTerminal` som returnerer en *SIPTerminal*. SIP siden av gatewayen er nå rede til å ta imot anrop fra SIP terminaler på utsiden av gatewayen.

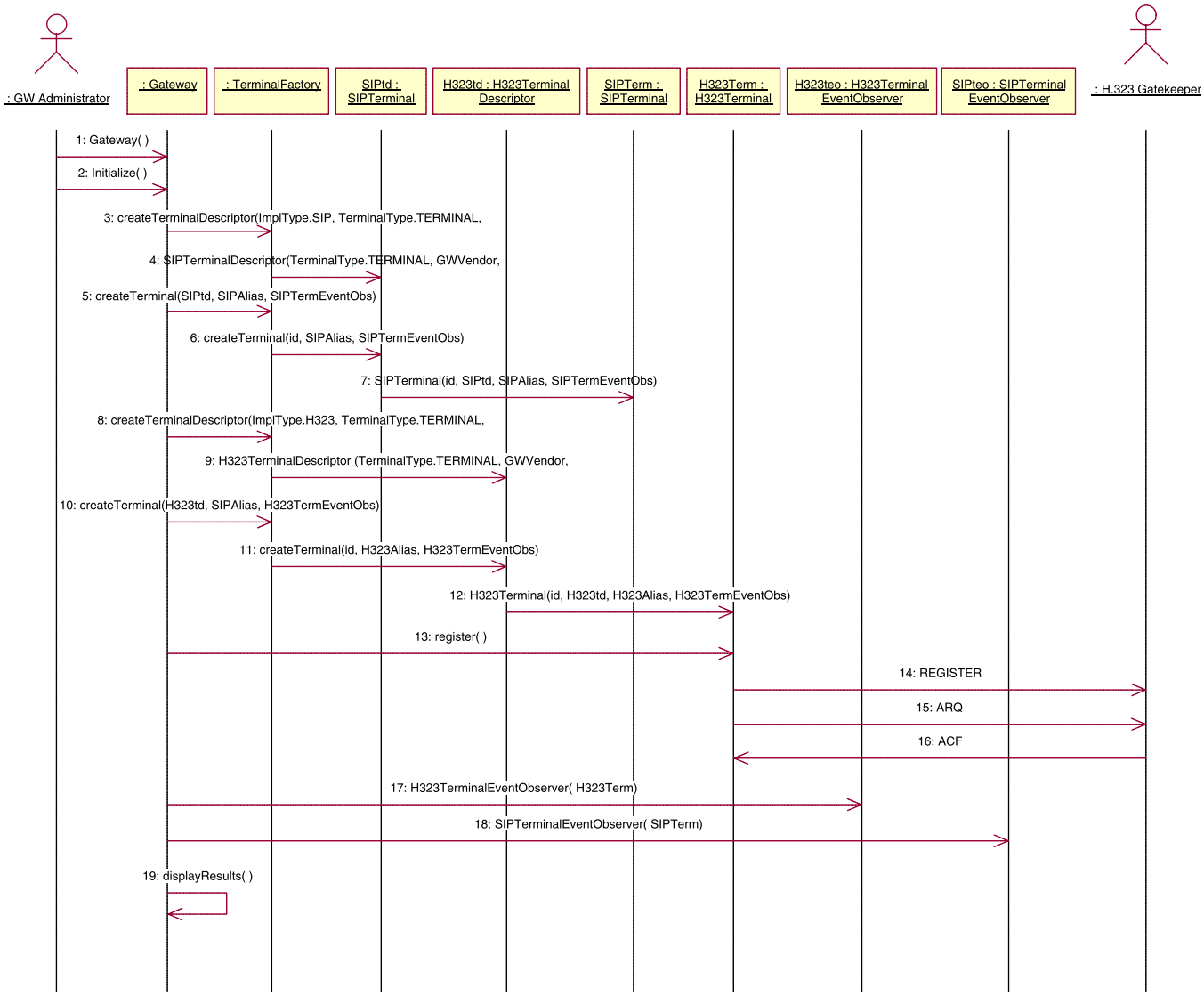
Den andre delen av sekvensdiagrammet tar for seg prosedyren for å skape en H.323 terminal og sette den istand til å ta i mot H.323 anrop. Dette gjøres helt likt som med SIP terminalen bortsett fra en ting; H.323 terminalen må registreres hos en Gatekeeper. Dette gjøres ved metodekallet `register()` på H323Terminalobjektet. Denne metoden utfører nødvendig RAS signalering for å registrer terminalen. Metoden ber samtidig om *Pregranted Admission* for å slippe å be om ressurser når et anrop skal settes opp mot H.323 miljøet.

Det bemerkes at det ikke hadde vært nødvendig å registrere H.323 terminalen hvis det ikke hadde vært en Gatekeeper involvert i dette scenarioet. Årsaken til at en Gatekeeper er med i bildet er dels at man ønsker å benytte seg av tilleggsfunksjonaliteten dette gir, dels at Ericssons ATK i utgangspunktet krever at en Gatekeeper anvendes.

`register()` metoden er en del av interfacet som ATK tilbyr, og må dermed implementeres i de protokollspesifikke delene. Dette gjør at det kan stilles spørsmålstegn ved hvordan denne metoden skal anvendes i SIP. SIP har ingen Gatekeeper, og dermed ikke samme behov for denne metoden. Derimot har SIP flere servertyper som kan tenkes være involvert i en registreringsprosedyre. Disse serverene er *Proxyserver* og *Registrar*. Proxyserveren har ikke behov for å vite noe om SIP klientene før de anvender den. Dermed har den ikke behov for noen form for forhåndsregistrering. En Registrarserver derimot, har ansvaret for å vite på hvilken eller hvilke SIP terminaler en bruker er innlogget. Dette forsvarer en `register()` metode i SIP implementasjonen. Vel å merke er det ikke noen Registarserver involvert i dette brukstilfellet, og `register()` metoden er dermed ikke anvendt under instansieringen av SIP terminalen.

I den siste delen av sekvensdiagrammet instansieres to objekter; H323TerminalEventObserver og SIPTerminalEventObserver. Disse objektene kalles fra eventsystemet blant annet når det kommer innkommende anrop. Disse to objektene er ikke implementert i ATK, men må være en del av implementasjonen til gatewayen.

Til slutt varsles administratoren av gatewayen om resultatet av initieringen.



Figur 6.5: Initiering SIP/H.323 gateway

6.2.6 Innkommende H.323 anrop

Sekvensdiagrammet i figur 6.6 på neste side viser et anrop til gatewayen fra en H.323 terminal. Siden modellen forutsetter at all signalering går via Gatekeeper sendes SETUP fra terminalen til Gatekeeper som anroper selve gatewayens H.323 terminalobjekt (H323Term). Det underliggende eventsystemet registrer at det kommer et innkommende anrop. Dette resulterer i at objektet H323Term instansierer et TerminatingCall objekt med H.323 implementasjon (H323Terminating). I instansieringen sendes det med hvilket H323Terminalobjekt som mottok anropet, og hvilken *socket*[3] anropet kom inn på.

For å varsle Gatewayobjektet om det innkommende anropet kalles metoden `incommingCall` på H323TerminalEventObserver objektet H323teo. Som nevnt tidligere er dette en callbackmetode i ATK. Denne metoden implementeres av ATK brukeren. I dette tilfellet blir det utvikleren av gatewayen. Videre vil H323teo objektet kalle `incommingH323Call` på Gateway objektet. Det er denne metoden som i realiteten tar seg av resten av stegene i sekvensdiagrammet. Dette kan illustreres med følgende pseudokode:

```
Result incomingH323Call(H323TermCall termCall)
{
  H323TermCall.proceeding(); //Signal to H.323 side something is going on
  SIPOrig = SIPTerm.createCall(SIPAlias,
                               bandwidth, SIPceo); //Create originating call
  SIPOrig.placeCall(); //Do the actual dial

  if (proceedingSIPCall() before timeout) //Wait for proceeding in n millisec
  {
    continue;
  }
  else
  {
    cancelCall;
  }

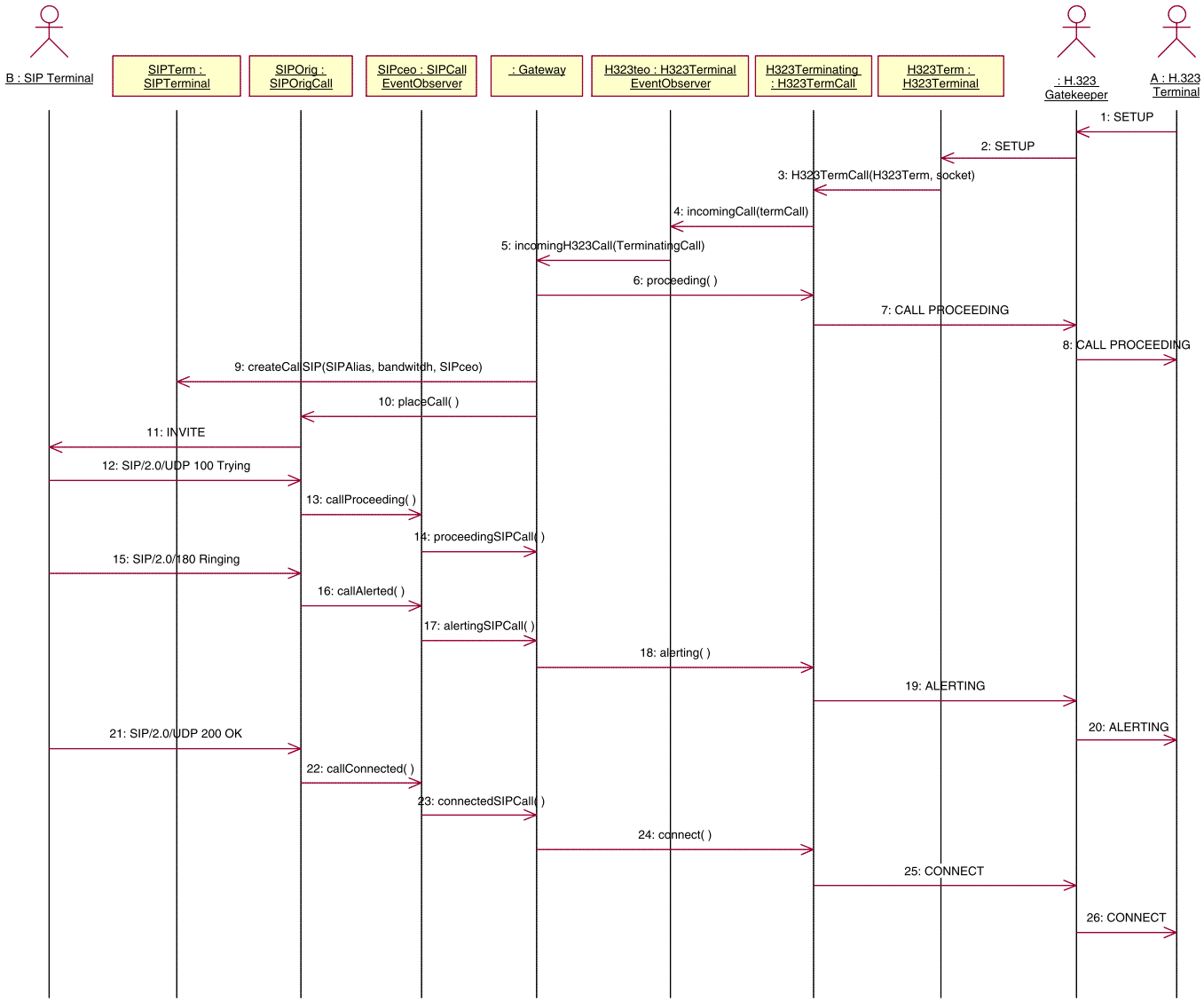
  if (alertingSIPCall before timeout) //Wait for alerting in n millisec
  {
    H323TermCall.alerting();
  }
  else
  {
    cancelCall;
  }

  if (connectedSIPCall() before timeout) //Wait for connected in n millisec
  {
    H323TermCall.connect();
  }
  else
  {
    cancelCall;
  }
}
```

I denne pseudokoden skisseres grovt en mekanisme for å oppdage feil. Denne mekanismen er primært tiltenkt selve oppsettet av anropet, og baserer

[3] IP socket som består av kombinasjonen IP adresse og portnummer.

seg i høy grad på *timere*. Dette kan brukes i selve oppsettet, men kan i mindre grad brukes under selve samtalen. Årsaken til dette er at en samtale ikke har noen definert varighet.



Figur 6.6: Sekvensdiagram innkommende H.323 anrop

6.2.7 Innkommende SIP anrop

Et innkommende SIP anrop som er vist i figur 6.7 på neste side har store likhetstrekk med H.323 anrop. Det som skiller disse to forskjellige anropene er primært hva som signaliseres fra den protokollspesifikke implementasjonen av Terminal- og Callobjektet. En annen forskjell er at H.323 løsningen impliserer en Gatekeeper. Dette gjør at signaleringen fra H.323 terminalen må gå via et mellomledd. SIP oppsettet som er skissert gjør ikke dette, men signaleringen hadde blitt lik hvis signaleringen gikk via en SIP proxyserver.

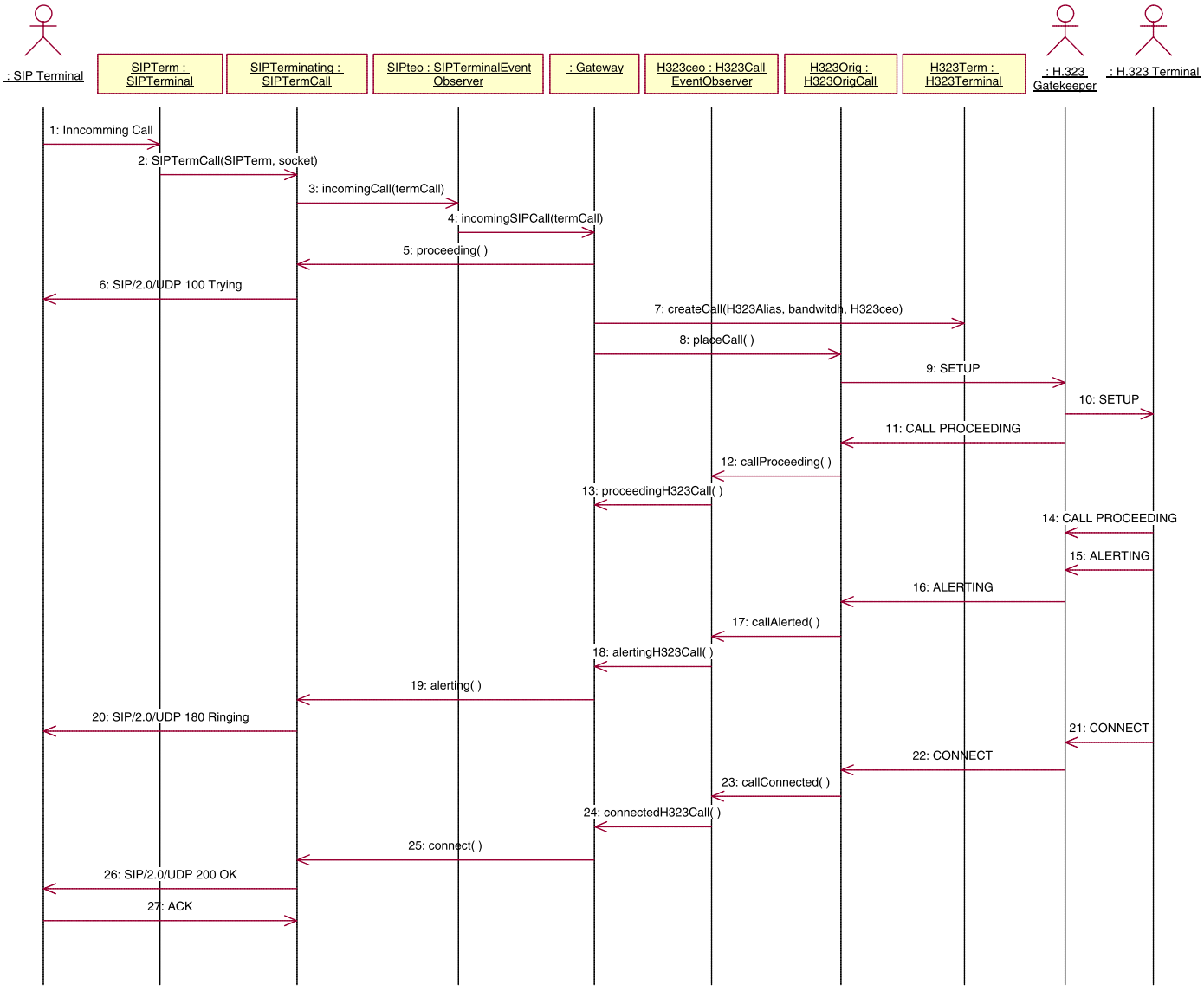
I likhet med innkommende anrop fra H.323, er `incomingSIPCall` metoden sentral for anrop fra SIP terminaler. Pseudokoden vil bli tilsvarende den som er er vist for `incomingH323Call`, bare at SIP og H.323 metoden vil bytte roller. Nå er det SIP siden som fungerer som terminerende side og H.323 siden som er den originerende.

```
Result incomingSIPCall(SIPTermCall termCall)
{
  SIPTermCall.proceeding(); //Signal to SIP side something is going on
  H323Orig = H323Term.createCall(H323Alias,
                                 bandwidth, H323ceo); //Create originating call
  H323Orig.placeCall(); //Do the actual dial

  if (proceedingH323Call() before timeout) //Wait for proceeding in n millisec
  {
    continue;
  }
  else
  {
    cancelCall;
  }

  if (alertingH323Call before timeout) //Wait for alerting in n millisec
  {
    SIPTermCall.alerting();
  }
  else
  {
    cancelCall;
  }

  if (connectedH323Call() before timeout) //Wait for connected in n millisec
  {
    SIPTermCall.connect();
  }
  else
  {
    cancelCall;
  }
}
```



Figur 6.7: Sekvensdiagram innkommende SIP anrop

6.2.8 Avslutte samtale fra H.323

Figur 6.8 på neste side viser en samtale som avsluttes fra en H.323 terminal. Verdt å merke seg er at mediakanalene allerede er tatt ned før starten på sekvensdiagrammet. Rekkefølgen som mediakanalene skal tas ned er i følge [ITU99, side 79] video, data og til slutt audiokanaler.

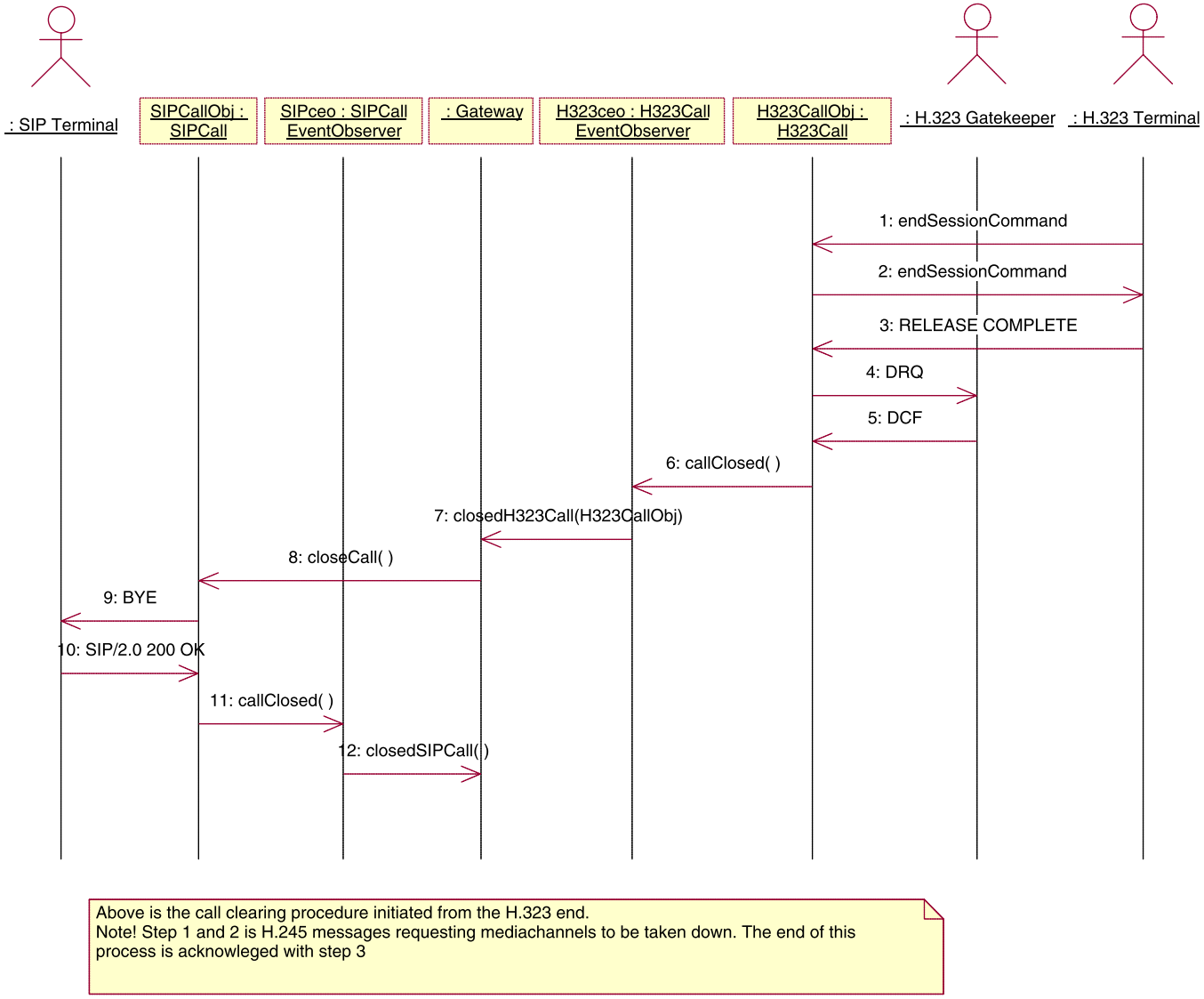
Det neste som skjer er at terminalen sender H.245 meldingen **endSessionCommand** i H.245 kontrollkanalen for å indikere at samtalen ønskes å tas ned. I dette tilfellet vil H323Callobjektet svare med **endSessionCommand** for å bekrefte dette. Når det andre endepunktet mottar denne meldingen sendes en Release Complete melding for å ta ned kanaler for kallsignalering. Steg 1 til 3 i sekvensdiagrammet viser dette. Legg merke til at dette er signalering som ikke involverer Gatekeeperen. Den berøres først når H323Callobjektet ber Gatekeeper om at samtalen frigjøres fra Gatekeeper. Dette gjøres ved at Disengage Request (DRQ) sendes til Gatekeeper. Denne meldingen bekreftes så med Disengage Confirmation (DCF). Etter dette er samtalen å regne som tatt ned og frigjort på H.323 siden.

At samtalen er tatt ned signaliseres til CallEventObserver objektet H323ceo ved at metoden `callClosed` kalles. Denne metoden anroper `closedH323Call`i Gatewayobjektet som har ansvaret for å ta ned anropet på SIP siden. Pseudokoden for denne metoden ser slik ut:

```
Result closedH323Call(H323Call H323CallObj)
{
  SIPCall SIPCallObj =
      returnSIPSide.get(H323CallObj); //Return SIP side of the call.
                                      //returnSIPSide is a Java Hashtableobject

  SIPCallObj.closeCall();

  if (closedSIPCall() before timeout)
  {
    return OK;
  {
  else
  {
    cancelCall;
  }
}
```



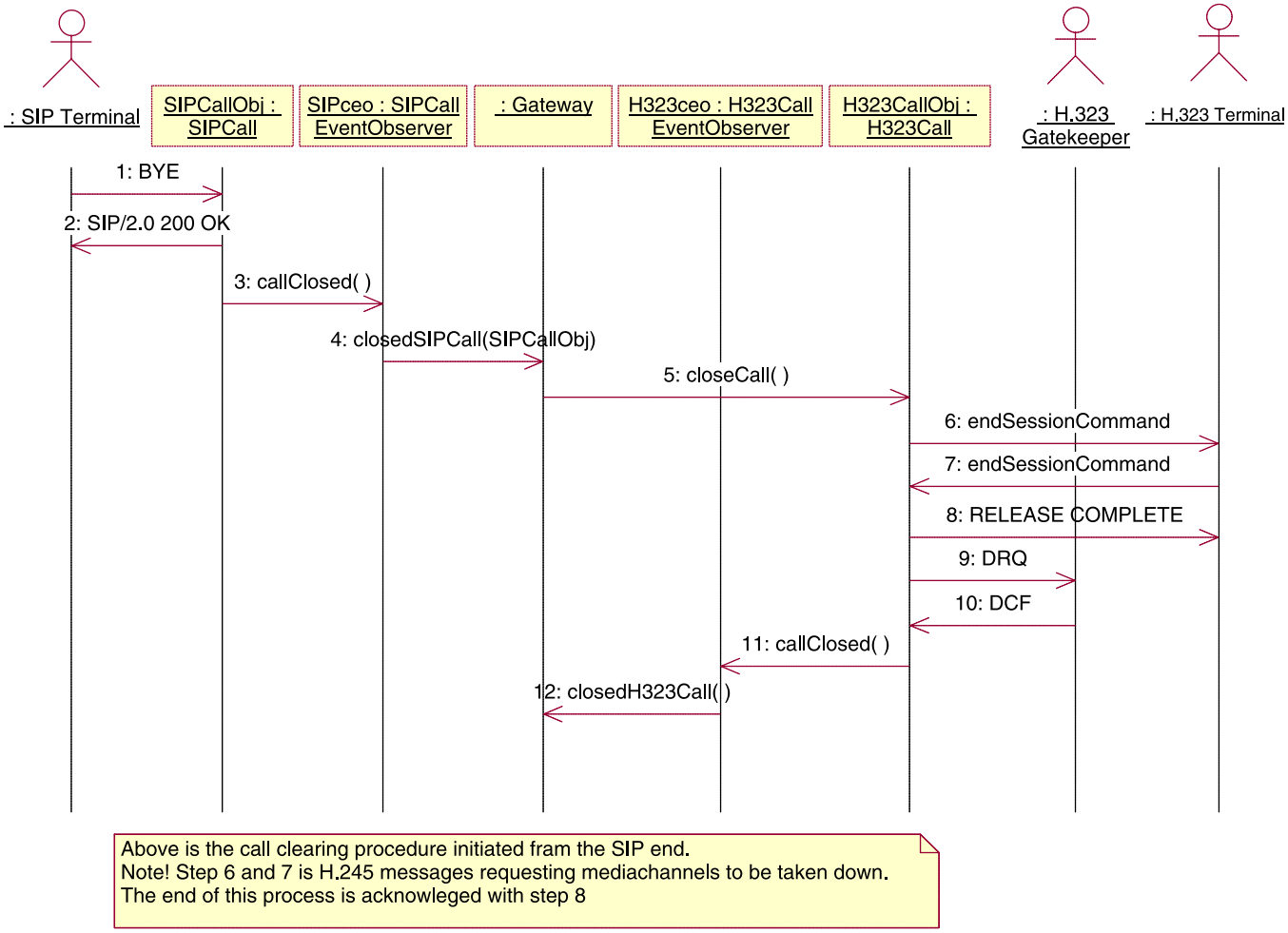
Figur 6.8: Sekvensdiagram for avsluttet samtale fra H.323 terminal

6.2.9 Avslutte samtale fra SIP

En samtale som avsluttes med initiativ fra en SIP terminal innledes med at terminalen sender BYE til SIPsiden av gatewayen. Dette er vist i figur 6.9 på neste side. SIPCallobjektet bekrefter denne meldingen med svaret `SIP/2.0 200 OK`. Dette er tilstrekkelig interaksjon med SIPterminalen for å bekrefte at anropet er tatt ned. Resten av prosedyren fra steg 3 er til forveksling lik den beskrevet over for H.323 anrop, men her styrer metoden `closedSIPCall` at anropet tas ned på det korresponderende H.323 anropet.

Det bemerkes at [HSSR99b] ikke beskriver noen prosedyre for hvordan mediakanalene skal tas ned, eller i hvilken rekkefølge. Dette blir noe som overlates til den enkelte protokollimplementasjon. Et tema som burde undersøkes nærmere i denne sammenhengen er hva som skjer med media etter at SIP signaleringen er tatt ned. Siden [HSSR99b] ikke eksplisitt spesifiserer hvordan media tas ned når `BYE` er sendt, er det grunn til å tro at det kan oppstå en situasjon hvor mediakanalene fortsatt er oppe selv om signaleringen er avsluttet. Dette kunne være en ønskelig situasjon for "ondskapsfulle" klienter som ønsker å lure systemene for å få rimelige telefonsamtaler. For selve betalingssystemene kan det da oppstå en situasjon hvor de registrer en kort samtale, mens samtalen i realiteten har vært en lang. Årsaken er at mediakanalene, og dermed selve samtalen, vært oppe mye lengere enn anropssignaleringen.

Metoden `closedSIPCall` beskrives ikke her fordi den bortsett fra at SIP og H.323 objektene bytter roller, vil være semantisk lik `closedH323Call`.



Figur 6.9: Sekvensdiagram for avsluttet samtale fra SIP terminal

6.2.10 Forandre mediaformater underveis i en samtale

Under en sesjon kan det være et behov for å forandre mediatypene eller sammensetningen, og det trengs funksjonalitet som kan fange opp en slik situasjon. Dette temaet er ikke modellert i dette kapittelet, men fortjener likevel å bli omtalt.

I H.323 vil et behov om å endre mediatypene eller sammensetning initiere en ny fase med H.245 mediaforhandling. Således er denne delen helt skilt fra vanlig kallsignalering, og ATK kan da lett skille ut H.245 meldinger. Hvis derimot ønsket er å initiere en ny mediaforhandling *fra* ATK, vil dette skje med eksplisitte metodekall ned i H.323 implementasjonen. Dermed skulle H.323 delen av ATK kunne oppfatte begge situasjonene og ha funksjonalitet for å utføre oppgavene.

Varsling av endringer i mediasituasjonen er ikke så adskilt i SIP. Nye mediakapabiliteter sendes avgårde med en ny `INVITE`. Denne meldingen er den samme som brukes for å sette opp en samtale. Derfor må det kunne skilles mellom hva meldingen skal brukes til. I følge spesifikasjonen til SIP [HSSR99b], skal derfor forandringer i mediasammensetningen signaleres med å sende en `INVITE` med den *opprinnelige* anrops ID'en. Dette skal ikke indikere et nytt anrop, men forandringer på et eksisterende anrop. Dette tilsier at SIP implementasjonen i ATK må ha funksjonalitet for å kunne skille mellom `INVITE` meldinger som er ment å indikere nye anrop, og meldinger som indikerer forandringer i et eksisterende anrop. Denne funksjonaliteten må videre generaliseres slik at den passer inn i ATKs virkemåte.

Det er grunn til å anta at dette er funksjonalitet som lar seg implementere i ATK og i gatewaymodellen. Men det er viktig at denne situasjonen modelleres godt før den implementeres. Grunnen til det er at dette er et område hvor det er stor forskjell på hvordan henholdsvis H.323 og SIP oppfører seg.

6.3 Oppsummering og konklusjon

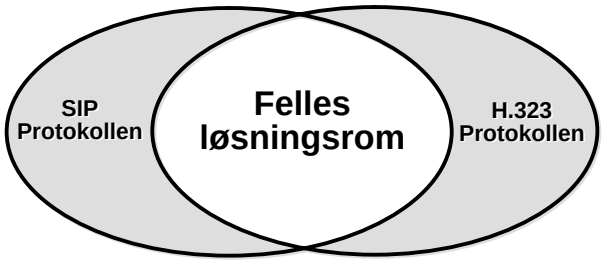
I dette kapittelet har ATKs design og funksjonalitet blitt gjennomgått. Dette har vært en sentral komponent i SIP/H323 gatewayen, og har i høy grad påvirket design og hvordan løsninger er utformet. Når det gjelder ATK er det et meget nyttig verktøy når man ønsker å ha tilgang til et H.323 miljø uten å måtte implementere H.323 i full skala. Abstraksjonen i ATK gir muligheten til å konsentrere seg om den overordnede funksjonaliteten. Det er denne tankegangen vi har ønsket å prøve ut ved å modellere og implementere SIP protokollen inn i ATKs klassehieraki.

Videre i kapittelet er SIP/H323 gatewayen presentert. Denne modellen er laget med den grunnleggende designbeslutningen om å la gatewayen stå imellom to komplette protokollmiljøer. I dette tilfelle vil det si et SIP miljø på den ene siden, og på den andre siden et H.323 miljø i form av en H.323 sone. Alternativet hadde vært som nevnt i [MH98, side 39], at gatewayen skulle muliggjøre å sette en enkelt SIP terminal inn i et H.323 miljø eller omvendt. Denne muligheten ble raskt forlatt fordi gatewayen antagelig

ville ha en begrenset nytteverdi. Vi ser det slik at den mest vanlige situasjonen ville være at en terminal i det ene miljøet ønsker å nå terminaler i det andre protokollmiljøet.

Modellen som er vist tar for seg enkle brukstilfeller, men samtidig de mest vanlige for VoIP samtaler. Altså A ringer B og ønsker en sesjon som inkluderer en eller flere mediatyper som begge forstår. Dette mener vi at skulle være mulig med det paradigmet som ligger til grunn for gatewayen; abstrahere funksjonaliteten for å oppnå en felles uttrykksform.

I en slik abstraksjon er det naturlig at noe av særtrekkene til den enkelte protokollen mistes. Dette kan illustreres med følgende figur:



Figur 6.10: Felles løsningsrom SIP og H.323

I det felles løsningsrommet ligger den delen av protkollfunksjonaliteten som har en en-til-en funksjonalitet mellom protokollene. Siden både SIP og H.323 er protokoller som er beregnet på IP-telefoni vil dette omfatte vesentlige deler av funksjonaliteten, og dermed vil det felles løsningsrommet være stort nok til å dekke de fleste standardsituasjoner. Dette løsningsrommet kan antagelig utvides ved å benytte seg av muligheten ATK gir for å aksessere protokollspesifikke deler av implementasjonene. Da kan en tenke seg at det kan være mulig å utvide løsningsrommet ved å lage en-til-mange relasjoner mellom protokollfunksjonalitetene. Et eksempel på dette kan være at et H.323 anrop med mediakapabilteter beskrevet som et logisk uttrykk, resulterer i en serie av SIP INIVITE med enklere kapabilitetsbeskrivelser. Så vil det svaret som passer best med H.323 anropet bli foretrukket.

En annen type design som kunne vært brukt var å la gatewayen grovt sett bestå av to deler:

- en del som oversetter direkte fra SIP til H.323.
- en del som oversetter direkte fra H.323 til SIP.

Denne løsningen ville resultert at ny funksjonalitet i gatewayen kunne implementeres raskere, men ville på den andre siden ført til at endringer i de respektive protokollene måtte gjøres to steder. Et annet ikke uvesentlig moment for at denne løsningen ble valgt bort var at det ville ført til at ATK ikke kunne anvendes.

Abstraksjon av protokoller generer til en viss grad redundans. Om denne redundansen er vesentlig vil i denne gatewayløsningen i hovedsak påvirkes av hvor mye ekstra objektinstansiering som må utføres, hvordan tråder opprettes, effektiviteten i Java og hvor kritisk tidsfaktoren er for signalering. Når det gjelder objektinstansiering bygger hele ATK på den filosofien at objektinstansiering skal holdes på et minimum, og at objekter skal referanseoverføres ved metodekall. Denne filosofien er videreført i arbeidet med SIP/H.323 gatewayen. Dermed skulle ekstra objektinstansiering holdes på et minimum.

Oppretting av tråder er en prosess som er relativt ressurskrevende, og kan påvirke ytelsen hvis det skjer under programkjøring i utstrakt grad. I denne gatewayløsningen vil hvert anrop ha minst to tråder. Multiplisert med antall anrop kan dette antallet fort bli betydelig. Dette løses ved at ATK oppretter en *trådpool* med et gitt antall tråder under oppstart. Tråder vil da tas fra trådpoolen ved behov og returneres etter bruk. Dette gjør at kostnaden ved bruk av tråder reduseres til det det koster å gjøre *contextswitching* mellom tråder. Denne kostnaden er svært liten siden alle trådene i en prosess deler minneområde for variabler.

Når det gjelder Java er språket i utgangspunktet avhengig av en *Virtual Machine* for å kunne eksekveres. Dette resulteres i noe overhead i form av at Javas bytekode må interpreteres og oversettes til maskinkode.

Det innføres også ekstra redundans i form av at abstraksjonen resulterer i en serie ekstra metodekall. Selv om disse metodekallene ikke resulterer i noen objektinstansiering, vil dette implisere noe ekstra tidsforbruk.

Det konstateres av dette at abstraheringen resulterer i redundans og dermed ekstra tidsforbruk. Det som gjenstår å vurdere i denne sammenhengen er om dette er kritisk. Den redundansen som oppstår vil antagelig føre til tidsforsinkelser i størrelse av millisekunder. Denne tidsforsinkelsen påvirker bare signaleringen mellom partene, og i utgangspunktet bare selve oppsettet av samtalen. Denne fasen er rimelig elastisk for mindre tidsforsinkelser, og i tillegg vil ikke forsinkelser i signalleringen påvirke kvaliteten på media. Dette resulterer i at det er rimelig å anta at redundansen som abstraheringen innfører ikke er kritisk.

Den modellen for en SIP/H.323 gateway som er beskrevet i dette kapittelet, kan ikke fungere som et knutepunkt mellom en SIP konferanse og en H.323 konferanse som til sammen involverer flere enn to parter. Dette har ligget utenfor det som vi har ansett som vårt problemområde. Samtidig er det ikke noe hinder for at den enkelte SIP eller H.323 terminal kan være del av henholdsvis en H.323 eller SIP multipartykonferanse.

Feilhåndtering er et moment som det tas lett på i dette kapittelet. Den verdensbildet som det er operert med under modelleringen er en perfekt verden hvor det aldri oppstår feil. Årsaken til dette er at det er forsøkt å vise hvordan konseptet med å abstrahere funksjonalitet kan løses. Det er så vidt hentydet til at en del feilsituasjoner kan oppdages ved bruk av timere tidligere i dette kapittelet. En slik metode vil fungere for noen feilsituasjoner, men på langt nær alle. Et annet moment er hvilken detaljeringsgrad selve Gatewayobjektet trenger på feilmeldingene. ATK er laget med tanke

på å bli brukt nær brukeren, derfor er det valgt å ha få detaljer på feilmeldingene som presenteres den applikasjonen som anvender ATK. Når ATK brukes i en gatewayløsning kan det hende det er behov for mer spesifikke feilmeldinger. Dette kan løses ved at gatewayobjektet aksesserer den protokollspesifikke delen av protokollimplementasjonene, men det vil i noe grad bryte ned abstraksjonsnivået.

Som antydet er dette et område gjenstående arbeid hvis gatewayløsningen skal kunne settes i produksjon i et virkelig miljø. Det må forventes at store deler av det gjenstående arbeidet må vies til å oppdage feilsituasjoner og håndtere dem riktig.

Til tross for det som er pekt på over, mener vi at vi har vist hvordan translasjon fra en signaleringsprotokoll til en annen kan foregå ved å abstrahere protokollfunksjonaliteten til en felles uttrykksform.

Kapittel 7

Implementasjon av gatewaymodellen

> «Det viser bare hva en kan oppnå når en gjør seg riktig flid,» sa Tussi. «Du skjønner hva jeg mener, Brumm? Du skjønner hva jeg mener, Nøff? Først bruker man sin forstand, og etterpå Arbeider man Hardt. Se godt på det. *Slik* skal hus bygges,» sa Tussi stolt og glad.
>
> — *A. A. Milne: Huset på Bjørnehjørnet*

FASEN dette kapittelet beskriver er implementasjonsfasen av vår modell. Således kan en si at dette er den delen av oppgaven som innebærer mest praktisk arbeide. Dette supplerer den tidligere modelleringsfasen som var mer preget av teoretisk arbeide. Dermed blir det et viktig moment i denne fasen å finne ut om vår modell er realiserbar.

Som det kan forventes av en implementasjonsfase, dukket det opp uforutsette problemer som måtte takles. Dette var problemer knyttet til utviklingsmiljøet og manglende dokumentasjon. Kapittelet vil utdype disse emnene nærmere sammen med en beskrivelse av den strategien vi la opp for implementasjonen, hva vi hadde tenke å implementere, og hva det ble tid til.

Til slutt følger en oppsummering og konklusjon av arbeidet som ble utført i denne fasen og hvilke oppgaver som er uløst.

7.1 Forutsetninger, mål og strategi

De føringene vi hadde for implementasjonen var at Java skulle benyttes som programmeringsspråk, Ericssons ATK skulle anvendes og at utviklingsmiljøet skulle være Borland JBuilder 3.0. I tillegg til dette måtte hele implementasjonsfasen foregå i Ericssons lokaler på Billingstad. Årsaken til dette er at kildekoden som det ble gitt tilgang til av sikkerhetsmessige grunner ikke måtte tas ut av Ericssons lokaler.

En annen forutsetning for implementasjonsfasen er den realiteten at en SIP klient som er bra, veldokumentert og ikke minst er rimelig, har vært svært vanskelig å oppdrive. Dette ble løst ved at vi bestemte oss for å emulere en SIP klient med svært begrensede egenskaper. Denne emuleringen skulle ikke ha noe som helst funksjonalitet for å overføre media.

Den strategien vi bestemte oss for å følge var å først implementere SIP i ATK slik at man kunne lage SIP endepunkter. Deretter implementere den avgrensede gatewaymodellen. På denne måten ville vi løse både oppgaven med å implementere SIP inn i ATK, og problematikken rundt den manglende SIP klienten. Med en fungerende SIP implementasjon i ATK ville et SIP endepunkt bestå av et lite antall linjer kode.

7.2 Implementasjonsfasen

For å kunne implementere en ny protokoll inn i ATKs klassehierarki er det nødvendig med god kjennskap til hvordan APIet er bygd opp. Denne kunnskapen ble ervervet ved dels å benytte seg av kodeeksempler lagt ved kildekoden til ATK, dels å utsette ATK for en *Reverse Engineering*[1]. I denne prosessen ble verktøyet Rational Rose benyttet for å forenkle oppgaven.

7.2.1 Implementasjon av grunnleggende komponenter

Når det ble vurdert å ha tilegnet seg tilstrekkelig kunnskap om ATK, og kodeeksempler kompilerte og kjørte, gikk implementasjonen over i en fase hvor det ble lagt vekt på å utvikle de grunnleggende komponentene i SIP delen. Dette kan summeres opp til å være:

- En UDP klient.
- En UDP tjener.
- En SIP parser.

Disse tre modulene ble utviklet som frittstående klasser. Dermed kan de lett brukes som en del av implementasjonen av ATK.

Meldinger i SIP baserer seg på UDP, derfor ble den første oppgaven å lage en enkel UDP klient og tjener. Klassen **UDPclient** ble implementert først, og består av konstruktører og metodekall som beskrevet under.

Denne parameterløse konstruktøren gjør ikke stort annet enn å sette bufferlengden til en standard størrelse.

```
public UDPClient()
{
  len = defaultBufferLength;
}
```

[1]En prosess hvor en bruker kildekoden som et utgangspunkt for å lage en modell. Det motsatte av den vanlige prosessen med å først lage modell så kildekode.

Konstruktøren under tar to parametere; hvilken maskin som skal være mottager og hvilket portnummer som skal anvendes.

```
public UDPClient(String hname, int pnbr)
{
  hostname = hname;
  portnumber = pnbr;
  len = defaultBufferLength;
}
```

Denne konstruktøren tar i tillegg til parameterene beskrevet over, et argument som setter ny bufferlengde.

```
public UDPClient(String hname, int pnbr, int blen)
{
  hostname = hname;
  portnumber = pnbr;
  len = blen;
}
```

I tillegg til konstruktørene har UDP klienten to `send` metoder. Den første av disse er den metoden med flest parametere. Parameterne er meldingen som skal sendes, mottaker og portnummer.

```
public void send(String request, String hname, int pnbr)
```

Java har en mengde med nyttige biblioteker som gjør at kodebiten som må til for å sende dataene blir svært liten. Linjene under er tilstrekkelig for å få oversendt selve meldingen.

```
DatagramSocket ds = new DatagramSocket(0);
InetAddress ia = InetAddress.getByName(hostname);
byte[] data = request.getBytes();
DatagramPacket outgoing = new DatagramPacket(data, data.length, ia , portnumber);
ds.send(outgoing);
```

Metoden under er tiltenkt de situasjonene hvor det er ønskelig enten å sende informasjon til den mottaker som er definert som standard, eller den som mottok en melding sist. Metoden kaller den `send` metoden som er beskrevet over, med standardverdiene for `hostname` og `portnumber`.

```
public void send(String request)
{
  send(request, hostname, portnumber);
}
```

Klassen **UDPServer** realiserer en enkel UDP server. Den har som oppgave å lytte på en port etter innkommende data, og deretter returnere disse dataene i en `String`.

Denne klassen kan instansieres på to måter. Enten ved at bare portnummeret det skal lyttes på sendes med, eller at også at lengden på bufferet angis. Signaturen på konstruktørene som løser disse oppgavene ser slik ut:

```
public UDPServer(int pnbr)
public UDPServer(int pnbr, int blen)
```

For å lytte etter data kalles en av metodene `waitForIncomming`. Disse metodene har følgene signatur:

```
public String waitForIncomming(int pnbr)
public String waitForIncomming()
```

Disse metodene er blokkerende og returnerer ikke til kallstedet før det er mottatt data fra porten. Returverdien fra disse metodene er en `String`, og inneholder dataene i UDP pakken.

Den neste sentrale komponenten er klassen **SIPParser**. Denne klassen realiserer en *meget* enkel SIP parser med svært begrenset funksjonalitet. For vårt bruk var det tilstrekkelig å sende en enkel SIP melding, og derfor ble parseren tilpasset etter dette.

Parseren tar grovt sett imot en `String` med hele SIP meldingen og returnerer et SIPMessageobjekt. Dette objektet inneholder alle felter i en SIP melding, og parseren har satt verdiene fra meldingen inn i de respektive feltene. Klassen SIPMessage er vist på side 139.

Rekkefølgen i parsingen er at først splittes hele meldingen opp i de enkelte linjene. Disse linjene blir satt inn i en `Vector`. Denne oppgaven blir tildelt metoden `separateLines`. Etter dette må hver enkelt linje parseres av metoden `parseLines`. I denne metoden settes dataene i meldingen inn i de respektive feltene i SIPMessageobjekt. Metoden brukeren vil kalle for å parse en melding vil dermed se slik ut:

```
*/
public SIPMessage parse(String messageString)
{
  return parseLines(separateLines(messageString));
```

Selv om denne klassen er ribbet for mye funksjonalitet og feiltesting, vil det grunnleggende konseptet være igjen i en utvidet parser. En utvidet parser ville basert seg på å lage en *Recursive Decendent* parser på grunnlag av BNF gramatikken oppgitt i [HSSR99b], og algoritmer og teknikker som er beskrevet i [KO97] for syntaktisk analyse.

De tre modulene beskrevet over ble så testet med testprogrammene UDPClientTest.java , UDPServerTest.java og Parsertest. Kildekoden til programmene er å finne på sidene 137, 138 og 145. Disse modulene fungerte i de testene som ble utført.

7.2.2 Implementasjon av SIP inn i ATK

Etter at de basale byggesteinene var ferdig var turen kommet til å implementere SIP inn i ATKs design. En naturlig framgangsmetode var da å implementere det som interfacene forlangte med metoder uten funksjonalitet. Altså selve rammeverket eller skjelettet for en SIP implementasjon i ATK.

På dette tidspunktet ble det bestemt at resten av prosjektet skulle utvikles med kildekodekontroll. Til dette formålet bruker Ericsson programmet *Clearcase*. Dette er et program som ivaretar behovet for versjonskontroll av kildekode. Clearcase gir i tillegg muligheten for å arbeide på både offentlige og private deler av kildekoden.

Denne beslutningen førte til at den kildekoden som allerede var laget ble overført til Clearcase. Dette førte til at alle *Pather* og *Classpather* i J-Builder måtte oppdateres. Denne oppgaven ble relativt omfattende fordi J-Builder ikke var helt konsekvent når ting ble oppdatert.

Rammeverket som beskrevet over ble nå laget og lagt inn i ATK. Dette fungerte tilsynelatende bra. Metodene var stort sett uten innhold, men det fungerte å instansiere objekter med SIP implementasjon.

Den neste fasen var å implementere mer funksjonalitet på dette rammeverket. I denne fasen ble det jobbet fra bunn av og oppover i klassehierarkiet. Det ble raskt konstatert at det asynkrone eventsystemet i H.323 delen av ATK også måtte være en del av SIP implementasjonen. Eventsystemet var dessverre ikke beskrevet i den dokumentasjonen som var tilgjengelig. Alternativet var da enten å belaste en av designerene, Frode Sivertsen på Ericsson, betydelig for å få forklart eventsystemet tilstrekkelig, eller gjøre et forsøk på å *trace* hvor og når de ulike eventene ble kalt. Det andre alternativet ble valgt for å slippe å forstyrre Frode Sivertsen i en allerede travel arbeidsdag.

Det viste seg da at den koden som hadde eksekvert uten problemer tidligere nå viste seg å generere feil under kjøring. Tiden var nå i ferd med å renne ut, og det ble bestemt at vi avsluttet implementasjonsfasen på dette nivået. Vi anså allikevel at vi allerede hadde fått tilstrekkelig erfaring og kunnskap om ATK til å vurdere hvor vidt det var mulig å realisere modellen vår.

Problemene som oppsto relaterer seg antagelig til flyttingen av koden inn i Clearcase, og kunne antagelig løses på relativt kort tid. Til tross for dette ble denne situasjonen vurdert til å ta så mye tid at det ville gå utover resten av oppgaven. Vi besluttet dermed å basere oss på det vi hadde kommet fram til så langt i implementasjonsfasen.

7.3 Problemer underveis

Det er nevnt over at det var problemer etter at kildekoden ble lagt inn i Clearcase. De problemene som er nevnt tidligere relaterer seg stort sett til oppdateringer av systemvariabler. En annet problem med Clearcase var at den ikke er helt stabil. Dette førte til at unødig mye tid gikk med til å sikre at alle nødvendige prosesser var i gang, og starte om Clearcase hvis de ikke var det. I tillegg til dette ble tilgangen til filene med kildekoden mye tregere etter at de var lagt inn under kildekodekontroll.

Et problem som ble oppdaget underveis var at det i interfacene til ATK er brukt noen protokollspesifikke typer. Dette gjør at generaliteten blir svekket. Årsaken til dette er at da må disse typene også brukes i implementasjonen av den enkelte protokollen. Et eksempel på dette finnes i interfacet TerminalDescriptor:

```
1 package se.ericsson.eto.mmts.atk;
2
3 import java.io.IOException;
4
5 import se.ericsson.eto.mmts.toolbox.protocol.h323.v2.common.TransportAdress;
6 import se.ericsson.eto.mmts.toolbox.protocol.h323.v2.common.Alias;
```

Linje nummer 5 og 6 importerer henholdsvis typene `TransportAdress` og `Alias`. Disse typene antas å ha, som det fremgår av importpathen, en

protokollspesifikk funksjon. Siden typene brukes på interfacenivå, må disse typene bli brukt i de respektive protokollimplementasjonene.

Det kan virke som disse datatypene kan være anvendelig for alle protokollimplementasjoner. For å vinne tilbake den generaliteten som er blitt tapt bør typedefinisjonene flyttes slik at de er en del av ATK pakken. Dette er vist under:

```
1 import se.ericsson.eto.mmts.atk.TransportAdress;
2 import se.ericsson.eto.mmts.atk.Alias;
```

Det neste som må vurderes er om disse datatypene er så generiske at de kan benyttes i protokollimplementasjonen uten tilpasning. Hvis de er det, kan typene uten videre endring flyttes inn som en del av ATK pakken.

Viser det seg at disse datatypene brukes forskjellig fra protokoll til protokoll det nødvendig å abstrahere dem. Dette kan gjøres ved at `TransportAdress` og `Alias` gjøres om til interfacer. Dermed kan datatypen implementeres spesielt for hver enkelt protokoll.

For vår del jobbet vi oss rundt dette problemet ved å bruke en H.323 type i SIP implementasjonen.

Eventsystemet ble den største utfordringen. Grunnen til dette er at systemet ikke er dokumentert tilstrekkelig av de som laget det. Oppgaven med å finne ut av hva som kalte hva og i hvilken rekkefølge ble for omfattende i forhold til den tiden vi hadde til rådighet.

7.4 Gjenstående arbeid

For å kunne vise det konseptuelle ved gatewaymodellen må SIP implementeres ferdig i ATK. Dette inkluderer i første rekke eventsystemet. For å implementere dette med de respektive callbackmetodene kreves det bedre kjennskap til hvordan kallene skjer i systemet. Dette kan løses enten ved at hele ATK traces, eller ved at utviklerne av eventsystemet dokumenterer det. Sekvensdiagrammer er en type dokumentasjon som ville vært til stor hjelp. Da vil det gå klart frem hva de ulike eventene resulterer i.

Ut over dette må de tomme metodene i SIP delen av ATK implementeres. Siden vårt mål bare var å konseptuelt vise paradigmet med å abstrahere protokollfunksjonalitet, behøver kun de mest sentrale metodene å implementeres.

Når de to oppgavene over er løst gjenstår realiseringen av den avgrensede gatewaymodellen. Dette burde ikke innebære mye arbeid fordi det bare er en bit av modellen som skal realiseres. I tillegg opererer modellen med en perfekt verden uten feilsituasjoner. Dette resulterer i at systemer for å oppdage og klare opp i feilsituasjoner ikke er nødvendig å implementere.

7.5 Oppsummering og konklusjon

I dette kapittelet er det beskrevet hvordan implementasjonen av gatewaymodellen forløp. Det første som ble implementert var altså det vi anså som

de mest grunnleggende modulene i systemet; UDP klienten, UDP tjenere og SIP parseren. I utviklingen av disse komponentene ble det lagt vekt på å bare legge inn nok funksjonalitet til å vise at det var mulig å lage en gateway mellom to VoIP protokoller ved å abstrahere protokollfunksjonaliteten.

Fasen med å lage et rammeverk for SIP implementasjonen som passet inn i ATKs klassediagram er også beskrevet i dette kapittelet. Denne fasen ble fullført, og vi så at det var mulig å instansiere en SIP implementasjon. Riktignok besto denne implementasjonen i grove trekk av tomme metoder, men vi fikk vist at rammeverket fungerte.

Dette ble fulgt opp av nytt stadium i prosessen som innebar implementasjon av mer funksjonalitet i SIP delen av ATK. Som nevnt før i kapittelet ble det konstatert at et tilsvarende eventsystem som i H.323 delen av ATK også måtte implementeres i SIP delen. Manglende dokumentasjon av dette systemet bidro til å gjøre den prosessen tung. Å forsøke å trace når de ulike eventene oppsto og hvilke metoder som ble kalt ble valgt som en fremgangsmåte for å få tilstrekkelig kunnskap om eventsystemet. Dette viste seg å ikke kunne gjennomføres på den korte tiden vi hadde til rådighet. Årsaken til dette var at programmer som hadde virket tidligere nå genererte feil under kjøring. Dette skyldtes beslutningen om å migrere kildekoden inn i kildekontrollsystemet Clearcase. Dermed markerte dette slutten på implementasjonsfasen.

De problemene som er nevnt i dette kapittelet kan grovt sett deles inn i to grupper:

- Problemer relatert til utviklingsmiljøet.
- Problemer relatert til hvordan ATK er bygd opp.

De fleste problemene som oppsto hører hjemme i den første kategorien. Dette var problemer som relaterte seg til å sette opp riktig miljøvariabler i utviklingsmiljøet, ustabilitet i JBuilder og vanskeligheter rundt migrasjon av kode inn i Clearcase. I tillegg til dette må det nevnes at bruk av Clearcase førte til at en mye tid ble brukt på å forvisse seg om at systemet fungerte tilfredsstillende.

Summen av dette ble at uforholdsmessig mye tid gikk med til å takle problemer i utviklingsmiljøet. Denne tiden kunne vært anvendt til koding.

Den andre kategorien med problemer skapte ikke mye forsinkelser, men påvirket heller graden av abstraksjon i ATK. Dette relaterer seg spesielt til at det ble anvendt to H.323 spesifikke typer i interfacedefinisjonen i ATK. Dette fører til at disse typene *må* anvendes i SIP implementasjonen. Dermed strider dette noe i mot selve intensjonen med ATK; *å tilby en abstrahert protokollfunksjonalitet gjennom et modulært, objektorientert grensesnitt*.

Årsaken til at slikt oppstår skyldes antagelig to ting. Det ene er at det er svært vanskelig å tenke abstrakt når en bare har et spesialtilfelle. En løsning hvor for eksempel SIP og H.323 hadde blitt implementert samtidig i ATK ville antagelig ført til at slike problemer hadde kommet opp i dagen

på et tidlig tidspunkt. Den andre årsaken til at det er brukt H.323 spesifikke typer i interfacet kan relatere seg til at utviklingen av ATK led under knappe tidsressurser.

En løsning på dette problemet ville være å abstrahere disse typene til interfacer. Dermed kunne protokollimplementasjonen sørge for at riktig type blir anvendt.

Det må understrekes at dette skapte minimale problemer for fremdriften. Løsningen ble for vår del å arbeide seg rundt problemet ved å importere de protokollspesifikke typene i SIP implementasjonen. Dette er en ting som naturligvis må rettes opp i senere versjoner.

Status ved slutten av implementasjonsfasen var at vi hadde realisert den grunnleggende strukturen i ATK, men manglet mye av funksjonaliteten. I tillegg har vi ikke fått implementert noe av selve gatewayen.

Selv om vi hadde håpet å komme lengere, har arbeidet som er blitt utført gitt oss en god innsikt i og forståelse av ATK. Det er også her det vesentligste implementasjonsarbeidet ligger. Når det er lagt inn tilstrekkelig funksjonalitet i ATK, skal det ikke mange programlinjer til for å realisere en avgrenset implementasjon av gatewaymodellen på side 61.

Vi mener med bakgrunn i kunnskapen som vi har tilegnet oss i denne fasen, at det er grunn til å anta at det er mulig å implementere en avgrenset del av gatewaymodellen. Dermed mener vi at det skal være mulig å realisere det paradigmet som ligger til grunn for modellen; *å lage en gateway mellom to protokoller for IP-telefoni ved å abstrahere protokollfunksjonaliteten*.

Kapittel 8

Media Gateway Control Protocol

> Brumm var gått inn i Hundremeterskogen, og nå stod han foran det som en gang hadde vært huset til Ugla.
> Det så ikke ut som noe hus lenger – det så ut som et tre som var blåst over ende, og så snart et hus ser sånn ut, så er det på tide at man prøver å finne seg et annet.
>
> — *A. A. Milne: Huset på Bjørnehjørnet*

PROTOKOLLER i IETF har i høy grad en dynamisk utvikling. Protokoller oppstår, eksisterer en stund, for så å bringe de beste egenskapene inn i nye protokoller. Protokoller innen IETF representerer sådan en evolusjonær tankegang. En slik evolusjon er det som er utgangspunktet for *Media Gateway Control Protocol* (MGCP) [ADE+99]. MGCP er sådan en sammenslåing av protokollene *Simple Gateway Control Protocol* (SGCP) [AH98] og *IP Device Control* (IPDC) [TCR98].

I dette kapittelet vil bakgrunn og motivasjon for MGCP beskrives. Videre vil kapittelet ta for seg hvilken funksjonalitet som ligger i MGCP.

8.1 Bakgrunn og motivasjon

Hele motivasjonen for å lage en protokoll som MGCP er behovet for, og eksistensen av, VoIP gatewayer. En VoIP gateway har som funksjon å formidle audio telefon samtaler over et IP basert nett. Et eksempel på dette kan være to vanlige telefoner som ønsker å snakke sammen og bruker Internett som transport seg imellom. For å realisere dette trengs det løsninger som oversetter signalering og tale fra vanlig telefoni til IP pakker, og fra IP tilbake til PSTN telefoni. Denne oppgaven overlates til en eller flere gatewayer.

En VoIP gateway kan grovt deles inn i to deler; en del som tar seg av konvertering av signalering, og en del som konverterer media. Denne inndelingen

gir grobunn for en tankegang om å dele inn de ulike delene i forskjellige fysiske (eller logiske) enheter. Disse delene kalles *Signalling Gateway* (SG) og *Media Gateway* (MG).

En slik dekomponering av gatewayer førte til arbeidet med å lage en distribuert gatewayløsning bestående av et antall av komponentene nevnt over. Hensikten med en distribuert GW er å kunne sette sammen ny funksjonalitet utfra allerede eksisterende komponenter. Samtidig vil den distribuerte gatewayen utad fungere som *en* VoIP gateway.

Under dette arbeidet viste det seg at når en protokoll oversettes til en annen vil det ofte bli et tap av informasjon. Dette er et resultat av at protokoller med tilsvarende funksjonalitet sjelden utfyller hverandre 100%. Løsningen blir da at man finner et felles løsningsrom for protokollene det skal oversettes mellom. I noen tilfeller kan det føre til at viktig informasjon går tapt. Som et konkret eksempel på dette nevner Telcordia Technologies tap av all informasjon relatert til testing av kontinunitet ved oversetting mellom protokollene SS7 (Signalling System No. 7) og Q.931 i sin presentasjon [Hui99] av SGCP[1]. Dette faktumet inspirerte til en ny arkitektur hvor funksjonalitet for kallkontroll ble eksportert fra gatewayene inn i en *Call Agent*. Call Agent vil sørge for den nødvendige abstraksjonen slik at en samling av gatewayer vil fremstå som en samlet distribuert løsning.

Ut fra dette melder seg et behov for en protokoll som kan kommunisere mellom de ulike delene av den distribuerte gatewayen. MGCP er ment å dekke dette behovet, og skal være en intern protokoll for distribuerte gatewayer.

Systemet vil bestå av en Call Agent som kan være distribuert over flere datamaskiner, og et sett med gatewayer. En typisk konfigurasjon kan være at det distribuerte gatewaysystemet på den ene siden har et grensesnitt mot en eller flere PSTN svitsjer, og på den andre siden et grensesnitt mot et H.323 konformt system. I dette systemet vil de respektive gatewayenes Call Agent ta seg av signaleringsfunksjonen, opprettholde tilstanden til anropet og styre MG'ene som konsentrerer seg om translasjon av audio signalene.

8.2 Forbindelsesmodell (Connection Model)

For å kunne abstrahere kall på en måte som gjenspeiler endepunkter og endepunktstyper trengs en *forbindelsesmodell* eller *Connection Model*.

MGCPs Connection Model, slik den er vist i figur 8.1 på neste side, består av:

- Connection.
- Endpoint.

En Connection er en abstraksjon av selve mediabiten i anropet. En Connection består av en eller flere endepunkter (Endpoints), og kan enten være punkt-til-punkt eller multipunkt (en til mange). Som eksempler på dette

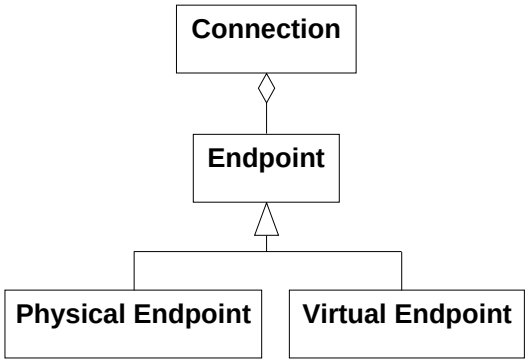
[1] forløperen til MGCP.

kan nevnes en telefonsamtale mellom to parter, og en telekonferanse hvor flere er involvert.

Connections blir opprettet av en Call Agent ved behov. Den vil få det innkommende anropet og se hva som trengs av mediatranslasjon. Deretter vil Call Agent opprette en Connection på en MG som kan håndtere den respektive mediakonverteringen.

Hvis en skal sammenligne begrepet Connection med H.323, kan en si at en Connection kan ligne en H.245 logisk mediakanal.

Endpoints er enten sendere eller mottakere av data og kan være fysiske eller virtuelle. Et eksempel på et fysiske Endpoint kan være et grensesnitt på en gateway som terminerer en analog telefonforbindelse. Virtuelle Endpoints kan være en audiokilde på en audioserver (f.eks. en musikkdatabase).



Figur 8.1: MGCP Connection Model

8.3 Omtalte anvendelsesområder

I [ADE+99][side 6] beskrives anvendelsesområdet for MGCP til å være kontroll av telefoni gatewayer fra eksterne elementer for kallkontroll. Med dette menes her Call Agents (CA) (også kalt *Media Gateway Controllers*). Som eksempler på hvor MGCP kan anvendes lister [ADE+99] opp:

- Trunking Gateways.
- Voice over ATM Gateways.
- Access Gateways.
- Residential Gateways.
- Business Gateways.
- Network Access Server.
- Linje- eller pakkesvitsjer som kan tilby et kontrollgrensesnitt til en CA.

8.3.1 Trunking Gateway

En *Trunking Gateway* (TG) er en gateway mellom *Switched Circuit Network* (SCN)[2] og VoIP. Denne løsningen vil gjøre det mulig for SCN svitsjer å bruke et pakkebasert nett (IP eller ATM) som *trunk*[3] i mellom seg.

En annen type TG er *Voice over ATM Gateway*. Forskjellen er at denne gatewayen står mellom et SCN- og ATM nett, mens en vanlig TG står mellom SCN nett.

8.3.2 Access Gateway

Access Gateway (AG) er en type GW som tilbyr *User to Network Interface* (UNI). Med dette menes at en AG gir et grensesnitt mellom tradisjonelle telefoner (ISDN og POTS) og et VoIP nettverk. Telefonen er fysisk knyttet til gatewayen via "pluggen i veggen".

En tenkt bruksmåte kan være som en slags hussentral knyttet til vanlige telefoner. Et eksempel kan være en konfigurasjon hvor det ene endepunktet er knyttet til en AG, mens det andre endepunktet er tilknyttet en offentlig SCN svitsj.

8.3.3 Residential Gateway

Residential Gateway (RG) er en AG for et lite antall audioterminaler som f.eks kan være samlokalisert via et kabelmodem. Et eksempel kan være fire PSTN telefoner i en husstand tilkoblet en RG i form av et slags kabeltvmodem.

8.3.4 Network Access Server

Løsningen en *Network Access Server* (NAS) vil tilby er å sørge for konvertering av modemsignaler til og fra SCN nettet, og levere dataaksess til pakkenettverket (IP). Dette kan sammenlignes med å bruke PPP for å koble opp mot en Internettleverandør (ISP). I en slik løsning vil et modem tilknyttet f.eks. til en PC koble seg opp til en NAS i form av en modempool. Denne modempoolen er igjen knyttet til et datanettverk. På denne måten brukes SCN nettet for å få aksess til datanettet og transport av datapakker.

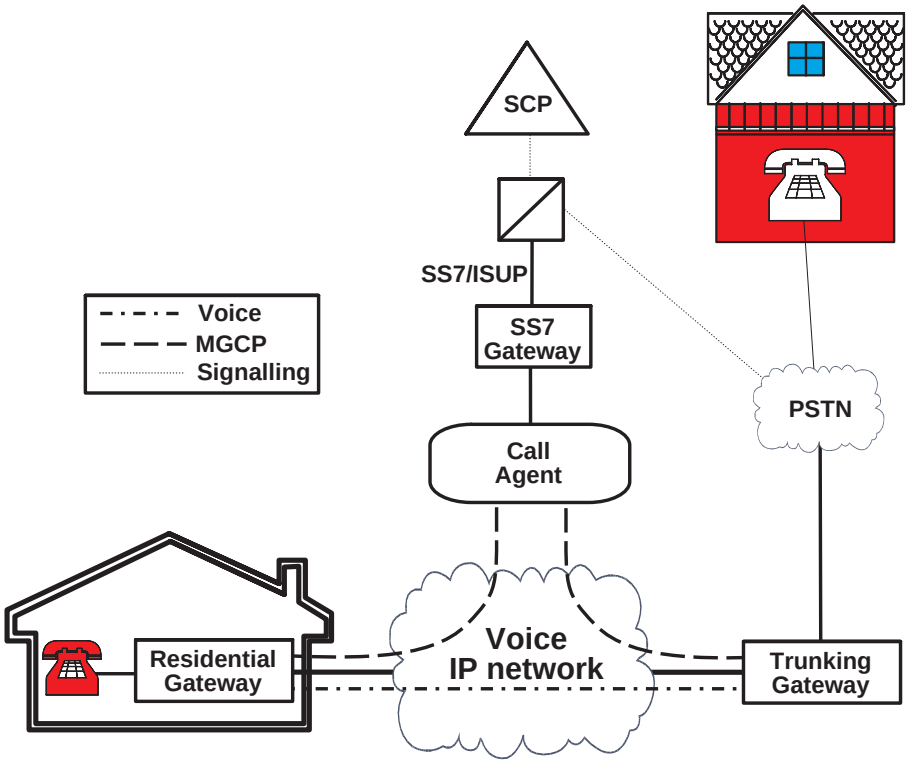
8.3.5 Business Gateway

Med en *Business Gateway* (BG) menes det her gatewayer som tilbyr et digitalt PBX grensesnitt til et VoIP nett.

[2] Linjesvitsjet nett. Det vi si det tradisjonelle telefonnettet.
[3] Sammenslåing av flere kall til en større datastrøm

8.4 Eksempel MGCP

Figur 8.2 som er tatt fra [HCMS99], viser et typisk oppsett hvor MGCP er involvert. En vanlig telefon er knyttet til et VoIP nettverk via en Residential Gateway. En Trunk Gateway er grensesnittet mot PSTN for den andre deltageren. En Call Agent vil sørge for å vedlikeholde nødvendige kalltilstander, og kontrollere RG og TG.



Figur 8.2: Eksempel MGCP VoIP til PSTN

8.5 Utvidete anvendelsesområder

De anvendelsesområdene som er nevnt over går grovt sett ut på å tilby VoIP i en eller annen form hvis man ser bort fra NAS. Muligheten for å anvende MGCP i en kontekst hvor det er flere mediatyper involvert (f.eks. video) i et kall, er et tema som vil bli vurdert i dette kapittelet.

For å kunne utvides til å håndtere andre typer media, som video og applikasjonsdeling, stilles det flere krav til protokollen:

- Protokollen må ha funksjonalitet som muliggjør formidling av ønskede mediatyper.

- Protokollen må kunne håndtere multipartysamtaler hvor ulike mediatyper er involvert (eks. en deltager har kun audio mens de to andre bruker video og audio).
- For å minske tiden det tar å sette opp samtaler er det ønskelig at protokollen støtter en effektiv teknikk for å forhandle hvilke media som skal anvendes.
- Protokollen må være så effektiv at maksimum forsinkelse er 300ms [PD96] fra avsender til mottaker. Dette tilsvarer en rundtur på 600ms.

8.5.1 Formidling av ønskede mediatyper i MGCP

I en MGCP løsning vil Call Agent opprette forbindelsen videre i nettet til mottakeren, og allokere de nødvendige ressursene. Ressursene allokeres i mediagatewayene ved hjelp av *Session Description Protocol* (SDP) (se kapittel 5 på side 43)

At MGCP bruker SDP for å beskrive sesjoner indikerer at det kan være en mekanisme tilgjengelig for å beskrive hvilke mediatyper som er ønskelig i en sesjon. Siden SDP har funksjonalitet for å beskrive multimediasesjoner med flere mediatyper, må eventuelle begrensninger da relatere seg til funksjonaliteten i MGCP. Denne begrensningen er beskrevet i kapittel 2.2 "Usage of SDP" i [ADE+99]. Der nevnes det at den aktuelle versjonen av MGCP (versjon 0.1) kun takler bruk av SDP til å beskrive audio og data CODEC'er.

Dette fører til at begrepet "multimediasesjon" får begrenset betydning. En sesjon kan i følge dette bestå av audio og/eller data.

Mediatypen i SDP meldingen kodes med *audio* eller *nas* (**n**etwork **a**ccess **s**ervice). Network Access Service er en tjeneste for autentisering og pålogging av nettverk som inngår i en Connection. Eksempler kan være at det trengs å opprettes en forbindelse til et nettverk hvor det kreves påloggings-ID og passord (for eksempel tilgang til et IP-nett med PPP og et oppringt modem). Dette kan muliggjøres med mediafeltet i SDP satt til *nas* i tillegg til at attributter med nødvendig tilleggsinformasjon sendes over (f.eks. autentiseringsprotokoll, id, passord . . .).

Dette gjør at begrepet data i den sammenhengen det er brukt i [ADE+99, kapittel 2.2] er ensbetydende med å oversende informasjon for å få nødvendig nettverksaksess. Videre betyr dette i praksis at audio er den eneste mediatypen som kan brukes i en samtale hvor MGCP er involvert.

Som et resultat av momentene over kan det konkluderes med at MGCP i dagens versjon ikke er definert for å formidle andre ønskede mediatyper enn audio i en multimediakonferanse.

8.5.2 Multiparty samtaler med ulike mediatyper i MGCP

I en verden med multimediaterminaler vil det eksistere utstyr med ulike egenskaper. Alt fra terminaler som kan håndtere audio, video, applika-

sjonsdeling og ulike dataoverføringer til enkle terminaler som kun takler audio. I multipartysamtaler vil det være normalt at deltagerne har endeutstyr med forskjellig kapabiliteter. Det vil i tillegg være en vanlig situasjon at to deltagere i en multipartysamtale kan håndtere audio og video mens den tredje bare audio.

Hvis en MGCP-løsning skal håndtere en slik situasjon kreves følgende:

- MGCP må kunne håndtere multipartysamtaler.
- MGCP må takle at endeutstyr kan ha ulike mediakapabiliteter i en sesjon.

I [ADE+99] brukes begrepene *Connection* og *Call*. De er definert slik at Connections er gruppert inn i Calls. En Connection består av to eller flere endepunkter [ADE+99, side 7], og et Call kan bestå av flere Connections [ADE+99, side 17]. Dette tilsier at MGCP er i stand til å takle multipartysamtaler ved å opprette et Call bestående av et nettverk av Connections, eller et Call bestående av en eller flere multipoint Connections.

Siden MGCP definerer Call begrepet slik vil det være mulig å ha en Call hvor det er flere parter involvert som anvender ulike mediasammensetninger i en sesjon. Det er ting som tyder på at oppsetting av en slik sesjon kan bli noe komplisert. MGCP draftet [ADE+99, side 29] understreker at den initierende SDP meldingen kun kan inneholde *en* mediatype. Dette er en begrensning som ligger i MGCP og ikke i SDP. Det betyr at enten må det være flere Connections mellom parter som skal anvende flere mediatyper, eller at det først settes opp en Connection med en mediatype for deretter "legge" til flere mediatyper.

Utfra dette kan det sies at det er sannsynlig at MGCP kan takle både multipartysamtaler og bruk av ulike mediakonstellasjoner i en sesjon. Det bemerkes at det ikke er beskrevet noen slike situasjoner i MGCP draftet. Derfor må dette tolkes som at protokollen i utgangspunktet kun er beregnet for audiosamtaler mellom to eller flere parter. Dette understrekes av beskrivelsen over om hvilke mediatyper det er lagt opp til i MGCP.

Diskusjonen om ulike mediaformater blir derfor noe akademisk. Den dreier seg i praksis om det i en Connection skal være et eller flere audioformater til samme deltager.

8.5.3 Forhandlingsteknikker for mediatyper

Slik det er i dag eksisterer det en rekke formater for koding av media. Eksempler på dette er videoformatene MPEG, H.261 og H.263. Disse formatene er ofte ikke kompatible med hverandre. Dette fører til et krav om at partene i en samtale må utveksle data (henholdsvis lyd og bilde) med hverandre i de samme mediaformatene. For å sikre dette kreves det en mulighet for å utveksle hvilke mediaformat som skal anvendes. Dette gjøres i en dertil egnet signaleringsprotokoll. I MGCPs tilfelle brukes SDP for dette formålet.

Den løsningen som er valgt for å utveksle mediaformat kan påvirke den tiden det tar å sette opp en samtale. Dette gjelder spesielt i tilfeller hvor partene bruker forskjellige formater med forskjellige preferanser. Dette kan

føre til mange rundturer mellom partene som brukes til å enes om et felles sett av mediaformater.

Antall rundturer kan reduseres hvis protokollen som brukes til signalering innehar et visst begrep om logikk i mediaforhandlingen. Med dette menes for eksempel at det følgende kan uttrykkes i protokollen:

(Audio med G.711 **OG** Video med H.263)
ELLER
(Audio med G.711 **OG** Video med MPEG)
ELLER
(Audio med G.711)

H.245 [ITU98c] har innebygd en logikk som muliggjør slike uttrykk i mediaforhandling. SDP derimot, som MGCP anvender for å beskrive mediaformater som skal anvendes i en sesjon, har ikke denne muligheten. I følge SDPs RFC [HJ98, side 1] sies det eksplisitt at den ikke er laget med den hensikt å forhandle om mediaformater. Dette gjør at MGCP ikke har noen mekanisme som støtter forhandling av media.

8.5.4 Betydning av effektivitet i MGCP

I følge [PD96] må summen av forsinkelser i form av prosesseringstid og propagasjonstid være mindre enn 300ms fra A til B for at det skal være mulig å føre en dialog. Dette gjelder når samtalen er igang. Tiden det tar å sette opp en samtale vil ikke lide av samme begrensingen, og kan være adskillig lengere. Dette er avhengig av hva lags nett en setter opp samtalen mellom. Et par målinger av oppsettstiden fra en GSM mobiltelefon til PSTN mobiltelefon, viste at dette tok omtrent 4 sekunder. Denne tiden er innenfor hva som er akseptabel, og viser at oppsettstiden ikke er så tidskritisk som tidsforsinkelser i mediaoverføring.

MGCPs hensikt er å kontrollere mediagatewayer for å sette opp og forandre kallsesjoner. Den er ikke involvert i selve mediastrømmene som brukes i et kall. Dette medfører at i den grad brukeren vil merke noe til effektiviteten, vil det være i form av kortere eller lengre tid for å sette opp eller ta ned en samtale.

Om MGCP har høy eller lav effektivitet vil ikke gå utover kvaliteten på mediastrømmene (som f.eks. talekvalitet), men heller påvirke hvor rask signaleringen er fra ende til ende (f. eks tiden det tar å sette opp en samtale).

8.5.5 Konklusjon bruksområde

MGCP slik protokollen fremstår i dag, er den først og fremst beregnet for å muliggjøre bruk av Internett til overføring av vanlige telefonsamtaler (kalt *toll bypass markedet*). Motivasjonen for dette er rimelige telefonsamtaler. Potensielle interessenter for dette vil være brukere som har hyppige telefonsamtaler til utlandet, eller bedrifter som har en eksisterende IP nettverkstopologi som de ønsker å utnytte til tradisjonell telefoni.

Som det er påpekt tidligere, har MGCP elementer i seg som kan indikere at protokollen har et potensiale i multiparty multimediakonferanser. Men dette er ikke mulig slik som dagens versjon av protokollen er definert.

8.6 Lokalisering av brukere i MGCP

MGCP adresserer ikke brukere, men endeutstyr. I MGCPs tilfelle vil det si "pluggen i veggen". Videre må denne "pluggen i veggen" relateres til en adresse for å kunne nås av eventuelle samtalepartnere. Gatewayen vil så sørge for å opprettholde en assosiasjon mellom dette endeutstyret og Internett ved hjelp av en nummerplan. Ved hjelp av denne nummerplanen dirigerer gatewayen samtaler til riktige adressater.

En slik defineringen av endeutstyr gjør at man i MGCP hverken kan snakke om begrepene brukermobilitet eller terminalmobilitet.

Et forbehold må tas når det gjelder terminalmobilitet. Hvis endepunktet er tilknyttet et trådløst LAN, vil det innebære en viss form for terminalmobilitet. Vel å merke er dette kun for et lite geografisk område som for eksempel innenfor en og samme bygning.

Kapittel 9

Media Device Control Protocol

> Ugla bodde i Kastanjeslottet, et gammelt herskapshus av edel årgang. Det var finere enn noe annet hus Brumm visste om, for det hadde *både* dørhammer og klokkesnor.
>
> — *A. A. Milne: Ole Brumm*

PROTOKOLLEN *Media Device Control Protocol* (MDCP) [Sij98] er i likhet med MGCP (på side 85) et forsøk på å lage en generisk protokoll for å kontrollere gatewayer. I motsetning til MGCP utvider MDCP fokus fra VoIP til også å omfatte multimedia multiparty konferanser.

MDCPs utalte mål [Sij99, side 3] er å spesifisere en generisk protokoll for å kontrollere mediaprosessorer[1] i et multitjeneste datanettverk, og å støtte ulike applikasjoner med samme protokoll. Nærmere spesifisert kan en tenke seg et scenario hvor det eksisterer ulike mediaprosessorer som gjør transkodinger til og fra forskjellige mediaformater. Ved å sette disse sammmen ved hjelp av et felles grensesnitt og en kontrollprotokoll kan det dannes virtuelle gatewayer med ny funksjonalitet. I likhet med MGCP blir dette en distribuert gatewayløsning.

9.1 Oppbyggning og arkitektur

Utarbeidelsen av MDCP baserer seg på, i følge [SBW+99], en dekomponering av gatewayer i følgende funksjonelle inndelinger:

- Media Gateway (MG).
- Media Gateway Controller (MGC).

[1] MDCP bruker dette begrepet og mediagatewayer om hverandre

- Signalling Gateway (SG).

En *Media Gateway*(MG) vil være tildelt oppgaven med CODEC translasjon. Den får altså et mediaformat inn, oversetter mediet til et annet format, og sender det ut.

I tillegg til dette trengs en enhet som tar vare på tilstanden til de ulike sesjonene. Denne oppgaven er tildelt *Media Gateway Controller* (MGC). MGC kontrollerer den overordende kallstatusen for et gitt antall audio- eller multimediakanaler. En MGC kontrollerer en eller flere MG'er.

I [SBW+99, side 9] defineres *Signalling Gateway* (SG) som en enhet tilkoblet SCN og som terminerer SS7 (Signalling System Number 7) eller en annen type signaleringslink. Typisk vil det her være signalering for flere samtidige kall. SG vedlikeholder bare de tilstandene om et kall som er nødvendig for protokollgrensesnittene. I dette tilfellet er det MGC på den ene siden og SNC på den andre.

Etter at de innkommende signaleringsdataene fra SNC er prosessert overleveres denne informajonen til den respektive MGC'en. En SG kan overføre signaleringsinformasjon til flere separate MGC'er for å gi en bedre utnyttelse av det kraftfulle SS7 grensesnittet.

En konsekvens av dekomponeringen som er gjort over er at man trenger en protokoll som tillater at MGC og MG kommuniserer seg imellom. Denne oppgaven er tiltenkt MDCP, og vil utgjøre det felles grensesnittet som muliggjør at de ulike mediagatewayene kan snakke sammen.

I det følgende beskrives definisjonene av noen sentrale begreper i MDCP. Dette er essensielt for å få en forståelse av den videre oppbyggningen av MDCP.

9.1.1 Kontrollmodell MDCP

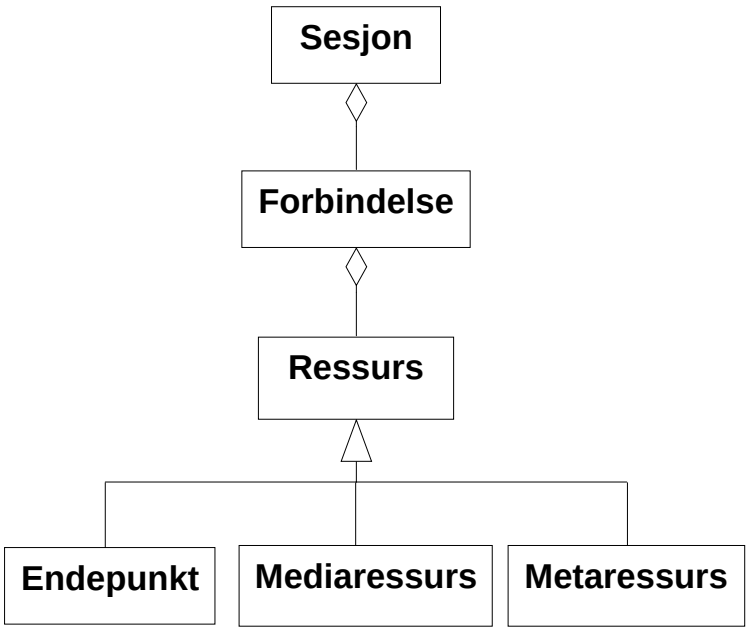
For kunne beskrive tilstander i et kall og hvilke ressurser som er involvert trengs det en *kontrollmodell*. I MDCPs kontrollmodell har begrepene *Session, Connection* og *Resource* hver sin eksplisitte betydning. Jeg velger i det følgende å oversette begrepene med henholdsvis *sesjon*, *forbindelse* og *ressurs* der hvor det er naturlig. Noen av termene som er brukt videre er noe friere oversatt, men dog ikke slik at det er i uoverenstemmelse med [Sij98].

I [Sij98, side 5] beskrives likevel designet som er valgt for kontrollmodellen i MDCP. I figur 9.1 på neste side vises designet modellert i UML. Det gjøres oppmerksom på at modellen ikke inneholder kardinalitet fordi dette ikke går helt klart fram av [Sij98].

Ressurser (resources)

En *ressurs* er et objekt som modellerer funksjonalitet i mediagatewayen, og kan i MDCP være av disse typene:

Endepunkt Den interne representasjonen av eksterne forbindelser fra MG til verden utenfor.



Figur 9.1: MDCP objekt arkitektur

Mediaressurs Ressurser som opererer på mediastrømmene, men som ikke har forbindelser utenfor MG.

Metaressurs Ressurser som går på tvers av flere sesjoner og er mediauavhengig. Eksempler på metaressurser kan være nummerplan og scriptingmodul.

Forbindelser (connections)

En *forbindelse* er en kombinasjon av *ressurser* som i hovedsak opererer unidireksjonalt med *en* mediatype og som forbinder *ett* eller *flere* endepunkter med andre endepunkt.

Flere *forbindelser* settes sammen i en *sesjon*. Eksempler på dette kan være:

- En punkt-til-punkt audio *sesjon* sammensatt av to unidireksjonale audio *forbindelser*.
- En multicast[2] *sesjon* bestående av en unidireksjonal *forbindelse* som er knyttet til en *multicast strøm*[3].
- En multipoint konferanse *sesjon* bestående av et antall unidireksjonale *forbindelser*.

[2]Pakker adressert med *multicast-adresser*. På denne måten skapes ikke unødvendig redundans i nettet. Pakken dupliseres ved behov i nettet etter hvert som det dukker opp forgreninger.

[3]En strøm av data (f.eks audio) fra en kilde til mange mange mottakere

- En punkt-til-punkt multimedia *sesjon* bygget opp av et antall audio, video og data *forbindelser*.

Sesjoner (sessions)

En *sesjon* er en generisk term for et kall, en konferanse e.l. Således er en telefonsamtale i likhet med en konferanse, en *sesjon*.

En Session bygges opp av et antall *forbindelser*.

9.2 Omtalte anvendelsesområder

I [Sij98, side 1] listes opp hvilke anvendelsesområder MDCP er tenkt brukt. Anvendelsesområdene som er omtalt i [Sij98], [SBW+99] og [Sij99], kan deles inn på følgende måte:

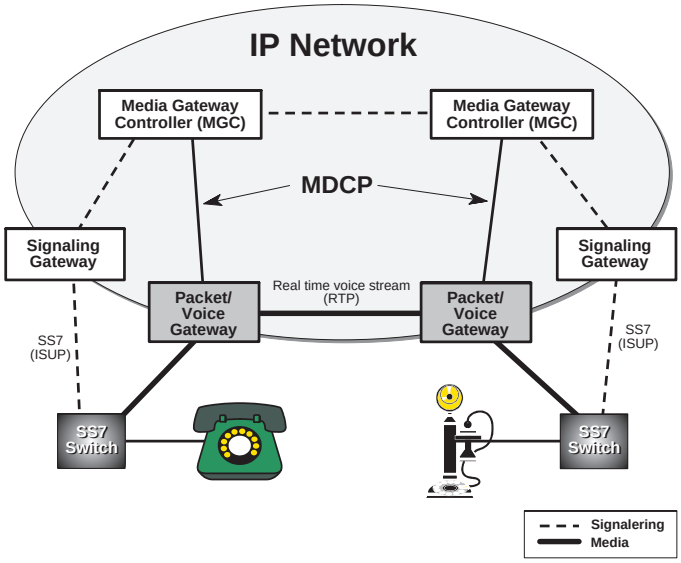
- VoIP Gatewayer.
- Network Access Servers.
- Multipoint Processors.
- Interaktive Voice Respons System.
- Transcoding Devices.

I [SBW+99, side 3] nevnes det at det opprinnelige anvendelsesområdet var VoIP gatewayer, men i ettertid ble det oppdaget at mye av den samme problematikken gjentok seg i andre typer gatewayer og media utstyr. Dermed med ble bruksområdet til MDCP utvidet med punktene over i tillegg til VoIP.

MDCP beskriver tre typer VoIP gatewayer; *Trunking Gateway*, *Access Gateway* og *Residential Gateway*. Disse tre typene GW'er er sammen med *Network Access Server*, beskrevet i kapittel 8.3 på side 87. Under vises det til eksempler, og anvendelsesområder som ikke er definert tidligere blir beskrevet.

9.2.1 Trunking Gateway

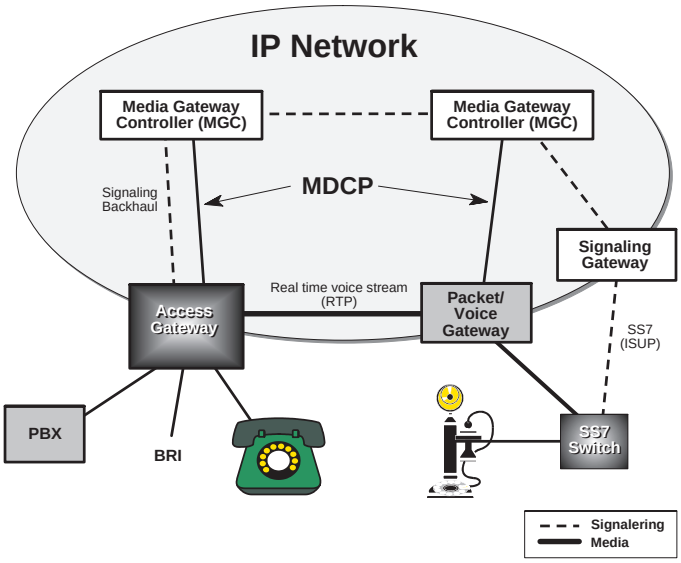
Et eksempel på bruk av *Trunking Gateway* (TG) er vist i figur 9.2 på neste side. Her vises hvordan det kan være mulig å bruke et pakkebasert nett som transport mellom to SS7 svitsjer. Gatewayløsningen vil i dette tilfellet tilby svitsjene et transparent grensesnitt mot hverandre. Det bemerkes at det ikke er spesifisert hva slags signalering det skal være mellom MGC, men det er grunn til å tro at dette for eksempel kan være SIP eller H.323.



Figur 9.2: MDCP Trunk Gateway

9.2.2 Access Gateway

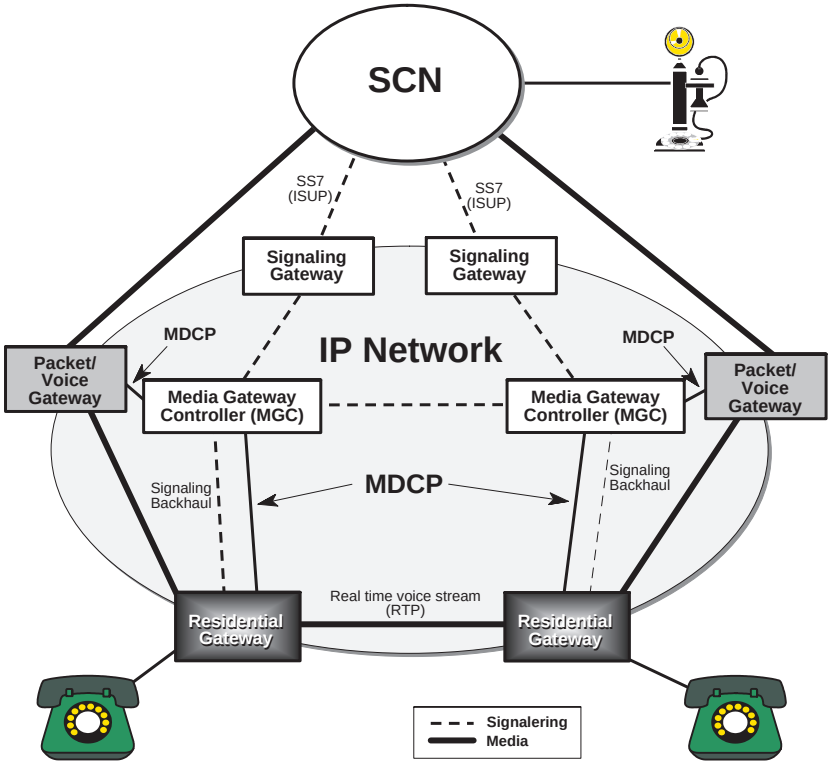
En tenkt bruksmåte for *Access Gateway* (AG) kan være som en slags hussentral knyttet til vanlige telefoner. Figur 9.3 viser en konfigurasjon hvor det ene endepunktet er knyttet til en aksess GW, mens det andre endepunktet er tilknyttet en offentlig SCN svitsj.



Figur 9.3: MDCP Access Gateway

9.2.3 Residential Gateway

Figur 9.4 viser et eksempel på bruk av *Residential Gateway* (RG). Her er fire PSTN telefoner i en husstand tilkoblet en RG i form av et slags "tvkabelmodem".



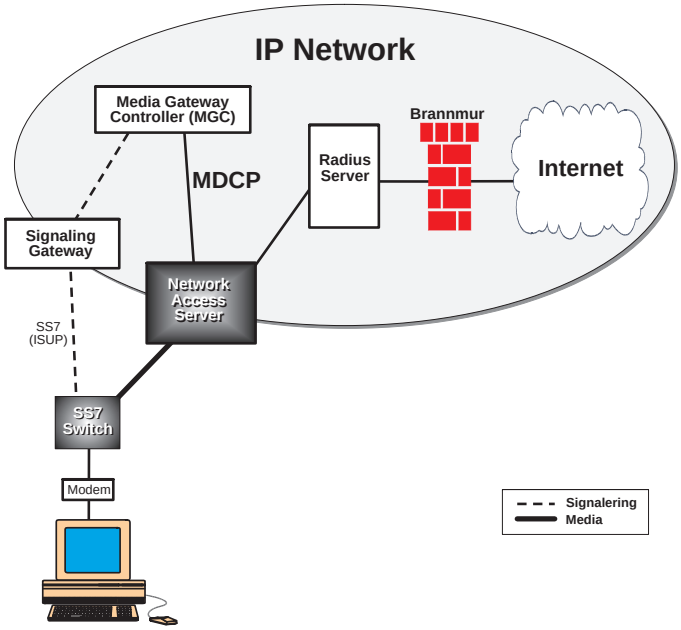
Figur 9.4: MDCP Residential Gateway

9.2.4 Network Access Server

En løsning hvor *Network Access Server* (NAS) er demonstrert i figur 9.5 på neste side. NAS vil sørge for å konvertere modemsignaler til og fra SCN nettet, og levere dataaksess til pakkenettverket (IP).

9.2.5 Multipoint Processors

En Multipoint Processor muliggjør at flere terminaler både på SCN og pakkenettet kan knyttes sammen i et kall (multipoint forbindelser).



Figur 9.5: MDCP Network Access Server

9.2.6 Interaktive Voice Respons System

Interaktive Voice Respons System (IVR) er et system som reagerer på *Dual Tone Multi Frequency* (DTMF) signaler og gir et stemmesvar. Typiske eksempler på dette vil være kontofon og billettbestillingsystemer over telefon.

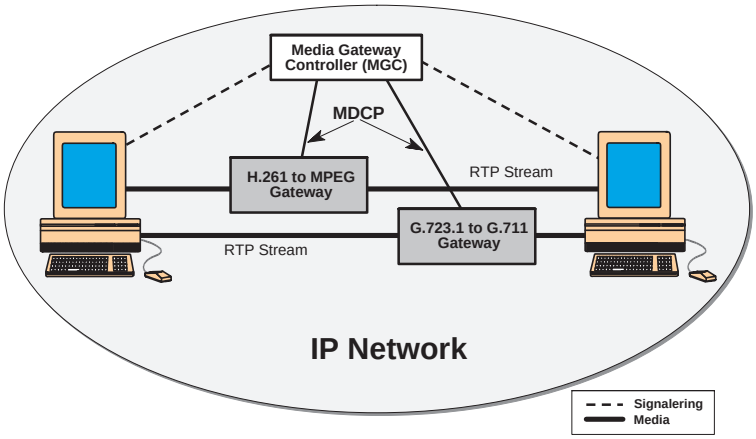
9.2.7 Transcoding Devices

Transcoding Devices brukes i et anvendelsesområde som primært har til oppgave å utføre CODEC translasjoner fra et mediaformat til et annet og vica versa.

Figur 9.6 på neste side viser en løsning for å konvertere fra GSM audio til audiformatet G.723.1.

9.3 MDCP anvendt som SIP/H.323 Gateway

For å vurdere hvor vidt MDCP er egnet eller ikke i denne sammenhengen, må man ta i betraktning den dekomponeringen av gatewayene som er hele motivasjonen for MDCP; Media Gateway (MG), Media Gateway Controller (MGC) og Signalling Gateway (SG). Hovedtankegangen bak MDCP er å kunne tilby en distribuert og allsidig gateway for media- og signaleringskonvertering.



Figur 9.6: MDCP CODEC translasjon

Med dette i bakhodet vil den videre vurderingen av MDCP i et scenario som SIP/H.323 gateway avhenge av den funksjonaliteten som er påkrevd. Det er to situasjoner som vil kreve svært forskjellig funksjonalitet av en GW:

1. Endepunktene bruker forskjellig protokoller for signalering, men mediaformatene (audio, video, data) er de samme.
2. Endepunktene bruker forskjellig protokoller for signalering, *og* ulike mediaformater.

I det første tilfellet er det i hovedsak snakk om en signaleringstranslasjon. Dette vil kunne løses av en *Signallinggateway* (SG). MDCP kan i denne sammenhengen settes til å samvirke med SG. Siden det bare er signaleringen som skal oversettes, vil MDCP her kun kontrollere en gateway. SG vil i tillegg få en svært spesifik oppgave, nemlig å translatere mellom SIP signalering og H.323 signalering. Behovet for å kommunisere med andre gatewayer vil også være lite.

Ser man på det andre tilfellet hvor det er snakk om translasjon av både signalering og mediaformat vil det være en oppgave som er betydelig mer krevende. Spesielt er mediatranslasjon en oppgave som krever stor prosesseringskapasitet. I dette bildet vil det være involvert både en SG og en eller flere mediagatewayer. Siden det er flere enheter involvert vil det være lettere å forsvare en introduksjon av MDCP for å kontrollere de ulike gatewayene.

Utifra dette kan det se ut til at MDCP er mest egnet i de tilfellene hvor de er flere enheter som skal operere sammen for å danne en virtuell distribuert gateway. I de tilfellene hvor det kun er snakk om å oversette signaleringen er det grunn til å tro at MDCP vil introdusere unødvendig redundans i løsningen.

Kapittel 10

Media Gateway Control

«Det er det det er,» sa Nøff fort. Og etter å ha tenkt lenge la han til:
«Bare varmere.»
«Hva er varmere?» spurte Kristoffer Robin.
«Andre siden av skogen, der huset til Tussi står,» sa Nøff.
«Huset *mitt*?» sa Tussi. «Huset mitt var her.»
«Nei,» sa Nøff fast. «Andre siden av skogen.»
«Fordi det er varmere,» sa Brumm.
«Jeg må da vel best vite hvor huset mitt stod»
«Kom og se,» sa Nøff bare og sprang i forveien.
«Det kan ikke være *to* hus,» sa Brumm. «ikke så nær hverandre.»
De kom til Bjørnehjørnet og der stod huset til Tussi, hyggeligere enn noensinne.
«Der kan dere se,» sa Nøff.
«Både inni og utenpå,» sa Brumm stolt.

— A. A. Milne: Huset på Bjørnehjørnet

At eksisterende protokoller konvergerer og smelter sammen til nye protokoller er ikke uvanlig hverken i IETF eller ITU. Det oppsiktsvekkende i tilfellet med *MEdia GAteway COntrol* (MEGACO) [CHG+99, Gro99], er at dette er resultatet av en sammenslåing av MGCP (side 85) fra IETF, og MDCP (side 95) fra ITU. Et samarbeide mellom IETF og ITU for å forene to konkurrerende protokoller hører definitivt med til sjeldenhetene. Fra et brukerperspektiv er dette en situasjon som er svært ønskelig. I stedet for å tviholde på prinsipper forener man det beste fra to verdener.

Dette kapittelet beskrives MEGACO på et nivå hvor det virker å være konsensus. Prosessen med spesifiseringen av MEGACO er, i skrivende stund, i en svært turbulent fase. Derfor konsentrer dette kapittelet seg mer om prosessen, prinsippene og problemer som må løses.

10.1 Bakgrunn og motivasjon

Det er to drivkrefter som er hovedmotivasjonen bak MEGACO. Den første er erkjennelsen av at en dekomponering av gatewayfunksjonalitet er en naturlig videreutvikling av gatewayarkitekturer. Den andre er det faktum at krefter både innen IETF og ITU allerede har jobbet med denne problemstillingen en stund, og har kommet fram til hver sin løsning av problematikken. Disse løsningene er henholdsvis MDCP og MGCP.

Det lå en stund an til en ny verbal krig mellom IETF og ITU. Dette skjedde ikke, og det ble enighet om å forene MGCP og MDCP i en ny protokoll kalt MEGACO/H.248. Betegnelsen for denne protokollen vil antagelig bli MEGACO innen IETF, mens den i ITU sannsynligvis blir kalt H.248. I det følgende vil betegnelsen MEGACO bli brukt for MEGACO/H.248.

Selv om toneangivende krefter både innen IETF og ITU har noen kjepphester de ønsker å ri (f.eks. om protokollen skal spesifiseres i et tekst- eller binærformat), preges diskusjonene av et ønske om å forenes i en standard.

10.2 Forbindelsesmodell (Connection Model)

I [CHG+99, side 9] beskrives MEGACOs *Connection Model* som en logisk representasjon i MG'en av de fysiske objektene som er involvert i en sesjon. Med dette menes at for eksempel hvert fysiske endepunkt som inngår i en samtale har sin tilsvarende logisk representasjon i MG'en. Disse logiske entitetene kan kontrolleres av en *Media Gateway Controller* (MGC). MGC er ansvarlig for å kontrollere tilstanden til den enkelte sesjonen i MG'en.

Den modellen som er valgt i forhold til Connections i MEGACO representerer en ny tankegang i forhold til MDCPs kontrollmodell (se figur 9.1 på side 97) og MGCPs Connection Model (se figur 8.1 på side 87). MGCP og MDCP modellerer grovt sett en sesjon som en aggregering av forbindelser med gitte ressurser, mens MEGACO generaliserer dette ytterligere til en sesjon i en gitt kontekst bestående av termineringer med visse egenskaper.

Hovedabstraksjonene i MEGACOs Connection Model er:

- Terminations.
- Context.

10.2.1 Terminations

Terminations er en enhet som kan være kilde til, og/eller mottaker av, en eller flere mediatyper. I en multimediakonferanse kan en Termination være multimedia. Det vil si at den sender og/eller mottar flere multimediastrømmer.

10.2.2 Context

En *Context* er en assosiasjon mellom en samling av Terminations som utgjør en enkelt sesjon. Konferansen kan være punkt-til-punkt eller multipunkt, og kan involvere ett eller flere mediatyper (f.eks. bare audio eller audio og video).

En sesjon bestående av to endepunkter som kommuniserer med hverandre vil typisk utgjøre en Context. Dette betyr at en Context kan beskrives som en gruppering av alle terminalene som inngår i samme sesjon. Vel å merke *kan* to endepunkter som snakker med hverandre ha to Contexter i samme MG. Et eksempel på dette kan være at en Context seg av audio konvertering, mens en annen Context tar seg av videokonvertering. Til tross for dette vil det vanlige være at all mediakonvertering i en sesjon foregår i samme Context. Et eksempel på dette er vist i figur 10.2 på side 107.

En Context som inneholder mer enn to Terminations bekriver en *Conference Bridge*. Med det menes for eksempel audio- eller videomiksing, video svitsjing eller kontroll av hvem som hører og ser hvem.

Det er en spesiell Context som nevnes i [Gro99], *Null Context*. Denne Contexten inneholder alle Terminations som for øyeblikket er ledige. Et eksempel på dette kan være de ledige linjene i en Access Gateway representert som Termionations i en Null Context.

MG har muligheten til å overføre Terminations til Null Context enten når en sesjon avsluttes eller det ikke er behov for den enkelte Termination i sesjonen. MG kan så overføre Terminations fra Null Context til aktive Contexter når den skal brukes på nytt i sesjon.

Alternativet til dette hadde vært å slette det enkelte Terminationobjekt når det ikke hadde vært mer behov for det i en sesjon. Dette ville ført til mye unødvendig arbeid fordi det enkelte Terminationobjekt måtte instansieres med riktige egenskaper for hver gang det skulle anvendes i en sesjon.

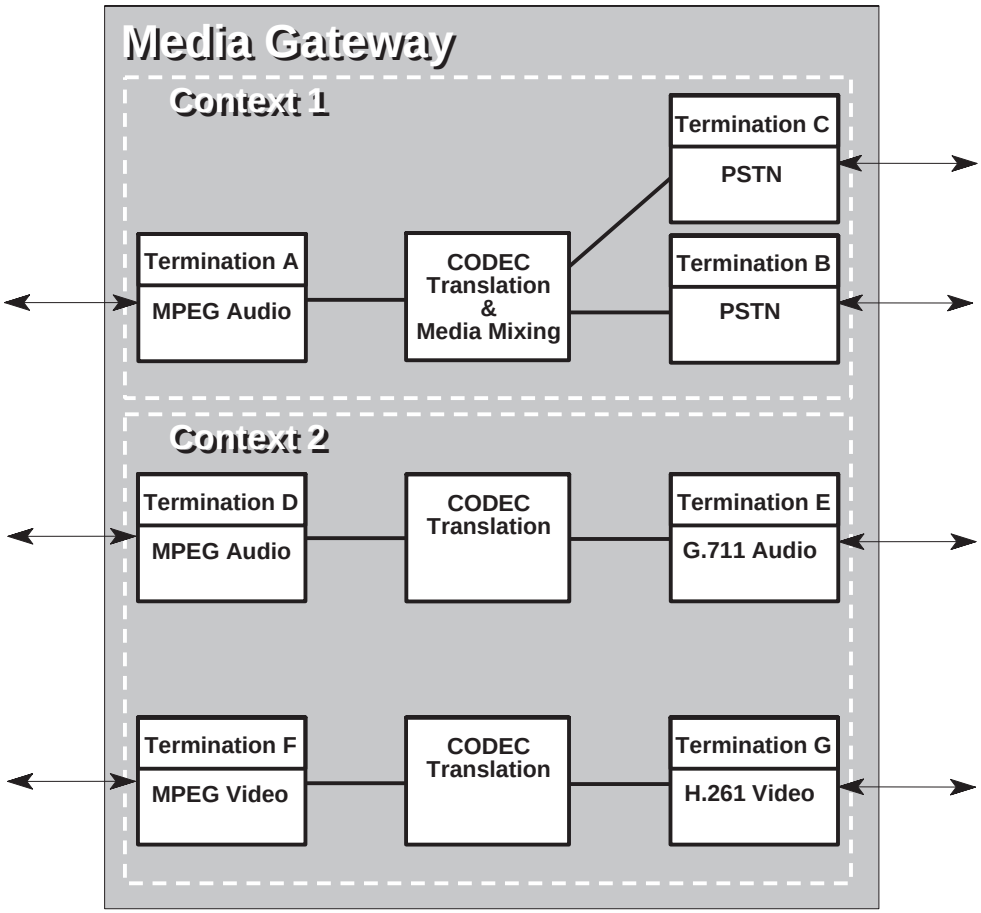
10.2.3 Eksempler på Context/Termination

For å utdype sammenhengen mellom Context og Termination er det enklest å ta for seg noen eksempler i form av tre figurer. I figurene representerer de tomme boksene en logisk assosiasjon mellom Terminations.

Ulike Contexter

Det første eksempelet, som er vist på figur 10.1 på neste side, viser to Contexter i en mediagateway. Selv om mediagatewayen er tegnet som en enhet kan den være fysisk oppdelt.

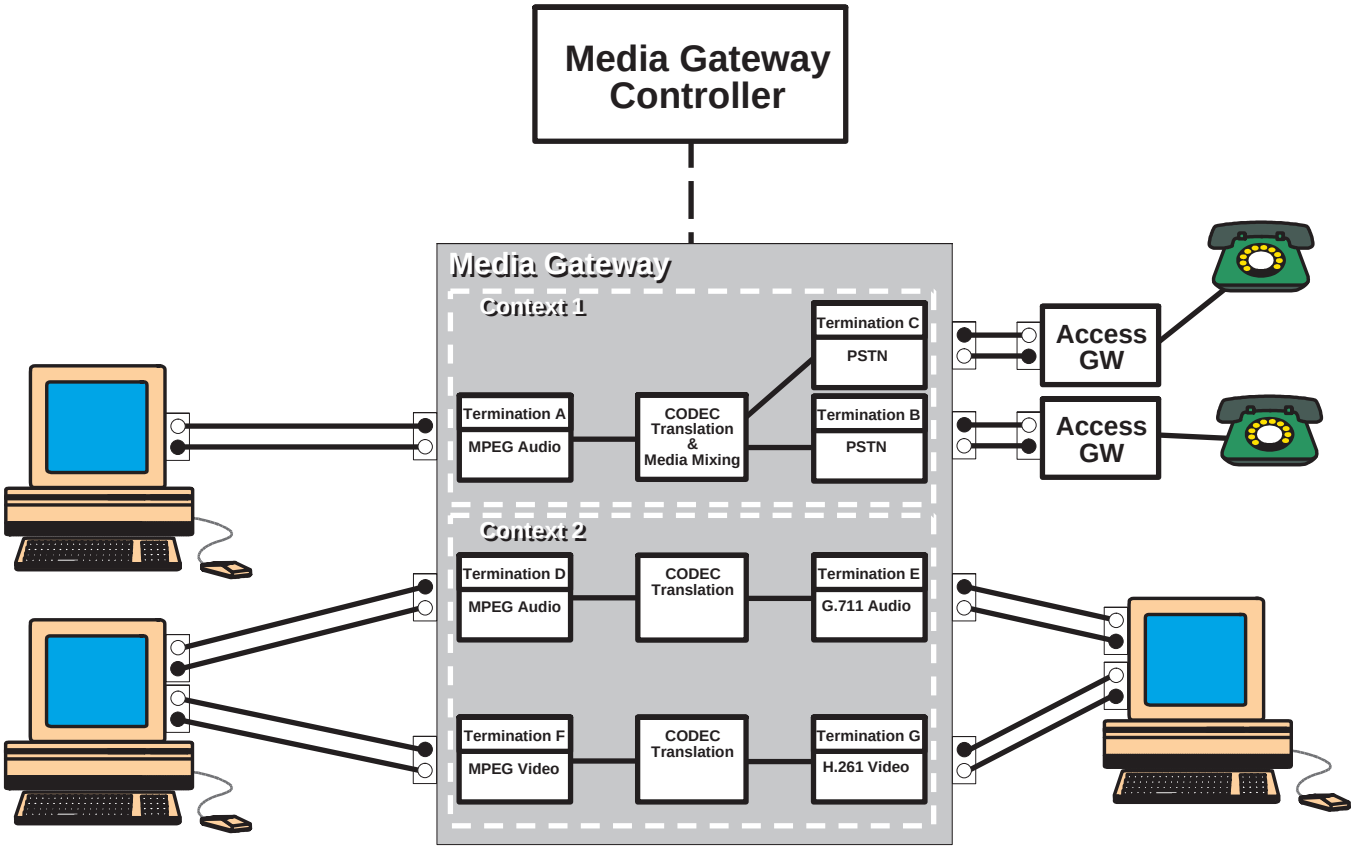
Den øverste Contexten representer en Conference Bridge (se over), hvor en MPEG audio Termination og to PSTN Terminations er knyttet sammen i en multipunktskonferanse ved hjelp av mediagatewayen. I denne Contexten har MG to oppgaver; CODEC translasjon og Media miksing.



Figur 10.1: MEGACO eksempel Connection Model

Nedre del av figuren vises en multimedia Context som involverer både konvertering av audio og video.

Figur 10.2 på neste side viser et eksempel på hvordan disse Contextene kan relaterer seg til fysiske endepunkter.



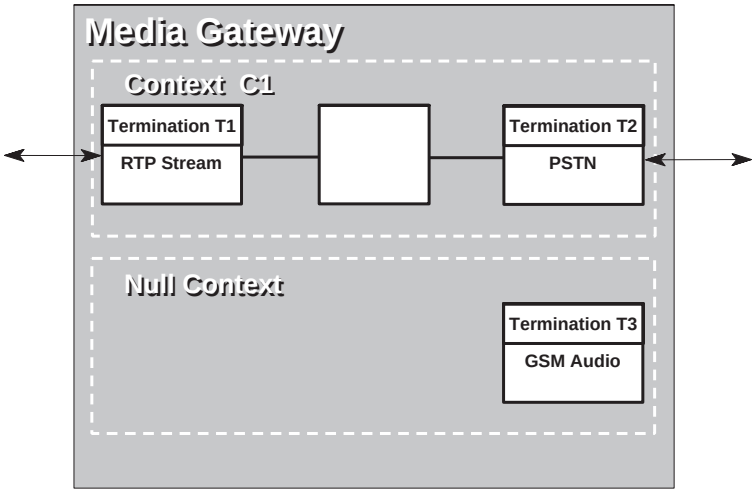
Figur 10.2: MEGACO Context relatert til fysiske endepunkter

Eksempel på bruk av Null Context

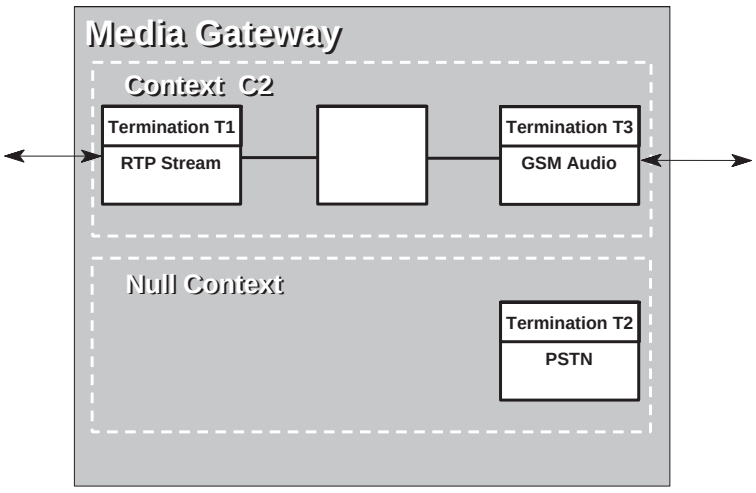
Tidligere er begrepet Null Context beskrevet, og figur 10.3 og 10.4 viser er eksempel på bruk.

I dette eksemplet er to Terminations aktive, T1 og T2, mens T3 for tiden ikke er med i noen aktiv sesjon. T3 er da blitt plassert i Null Context.

I figur 10.4 er T3 igjen aktiv sammen med med T1. T2 er nå satt i Null Context i for å vente på å bli reaktivert.



Figur 10.3: T3 i Null Context, T1 og T2 aktive



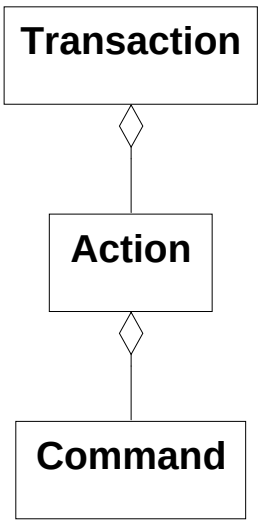
Figur 10.4: T2 i Null Context, T1 og T3 aktive

10.3 MEGACOs transaksjonsmekanisme

MEGACO definerer en transaksjonsmekanisme. Denne mekanismen gjør det mulig å gruppere kommandoer som sendes fra MGC til MG. Oppbygningen av en transaksjon skjer i henhold til modellen på figur 10.5.

En transaksjon (Transaction) består av en eller flere handlinger (Actions). Handlingene er igjen delt opp i en serie av kommandoer (Commands). Det bemerkes at kommandoene i en handling (Action) må relatere seg til samme Context.

En slik oppdeling som beskrevet over, gjør at kommandoer kan få felles skjebne; handlinger og kommandoer gruppert i en transaksjon blir atomære. Det vil si at enten er *hele* transaksjonen vellykket, eller så feiler *hele* transaksjonen.



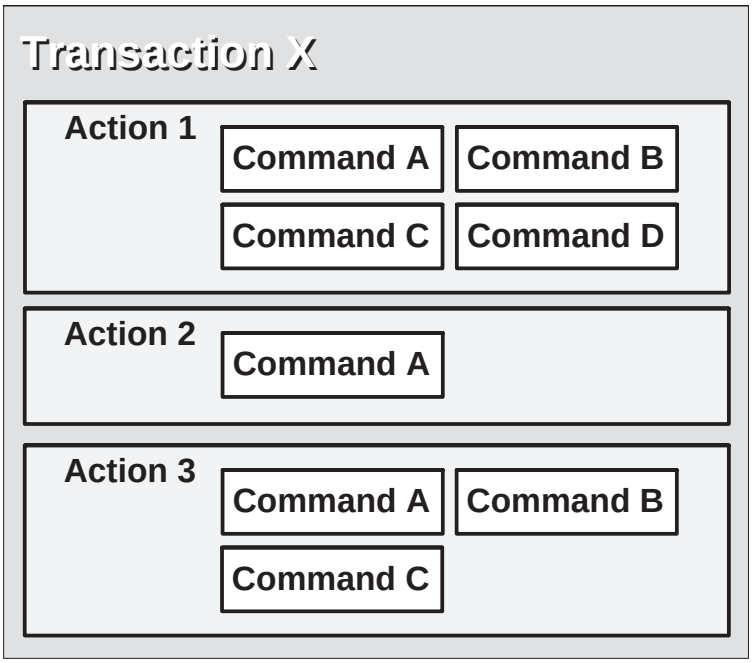
Figur 10.5: Modell for MEGACOs transaksjoner

Figur 10.6 på neste side viser et eksempel for å illustrere hvordan Transactions, Actions og Commands relaterer seg til hverandre.

10.4 Anvendelsesområder

Siden MEGACO er et resultat av sammenslåingen av MDCP og MGCP vil naturlig nok anvendelsesområdene være de samme som disse protokollene.

MEGACO har en svært generell forbindelsesmodell (Connection Model). Dette gjør det sannsynlig at MEGACO kan brukes i andre områder enn det som var opprinnelige tilsiktet. En av disse områdene er en SIP/H.323 gateway som omtales nedenfor.



Figur 10.6: Eksempel MEGACO transaksjon

10.5 Ting under utarbeidelse

Forløpig er MEGACO i en tidlig konstruksjonsfase og mye av detaljene står igjen å spesifisere. Dette er blant annet hvordan mediakapabiliteter skal utveksles mellom MGC og MG. Noen mener at H.245 er for tung, mens andre mener at SDP blir for enkel. En av manglene med SDP som er påpekt er at det ikke er mulig å bygge opp kapabilitetssett bestående av logiske uttrykk som OG og ELLER.

Andre påpeker at siden MEGACO er en intern protokoll i et distribuert system, er kanskje hverken SDP eller H.245 den beste løsningen. Det trekkes fram her at det mest ideelle kunne være å spesifisere en egen mekanisme for kapabilitetsutveksling i MEGACO.

Dette og andre ting er områder som ligger i støpeskjeen i dag. Blant annet er det lagt frem et draft som foreslår at både API spesifikasjon og syntaks for MEGACO bør lages i OMG-IDL. Dette vil gjøre den distribuerte tankegangen i MEGACO enda mer fremtredene.

Det forventes at MEGACO er nærmere en avklaring på disse temaene i løpet av høsten 1999.

10.6 MEGACO anvendt som SIP/H.323 gateway

I et slikt scenario vil det foreligge omtrent de samme argumentene som for MDCP i kapittel 9.3 på side 101. Altså den store nytten kommer først når man må foreta en CODEC translasjon på veien fra mottaker til avsender. Men det er et moment som kommer inn her som ikke fremtrer så klart i MDCP; hvis en MEGACO gateway implementeres med tanke på at den *både* skal få funksjonalitet som gateway og Conferencing Bridge (i H.323 sammenheng; MCU), vil fortjenesten av å bruke en MEGACO løsning bli større. Man får mer funksjonalitet inn i samme løsningen.

10.7 Konklusjon MEGACO

MEGACO virker som en protokoll og et konsept med et stort potensiale innen området med distribuerte gatewayer. Bare det at krefter innen IETF og ITU har gått i sammen om å lage *en* protokoll som forener MDCP og MGCP virker lovende.

Den generell forbindelsesmodellen som er valgt gjør det svært trolig at MEGACO kan brukes i andre sammenhenger enn de som er påtenkt i utgangspunktet. Dette i tillegg til de tankene som er gjort rundt OMG-IDL, gjør at MEGACO har stort potensiale som en distribuert løsning for gatewayer.

Selv om MEGACO har et stort potensiale står det fortsatt igjen mye å bli enig om og å spesifisere. Dette gjør at MEGACO er et stykke fra å kunne være klar for implementasjon.

Kapittel 11

Utvidelse av gatewaymodellen

> Brumm visste hva han mente, men siden han var en bjørn med Bare Liten Forstand, så kunne han ikke finne de riktige ordene.
>
> — *A. A. Milne: Huset på Bjørnehjørnet*

GATEWAYMODELLEN som er blitt presentert tidligere i kapittel 6.2 har ikke hatt mulighet for noen form for mediakonvertering. Oversetting av mediaformater har vært et aspekt som ikke er berørt i modellen, og dette er områder som krever løsninger som er godt gjennomtenkte og har høy ytelse. Vi snakker her om enheter som må kunne oversette fra det ene mediaformatet til det andre i tilnærmet sanntid. Som en naturlig følge av dette er mediagatewayer i høy grad hardwarenære og spesialiserte enheter.

I dette kapittelet presenteres en utvidet modell av SIP/H.323 gatewayen som håndterer mediatranslasjon. Denne modellen vil basere seg på en tankegang om en distribuert gatewayløsning. Dette innebærer at den primære oppgaven til gatewayen er å oversette mellom SIP og H.323, mens eventuell mediatranslasjon vil foregå i enheter som er utenfor selve gatewayen.

11.1 Presentasjon av konseptet

Ideen bak en distribuert gateway er motivert av det faktum at oversetting av signalering mellom SIP og H.323 er adskillig mindre prosessorkrevende enn mediatranslasjon.

Vi ville forsøke å lage en modell som baserte seg på frittstående mediagatewayer som har et felles grensesnitt for kontroll. For å få til dette trengs en protokoll som kan uttrykke et felles språk for de enhetene som skal kommunisere sammen. Det trengs med andre ord en protokoll for å styre

den distribuerte mediagatewayen. Tre slike protokoller er blitt beskrevet tidligere; MGCP, MDCP og MEGACO.

Av disse protokollene er MEGACO-protokollen valgt som et utgangspunkt for den løsningen som presenteres her. Årsaken er at denne protokollene representerer en sammenslåing av MGCP og MDCP. MEGACO er dermed støttet både av IETF og ITU.

Som beskrevet i kapittel 10 på side 103, deler denne protokollen opp en distribuert gateway i to hovedbestanddeler:

- En eller flere mediagatewayer (MG).
- En Media Gateway Controller (MGC).

MGC er den enheten som har ansvaret for å sørge for at endepunkt blir assosiert med en MG, å vedlikeholde og overvåke tilstanden på mediatranslasjonen.

Dette konseptet passer godt overens med vår gatewaymodell, og gjør at å utvide den med mediatranslasjon kan realiseres ved å implementere en MGC inn i modellen.

11.2 Utvidelse av modellen

Figur 11.1 på neste side viser hvordan utvidelsen av klassehierarkiet vil se ut.

I dette klassedigrammet er det ingen andre forskjeller fra modellen på side 50 enn at klassen MediaGatewayController og noen metoder i Gatewayklassen er lagt til. Grunnen til at det ikke er flere endringer involvert er designet som er lagt til grunn. Designet tillater at Gatewayobjektet har full oversikt over mediakapabilitetene gjennom objektet MediaDescriptor.

Dette gjør at det er *kun* Gatewayklassen som trenger mer funksjonalitet i forhold til den tidligere modellen. Følgende nye metoder må legges til:

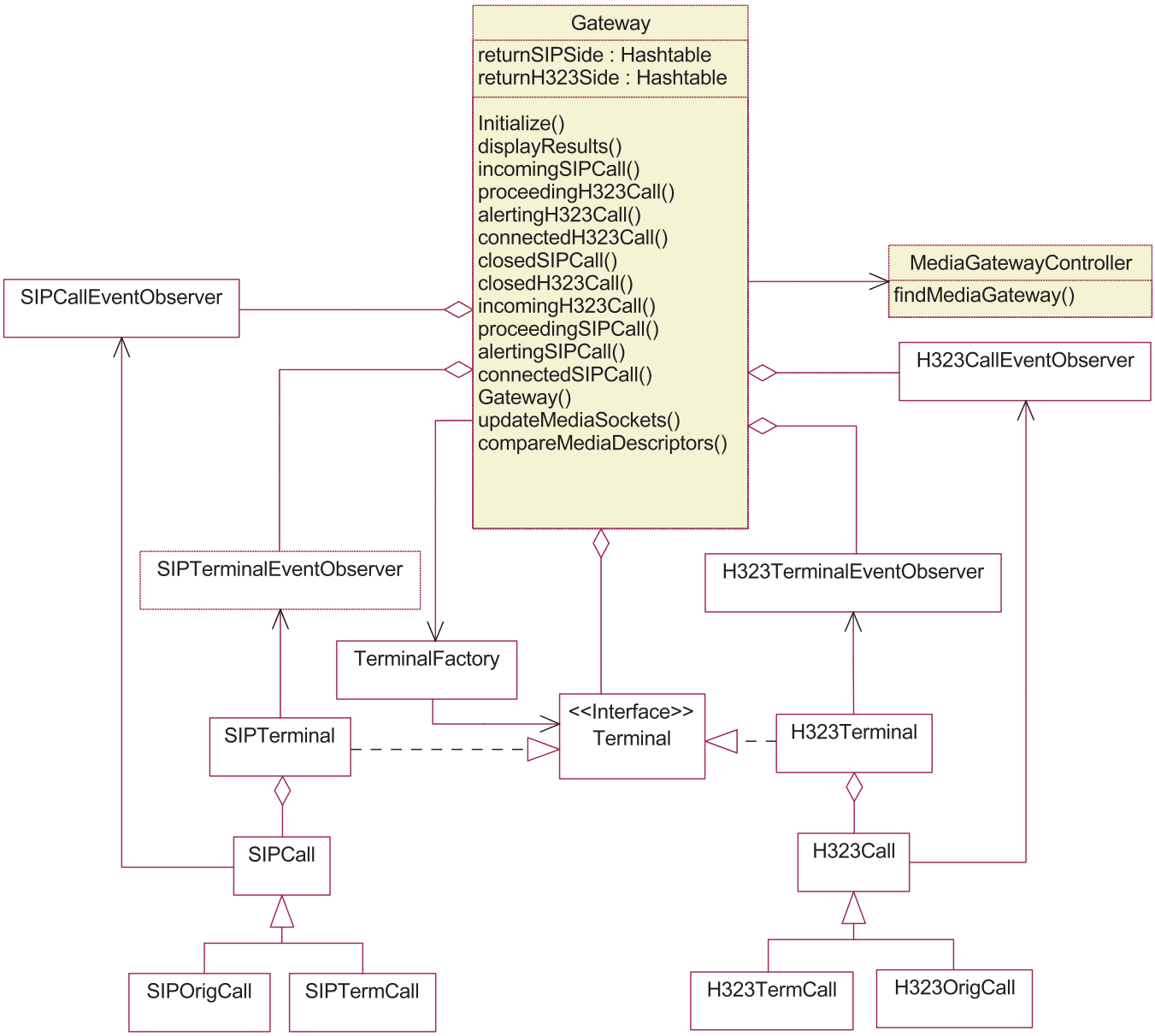
compareMediaDescriptors() Denne metoden sammenligner to MediaDescriptors for å sjekke om mediatypene er sammenfallende.

updateMediaSockets() Etter å ha fått returnert IP adresser og portnummer fra MediaGatewayController kalles denne metoden for å formidle de respektive *socket*[1] til endepunktene.

Klassen MediaGatewayController er ansvarlig for all nødvendig signalering med mediagatewayene, og trer altså inn i rollen som MGC beskrevet over. MediaGatewayController vil dermed bruke MEGACO protokollen for å kommunisere med de anvendte mediagatewayene.

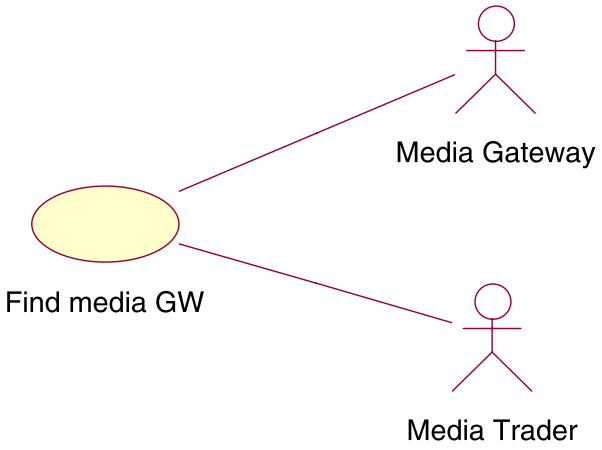
Således må MediaGatewayControllerklassen implementere det av MEGACO [CHG+99] protokollen som er nødvendig for kunne utføre mediatranslasjon.

[1] En socket utgjør et par bestående av en IP adresse og et portnummer. Dette beskriver f.eks. et punkt hvor en CODEC nås.



Figur 11.1: Klassediagram utvidet gatewaymodell

11.3 Et konkret brukstilfelle



Figur 11.2: Use case Find Mediagateway

Brukstilfellet i figur 11.2 vil konkretisere og klargjøre de egenskapene som er tiltenkt utvidelsen av modellen.

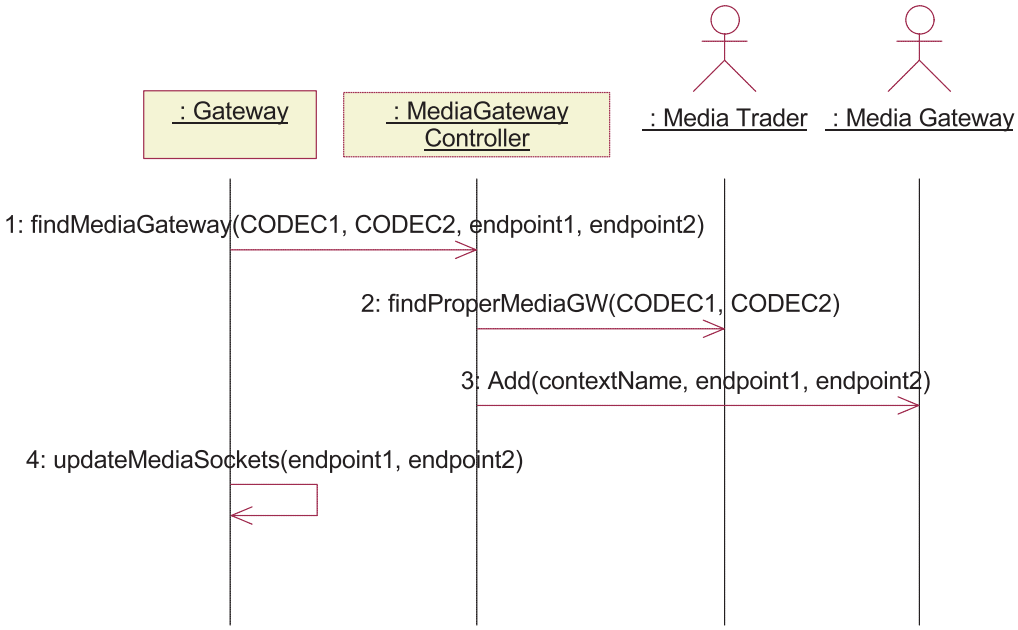
I dette brukstilfellet har Gatewayobjektet allerede oppdaget at endepunktene ikke har noe felles mediaformat ved hjelp metoden `compareMediaDescriptors()`.

Sekvensdiagrammet på figur 11.3 på neste side viser hvordan forløpet er videre. Først vil Gatewayobjektet kalle metoden `findMediaGateway`. Som parameter gis mediaformatene det skal konverteres mellom. MediaGatewayControllerobjektet vil så finne riktig MG på en "velkjent måte". Hvordan MGC skal kjenne til tilgjengelige MG'er går ikke fram av [CHG+99]. Derfor har vi valgt en løsning hvor tilgjengelige MG'er finnes på en Media Trader. På denne enheten er MG'er registrert etter hvilke mediaformater de kan konvertere mellom. MediaGatewayControllerobjektet forespør Media Trader med metodekallet `findProperMediaGW`, og får returnert en referanse til en MG som svarer til forespørselen. Deretter vil MediaGatewayControllerobjektet opprette en MEGACO Context[2] på MG'en med VoIP endepunktene som er involvert i det aktuelle anropet. MediaGatewayControllerobjektet vil så returnere adressene til de respektive CODEC inngangene i MG'en tilbake til Gatewayobjektet. Den enkelte mediasocket blir så formidlet til det riktige SIP eller H.323 endepunktet med metoden `updateMediaSockets`.

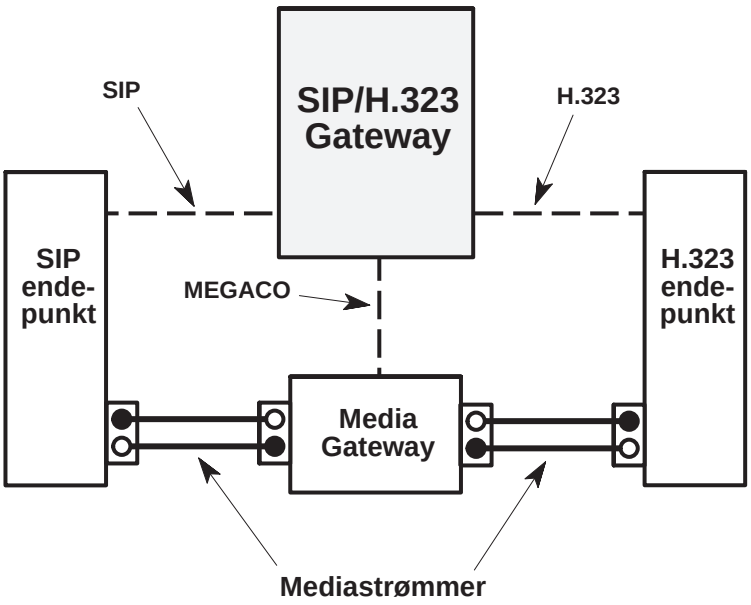
Nå vil henholdsvis SIP og H.323 endepunktene sende sine mediastrømmer via en MG, men de vil ikke være oppmerksom på dette mellomleddet. Det er det Gateway- og MediaGatewayControllerobjektet som holder rede på, og sørger for å videreformidle eventuelle tilstandsendringer.

Figur 11.4 viser hvordan signaleringen og mediastrømmene vil foregå.

[2]En Context kan sammenlignes med *sesjonsbegrepet*. Altså de endepunktene som inngår i



Figur 11.3: Sekvensdiagram for å finne en mediagateway



Figur 11.4: Signalering og mediastrømmer i SIP/H.323 GW

en sesjon.

11.4 Evaluering av løsningen

I dette kapittelet har vi beskrevet et forslag til hvordan mediatranslasjon kan løses uten å ha en mediagateway implementert inn i SIP/H.323 gatewayen. Løsningen bygger på ideen om en distribuert gateway med forskjellige spesialiserte enheter. Den protokollen som brukes for kommunisere mellom disse distribuerte enhetene er MEGACO. Ved å anvende seg av dette konseptet reduseres behovet for endringer i designet som er lagt til grunn for SIP/H.323 gatewayen. Hele MEGACO løsningen er implementert i en ny klasse, og Gatewayobjektet spør bare denne klassen etter MG'er og får returnert adressen til dem. Oppgaven til Gatewayobjektet blir da bare å videreformidle dette til de respektive SIP og H.323 endepunktene.

Det må sies at MEGACO er en ny protokoll. Det første stabile draftet av protokollen ble sluppet i august 1999. Dermed vil det finnes få eller ingen kommersielle mediagatewayer som støtter protokollen. Til tross for dette er det sider ved protokollen som tilsier at den vil få større betydning i fremtiden. Et moment er at den tillater leverandører å spesialisere seg på enten å konvertere signalering eller media. Dermed kan MEGACO brukes for å få et felles grensesnitt mot mediagatewayer uavhengig av leverandører. Standarden som sådan åpner også for tredjepartsleverandører som spesialiserer seg på å tilby mediakonvertering på sesjonsbasis. En mediagateway kan da gjøres tilgjengelig via en form for mediatrader, og leies for den enkelte sesjonen.

En annen grunn til at MEGACO kan få stor utbredelse er at den er støttet av både ITU og IETF. Dette gjør at protokollen får ekstra tyngde.

Den løsningen som er presentert i dette kapittelet er en skisse, men har momenter i seg som taler for at den kan være realiserbar. En ting er at den er enkel å implementere som en utvidelse av den opprinnelige gatewaymodellen. I dette ligger at det ikke er nødvendig å røre noe av den underliggende strukturen i modellen for å implementere en mediagatewayløsning. I tillegg har MEGACO en relativ enkel oppbygning som forenkler selve implementasjonen av protokollen. Et siste moment er at denne type løsning forenkler arbeidet ved å lage gatewayer for signalering. Dette er et resultat av at mediatranslasjon kan overlates til komponenter med et standard grensesnitt. Gatewayer for signalering av IP-telefoni kan dermed basere seg på at de fleste anrop bruker samme CODEC, og raskt koble opp en løsning for mediatranslasjon når dette ikke er tilfelle.

Videre utarbeiding av dette konseptet må også innebære at feilsituasjoner og andre hendelsesforløp må undersøkes nærmere. Et eksempel på dette kan være en situasjon hvor en av partene ønsker å skifte mediaformat underveis i sesjonen. Brukstilfellet i dette kapittelet tar kun for seg å finne en mediagateway ved oppsett av samtalen.

Som en konklusjon kan det sies at modellen som er presentert er tiltalende av årsaker nevnt over, og lar seg sannsynligvis implementere. Til tross for dette krever modellen videre bearbeiding for å avklare flere brukstilfeller og hvordan feilsituasjoner skal håndteres.

Kapittel 12

Oppsummering og diskusjon

> «Det kommer bare av at noen ikke har store nok inngangsdører,» sa Brumm grettent.
> «Det kommer bare av,» sa Sprett strengt, «at noen spiser for mye. Jeg tenkte det nok,» sa Sprett, «men jeg ville ikke si det,» sa Sprett, «at en av oss spiste for mye,» sa Sprett, «og jeg visste at det ikke var *meg*,» sa han.
>
> — *A. A. Milne: Ole Brumm*

DETTE kapittelet vil forsøke å trekke noen slutninger av arbeidet som er gjort, og oppsummere innholdet i denne rapporten. Videre vil vi forsøke å sette resultatet vi har kommet frem til i et større perspektiv, og mot slutten diskutere videre arbeid med modellen og det konseptet vi har presentert.

12.1 Oppsummering av oppgaven og rapporten

For å rekapitulere utgangspunktet for oppgaven er det ønskelig å gjenta selve problemstillingen:

> *Er det mulig å realisere en gateway mellom SIP og H.323 basert på abstraksjon av funksjonaliteten i de respektive protokollene?*

Denne problemstillingen krever en god del bakgrunnskunnskap. Det er blant annet nødvendig med kjennskap til protokollene SIP og H.323, og deres anvendelsesområde.

Det er dette de innledende kapitler forsøker å formidle. Leseren får i kapittel 2 en introduksjon til begrepet VoIP. Videre gis et kort sammendrag av

historikken bak begrepet og hvilke motiver som ligger til grunn for ønsket om å anvende IP som kjerneteknologi for telefoni. Det pekes på fire grunnleggende motivasjoner for dette.

- Rimelige telefonsamtaler.
- Lette utviklingen av tjenester relatert til telefonnettet.
- Vedlikehold av et pakkebasert nett koster vesentlig mindre enn vedlikehold av det tradisjonelle linjesvitsjede nettet.
- Nye muligheter for tjenesteinovasjon og integrasjon med andre typer datatjenester.

De tre nederste er aspekter som virker svært tiltalende på teleoperatørene. Det første punktet er en vesentlig faktor for brukerene. Disse punktene kan tilsammen være en pådriver i utbredelsen av VoIP.

Konsekvensene av å bytte ut kjerneteknologien i dagens telekommunikasjonsnett er, som kapittel 2 påpeker, store. Derfor vil det ta lang tid og kreve store ressurser å oppgradere dagens telenett. I denne perioden fordres det løsninger som muliggjør sameksistens med gamle systemer.

Basiskunnskap om VoIP og protokollene SIP er helt essensielle for å forstå modellen som ligger til grunn for oppgaven. Kapitlene 3, 4 og dels kapittel 5 er tilegnet dette formålet. I disse kapitlene fremstilles de respektive protokollene på en slik måte at leseren har en mulighet til å forstå tankegangen og de grunnleggende konseptene bak protokollene. Kapittel 3 summerer opp H.323 som ITUs bidrag til VoIP, og i ITU tradisjonen tro tar standarden for seg mange aspekter. Dette fører til at standarden er svært omfattende, men sikrer samtidig at forskjellige H.323 løsninger kan kommunisere med hverandre. SIP, slik det påpekes i kapittel 4, har aldri tatt mål av seg til å dekke et så stort område som H.323. Dette resulterer i en standard som det hverken er krevende å forstå eller implementere. En av fordelene ved måten SIP er spesifisert er at det er enkelt å legge til nye momenter i protokollen eller teste ut for eksempel nye CODEC'er uten å bryte standarden. Dette går på bekostning av interoperabiliteten mellom ulike SIP løsninger. Årsaken til dette er, som kapittel 4 nevner, at det ikke er definert et lite felles løsningsrom som alt SIP utstyr må støtte.

Kapittel 6 på side 47 skildrer den modellen vi har utviklet for en SIP/H.323 gateway. ATK er en viktig bestandel som ligger til grunn for mye av modellen. Den sentrale funksjonen til ATK i modellen gjør at den vies atskillig oppmerksomhet i dette kapittelet. Her tas både ATKs design og virkemåte opp i detalj. Virkemåten blir illustrert ved hjelp av eksempler. I tillegg beskrives hvordan SIP kan implementeres inn i ATKs klassehierarki. Med dette sørger ATK for mye av den abstraksjonen vi søker etter i gatewayløsningen.

Den mest sentrale delen av kapittelet beskriver modellen og de forutsetningene som ligger til grunn. Vi ønsket å se på det konseptuelle ved å lage en gateway basert på abstraksjon av protokollfunksjonalitet. Derfor var vi nødt til å begrense modellen til å omfatte noen valgte brukstilfeller med gitte forutsetninger. De valgte brukstilfellene ble:

- Initiering av gatewayen.
- Innkommende anrop fra en SIP terminal.
- Innkommende anrop fra en H.323 terminal.
- Terminering av anrop fra en SIP terminal.
- Terminering av anrop fra en H.323 terminal.

Disse brukstilfellene står utførlig beskrevet i kapittel 6. Resultatet av dette, sammen med ATKs design, ble klassediagrammet som er vist på side 61. Dette klassehierarkiet vil tilby den abstraksjonen som vi søker etter.

På slutten av kapittelet diskuteres løsningen, og hvor mye funksjonalitet som blir borte i generaliseringen.

For å teste om modellen er realiserbar ønsket vi å implementere deler av den. Denne prosessen er beskrevet i kapittel 7. Her har vi beskrevet det vi rakk å implementere og de problemene som oppsto underveis. Vi konkluderer i kapittelet med å anta at modellen er realiserbar, men at det gjenstår en del arbeid for å få et system som fungerer utenfor et lukket testmiljø.

Modellen beskrevet så langt har ikke tatt for seg hvordan eventuelle CODEC translasjoner kan foregå. Vi ville skissere en løsning for dette som passet best mulig inn i vår modell. En protokoll for distribuerte mediagatewayer viste seg å nettopp være en slik løsning. Kapittel 8, 9 og 10 beskriver og evaluerer tre slike alternativer; MGCP, MDCP og MEGACO. Vi valgte MEGACO fordi den representerer en sammenslåing av MGCP og MDCP, og fordi den er valgt som en felles protokoll for IETF og ITU.

Kapittel 11.4 viser hvordan MEGACO er anvendt for å utvide modellen til å takle mediatranslasjon. Implikasjonene dette har på modellen er minimale. Det som trengs er en klasse som implementerer en MEGACO Media Gateway Controller, og noen nye metoder i Gatewayklassen.

Vi konkluderer dette kapittelet med at den utvidete modellen ikke er realiserbar i dag fordi MEGACO er i en så tidlig fase i utviklingen, og dermed støtter ingen mediagatewayer standarden. Til tross for dette mener vi at en MEGACO løsning har et stort potensiale for fremtiden. Noe av grunnen til det er at dette er et felles satsningsområde for IETF og ITU, og at leverandører av mediagatewayer vil ha interesser i å få etablert et standard grensesnitt for sine produkter.

12.2 Evaluering og diskusjon av løsningen

Et sentralt aspekt ved oppgaven er vi at har valgt å abstrahere protokollfunksjonaliteten. Dette valget vil implisere hvilke anvendelsesområder som er aktuelt for SIP/H323 gatewayen. Abstraksjon innebærer generalisering som igjen impliserer at en gir slipp på det særegne. For gatewayløsningens del innebærer det at situasjoner som er særegne i den enkelte protokoll blir ofret til fordel for den funksjonaliteten som er felles. Et eksempel på funksjonalitet som ikke er direkte overlappende er hvordan forhandling om

mediakapabiliteter forgår i H.323. I H.323 kan det bygges opp komplekse logiske uttrykk for mediapreferanser. Dette finnes det ingen mulighet for i SIP, og denne funksjonaliteten kan gå tapt i en konvertering.

Det felles løsningsrommet for SIP og H.323 kan antakelig utvides ved at det etableres en-til-mange og mange-til-en relasjoner mellom de ulike funksjonalitetene i protokollene. For eksempel kan situasjonen beskrevet over med mediaforhandling, splittes opp i flere INVITE meldinger med enkle uttrykk for mediapreferanser.

Til tross for dette vil den gatewayløsningen vi har skissert først og fremst være myntet på det gjengse tilfellet. Med dette menes de mest vanlige situasjonene som forekommer i en sesjon mellom VoIP endepunkter. Dette medfører at den løsningen vi har beskrevet antakelig passer best for miljøer som etterspør en enkel løsning for at SIP og H.323 terminaler kan ha muligheten til å kommunisere med hverandre. En kan da se for seg gatewayløsningen som en komponent i enten et SIP eller H.323 miljø som takler enkle VoIP sesjoner på tvers av protokollene. Vi ser for oss at disse miljøene har et relativt lite antall samtidige sesjoner gjennom SIP/H.323 gatewayen. Dermed utelukkes miljøer som preget av et høyt antall samtidige sesjoner. Dette impliserer at den modellen vi har lagt frem neppe er egnet i sentrale telekommunikasjonsløsninger med tusenvis av samtidige anrop. Det er mer naturlig å se for seg denne SIP/H.323 gatewayen som en komponent i et H.323 eller SIP domene[1], og at den muliggjør kommunikasjon med det motsvarende protokollmiljøet.

Et alternativ til abstraksjon som vi har valgt som paradigme hadde vært en mer todelt gatewayløsning. En del ville tatt seg av konverteringen fra SIP til H.323, mens en annen uavhengig del ville vært ansvarlig for konvertering fra H.323 til SIP. En slik løsning ville antakelig skalere bedre til store forhold enn den løsningen vi har valgt. Årsaken til dette er at en ikke får noe redundans i form av å først abstrahere funksjonalitet til et felles språk, for deretter spesialisere den respektive funksjonaliteten ned i protokollimplementasjonen. Hvis denne løsningen i tillegg kan gjøres tilstandsløs vil dette gjøre at den har muligheter til å skalere til store forhold som krever stor kapasitet. Et anvendelsesområde kan da være som en gatewayløsning plassert sentralt i et telekommunikasjonsnett.

En annen fordel med denne løsningen er at implementasjon av ny funksjonalitet som bare påvirker translasjon fra SIP til H.323 eller omvendt, kun må gjøres et sted. En ulempe med en slik løsning vil være at grunnleggende forandringer i protokollene må implementeres i begge konverteringsdelene av gatewayen.

Hvis en skal betrakte den modellen vi har foreslått litt på avstand kan en godt si at den opererer med en synsvinkel som har et sterkt teknologisk og pragmatisk preg. Vi forsøker å løse det basale problemet om hva som må til for å etablere en samtale mellom endepunktene A og B når de bruker forskjellige protokoller. Politiske, økonomiske eller andre aspekter som måtte påvirke modellen har ikke vært et tema. Det kan vise seg at en nærmere kartlegging av modellens omgivelser, eller *enterprise*, kan avsløre vesentlige momenter som modellen har oversett.

[1] Med dette mener vi en størrelse som tilsvarer en større bedrift som for eksempel Statoil.

En mulighet for å identifisere faktorene som påvirker systemet kan være å anvende *Enterprise Viewpoint* i RM-ODP [IT95]. Jan A. Audestad skriver i [Aud99, kapittel 4] hvilke faktorer som skal identifiseres. Dette er momenter som vil påvirke systemet på ulik måte. Målet er å avdekke alle kilder som har innvirkning. Det må understrekes at hverken SIP eller H.323 har noe definert enterprise i henhold til RM-ODP, men teknikken beskrevet over kan være et hjelpemiddel for å få klarhet i forholdene som påvirker gatewayløsningen, og dermed avsløre svakheter med SIP/H.323 gatewaymodellen slik den fremstår i dag.

Hvordan samtaler tas ned er et eksempel på et område hvor protokollenes ulike enterprise påvirker gatewayløsningen. Vi oppdaget under modelleringen av gatewayen at SIP ikke berører hvordan media tas ned, mens i H.323 er dette veldefinert. SIP definerer altså slike momenter til å være utenfor sitt enterprise, men dette er temaer som må kunne håndteres i en gatewayløsning. Spesielt når gatewayen skal håndtere konvertering av mediaformater. Gatewayen må da implementere funksjonalitet som sørger for at mediakanaler blir tatt ned når samtalen avsluttes. Et annet eksempel på forskjeller i SIP og H.323s enterprise er hvor vidt reservasjonsmekanismer er et tema. I H.323 vies dette oppmerksomhet, mens det i SIPs nåværende versjon er dette et lite omtalt emne. Med all respekt må det sies at dette er et tema som det arbeides med i SIP miljøet, og [MRM+99] viser resultater fra dette arbeidet.

12.3 Videre arbeid

Flere steder i oppgaven er det antydet at det står igjen en del arbeid med SIP/H.323 gatewayen. Dette avsnittet vil oppsummere de momentene vi mener må fullføres for å få et system som vil fungere i et virkelig miljø.

Det første som bør gjøres er å implementere ferdig modellen slik den ser ut i dag. Dette innebærer å sluttføre arbeidet av SIP delen i ATK, og implementere selve Gatewayklassen.

Når dette er ferdig må modellen utvides med mer funksjonalitet. I dette ligger det at gatewayløsningen må kunne håndtere feilsituasjoner, og at det modelleres funksjonalitet utover de brukstilfellene vi har sett på. Eksempler på slik funksjonalitet kan være tilleggstjenster som viderekobling og samtale venter.

Hvis en slik videreutviklet modell implementeres burde den bli testet med kommersielle SIP og H.323 klienter på hver sin side av gatewayen. Dette burde gi en god pekepinn for hvordan gatewayløsningen vil oppføre seg under drift i et virkelig miljø.

Et moment som må testes ut før SIP/H.323 gatewayen kan settes ut i et virkelig miljø er hvordan den skalerer og løsningens ytelse. Dette kan gjøres ved at man simulerer et økende antall samtidige anrop mot gatewayen, og observerer når ytelsen blir for dårlig. Denne måten gir en bra antydning for hvilken ytelse løsningen har, og hvor godt den skalerer. Det må bemerkes at en slik simulering gir en god antydning, men at systemet først kan bli skikkelig prøvd ut i et virkelig miljø med ekte samtaletrafikk.

Vi presenterte i avsnitt 11.4, en utvidelse av modellen basert på MEGACO. Dette er et arbeid som krever at nødvendig funksjonalitet modelleres inn i løsningen etter hvert som MEGACO standarden modnes og leverandører av mediagatewayer støtter den. Et poeng kunne være å modellere klassen MediaGatewayController som et Interface. Da ville det vært mulig å lage implementasjoner av denne klassen som støtter dagens leverandørspesifikke grensesnitt mot mediagatewayer. Samtidig ville en beholde SIP/H.323 gatewayens grensesnitt mot klassen MediaGatewayController.

Et annet perspektiv som hadde vært interessant er å sette SIP/H.323 gatewayen inn i sammenhenger med multimediakonferanser med flere deltagere. Hvordan dette ville samspille med gatewayløsningen, og hvor en eventuell konferansekontroll skal være i forhold til gatewayen vil være spørsmål som er naturlig å stille i en slik sammenheng. Dette vil kreve løsninger for mediamiksing, *Floorcontrol*[2] og mer detaljerte løsninger for samtalekontroll.

Å se gatewayen i perspektiv som involverer trådløse elementer og mobiltet er også et interessant tema. Hvor vidt dette vil påvirke gatewayens virkemåte er et emne som kan være verdt å studere nærmere.

[2]Mekanisme som sørger for at lyd og bilde sendes av den som for tiden snakker eller er tilordnet er lederrolle i konferansen.

Bibliografi

[ADE+99] Mauricio Arango, Andrew Dugan, Isaac Elliott, Christian Huitema og Scott Pickett. Media Gateway Control Protocol (MGCP). *draft-huitema-megaco-mgcp-v01r1-05.txt*, 21. februar 1999.

[AH98] Mauricio Arango og Christian Huitema. Simple Gateway Control Protocol (SGCP). *draft-huitema-sgcp-v1-02.txt*, 30. juli 1998.

[Aud99] Jan A. Audestad. *Distributed Processing in Telecommunications*. UNIK, 1999.

[CHG+99] Fernando Cuervo, Brian Hill, Nancy Greene, Christian Huitema, Abdallah Rayhan, Brian Rosen, John Segers Megaco protocol. *draft-ietf-megaco-protocol-05.txt*, 21. september 1999.

[Dat98] DataBeam. A Primer on the H.323 Series Standard, Mai 1998.

[deC99] Linden deCarmo. Internet Telephony Protocols. *Dr. Dobb's Journal*, side 30–39, Juli 1999.

[DRS99] Mark Day, Jonathan Rosenberg og Hiroyasu Sugano. A Model for Presence and Instant Messaging. *draft-ietf-impp-model-03.txt*, Mai 1999.

[ETS99] ETSI. About ETSI. *ETSI*, September 1999. *http://www.etsi.org/*.

[Gro99] MEGACO Working Group. MEGACO/H.GCP. *Preliminary draft*, 25. juni 1999.

[HCMS99] Christian Huitema, Jane Cameron, Petros Mouchtaris og Darek Smyk. An Architecture For Residential Internet Telephony Service. *IEEE Inernet Computing*, side 73–82, May/June 1999.

[HJ98] M. Handley og V. Jacobson. SDP: Session Description Protocol. *Request for Comments: 2327*, April 1998.

[HSSR99a] Handley, Schulzrinne, Schooler og Rosenberg. SIP: Session Initiation Protocol. *draft-ietf-mmusic-sip-12.ps*, 15. Januar 1999.

[HSSR99b] M. Handley, H. Schulzrinne, E. Schooler og J. Rosenberg. SIP: Session Initiation Protocol. *Request for Comments: 2543*, Mars 1999.

[Hui99] Christian Huitema. SGCP – Simple Gateway Control Protocol, 1999. Telcordia Tecnologies.

[IT95] ITU-T. Reference Model of Open Distributed Processing. RM-ODP. *ITU-T X.901*, Mai 1995.

[ITU93a] ITU-T. *ISDN user-network interface layer 3 specification for basic call control*. International Telecomminication Union, 1993.

[ITU93b] ITU-T. Q-series Intelligent Network Recommondation Structure. *Rec. Q.1200*, Mars 1993.

[ITU98a] ITU. International Telecommunication Union. *ITU Telecom Information Exchange Services*, 1998. *http://www.itu.int/home/Aboutsite/index.html*.

[ITU98b] ITU-T. *H.225.0 – Call Signalling Protocol and Multimedia Stream Packetization for Packet Based Multimedia Communication Systems*. Series H: Audiovisual and Multimedia Systems. International Telecomminication Union, 17 April 1998.

[ITU98c] ITU-T. *H.245 – Control protocol for multimedia communication*. Series H: Audiovisual and Multimedia Systems. International Telecomminication Union, Februar 1998.

[ITU99] ITU-T. *H.323 – Packet-based multimedia communications systems*. Series H: Audiovisual and Multimedia Systems. International Telecomminication Union, version 3 utgave, Mai 1999.

[KHN99] Lill Kristiansen, Jan Holm og Borge Nilsen. Splitting the GK and some reasons to do so. *Tiphon 13*, 30. April 1999. L.M. Ericsson.

[KO97] Stein Krogdahl og Chistian-Emil Ore. *Kompendium 47, Om syntaks og syntaksanalyse*. UNIPUB, Akademika as, August 1997.

[MH98] Eirik Maus og Peter D. Holmes. Sammenligning av SIP og H.323. *IMEDIA*, IMiS Kernel(10), Oktober 1998.

[MRM+99] W. Marshall, K. Ramakrishnan, E. Miller, G. Russell, B. Beser, M. Mannette, K. Steinbrenner, D. Oran, J. Pickens, P. Lalwaney, J. Fellows, D. Evans, K. Kelly og F. Andreasen. DCS Architecture. *draft-dcsgroup-mmusic-arch-00.txt*, Juni 1999.

[PD96] Larry L. Peterson og Bruce S. Davie. *Computer Networks: A System Approach*, kapittel 9, side 452. Morgan Kaufmann Publishers Inc, 1996.

[PPY+99] Christos A. Polyzois, K. Hal Purdy, Ping-Fai Yang, David Shrader, Henry Sinnreich, François Ménard og Henning Schulzrinne. From POTS to PANS, A Commentary on the Evolution to Internet Telephony. *IEEE Internet Computing*, side 83–91, Mai-Juni 1999.

[SBW+99] Paul Sijben, Mike Buckley, John Wachter, John Seegers og Chia Li. Media Device Control Protocol. *draft-sijben-megaco-mdcp-01.txt*, 9. Februar 1999.

[Sch99] Henning Schulzrinne. Converging on Internet Telephony, Service for Telecom Version 2. *IEEE Internet Computing*, side 40–43, Mai-Juni 1999.

[Sec94] IETF Secretariat. The tao of IETF – A Guide for New Attendees of the Internet Engineering Task Force. *Request for Comments: 1718*, November 1994. *http://www.ietf.org/rfc/rfc1718.txt*.

[Sij98] Paul Sijben. MDCP, Media Device Control Protocol. *Tiphon 11*, 18. Desember 1998. Lucent Technologies.

[Sij99] Paul Sijben. Toward network independent media processing: Media Device Control Protocol. *Slide Presentation*, 10. Januar 1999. Lucent Technologies.

[Siv98] F. Sivertsen. *Access Toolkit User Guide*. Ericsson AS, 12. august 1998. For internal Ericsson use only!

[TCR98] P. Tom Taylor, Pat R. Calhoun og Allan C. Rubens. IPDC Base Protocol. *draft-taylor-ipdc-00.txt*, Juli 1998.

[TIP99] TIPHON. Telecommunications and Internet Protocol, Harmonization Over Networks. *ETSI*, August 1999. *http://www.etsi.org/tiphon/*.

Vedlegg

Vedlegg A

Forkortelser

Tabell A.1: Forkortelser

AG	Access Gateway
ACF	Admission Confirmation
API	Application Programmers Interface
ARJ	Admission Reject
ARQ	Admission Request
ASN.1	Abstract Syntax Notation 1
ATK	Access Toolkit
BG	Business Gateway
CA	Call Agent
CODEC	COmpressor DECompressor
DARTnet	DARPA Research Testbed network
DCF	Disengage Confirmation
DNS	Domain Name System
DRQ	Disengage Request
DTMF	Dual Tone Multi Frequency
GK	Gatekeeper
IETF	Internet Engineering Task Force
IP	Internet Protocol
ISP	Internet Service Provider
ITU	International Telecommunications Union
LAN	Local Area Network
LDAP	Lightweight Directory Access Protocol
MC	Multipoint Controller
MCU	Multipoint Control Unit
MD	Media Device
MDC	Media Device Controller
MEGACO	MEdia GAteway COntrol
MGC	Media Gateway Controller

fortsetter neste side

fortsatt fra forrige side	
MMoIP	MultiMedia over IP
MP	Multipoint Processor
MDCP	Media Device Control Protocol
MG	Media Gateway
MGC	Media Gateway Controller
MPEG	Moving Picture Expert Group
NAS	Network Access Servers
nas	**n**etwork **a**ccess **s**ervice
POTS	Plain Old Telephony System
PSTN	Public Switched Telephony System
RAS	Registration Admission Status
RFC	Request For Comments
RG	Residential Gateway
RTP	Real-time Transport Protocol
RTCP	Real-time Transport Control Protocol
RTSP	Real Time Streaming Protocol
SAP	Session Announcement Protocol
SCN	Switched Circuit Network
SCP	Service Control Point
SDP	Session Description Protocol
SG	Signalling Gateway
SGCP	Simple Gateway Control Protocol
SIP	Session Initiation Protocol
SS7	Signalling System Number 7
TCP	Transmission Control Protocol
TG	Trunking Gateway
UDP	User Datagram Protocol
UNI	User to Network Interface
VoIP	Voice over IP
WAN	Wide Area Network

Vedlegg B

Kildekode

B.1 Basiskomponenter

B.1.1 UDPClient.java

```
2  //Title:        SIP implementation
   //Version:
4  //Copyright:    Copyright (c) 1999
   //Author:       Rune Schumann
6  //Company:      IFI, UiO
   //Description:  This project is trying to implement basic
8  //features of SIP to demonstrate conseptual
   //functionalities
10 package se.ericsson.eto.mmts.atk.impl.sip;

12 import java.net.*;
   import java.io.*;
14
   /**
16 * This class provides functionality to send a String to a given
   * hostname and portnumber as a UDP packet.
18 */
   public class UDPClient
20 {
     String hostname;
22   public int portnumber;  //Portnumber to send
     public int len;         //Bufferlength
24   protected static int defaultBufferLength = 65507;

26   /**
     * The parameterless constructor.
28   */
     public UDPClient()
30   {
       len = defaultBufferLength;
32   }

34   /**
     * This constructor takes portnumber and hostname as arguments.
36   */
     public UDPClient(String hname, int pnbr)
```

```
38    {
        hostname = hname;
40      portnumber = pnbr;
        len = defaultBufferLength;
42    }

44    /**
      * This constructor takes portnumber, hostname and bufferlength
46    * as arguments.
      */
48    public UDPClient(String hname, int pnbr, int blen)
      {
50      hostname = hname;
        portnumber = pnbr;
52      len = blen;
      }
54
      /**
56    * This function sends a String to a given hostname and portnumber.
      * NOTE! In this version exception are caught anf not send to the caller.
58    * In future systems an errorsystem should deal with exceptions.
      */
60    public void send(String request, String hname, int pnbr)
      {
62      hostname = hname;
        portnumber = pnbr;
64
        try
66      {
          DatagramSocket ds = new DatagramSocket(0);
68        InetAddress ia = InetAddress.getByName(hostname);
          byte[] data = request.getBytes();
70        DatagramPacket outgoing = new DatagramPacket(data, data.length, ia , portnumber);
          ds.send(outgoing);
72      } //end try
        catch (UnknownHostException e)
74      {
          System.err.println(e);
76      } // end catch
        catch (SocketException e)
78      {
          System.err.println(e);
80      } // end catch
        catch (IOException e)
82      {
          System.err.println(e);
84      } // end catch
      }
86
      /**
88    * This function takes a String as an argument and sends it to the prior
      * defined hostname and portnumber.
90    */
      public void send(String request)
92    {
        send(request, hostname, portnumber);
94    }
  }
```

B.1.2 UDPServer.java

```
2  //Title:        SIP implementation
   //Version:
4  //Copyright:    Copyright (c) 1999
   //Author:       Rune Schumann
6  //Company:      IFI, UiO
   //Description:  This project is trying to implement basic
8  //features of SIP to demonstrate conseptual
   //functionalities
10 package se.ericsson.eto.mmts.atk.impl.sip;

12 import java.net.*;
   import java.io.*;
14

16 /**
   * This class will provide simple UDP server functionalites.
18 * After initiated with portnumber the server listens and sends any
   * incomming packets to a respond function. This function does what
20 * ever wanted. At the current time, the task of respond is to return
   * incomming data in a String.
22 */
   public class UDPServer
24 {

26   private DatagramPacket incoming;
     public int portnumber;  //Portnumber to listen at
28   public int len;         //Bufferlength
     protected static int defaultBufferLength = 65507;
30


32
     public UDPServer(int pnbr)
34   {
       portnumber = pnbr;
36     len = defaultBufferLength;
     }
38
     public UDPServer(int pnbr, int blen)
40   {
       portnumber = pnbr;
42     len = blen;
     }
44
     /**
46   * This fuction is waiting for an incomming datagram packet. And returns
     * the data as a String to the caller.
48   * NOTE! The function is blocking and must be called again to wait for
     * another incoming UDP packet.
50   */
     public String waitForIncomming(int pnbr)
52   {
       String message = new String();
54     DatagramSocket ds;
       portnumber = pnbr;
56
       try
58     {
         ds = new DatagramSocket(portnumber);
60       byte[] buffer = new byte[len];
```

```
        try
62      {
          incoming = new DatagramPacket(buffer, buffer.length);
64        ds.receive(incoming);
        }
66      catch (IOException e)
        {
68        System.err.println(e);
        }
70    }
      catch (SocketException se)
72    {
        System.err.println(se);
74    }//end catch
      return(respond(incoming));
76  }

78  public String waitForIncomming()
    {
80    return waitForIncomming(portnumber);
    }
82
//  private String respond(DatagramSocket ds, DatagramPacket dp) //For future use
84  private String respond(DatagramPacket dp)
    {
86    return new String(dp.getData(), 0, dp.getLength());
    }
88 }
```

B.1.3 UDPClientTest.java

```
2 //Title:        SIP implementation
  //Version:
4 //Copyright:    Copyright (c) 1999
  //Author:       Rune Schumann
6 //Company:      IFI, UiO
  //Description:  This project is trying to implement basic
8 //features of SIP to demonstrate conseptual
  //functionalities
10 package se.ericsson.eto.mmts.atk.impl.sip;
  import java.io.*;
12
  public class UDPClientTest
14 {

16   public UDPClientTest()
     {
18   }

20   public static void main(String[] args)
     {
22     String request = args[0];
       UDPClient client = new UDPClient("ETO01089", 1000);
24     client.send(request);
     }
26 }
```

B.1.4 UDPServerTest.java

```
2 //Title:        SIP implementation
  //Version:
4 //Copyright:    Copyright (c) 1999
  //Author:       Rune Schumann
6 //Company:      IFI, UiO
  //Description:  This project is trying to implement basic
8 //features of SIP to demonstrate conseptual
  //functionalities
10 package se.ericsson.eto.mmts.atk.impl.sip;

12 public class UDPServerTest
   {
14
     public static void main(String[] args)
16   {
       String response;
18     UDPServer server = new UDPServer(1000);
       response = server.waitForIncomming();
20     System.out.println(response);
     }
22 }
```

B.1.5 SIPMessage.java

```
//Title:        SIP implementation
//Version:
//Copyright:    Copyright (c) 1999
//Author:       Rune Schumann
//Company:      IFI, UiO
//Description:  This project is trying to implement basic
//features of SIP to demonstrate conseptual
//functionalities
package se.ericsson.eto.mmts.atk.impl.sip;

import se.ericsson.eto.mmts.atk.impl.sip.*;

/**
* This class represent a parsed version of the SIP message.
*/
public class SIPMessage
{
  public String messageType;
//  public SIPMethodType methodType;
  public String methodType;
  public String uri;
  public String version;
  public String statusCode;
  public String reason;
  public String callID;
  public String contentType;
  public String contentLength;
  public String cSeq;
  public String from;
  public String to;
  public String via;
  public String media;

  public SIPMessage()
  {
  }
}
```

B.1.6 SIPMethodType.java

```
2  //Title:        SIP implementation
   //Version:
4  //Copyright:    Copyright (c) 1999
   //Author:       Rune Schumann
6  //Company:      IFI, UiO
   //Description:  This project is trying to implement basic
8  //features of SIP to demonstrate conseptual
   //functionalities
10 package se.ericsson.eto.mmts.atk.impl.sip;
   import java.io.*;
12 import java.util.*;
   import se.ericsson.eto.mmts.toolbox.fundamental.Enumerated;
14
   public final class SIPMethodType extends Enumerated
16 //public final class SIPMethodType
   {
18   public static final SIPMethodType INVITE = new SIPMethodType("INVITE");
     public static final SIPMethodType ACK = new SIPMethodType("ACK");
20   public static final SIPMethodType OPTIONS = new SIPMethodType("OPTIONS");
     public static final SIPMethodType BYE = new SIPMethodType("BYE");
22   public static final SIPMethodType CANCEL = new SIPMethodType("CANCEL");
     public static final SIPMethodType REGISTER = new SIPMethodType("REGISTER");
24
     private SIPMethodType(String messageID)
26   {
       super(messageID);
28   }
   }
```

B.1.7 SIPParser.java

```
2  //Title:        SIP implementation
   //Version:
4  //Copyright:    Copyright (c) 1999
   //Author:       Rune Schumann
6  //Company:      IFI, UiO
   //Description:  This project is trying to implement basic
8  //features of SIP to demonstrate conseptual
   //functionalities
10 package se.ericsson.eto.mmts.atk.impl.sip;
   import java.io.*;
12 import java.util.*;
   import java.text.*;
14 import se.ericsson.eto.mmts.atk.impl.sip.*;

16 /**
   * The main purpose of this class is to provide a very simple
18 * SIP parser.
   * The parser has limited functionality at the current version,
20 * but this can be enhanced by implenting a truely Recursive
   * Decendant parser based on SIP's BNF gramatics.
22 *
   * Note! The parser parses just a subset of the SIP protocol, and completly
24 * overlooks any SDP messages. This parser uses a field "m:" to signal
   * mediacapabilities. This field i NOT a SIP or SDP field. It is just
26 * chosen this solution to simplify the parser.
   */
28 public class SIPParser
   {
30   /**
     * This function separates the lines in a SIP message and
32   * places them as String occurencies in a Vector
     */
34   public Vector separateLines(String message)
     {
36     Vector messageVector = new Vector();
       String element = new String();
38     int start = 0;
       int i;
40     boolean firstWhite = true;

42     for (i=0; i<message.length(); i++)
       {
44       if (message.charAt(i) == '\n')
         {
46         if (firstWhite)
           {
48           element = message.substring(start,i);
             messageVector.add(element);
50           firstWhite = false;
             start = i;
52         }
         }
54       else if (firstWhite == false)
         {
56         start = i;
           firstWhite = true;
58       }
       }
60
```

```
      //The last line has  no newline
62    element = message.substring(start, message.length()); //Let's store the value
      messageVector.add(element);
64
      return messageVector;
66  }

68  /**
    * The function parses each line in the SIP message to obtain
70  * the header name and a value.
    * Note! Induvidual steps has to be taken to process the first
72  * line in the message. This is the request/response line describing
    * the message type (i.e INVITE), URI an SIP version.
74  * When all lines is parsed, the result is returned in a
    * SIPMessage object.
76  * Note 2! For simplicty reasons this function takes for
    * granted that field are separted with ONE space.
78  * Note 3! To simplify parsing for media this example uses
    * a 'm' as the fieldname containing the mediaformat. This
80  * is NOT SIP.
    */
82  public SIPMessage parseLines(Vector messageVector)
    {
84    SIPMessage message = new SIPMessage();
      String line = new String();
86    String fieldname;
      String value;
88    int start = 0;
      int i;
90
      Enumeration e = messageVector.elements();
92
      line = (String)e.nextElement();
94
      //We have to test if the message is a Request or Respons
96    if (line.charAt(0) == 'S') //Has to be a respons! Starting with SIP version
      {
98      parseResponseLine(line, message);
      }
100   else //Not starting with "SIP/X.XX", has to be a request!
      {
102     parseRequestLine(line, message);
      }
104
      //The message body is now starting, split lines in fieldname and value
106   while (e.hasMoreElements() && line.charAt(0) != '\n')   //The SDP message
                                                             //start after a newline
108   {
        line = (String)e.nextElement();
110
        for (i=0;
112         !Character.isWhitespace(line.charAt(i)) && i<line.length();
            i++);
114     //Okay, we should now be on the first blank!
        fieldname = line.substring(start, i); //Let's store the fieldname
116     //It's just just one pharse left starting at i++, so:
        value = line.substring(++i, line.length()); //Let's store the value
118
        //Values must be placed to corresponding attributes in the SIPMessage
120     placeValue(fieldname, value, message);
      }
122
```

```
    return message;
  }

  /**
   * This parses a Response line to extract SIP-version, Status code and
   * Reason.
   * Note! This function assumes that fields are separated by ONE space.
   */
  private void parseResponseLine(String line, SIPMessage message)
  {
    int start = 0;
    int lineLength = line.length(); //Used several times, let's store it!
    int i;

    message.messageType = "RESPONSE";

    for (i=0;
        !Character.isWhitespace(line.charAt(i)) && i<lineLength;
        i++);
    //Okay, we should now be on the first blank!
    message.version = line.substring(start, i); //Let's store the version

    for (start = ++i; //Increment first, then set start (a litle obscure, but..)
        !Character.isWhitespace(line.charAt(i)) && i<lineLength;
        i++);
    //Okay, we should now be on the second blank!
    message.statusCode = line.substring(start, i); //Let's store the statuscode

    //It's just just one pharse left starting at i++, so:
    message.reason = line.substring(++i, lineLength); //Let's store the reason
  }

  /**
   * This parses a Request line to extract SIP-method, SIP URI and
   * SIP-version.
   * Note! This function assumes that fields are separated by ONE space.
   */
  private void parseRequestLine(String line, SIPMessage message)
  {
    int start = 0;
    int lineLength = line.length(); //Used several times, let's store it!
    int i;

    message.messageType = "REQUEST";

    for (i=0;
        !Character.isWhitespace(line.charAt(i)) && i<lineLength;
        i++);
    //Okay, we should now be on the first blank!
    message.methodType = line.substring(start, i); //Let's store the methodtype

    for (start = ++i; //Increment first, then set start (a litle obscure, but..)
        !Character.isWhitespace(line.charAt(i)) && i<lineLength;
        i++);
    //Okay, we should now be on the second blank!
    message.uri = line.substring(start, i); //Let's store the URI

    //It's just just one pharse left starting at i++, so:
    message.version = line.substring(++i, lineLength); //Let's store the reason
  }

  /**
```

```
    * This function looks up the fieldname in SIPMessage and stores the
186 * corresponding value
    * Note! This function assumes that the fieldname exists as attribute
188 * in SIPMessage.
    */
190 private void placeValue(String field, String value, SIPMessage message)
    {
192   if (field.compareToIgnoreCase("Via:") == 0)
      {
194     message.via = value;
        return;
196   }
      if (field.compareToIgnoreCase("From:") == 0)
198   {
        message.from = value;
200     return;
      }
202   if (field.compareToIgnoreCase("To:") == 0)
      {
204     message.to = value;
        return;
206   }
      if (field.compareToIgnoreCase("Call-ID:") == 0)
208   {
        message.callID = value;
210     return;
      }
212   if (field.compareToIgnoreCase("Cseq:") == 0)
      {
214     message.cSeq = value;
        return;
216   }
      if (field.compareToIgnoreCase("Content-Length:") == 0)
218   {
        message.contentLength = value;
220     return;
      }
222   if (field.compareToIgnoreCase("m:") == 0)
      {
224     message.media = value;
        return;
226   }
    }
228
    /**
230 * Basically this funtion calls the above functions separateLines
     and parseLines.
232 */
    public SIPMessage parse(String messageString)
234 {
      return parseLines(separateLines(messageString));
236 }
  }
```

B.1.8 Parsertest.java

```
2 //Title:        SIP implementation
  //Version:
4 //Copyright:    Copyright (c) 1999
  //Author:       Rune Schumann
6 //Company:      IFI, UiO
  //Description:  This project is trying to implement basic
8 //features of SIP to demonstrate conseptual
  //functionalities
10 package se.ericsson.eto.mmts.atk.impl.sip;

12 import java.io.*;
  import se.ericsson.eto.mmts.atk.impl.sip.*;
14 import java.util.*;

16 public class Parsertest
  {
18
    public static void main(String[] args)
20  {
      String test = "INVITE sip:watson@boston.bell-telephone.com SIP/2.0\n"
22                + "Via: SIP/2.0/UDP 169.130.12.5\n"
                  + "To: T. A. Watson <sip:watson@bell-telephone.com>\n"
24                + "From: A. Bell <a.g.bell@bell-telephone.com>\n"
                  + "CALL-ID: 3298420296@kton.bell-tel.com\n"
26                + "CSeq: 1 INVITE\n"
                  + "m: audio 3456 RTP/AVP 0 3 4 5";
28
      System.out.println(test);
30
      SIPParser p = new SIPParser();
32    SIPMessage sip = p.parse(test);

34    System.out.println("\n\n"
                         + sip.messageType + "\n"
36                       + sip.methodType + "\n"
                         + sip.uri + "\n"
38                       + sip.version + "\n"
                         + sip.via + "\n"
40                       + sip.to + "\n"
                         + sip.from + "\n"
42                       + sip.callID + "\n"
                         + sip.cSeq + "\n"
44                       + sip.media);

46    test = "SIP/2.0 100 Trying\n"
                + "Via: SIP/2.0/UDP kton.bell-tel.com\n"
48              + "From: A. Bell <sip:a.g.bell@bell-tel.com>\n"
                + "To: T. Watson <sip:watson@bell-tel.com> ;tag=37462311\n"
50              + "Call-ID: 3298420296@kton.bell-tel.com\n"
                + "CSeq: 1 INVITE\n"
52              + "Content-Length: 10";

54    System.out.println("\n----------Test 2--------\n" + test);
      sip = p.parse(test);
56

58    System.out.println("\n----------Test 2--------\n"
                         + sip.messageType + "\n"
60                       + sip.version + "\n"
```

```
                        + sip.statusCode + "\n"
62                      + sip.reason + "\n"
                        + sip.via + "\n"
64                      + sip.to + "\n"
                        + sip.from + "\n"
66                      + sip.callID + "\n"
                        + sip.cSeq + "\n"
68                      + sip.contentLength);

70
  //    Vector testVec;
72 //    testVec = p.parseLines(test);

74 //    Enumeration e = testVec.elements();
  //  while (e.hasMoreElements())
76 //    {
  //      System.out.println(e.nextElement());
78 //    }
    }
80 }
```

B.2 SIP implementasjon i ATK

B.2.1 SIPTerminalDescriptor.java

```
2  //Title:        ATK Projectfile
   //Version:
4  //Copyright:    Copyright (c) 1999
   //Author:       Rune Schumann
6  //Company:      IFI, UiO
   //Description:  This project is trying to implement basic
8  //features of SIP to demonstrate conseptual
   //functionalities
10 package se.ericsson.eto.mmts.atk.impl.sip;
   import java.io.IOException;
12
   import se.ericsson.eto.mmts.atk.MediaCodec;
14 import se.ericsson.eto.mmts.atk.Terminal;
   import se.ericsson.eto.mmts.atk.TerminalEventObserver;
16 import se.ericsson.eto.mmts.atk.TerminalType;
   import se.ericsson.eto.mmts.atk.TerminalDescriptor;
18 import se.ericsson.eto.mmts.atk.TerminalVendor;

20 import se.ericsson.eto.mmts.toolbox.protocol.h323.v2.common.TransportAddress;
   import se.ericsson.eto.mmts.toolbox.protocol.h323.v2.common.Alias;
22
   import se.ericsson.eto.mmts.atk.impl.sip.SIPTerminal;
24
   public class SIPTerminalDescriptor implements TerminalDescriptor
26 {
     private TerminalVendor termVendor;
28   private TerminalType termType;
     private TransportAddress gktsap; //Not really used in SIP ... maybe a SIP server?
30   private MediaCodec videoCodec;
     private MediaCodec audioCodec;
32

34   public SIPTerminalDescriptor()
     {
36   }

38
     public SIPTerminalDescriptor(TerminalType type, TerminalVendor vendor,
40                                TransportAddress tsap)
     {
42     termType = type;
       termVendor = vendor;
44     gktsap = tsap;
     }
46

48   /**
      * A description of what role the Terminal has (e.g. terminal, mcu, domain
50    * controller or gateway).
      * @param type - the terminal type
52    */
     public void setTerminalType(TerminalType type)
54   {
       //DEBUG**** No implemetation ... Just SIP Terminal
56   }
```

```
58  /**
     * This method will return a copy of the internally stored data (i.e.
60   * changing anything in the returned object will not affect the Terminal
     * ´Descriptors information).
62   * @return TerminalType the type of terminal (e.g. TERMINAL or GATEWAY)
     */
64  public TerminalType getTerminalType()
    {
66    return TerminalType.TERMINAL;
    }
68
    /**
70   * A description of the Terminal vendor. Describes country an d extension specific
     * (e.g. T35 specifics).
72   * @param vendor – an object describing the ´terminals vendor
     */
74  public void setTerminalVendor(TerminalVendor vendor)
    {
76    termVendor = vendor;
    }
78

80  /**
     * This method will return a copy of the internally stored data (i.e.
82   * changing anything in the returned object will not affect the Terminal
     * ´Descriptors information).
84   * @return TerminalVendor contains terminal vendor information
     */
86   public TerminalVendor getTerminalVendor()
     {
88     return termVendor;
     }
90
    /**
92   * The address of the Domain Controller to which the terminal is associated or
     * shall be associated.
94   * @param tsap – the domain controller address
     */
96  public void setDomainControllerAddress(TransportAddress tsap)
    {
98    gktsap = tsap;
    }
100
    /**
102  * The address of the Domain Controller to which the terminal is associated
     * @return TransportAddress – the domain controller address
104  */
    public TransportAddress getDomainControllerAddress()
106 {
      return gktsap;
108 }

110 /**
     * The default audio codec is used by this terminal object, if nothing else
112  * is explicitly ordered during a call setup.
     * @param codec – the default audio codec
114  */
    public void setDefaultAudioCodec(MediaCodec codec)
116 {
      audioCodec = codec;
118 }
```

```
120     /**
         * The default audio codec is used by this terminal object, if nothing else
122      * is explicitly ordered during a call setup.
         * @return MediaCodec - the default audio codec
124      */
        public MediaCodec getDefaultAudioCodec()
126     {
          return audioCodec;
128     }

130     /**
         * The default video codec is used by this terminal object, if nothing else
132      * is explicitly ordered during a call setup.
         * @param codec - the default video codec
134      */
        public void setDefaultVideoCodec(MediaCodec codec)
136     {
          videoCodec = codec;
138     }


140
        /**
142      * The default video codec is used by this terminal object, if nothing else
         * is explicitly ordered during a call setup.
144      * @return MediaCodec - the default video codec
         */
146     public MediaCodec getDefaultVideoCodec()
        {
148       return videoCodec;
        }
150

152     public Terminal createTerminal(int termId, Alias[] alias,
                                       TerminalEventObserver observer)
154       throws IOException
          {
156 //          return new SIPTerminal(termId, this, alias, observer); //DEBUG**** Constructor not yet
                                                                        //implemented!
158         return new SIPTerminal();
          }
160
    }
```

B.2.2 SIPTerminal.java

```
2  //Title:        SIP implementation
   //Version:
4  //Copyright:    Copyright (c) 1999
   //Author:       Rune Schumann
6  //Company:      IFI, UiO
   //Description:  This project is trying to implement basic
8  //features of SIP to demonstrate conseptual
   //functionalities
10 package se.ericsson.eto.mmts.atk.impl.sip;

12 import java.util.Vector;
   import java.net.UnknownHostException;
14
   import se.ericsson.eto.mmts.atk.Terminal;
16 import se.ericsson.eto.mmts.atk.TerminalEventObserver;
   import se.ericsson.eto.mmts.atk.TerminalState;
18 import se.ericsson.eto.mmts.atk.OriginatingCall;
   import se.ericsson.eto.mmts.atk.CallEventObserver;
20 import se.ericsson.eto.mmts.atk.ResultValue;
   import se.ericsson.eto.mmts.atk.Reason;
22
   import se.ericsson.eto.mmts.toolbox.protocol.h323.v2.common.Alias;
24 import se.ericsson.eto.mmts.toolbox.protocol.h323.v2.common.AliasList;
   import se.ericsson.eto.mmts.toolbox.protocol.h323.v2.common.TransportAddress;
26
   import se.ericsson.eto.mmts.toolbox.asynch.runnableserver.RunnableServerPool;
28
   import EDU.oswego.cs.dl.util.concurrent.Executor;
30

32 public class SIPTerminal implements Terminal
   {
34   /** Contains a reference to the thread provider.*/
     // use setMaximumPoolSize to change the size;
36
     // N.B: this pool is also used by H323Call in its TaskManager.
38   // So its only one pool for the entire ATK.
     static RunnableServerPool executor = new RunnableServerPool(30);
40
     private AliasList aliases;
42   private TransportAddress gkAddr;

44   public SIPTerminal()
     {
46   }


48
     /**
50    * The Terminal alias by which the Terminal is identified.
      * @param alias - the alias associated with the terminal object
52    */
     public void setAlias(Alias alias)
54   {
     }
56


58   /**
      * The Terminal aliases by which the Terminal is identified.
60    * @param alias - the aliases associated with the terminal object
```

```
     */
62   public void setAlias(Alias[] aliases)
     {
64   }

66   /**
      * @return Alias - the primary alias associated with the terminal object
68    */
     public Alias getAlias()
70   {
       return aliases.size() > 0 ? (Alias) aliases.at(0) : null;
72   }

74   /**
      * @return Alias[] - all aliases asscociated with the terminal object
76    */
     public Alias[] getAliases()
78   {
       return aliases.toArray();
80   }


82
     /**
84    * Performs a discovery of available domain controller
      * (e.g. H323 gatekeepers).
86    * This method will broadcast to a well-known multicast address if the
      * underlying protocol supports it.
88    *
      * Note: This version of the ATK ALWAYS throws UnsupportedOperationException
90    *
      * <pre>
92    *     Vector v = Terminal.discover();
      *     // assume v is not null
94    *     Pair p = v.elementAt(0);
      *     String id = (String) p.first;
96    *     TransportAddress tsap = (TransportAddress) p.second;
      *     // now call Terminal.setDomainController(TransportAddress), or
98    *     // Terminal.setDomainController(TransportAddress, String)
      * </pre>
100   *
      * @return a Vector of Pair, where each pair contains an id (String) and a
102   * transport address (TransportAddress) of a domain controller responding
      * to the broadcast, or "null" if no response.
104   *
      * @exception java.lang.UnsupportedOperationException
106   * @exception java.lang.IllegalStateException if state is Terminated.
      *
108   * @see java.util.Vector
      * @see com.objectspace.jgl.Pair
110   */
     public Vector discover()
112  {
       return new Vector();
114  }

116  /**
      * Performs the same action as a discover without arguments but it makes use
118   * of the provided domain controller address.
      * @param tsap - the address to use during the discover procedure
120   *
      * @return ResultValue - the result of the discover procedure
122   *
```

```
     */
124  public ResultValue discover(TransportAddress tsap)
     {
126    return ResultValue.SUCCESS;
     }
128
     /**
130   * Performs a discover towards all domain controller adresses specified in the
      * argument. Domain controllers will be prioritized according to the order of
132   * addresses in the argument (first address have highest priority). The first
      * Domain Controller responding successful will be used.
134   * @param tsaps - a number of addresses to use during the discover procedure
      * @return ResultValue - the result of the discover procedure
136   */
     public ResultValue discover(TransportAddress[] tsaps)
138  {
       return ResultValue.SUCCESS;
140  }

142
     /**
144   * Tries to register a Terminal with a domain controller (e.g. a H323 gatekeeper).
      * Default values will be used for mandatory protocol elements.
146   * @return ResultValue - the result of the register procedure
      */
148  public ResultValue register()
     {
150    return ResultValue.SUCCESS;
     }
152
     /**
154   * @param timeToLive - defines for how long the registration is considered valid
      * @return ResultValue - the result of the register procedure
156   */
     public ResultValue register(int timeToLive, boolean keepAlive)
158  {
       return ResultValue.SUCCESS;
160  }

162  /**
      * Tries to unregister a Terminal from a domain controller.
164   * @return ResultValue - the result of the unregister procedure
      */
166  public ResultValue unregister()
     {
168    return ResultValue.SUCCESS;
     }
170
     /**
172   * Set the address of the Domain Controller to be used in the next register
      * request. Please note that this will override the setting in the
174   * TerminalDescriptor. You can call this method with tsap = "null" to revert
      * to the address specified in the TerminalDescriptor.
176   * Note: it can only be called when the Terminal is in an UNREGISTERED
      * state. (Before you call register or after having call unregister)
178   *
      * @param tsap - the domain controller address
180   *
      * @exception java.lang.IllegalStateException
182   *
      * @see TerminalDescriptor#setDomainControllerAddress(TransportAddress)
184   */
```

```
    public void setDomainControllerAddress(TransportAddress tsap)
186 {
    }
188
    /**
190  * Same as setDomainControllerAddress(TransportAddress), but you can
     * also specified the id of the domain controller you wish to register with.
192  *
     * @param tsap - the domain controller address
194  *
     * @exception java.lang.IllegalStateException
196  *
     * @see #setDomainControllerAddress(TransportAddress)
198  */
    public void setDomainControllerAddress(TransportAddress TransportAddress,
200                                         String dcnid)
    {
202 }

204 /**
     * The address of the Domain Controller to which the terminal is/or prefer
206  * to be associated with. This is normally the same as the one specified
     * in the TerminalDescriptor, but can be overridden with
208  * Terminal.setDomainControllerAddress.
     * Please note that a domain controller can redirect a request to another
210  * domain controller, in which case the ATK will internally override the
     * domain controller setting for this Terminal as a result of the response
212  * from the initial request.
     *
214  * @return TransportAddress - the domain controller address currently active
     * for this Terminal, or if the Terminal is not registered, the prefered
216  * domain controller address.
     */
218 public TransportAddress getDomainControllerAddress()
    {
220   return gkAddr;
    }
222
    /**
224  * This method will cancel any active calls related to this Terminal.
     * The method will also shutdown the call listening port (no more incoming calls).
226  * It should be used when a Terminal object is to be removed
     */
228 public void delete()
    {
230 }

232 /**
     * The reason will be mapped and sent to any opposite parts involved in active
234  * calls with this terminal object.
     * @param reason  The reason for deletion (this reason is used when releasing calls)
236  */
    public void delete(Reason reason)
238 {
    }
240
    /**
242  * Sets a Terminal Event Observer to observe a specific Terminal object. The Terminal
     * Event Observer class must implement the TerminalEventObserver interface.
244  * @param observer - the object listening for terminal callbacks
     */
246 public void setObserver(TerminalEventObserver observer)
```

```
    {
248 }

250 /**
     * Creates an Originating Call object, which is used when making outgoing
252  * calls from a Terminal object. The method assumes that default bandwidth
     * will be used.
254  * @param destAlias - the destination alias
     * @return OriginatingCall - the call object used for the outgoing call
256  * @exception IllegalStateException
     */
258 public OriginatingCall createCall(Alias destAlias) throws UnknownHostException
    {
260   return new SIPOrigCall(); //DEBUG****
    }
262
    /**
264  * The Bandwidth specified should match the bandwidth required by the codecs in
     * use.
266  * @param destAlias - the destination alias
     * @param bandwidth - the maximum bandwidth that will be used during the call
268  * @return OriginatingCall - The call object used for the outgoing call
     * @exception IllegalStateException
270  */
    public OriginatingCall createCall(Alias destAlias, int bandwidth)
272      throws UnknownHostException
    {
274   return new SIPOrigCall(); //DEBUG****
    }
276

278 /**
     * The provided Call Event Observer must implement the CallEventObserver interface.
280  * @param destAlias - the destination alias
     * @param bandwidth - the maximum bandwidth that wil be used during the call
282  * @param observer - the call object callback listener
     * @return OriginatingCall - The call object used for the outgoing call
284  * @exception IllegalStateException
     */
286 public OriginatingCall createCall(Alias destAlias, int bandwidth,
                                    CallEventObserver observer)
288      throws UnknownHostException
    {
290   return new SIPOrigCall(); //DEBUG****
    }
292

294 public TerminalState getState()
    {
296   return TerminalState.REGISTERED;
    }
298
}
```

B.2.3 SIPMediaDescriptor.java

```
//Title:        ATK Projectfile
//Version:
//Copyright:    Copyright (c) 1999
//Author:       Rune Schumann
//Company:      IFI, UiO
//Description:  Well, had to do this to get the sucker
//compiling
package se.ericsson.eto.mmts.atk.impl.sip;

import se.ericsson.eto.mmts.atk.*;
import se.ericsson.eto.mmts.toolbox.protocol.h323.v2.common.TransportAddress;

public class SIPMediaDescriptor implements MediaDescriptor
{
  private int mediaId;
  private MediaType mediaType;
  private MediaCodec mediaCodec;
  private TransportAddress mediaChannel;
  private TransportAddress mediaControlChannel;

  public SIPMediaDescriptor()
  {
  }


  /**
   * Instruct the Terminal to use the default audio codec set in
   * the TerminalDescriptor used to create the Terminal
   * @see #TerminalDescriptor.setDefaultAudioCodec
   * @exception NullPointerException if default audio is not set
   * in TerminalDescriptor
   */
  public void useDefaultAudio()
  {
  }


  /**
   * Instruct the Terminal to use the default video codec set in
   * the TerminalDescriptor used to create the Terminal
   * @see #TerminalDescriptor.setDefaultAudioCodec
   * @exception NullPointerException if default video is not set
   * in TerminalDescriptor
   */
  public void useDefaultVideo()
  {
  }


  /**
    Media Channel Id is used for identification of a specific media channel.
  */
  public void setMediaId(int id)
  {
  }

  public int getMediaId()
  {
    return mediaId;
```

```
    }
62
    /**
64   * Set the media type
     * @deprecated Use setMediaCodec(MediaType, MediaCodec)
66   * @see #setMediaCodec(MediaType, MediaCodec)
     */
68  public void setMediaType(MediaType type)
    {
70  }

72  public MediaType getMediaType()
    {
74    return mediaType;
    }
76
    /**
78   * Set the media codec.
     * Can be G.711, GSM EFR, or any valid type of codec the terminal owner can
80   * handle.
     * @deprecated Use setMediaCodec(MediaType, MediaCodec)
82   * @see #setMediaCodec(MediaType, MediaCodec)
     */
84  public void setMediaCodec(MediaCodec codec)
    {
86  }

88  public void setMediaCodec(MediaType type, MediaCodec codec)
    {
90  }

92  public MediaCodec getMediaCodec()
    {
94    return mediaCodec;
    }
96
    /**
98    Specifies which transport address to use for the actual media stream.
    */
100 public void setMediaChannel(TransportAddress address)
    {
102 }

104 public TransportAddress getMediaChannel()
    {
106   return mediaChannel;
    }
108
    /**
110   Specifies which transport address to use for the control channel associated
      the specified media channel.
112 */
    public void setMediaControlChannel(TransportAddress address)
114 {
    }
116
    public TransportAddress getMediaControlChannel()
118 {
      return mediaControlChannel;
120 }

122
```

```
124
    }
```

B.2.4 SIPCall.java

```
2  //Title:        SIP implementation
   //Version:
4  //Copyright:    Copyright (c) 1999
   //Author:       Rune Schumann
6  //Company:      IFI, UiO
   //Description:  This project is trying to implement basic
8  //features of SIP to demonstrate conseptual
   //functionalities
10 package se.ericsson.eto.mmts.atk.impl.sip;
   import java.util.*;
12
   import se.ericsson.eto.mmts.atk.Call;
14 import se.ericsson.eto.mmts.atk.CallEventObserver;
   import se.ericsson.eto.mmts.atk.CallState;
16 import se.ericsson.eto.mmts.atk.CallStatus;
   import se.ericsson.eto.mmts.atk.ResultValue;
18 import se.ericsson.eto.mmts.atk.Reason;
   import se.ericsson.eto.mmts.atk.MediaDescriptor;
20
   import se.ericsson.eto.mmts.toolbox.asynch.runnableserver.TaskManager;
22
   import EDU.oswego.cs.dl.util.concurrent.ClockDaemon;
24 import EDU.oswego.cs.dl.util.concurrent.Executor;
   import EDU.oswego.cs.dl.util.concurrent.ThreadFactory;
26

28 /**
   * SIPCall is the protocoll spesific implementation of the ATK interface "call".
30 * This classis a part of the SIP implemetation of the ATK.
   */
32 public class SIPCall implements Call
   {
34
     protected static TaskManager taskMgr =
36       new TaskManager(SIPTerminal.executor);

38   /** The CallEventObserver which will receive the call events as callbacks */
     protected CallEventObserver ceo;
40
     /** A runnable executor executing callbacks provided via the CallEventObserver */
42   protected Executor executor;

44   /** A IntervalExecutor that sends IRR messages common to ALL calls */
     private static ClockDaemon intervalExecutor = new ClockDaemon();
46
     /** an id for this call's irr sender **/
48
     private Object irrSenderId;
50
     static
52     {
         intervalExecutor.setThreadFactory(new ThrFact());
54     }


56
     public SIPCall()
58   {
     }
60
```

```
62  /**
      Sets a CallEventObserver that will receive callbacks then a
64    call event accurs
    */
66  public void setObserver(CallEventObserver observer)
    {
68  }

70  /**
      Called then an ongoing call is to be ended "normaly",
72    eg. the user hangs up
      @return ResultValue
74  */
    public ResultValue closeCall()
76  {
      return ResultValue.SUCCESS;
78  }

80  /**
      Used then an ongoing call is to be ended by "not normal" reasons,
82    eg. network failure, timeout in call setup.
      @return ResultValue
84  */
    public ResultValue cancelCall(Reason reason)
86  {
      return ResultValue.SUCCESS;
88  }

90  /**
      Will provide the user with the callstate
92    @return CallState
    */
94  public CallState getCallState()
    {
96    return CallState.INITIATED;
    }
98

100 /* NOTE! Functions regarding mediadescriptors is not implemented
    *       properly in this version!!
102 */

104 /**
      Will provide the user the MediaDescriptor(s) which contains information
106   about the media streams the opposite side can transmit.
      @return Vector
108 */
    public Vector getPeerXmitMediaDescriptorList()
110 {
      return new Vector(); //****DEBUG
112 }

114 /**
      Will provide the user the MediaDescriptor(s) which contains information
116   about the media streams the opposite side can recive.
      @return Vector
118 */
    public Vector getPeerRcvMediaDescriptorList()
120 {
      return new Vector(); //****DEBUG
122 }
```

```
124   /**
        Will provide a MediaDescriptor, which is used to inform the opposite
126     side which media streams the caller is able to transmit
        @return MediaDescriptor
128   */
      public MediaDescriptor addXmitMediaDescriptor()
130   {
        return new SIPMediaDescriptor();
132   }

134   /**
        Will provide a MediaDescriptor, which is used to inform the opposite
136     side which media streams the caller is able to receive
        @return MediaDescriptor
138   */
      public MediaDescriptor addRcvMediaDescriptor()
140   {
        return new SIPMediaDescriptor();
142   }

144   /**
        Will clear the XmitMediaDescriptorList
146   */
      public void clearXmitMediaDescriptorList()
148   {
      }
150
      /**
152     Will clear the RcvMediaDescriptorList
      */
154   public void clearRcvMediaDescriptorList()
      {
156   }


158
      protected abstract class CallBack implements Runnable
160   {
        public abstract void callback();
162
        public void execute()
164     {
          if(ceo != null)
166       {
            try
168         {
              executor.execute(this);
170         }
            catch (InterruptedException ie)
172         {
            }
174       }
        }
176
        public void run()
178     {
          try
180       {
            callback();
182       }
          catch (Exception ex)
184       {
```

```
            System.err.println("EXCEPTION: " + ex);
186         ex.printStackTrace(System.err);
          }
188     }
    }
190

192 protected class CallProceedingCallBack extends CallBack
    {
194   public CallProceedingCallBack( CallProceeding cp )
      {
196     this.cp = cp;
      }
198
      public void callback()
200   {
        ceo.callProceeding();
202   }

204   private CallProceeding cp;
    }
206

208 protected class ReleaseCompleteCallBack extends CallBack
    {
210   public ReleaseCompleteCallBack( ReleaseComplete rc )
      {
212     this.rc = rc;
      }
214
      public void callback()
216   {
        CauseValue causeValue = null;
218
        try
220     {
          Cause cause = rc.getCause();
222
          if ( cause == null || (causeValue = cause.getCauseVal()) == null )
224       {
            // see if we can use ReleaseCompleteReason
226         ReleaseCompleteReason rcr = rc.getUserInformation().getReleaseCompleteReason();

228         causeValue = H323ConvertTable.convertReleaseCompleteReasonToCauseValue(rcr);
          }
230     }
        catch (NullPointerException npe)
232     {
          causeValue = CauseValue.unspecified;
234     }
        finally
236     {
          if( causeValue == CauseValue.normalCallClearing )
238         ceo.callClosed();
          else
240         ceo.callCancelled(H323ConvertTable.convertCauseToReason(causeValue));
        }
242
      }
244
      private ReleaseComplete rc;
246 }
```

```
248   protected class AlertingCallBack extends CallBack
      {
250     public AlertingCallBack( Alerting al )
        {
252       this.al = al;
        }
254
        public void callback()
256     {
          ceo.callAlerted();
258     }

260     private Alerting al;
      }
262
      protected class ConnectCallBack extends CallBack
264   {
        public ConnectCallBack( Connect msg )
266     {
          this.msg = msg;
268     }

270     public void callback()
        {
272       ceo.callConnected();
        }
274
        private Connect msg;
276   }

278
      protected class ProgressCallBack extends CallBack
280   {
        public ProgressCallBack( Progress msg )
282     {
          this.msg = msg;
284     }

286     public void callback()
        {
288       // potential null pointer exception caught in super.run()
          ceo.callProgress(msg.getProgressIndicator());
290     }

292     private Progress msg;
      }
294
      protected class StatusCallBack extends CallBack
296   {
        public StatusCallBack( Status msg )
298     {
          this.msg = msg;
300     }
302
        public void callback()
304     {
          // potential null pointer exception caught in super.run()
306       ceo.callStatus(new CallStatus(msg.getCallState().getCallStateValue(),
                                        msg.getCause().getCauseVal()));
308
```

```
    }

    private Status msg;
  }

  // Writes a trace prinout
  private void trace(String s) {
    if(trace)
      System.out.println(" == H323Call : " + s + " == ");
  }

  private static class ThrFact implements ThreadFactory
  {
    public Thread newThread(Runnable runnable)
    {
      return new Thread(runnable, "irrsender-" + nextNum());
    }

    private static int next;

    private static synchronized int nextNum()
    {
      return ++next;
    }
  }

}
```

B.2.5 SIPOrigCall.java

```
2  //Title:        SIP implementation
   //Version:
4  //Copyright:    Copyright (c) 1999
   //Author:       Rune Schumann
6  //Company:      IFI, UiO
   //Description:  This project is trying to implement basic
8  //features of SIP to demonstrate conseptual
   //functionalities
10 package se.ericsson.eto.mmts.atk.impl.sip;

12 import se.ericsson.eto.mmts.atk.*;
   import se.ericsson.eto.mmts.atk.impl.sip.*;
14 import se.ericsson.eto.mmts.atk.ResultValue;

16 public class SIPOrigCall extends SIPCall implements OriginatingCall
   {
18
     public SIPOrigCall()
20   {
     }
22
     /**
24     Used then making an outgoing call
       @return ResultValue
26   */
     public ResultValue placeCall()
28   {
       return ResultValue.SUCCESS;
30   }

32   /**
       Used to indicate then the media is allowed to be sent.
34     @param MediaWaitForConnect the MediaWaitForConnect Flag
     */
36   public void setMediaWaitForConnect(boolean wait)
     {
38   }

40 }
```

B.2.6 SIPTermCall.java

```
2  //Title:        SIP implementation
   //Version:
4  //Copyright:    Copyright (c) 1999
   //Author:       Rune Schumann
6  //Company:      IFI, UiO
   //Description:  This project is trying to implement basic
8  //features of SIP to demostrate conseptual
   //functionalities
10 package se.ericsson.eto.mmts.atk.impl.sip;

12 //import se.ericsson.eto.mmts.atk.*;
   import se.ericsson.eto.mmts.atk.TerminatingCall;
14 import se.ericsson.eto.mmts.atk.ResultValue;

16 import se.ericsson.eto.mmts.toolbox.protocol.h323.v2.common.Alias;
   import se.ericsson.eto.mmts.toolbox.protocol.h323.v2.q931.ProgressIndicator;
18
   public class SIPTermCall extends SIPCall implements TerminatingCall
20 {

22   public SIPTermCall()
     {
24   }


26
     /**
28     Called after accepting an incomming call
       @return ResultValue
30   */
     public ResultValue proceeding()
32   {
       return ResultValue.SUCCESS;
34   }

36   /**
       Can only be called after proceeding() has been called, and before
38     connect() is called. Used then information, which cannot be provided
       in other messages must be provided to the called side.
40     @return ResultValue
     */
42   public ResultValue progress(ProgressIndicator pi)
     {
44     return ResultValue.SUCCESS;
     }
46
     /**
48     Called then the called terminal is "alerting"
       @return ResultValue
50   */
     public ResultValue alerting()
52   {
       return ResultValue.SUCCESS;
54   }

56   /**
       Called then the called terminal is ready to connect to calling terminal
58     @return ResultValue
     */
60   public ResultValue connect()
```

```
    {
62      return ResultValue.SUCCESS;
    }
64
    /**
66    Used to indicate then the media is allowed to be sent. If set to true
      the media will be sent as soon as possible.
68    @return MediaWaitForConnect
    */
70  public boolean willMediaWaitForConnect()
    {
72    return true;
    }
74
    /**
76   * @return Calling party's aliases
     */
78  public Alias[] getCallingAlias()

80  {

82    Alias[] a = new Alias[5];

84    return a;

86  }

88
}
```