SIP Servlet containere og frameworks

Et speciale udarbejdet af:
Mads Danquah — mads@danquah.dk

Vejledere:
Arne John Glenstrup — panic@itu.dk
Anders Fosgerau — af@com.dtu.dk

1. november 2005

IT-Universitetet i København
Internet- og softwareteknologi
Efteråret 2005

Resumé - Dansk

Specialet behandler krav og motiverende faktorer for implementeringen af et SIP Servlet framework, og reflekterer over betydningen af åbne standarder i fremtidens kommunikations-netværk.

Som en introduktion til SIP Servlet frameworket er en SIP Servlet Container blevet designet og delvist implementeret. Undervejs bliver SIP samt eksisterende frameworks til implementering af tele og web-tjenester diskuteret. Den samlede viden fra disse diskussioner bruges efterfølgende til udarbejdelsen af et framework baseret på SIP Servlets.

Specialet konkluderer at det er muligt for ressourcesvage aktører at implementere en SIP Servlet container. Designet af Frameworket er udarbejdet på baggrund af systematiske overvejelser, og det vurderes at en implementering af designet vil simplificere udviklingen af SIP Servlet applikationer. Designet er ikke blevet implementeret. Endelig konkluderer specialet, at frit tilgængelige implementeringer af de forskellige åbne standarder, der benyttes i SIP baserede kommunkationsnetværk, vil have en gunstig virkning på fremtidens kommunikations-marked.

Nøgleord: SIP (RFC 3261), SIP Servlets (JSR 116), Containere, J2EE, Java, JAIN

Resumé - English

This thesis examines the requirements and motivating factors for implementing a SIP Servet framework, and reflects on the importance of using open standards in the communications network of the future.

As an introduction to the SIP Servlet framework, a SIP Servlet Container has been designed and partially implemented. Along the way, SIP and existing frameworks for implementing tele- and web-services are discussed. The results of these discussions are then used to compose a framework based on SIP Servlets.

The thesis concludes that it is possible for resource-constrained parties to implement a SIP Servlet container. The framework design is based on systematic considerations, and it is estimated that an implementation of the design will simplify the development of SIP Servlet applications. The design was not fully implemented. Finally, the thesis concludes that the free availability of implementations of the open standards that are used in SIP-based communications-networks, will provide favorable effects on the communications-market of the future.

Keywords: SIP (RFC 3261), SIP Servlets (JSR 116), Containers, J2EE, Java, JAIN

Indhold

Afsnit 1

Forord

Dette speciale er udarbejdet i 2005 på Københavns IT-Universitet. Forarbejdet til specialet blev indledt i efteråret 2004, hvor rapporten "Telefoni over Internettet - en introduktion til tekniske og juridiske problemstillinger"[Dan04], blev udarbejdet som et 4-ugers projekt. Dette projekt kan med fordel læses som en generel indledning til IP-telefoni.

Jeg vil gerne benytte lejligheden til at takke en række personer, der har muliggjort projektet. Første og fremmest vil jeg gerne takke mine to vejledere Arne John Glenstrup og Anders Fosgerau. Undervejs i projektet har Sven Larsen og Erik Lund-Jensen fra TDC samt Brian Risbæk fra Ericsson Danmark bidraget med et værdifuldt indblik i brugen af IP-telefoni og SIP i erhvervslivet.

Sidst, men ikke mindst, vil jeg takke Kirsten Madsen, Ulf Holm Nielsen og Thomas Riisbjerg for at have påtaget sig det store arbejde med korrekturlæsning og kritik af projektet.

Del I

Introduktion

Afsnit 2

Introduktion til projektet

Dette første afsnit vil give en generel introduktion til projektet. Der vil først blive givet en introduktion til de problemstillinger, der gør standardplatforme til teletjenester interessante. Herefter følger en gennemgang af grundlæggende teori for SIP-protokollen, der benyttes i mange standardplatforme.

2.1 Udviklingen i telekommunkationsnetværk og teletjenester

Siden telefonnetværk blev opfundet har de udviklet sig fra at være analoge til i dag at være næsten udelukkende digitale. Undervejs har teletjenester også gennemgået en stor udvikling fra oprindeligt at være hardwarebaserede og stærkt distribueret til i dag at være langt mere softwarebaseret og centraliserede. Hvor operatører i telenettenes tidlige år skulle placere en hardware-implementering af hver af sine tjenester i alle sine centraler, kan en operatør i dag have en enkelt central software implementering af hver tjeneste.

Udviklingen i telenettene og de tilhørende tjenester har i langt tid været drevet af operatørernes egne behov. Centraliseringen har betydet, at operatørerne har kunne spare udgifter til vedligeholdelse af redundant udstyr i sine centraler, og indførslen af softwarebaserede tjenester har betydet, at den enkelte operatør i langt højere grad selv kan udvikle og vedligeholde sine tjenester. Digitaliseringen af det underliggende netværk har bl.a. betydet, at en central kan behandle langt flere opkald, og at man kan transportere langt flere opkald over det fysiske telenet. På denne måde kan de fleste tiltag i udviklingen af telenet føres tilbage til besparelser for operatører.

I de seneste år har mobiltelefoni, Internettet og IP-telefoni dog hver især været med til at presse operatørerne til at sætte nye prioriteter for udviklingen. Med mobiltelefoni og Internettet er kunder generelt blevet langt mere vant til personaliserede tjenester og mulighed for selvbetjening. Mobiltelefoni har bragt et helt nyt marked med sig, og det har betydet et stærkt øget behov for udvikling af nye tjenester. Samtidig har IP-telefoni banet vejen for masse nye operatører, der i kraft af at telefonien transporteres over de lettere tilgængelige og billigere IP-netværk, ikke længere er afhængige af at kunne leje sig ind på konkurrenters telenetværk. Dette muliggør at selv små operatører kan konkurrere med større operatører på såvel udbud af tjenester som på pris. Den øgede konkurrence og det stigende behov for udvikling

af nye tjenester vil betyde, at de store operatører må indstille sig på at skulle kunne implementere og idriftsætte nye tjenester i et langt højere tempo end før. Vel og mærke med de samme ressourcer, da prisen på IP-telefoni-produkter skal være konkurrencedygtig.

Mens interoperabillitet mellem telenetværk har været vigtig for enhver operatør, har standardisering af tjeneste-platforme, der f.eks. ville muliggøre hardware-uafhængige tjenester, ikke været højt prioriteret. Leverandører af teleudstyr har været tilfredse med at operatøren bandt sig direkte til deres udstyr, og softwareudviklere med den nødvendige specialviden til at implementere tjenester til en given platform, har ikke haft problemer med at finde arbejde. Det har derfor ofte ikke været muligt at bruge en teleapplikation udviklet til én leverandørs hardware på en anden leverandørs hardware. Dette er stort problem for en operatør, der ønsker at benytte sig af tjenester udviklet af 3. part. Ikke kun fordi det kan være sværere at finde leverandører til ens egen platform, men også fordi en operatør vil stå i en dårlig forhandlingssituation hvis det ikke er muligt frit at vælge imellem forskellige 3.-parts leverandører p.ga. hardware-inkompatibilitet. Samtidig vil en 3.part leverandørs produkter appellere til et langt større marked, disse fungerer på flest mulige platforme og ikke er bundet til en enkelt leverandør-specifik platform.

I de seneste år har der derfor været stort fokus på udvikling af standard-platforme. En lang række forskellige organisationer har givet hvert sit bud på hvordan en sådan platform kan se ud, heriblandt SUN Microsystems, The International Telecommunications Union(ITU) og The Parlay Group(Et konsortium der udvikler API særligt til brug i teleapplikationer). Disse tiltag vil blive diskuteret yderligere i afsnit 11 i del III. Fælles for disse platforme er, at de definerer en række standardenheder i et netværk der arbejder sammen for at løse forskellige problemstillinger som er relevante for kommunikations-netværk. F.eks. tilbyder de fleste platforme standardenheder til registrering af brugere, uafhængighed af underliggende netværk, standard-grænseflader der sætter en tjeneste i stand til at præsentere sig forskelligt alt efter en kundes abonnement osv. I den seneste tid har protokollen SIP vundet indpas som standard protokol til koordinering af alle disse elementer.

2.2 Formel beskrivelse af projektet

2.2.1 Problemformulering

Projektet har til hovedmål at designe et framework der kan benyttes til udvikling af SIP-baserede tjenester baseret med SIP Servlets.

Sekundært har projektet til mål at demonstrere hvordan det med åbne standarder og frit tilgængelig teknologi er muligt at lave en implementering af en SIP Servlet container.

2.2.2 Afgrænsning

Implementeringen af SIP Servlet containeren vil kun opfylde udvalgte dele af SIP Servlet specifikationen. Disse dele bliver valgt ud fra en vurdering af hvad hvilke dele der er mest vitale for en container.

Projektet vil udelukkende beskæftige sig med platforme tilgængelige for Java-baserede teknologier.

Målgruppen for dette projekt er personer med interesse i udvikling af teletjenster. Der forudsættes grundlæggende kendskab til Java, samt netværks-protokoller.

2.2.3 Metode

Første del af projektet vil gennemgå grundlæggende teori for SIP og IP-telefoni. Den næste del vil introducere SIP Servlets, og efterfølgende designe og implementere en SIP Servlet container. Erfaringer draget fra implementeringen af containeren kombineres i tredje del med en gennemgang af en række forskellige applikations-frameworks der har relevans for SIP Servlet applikationer. På baggrund af denne kombinerede viden designes et applikations-framework baseret på SIP Servlets i slutningen af del tre. Endeligt opsummeres projektet i den fjerde del.

2.2.4 Læsevejledning

Projektet er opdelt i fem dele der kan læses selvstændigt. Dog vil de enkelte dele forudsætte at man har kendskab til teorien der er gennemgået i de foregående.

Del I, side 10 - Introduktion
Introduktion til projektet samt til IP-telefoni, telekommunkationsnetværk og SIP. Denne del kan springes over såfremt man på forhånd kender til disse teknologier og ikke er interesseret i de formelle projekt-detaljer.

Del II, side 23 - Implementering af en SIP Servlet Container
Beskrivelse af implementeringen af en SIP Servlet container, der kan benyttes til at implementere teletjenester baseret på SIP. Denne del henvender sig hovedsageligt til personer der ønsker indblik i hvordan SIP Servlets og tilhørende containere fungerer.

Del III, side 53 - Design af et SIP Servlet applikations-framework
Denne del gennemgår en række eksisterende frameworks og applikations-platforme til implementering af tele- og web-applikationer. Efterfølgende designes et SIP Servlet applikations-framework, og en række anvendelser af dette beskrives.

Del IV, side 80 Konklusion
Denne sidste del opsummere de tre foregående dele, og reflekterer over de forskellige problemstillinger de behandler.

Del V, side 84 Bilag

Gennemgang af teori er spredt ud over de forskellige dele således at teori gennemgås umiddelbart op til det afsnit der forudsætter den. Følgende teori gennemgås:

"IP-telefoni": afsnit 3.1, side 15 En introduktion til begrebet IP-telefoni

Session Initiation Protocol og kommunikationsnetværk: afsnit 3.2, side 15:
En indtroduktion til den grundlæggende kommunikations-protokol SIP

Komponentbaseret udvikling: afsnit 4.1, side 23 beskrivelse af fordelen ved anvendelse af komponenter.

J2EE: afsnit 4.2, side 24 summarisk beskrivelse af SUN's enterprise platform

Servlets: afsnit 4.3, side 25 introduktion til Servlets, samt de to specialiseringer HTTP og SIP Servlets

Parlay/OSA og Parlay/X: afsnit 11.1, side 56 en hyppigt anvendt teleapplikationsplatform

JAIN og tilhørende teknologier: afsnit 11.2.1, side 57 et tiltag fra SUN der søger at tilbyde en platform til implementering af særligt SIP-baserede applikationer

Struts: afsnit 11.3, side 58 et udbredt Model-View-Controller webapplikationsframework

Java Server Faces: afsnit 11.4, side 59 et standardiseret Model-View-Controller framework designet af bla. SUN

Spring og Spring WebFlow: afsnit 11.5, side 60 et applikations-framework der blandt andet tilbyder Inversion-of-Control og et MVC framework

Afvigelse fra projektaftalen

- Rapportsprog er ændret fra engelsk til dansk
- Den oprindelige problemformulering

  > Et ideelt rammeværktøj til implementering af en SIP-baseret tjeneste vil lade programmøren fokusere udelukkende på implementering af tjenesten, og lade udformningen af implementeringen være mindst muligt påvirket af den underliggende teknologi. Dette har netop været målet for mange rammeværktøjer til web-baserede tjenester, og disse bør derfor kunne bruges som forgangseksempler.
  >
  > Et ideelt rammeværktøj bør benytte sig af Model-View-Controller paradigmet med Apache Struts som forgangseksempel. Brugerinteraktionen bør være hændelsesorienteret med Java Server Faces som forgangseksempel.

  Er ændret til

  > Projektet har til hovedmål at designe et framework der kan benyttes til udvikling af SIP-baserede tjenester baseret med SIP Servlets.
  >
  > Sekundært har projektet til mål at demonstrere hvordan det med åbne standarder og frit tilgængelig teknologi er muligt at lave en implementering af en SIP Servlet container.

Tekniske forhold

Dokumentet er sat i LaTeX, og har været versioneret i Sub Version (`http://subversion.tigris.org/`). Alle figurer er udarbejdet af Mads Danquah med Adobe Illustrator stillet til rådighed af IT-Universitetet.

Til udvikling af Javaapplikationerne i dette projekt, har jeg nbenyttet Eclipse, der er et gratis Java-IDE udviklet af IBM. Eclipse indeholder mange andre nyttige værktøjer, og kan hentes på adressen `http://www.eclipse.org`.

SIP-applikationer har været afprøvet med værktøjet SIPp, et gratis væktøj oprindeligt udviklet af Hewlett-Packard. SIPp kan downloades på adressen `http://sipp.sourceforge.net/`

Afsnit 3

"IP-telefoni" og SIP

Dette afsnit vil gennemgå den grundlæggende teori om SIP og IP-telefoni der er nødvendig for forståelsen af resten af specialet.

3.1 "IP-telefoni"

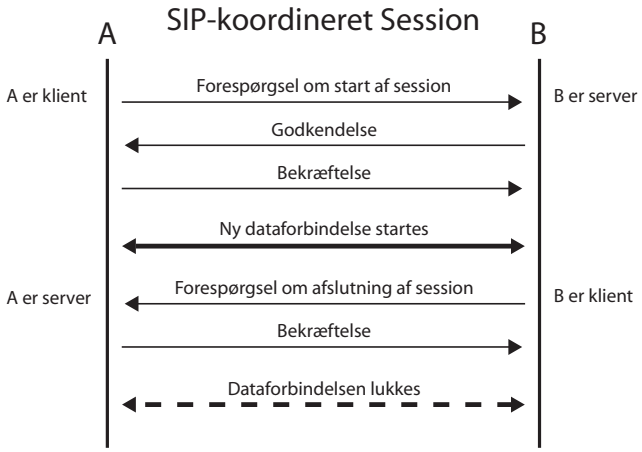
Navnet IP-telefoni bliver i dag brugt om en lang række vidt forskellige implementeringer, der benytter sig af vidt forskellige teknologier. Navnet har sandsynligvis vundet indpas fordi det til en vis grad lyder velkendt. Alle har en idé om, hvad telefoni er, og Internet Protocol (IP) leder hurtigt tankerne hen på Internettet, som de fleste i dag kender. Navnet er dog ikke lige beskrivende for alle produkter, der kategoriseres som IP-telefoni, og det er derfor vigtigt at gøre sig klart præcis hvad, der menes, når man taler om IP-telefoni i en given sammenhæng. I dette projekt betegner IP-telefoni et kommunikations netværk, der bruger IP som underliggende netværkslag, og som koordinerer dataforbindelser v.ha. "Session Initiation Protocol" (SIP). SIP beskrives i næste afsnit.

I sin mest simple form er IP-telefoni en talekommunikationstjeneste, der er baseret på IP. Der findes i dag mange forskellige teknologier, der implementerer denne form for IP-telefoni. Blandt de mest populære kan nævnes Skype[Sky], en gratis Internetbaseret telefoniapplikation, der benytter et underliggende peer-to-peer netværk, og Microsoft Netmeeting[Mic] der er en Internetbaseret konferenceapplikation. Et af problemerne ved betegnelsen "IP-telefoni" er, at mange af de eksisterende "IP-telefoni"-netværk kan tilbyde kommunikation med meget andet end tale. Det er sandsynligt at man i fremtiden vil se nye "IP-telefoni"-tjenester der ikke nødvendigvis har tale som omdrejningspunkt.

3.2 Session Initiation Protocol og kommunikationsnetværk

SIP er en klient/server protokol, der er kraftigt inspireret af HyperText Transfer Protocol (HTTP). SIP er ligesom HTTP tekst-baseret, og bygger på en række beskeder, der bliver sendt mellem to parter. En besked består af et hoved, indeholdende en række nøgle/værdi-par, og en krop. Hovedet indledes med enten en statuskode eller et metodenavn. I modsætning til HTTP, der er en klient/server protokol, er SIP en Peer-to-Peer protokol hvor rollen server og klient kan skifte. I HTTP sender

en klient env forespørgsel (request) til en server, hvorefter serveren sender et svar (response) tilbage. I SIP kan begge parter i kommunikationen sende forespørgsler og svar. Da måden, de enkelte forespørgsler og svar bliver behandlet undervejs, ikke altid er afhængig af, om der er tale om en forespørgsel eller svar, omtales de ofte blot som beskeder. Processen fra en SIP-kommunikations start til slut kaldes en dialog.



Figur 3.1: En SIP-koordineret dataforbindelse

Som navnet (Session Initiation Protocol) antyder, er SIP's rolle at starte (initiate) en session. Sessionen, der refereres til i navnet, er en dataforbindelse, der etableres parallelt med den, SIP-beskederne bliver sendt over (se figur 3.1). SIP bruges udelukkende til koordinering, og kan ikke selv transportere den "rå" data. I telefoniverdenen kaldes koordineringen af opkald signalering, og bliver i traditionelle telenet udført med protokollen Signaling System 7 (SS7). Når to parter i et SIP-netværk ønsker at starte en forbindelse, sender den ene part en forespørgsel til den anden. Denne forespørgsel er den første af 3 beskeder i et three-way handshake, hvori de endelige detaljer omkring forbindelsens karakter ligges fast. Herefter oprettes forbindelsen, og data kan nu udveksles mellem de to parter. Undervejs i forbindelsen bruges SIP til at koordinere forskellige ændringer ved forbindelsen såsom skift af lyd-codec, viderestilling af opkaldet, osv. Endelig bruges SIP også når en af parterne ønsker at nedlægge forbindelsen. Peer-to-peer egenskaberne ved SIP kan bl.a. ses ved at det ikke nødvendigvis er den part, der sendte den oprindelige forespørgsel, der sender forespørgslen om at nedlægge forbindelsen.

3.2.1 Enheder i et SIP-netværk

Et SIP-netværk består af en række elementer, der alle kan kommunikere med hinanden. Alle elementer skal have en implementering af et TCP og UDP-transportlag, og skal kunne adressere hinanden via IP. Netværket består af to slags elementer: endepunkter og servere. Endepunkter kan f.eks. være en IP-telefon, eller en gateway til et andet netværk som f.eks. et traditionelt telenet. Servere udbyder tjenester til endepunkterne i netværket. Det kan f.eks. være tjenester som viderestilling, registrering af bruger-data og adgangskontrol. SIP-specifikationen(RFC 3261) beskriver en række standard-roller, en server kan have:

Proxy - Kan agere stedfortræder for et endepunkt. En udbyder kan f.eks. stille

en proxy til rådighed for sine kunder, der håndterer alle de tekniske detaljer omkring routing af opkald. På denne måde behøver en IP-telefon ikke at have kendskab til hvordan udgående opkald skal behandles, men kan blot sende dem til sin proxy.

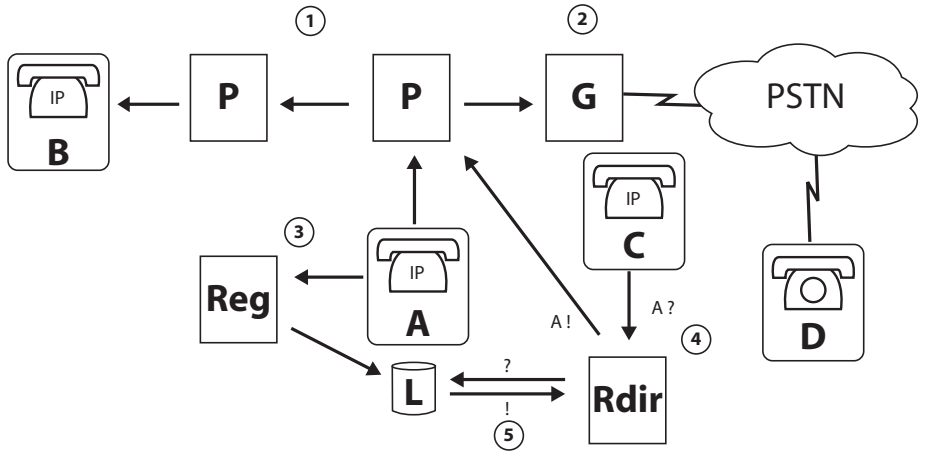
Lokations server - En generel term fra [ea02] for en enhed, der kan oplyse, hvordan man kontakter et endepunkt. En lokations-server er ikke nødvendigvis direkte forbundet til SIP-netværket, men kan f.eks. benyttes af en Redirect-server til viderestilling til en brugers faktiske adresse.

Redirect - Kan viderestille en SIP-session. Bruges f.eks. til parring af en brugers "brugervenlige" adresse som bruger@firma.dk til brugerens egentlige adresse.

Registrar(registrerings-server) - Modtager registrerings forespørgsler fra UA'er og kan opdatere lokations-servere.

User Agent Server(UAS) - en generel SIP Server der ikke falder i en af de ovennævnte kategorier.

Disse roller bliver brugt som byggesten i specifikationen, men er udelukkende roller og har ikke nogen direkte relation til en særlig implementering. Figur 3.2 giver et eksempel på disse roller i aktion.



Figur 3.2: (1) A kan ringe til B via deres proxies, (2) A kan ringe til D via en PSTN-gateway, (3) A oplyser sin faktiske adresse til en lokations-server via en registrerings server, (4) C bliver viderestillet til A's faktiske adresse via en redirect-server der benytter en lokations-server

3.2.2 Tilhørende protokoller

SIP bruges som før nævnt kun til signalering. For at løse de resterende opgaver bruges en række andre protokoller.

SDP Session Description Protocol bruges til at signalere en parts præferencer under det indledende three-way handshake. Præferencer kan f.eks. være hvilke lyd-formater parten understøtter, eller om parten kan modtage video-opkald.

RTP Real-Time Protocol er en protokol, der bruges til at transportere data, der har et realtids krav. RTP-pakker indeholder blandt andet information om hvilken rækkefølge, de blev afsendt i og på hvilket tidspunkt, således at modtageren kan være sikker på at afspille pakkerne i rigtig rækkefølge. Disse informationer betyder blandt andet, at en modtager kan se forskel på, om der ikke bliver sendt data p.g.a. netværksfejl eller stilhed hos afsenderen, hvis der kommer en pause i pakkestrømmen. RTP kan blandt andet bruges til at transportere video og tale mellem 2 eller flere parter.

RTCP Real-Time Control Protocol bruges til at sende kontrol-data relateret til en RTP-strøm. Det kunne f.eks. være statistik om pakketab, således at afsenderen kan justere den hastighed, hvormed data bliver sendt. En anden interessant anvendelse kunne være at oplyse navnet på personen, der taler, til brug i konference-samtaler.

RTSP Real-Time Streaming Protocol bruges til at fjernstyre optagelse og afspilning af forskellige former for medier over Internettet. Den bruges f.eks. ved implementering af en telefonsvarer, dels som optager, når der bliver ringet til en person, der ikke er til stede, dels når personen senere vil aflytte sine beskeder.

RSVP Ressouce ReServation Protocol bruges til at reservere ressourcer som båndbredde mellem to parter således at der kan stilles garantier for den tekniske kvalitet i et opkald.

3.2.3 Metodetyper og svarkoder

De grundlæggende metodekald er:

INVITE Forespørgere om påbegyndelse af en dialog. En INVITE-besked sendes af et endepunkt, der ønsker at forbinde til et andet endepunkt.

ACK bekræfter, at en part har modtaget en anden parts bekræftelse af en INVITE-besked. Dette dobbeltcheck er nødvendigt, da SIP ikke gør sig nogle antagelser om det underliggende netværk, og derfor ikke kan forvente pålidelig levering af pakker.

BYE Bruges til at afslutte en dialog og en eventuel tilhørende data-forbindelse.

CANCEL Afbryder en dialog, der er ved at blive sat op. Er dialogen først etableret, bruges BYE.

REGISTER Bruges til at registrere et UA hos en Registrar, der derefter f.eks. kan opdatere en lokations-server.

OPTIONS Forespørger en server om hvilke funktionaliteter den understøtter.

Svarkoder i SIP er struktureret på samme måde som i HTTP, hvor det første tal i koden angiver hvilken af 6 typer svar, der er tale om:

1xx (100-199) Informationer og løbende beskeder. Bruges undervejs i en kommunikation.

2xx (200-299) Positive svarbeskeder, f.eks. at røret på telefonen nu er blevet løftet og samtalen kan begynde.

3xx (300-399) Viderestillingsbeskeder der som regel vil indeholde en alternativ kontaktadresse.

4xx (400-499) Negative svar: Der er opstået en fejl, og kilden til fejlen findes på klientsiden. Forespørgslen kan gentages, når fejlen er rettet hos klienten.

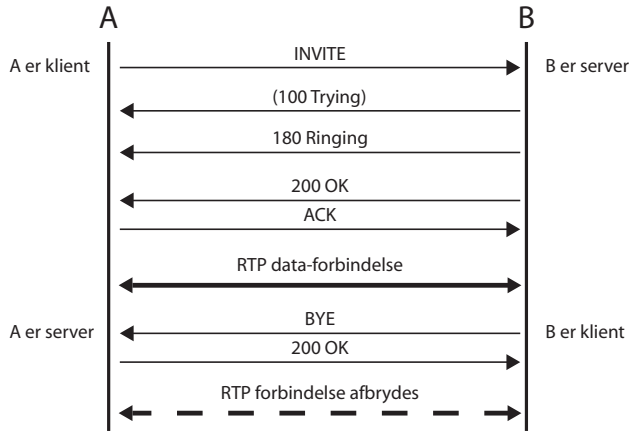
5xx (500-599) Negative svar: Der er opstået en fejl, og kilden til fejlen findes på serversiden. Sendes forespørgslen til en anden server, kan den muligvis lykkes.

6xx (600-699) Globale fejl: Handlingen er ikke mulig, og bør ikke forsøges igen, f.eks. hvis en bruger ikke ønsker at modtage et opkald fra en bestemt anden bruger.

3.2.4 Praktiske eksempler

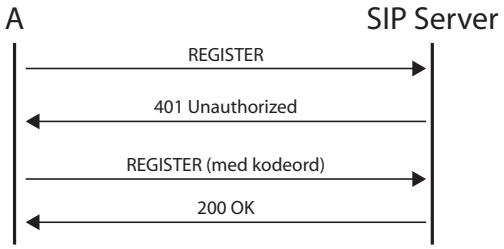
Følgende eksempler viser hvordan de grundliggende SIP-metoder og svarkoder benyttes i de mest basale scenarier.

Figur 3.3: SIP bruges til at koordinere en RTP-forbindelse mellem to brugere. Undervejs skifter klient/server rollen. Dette er illustreret ved at A sender forespørgslen først i dialogen, mens B fungerer som klient i slutningen.



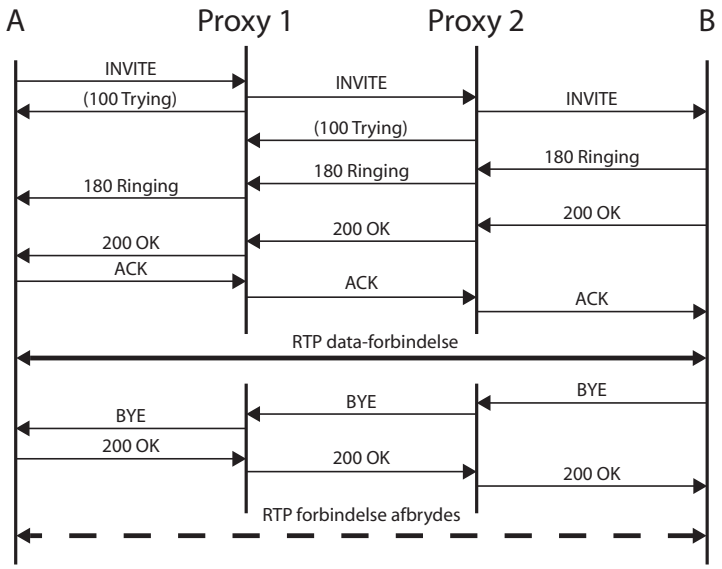
Figur 3.3: Oprettelse og nedlæggelse af en SIP-koordineret session

Figur 3.4: Bruger A registrerer sig hos en registrerings-server. Det kan være påkrævet at opgive et brugernavn/kodeord eller anden identifikation sammen med en registrerings forespørgsel. Dette er illustreret ved den første forespørgsel, der besvares med en 4xx fejlkode som svar.



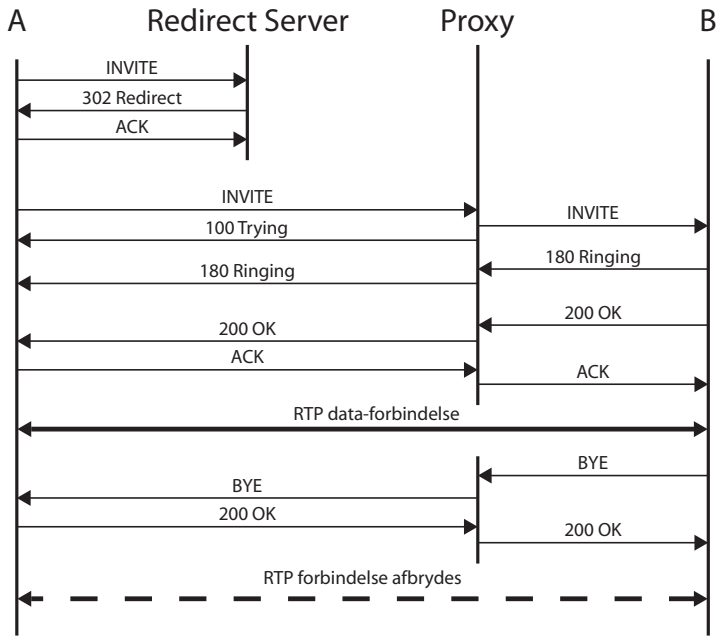
Figur 3.4: Registrering af bruger A hos en registrerings-server

Figur 3.6: Som regel vil en SIP-bruger benytte sig af en proxy-server, der under opkaldet agerer stedfortræder for brugeren. Dataforbindelsen mellem de to brugere vil dog som regel laves direkte for at minimere afstand og forsinkelse af data.



Figur 3.5: Oprettelse og nedlæggelse af en SIP-koordineret session mellem bruger A og B via proxy-servere

Figur 3.6: Ved at bruge en redirect server kan en bruger nås med en logisk adresse. Redirect serveren kan evt. kommunikere med en lokations-server, der via en tidligere registrering fra bruger B kender til B's faktiske adresse.



Figur 3.6: En redirect-server viderestiller en INVITE til en anden modtager

Del II

Implementering af en SIP Servlet container

Afsnit 4

Introduktion

Del II beskriver implementering af en SIP Servlet container, en type server, der kan indgå som en central del af SIP-baserede kommunikations-netværk. Målet med at lave en sådan implementering er, at vise at man ved at benytte åbne standarder i kommunikations-netværk, åbner mulighed for det selv med få ressourcer bliver muligt at implementere de platforme netværket og dets tjenester hviler på.

De første afsnit vil give en introduktion til den teori, der er nødvendig for at følge implementeringen. Der vil blive gennemgået teori om

Komponentbaseret udvikling En udviklingsmodel, der opfordrer til genbrug og understøtter 3. parts udvikling.

J2EE og Komponent/Container modellen Javas enterprise platform, der bl.a. benyttes i teleindustrien og understøtter komponentbaseret udvikling.

Servlets en meget succesfuld J2EE komponent.

De efterfølgende afsnit vil gennemgå analyse af problemstillinger og design af en SIP Servlet container-implementering. Endeligt vil de sidste afsnit beskrive implementering af containeren, afprøvning af den, og evaluering af implementationen.

4.1 Komponentbaseret udvikling

En softwarekomponent er indkapslet funktionalitet, der kan tilgås via klart definerede grænseflader[Ole03]. Komponentbaseret udvikling er en udviklingsform, hvor man benytter sig af en eller flere komponenter til at opbygge sin applikation. Der findes mange forskellige komponentbaserede applikationsplatforme, heriblandt Microsofts .NET og J2EE, der vil blive beskrevet i næste afsnit. Først vil vi dog se på komponenter og komponentbaseret udvikling generelt, og de fordele og ulemper de kan føre med sig.

Fordelen ved at have en komponent med klart definerede grænseflader er, at hvis grænsefladerne er korrekt defineret, behøver applikationen der benytter sig af en given komponent, ikke at være bekendt med hvordan den indkapslede funktionalitet er implementeret. Dette tillader ideelt en udvikler udelukkende at koncentrere sig om *hvad* en komponent gør, og ikke *hvordan*. Ved at standardisere og offentliggøre de grænseflader, en given type komponent tilbyder, åbnes der en række muligheder, heriblandt:

Portabillitet Da applikationer på forhånd kender til grænsefladen til komponenten, er det muligt at implementere flere applikationer, der kan benytte den samme type komponent. Dette gør det muligt at flytte komponeten fra én applikation til en anden uden at skulle ændre på hverken implementeringen af komponenten eller applikationen.

3.parts udvikling Omvendt kan en leverandør af komponenter også implementere sin komponent uden at have kendskab til applikationen, der skal benytte den. Dette åbner markedet for 3.parts udviklere, der kan sælge standardiserede komponenter. Denne type af komponenter bliver ofte betegnet "Commercial-Off-The-Shelf" (COTS) komponenter.[MOPS02]

Genbrug Ved at lade forskellige applikationer tilgå en komponent igennem de samme standardiserede grænseflader, kan den samme komponent genbruges mellem flere applikationer

Komponenter kan benyttes i forskellige grader alt efter programmeringssprog, applikationsplatforme og type af applikationer. Brugen kan spænde fra en enkelt komponent i en applikation, til hele applikationer udelukkende bygget op af standardkomponenter. Samtidig kan en komponent i sig selv gå fra at være en lille applikation, der udfører en simpel matematisk beregning, til at være en kæmpe applikation med hundredevis af elementer. Begge applikationer kan omtales som komponenter, så længe de indkapsler én eller flere funktionaliteter, der kan tilgås via klart definerede grænseflader.

4.2 J2EE

J2EE, der står for "Java 2 Platform, Enterprise Edition", en er specifikation, der beskriver en infrastruktur til implementering af "virksomheds-applikationer". Disse er typisk kendetegnede ved at skulle kunne leve op til en række krav så som skalering, sikkerhed, interoperabillitet med andre systemer.

J2EE består grundlæggende af fem grundelementer[PC00]

En specifikation - der beskriver de forskellige krav en implementering af et J2EE kompatibelt produkt opfylde.

En programmeringsmodel - der beskriver hvordan de forskellige dele af J2EE bør benyttes.

En underliggende Java platform - J2EE platformen er afhængig af en underliggende Java 2 Standard Edition

En referenceimplementering (RI) - der demonstrerer hvordan specifikationen kan implementeres. Referenceimplementeringen vil som regel ikke egne sig til produktionsbrug, men vil være tilstrækkelig til udviklingsbrug

Et Teknologi Kompatibilitets Kit (TCK) - der kan benyttes af en leverandør af en given J2EE server til at teste, om implementeringen lever op til specifikationen.

J2EE specifikationen refererer til en lang række teknologier, der bindes sammen for at tilbyde en passende infrastruktur. En implementering af en J2EE kompatibel server skal stille implementeringer af et antal af specifikationer til rådighed. De

forskellige specifikationer udvikles oftest i et samarbejde mellem en række interesserede parter, og vil som regel lade alle detaljer, der ikke kan risikere at bryde kompabiliteten imellem forskellige implementeringer, stå åbne. På denne måde holdes markedet for forskellige implementeringer, der f.eks. kan have fokus på forskellige kvaliteter, åbent.

J2EE bygger på en "Komponent/Container" model, hvori en komponent kan placeres i en container, der tager sig af en lang række af de problemstillinger, der ellers kunne komplicere implementeringen af komponenten, de såkaldt "Cross-cutting Concerns"[PC00]. Ved at lade containeren tage sig af de mere komplicerede ikke applikations-specifikke detaljer som f.eks. sikkerhedskontrol og loadbalancering, kan udvikleren af en given komponent koncentre sig om den centrale funktionalitet af sin komponent. Dette bevirker også, at der i en given applikation kun vil være én implementering af håndteringen af de oftest meget komplicerede cross-cutting concerns, og udviklingsressourcerne kan derfor koncentreres på ét sted. På denne måde kan kvaliteten af løsningen oftest øges, og denne kvalitet vil videregå til den enkelte komponent.

En container tilbyder normalt følgende til en komponent[PC00]

- En Java 2 Standard Edition installation
- En række J2EE API'er
- En implementering af disse API'er
- En standardiseret metode til at sætte en komponent i drift ("Deployment")
- En mulighed for at administrere komponenter installeret i containeren ("Management")

4.3 Servlets

Servlets er én af de fire typer komponenter, som J2EE specifikationen beskriver. En servlet er en komponent, der kan udvide en given servers funktionalitet, og er i sin grundform protokol-uafhængig. Servlet specifikationen beskriver hvordan en container grundlæggende skal forholde sig til en servlet så som hvordan servletten skal dannes og nedlægges, hvordan servletten får adgang til eksterne ressourcer og en række andre generelle problemstillinger. For rent faktisk at kunne benytte servlets implementeres protokol-specifikke specialiseringer af Servlet-typerne, og hørende containere der kan håndtere de pågældende typer.

Den mest velkendte type af servlets er HTTP Servlets, der benyttes i en web-container.

4.3.1 HTTP Servlets, JSP og web-frameworks

HTTP Servlets var oprindelig beregnet til udviklere af webapplikationer, men viste sig hurtigt at være for besværlige at bruge i deres rene form. Således ville visse servlets indeholde mere HTML-kode end Java-kode, og dermed blive uoverskuelige. Dette resulterede i udviklingen af Java Server Pages (JSP)[Man04], der

tillod udvikleren at bruge stumper af Java-kode i sine HTML-dokumenter. Efterhånden som web-applikationer blev mere og mere populære og kravene til web-applikationerne steg kom der en lang række HTTP Servlet-baserede frameworks til. Hvert framework havde sine fordele og ulemper, men fælles for dem alle var, at de i en eller anden grad abstraherede forskellige uønskede egenskaber ved HTTP Servlets væk, og i nogle tilfælde tilføjede nye egenskaber.

Blandt de mest populære frameworks og teknologier baseret på HTTP Servlets er i skrivende stund:

Struts et framework der især fokusere på opdeling i Model View og Controller (MVC)

Spring et framework der ud over at kunne tilbyde en MVC-arkitektur benytter sig meget af Inversion-Of-Control(IOC) og benytter sig stærkt af komponenter.

JavaServerFaces (JSF) industri-standard fra SUN der ud over en MVC-arkitektur og IOC lader programmøeren udvikle sin webapplikation i et event-baseret paradigme.

4.3.2 SIP Servlets

SIP Servlets er en specialisering af Servlets tiltænkt applikationer der skal kunne interagere med et SIP-netværk. SIP Servlets forsøger at abstrahere SIP væk for applikations-programmøren, således at fokuset kan holdes mere på de applikationsrelevante problemstillinger, og mindre på de for applikationen mindre relevante SIP-specifikke problemstillinger. Dette gøres ved at lade containeren håndtere opgaver som retransmittering, persistering af tilstandsdata mellem beskeder og delegering af de forskellige typer af beskeder mellem sine SIP Servlets. Én af fordelene ved SIP Servlets er at de i høj grad minder om HTTP Servlets, og derfor vil virke bekendte for HTTP Servlet programmører. Derudover passer de p.g.a. deres slægtskab med HTTP Servlets godt ind i J2EE arkitekturer. SIP Servlet-specifikationen er dog ikke selv er en J2EE specifikation. Endelig er det muligt at kombinere en SIP Servlet og en HTTP Servlet container og på denne måde lave en applikation, der spænder over begge protokoller.

Afsnit 5

Analyse og Kravspecifikation

Dette kapitel vil analysere problemstillingerne omkring implementeringen af en SIP Servlet container. En SIP Servlet container skal først og fremmest opfylde kravene i den officielle SIP Servlet API specifikation. Derudover vil der være en række mindre funktionelle krav til implementeringen. Disse krav vil analysen forsøge at afdække igennem en diskussion af en række forskellige problemstillinger.

Første underafsnit vil give en introduktion til hvad SIP Servlet API specifikationen er. De efterfølgende afsnit vil udføre selve analysen.

5.1 SIP Servlet API Specifikationen (JSR-116)

SIP Servlet API Specifikationen er en specifikation af det API en SIP Servlet og dens container kommunikerer igennem. Specifikationen er standardiseret igennem "The Java Community Process" (JCP), en åben proces drevet af SUN til brug for udvikling af Java relaterede standarder. Standarderne bliver udviklet af ekspertgrupper der dannes af JCP medlemmer, som har interesse i at en given standard bliver udviklet. Enhver kan blive JCP medlem, men som regel er ekspertgrupper opbygget af repræsentanter fra forskellige virksomheder. Ekspertgruppen indleder standardiseringen ved at indsende en anmodning, en såkaldt "Java Specification Request" (JSR), der beskriver problemstillingen. Efterfølgende udvikler medlemmerne i ekspertgruppen standarden og tilhørende dokumentation i fællesskab. Servlet API Specifikationen benævnes i JCP sammenhæng som JSR-116, og blev udviklet af blandt andet IBM, Dynamicsoft og Ericsson. Med JSR-116 følger der, som det er tilfældet med de fleste JSR'er, en reference-implementering(RI), samt et teknologikompabilitets test-kit(TCK), der bruges til at teste om en implementering følger standarden. Reference-implementeringen er udelukkende tiltænkt udviklingsbrug, reference og inspiration for andre implementeringer af JSR-116.

JSR-116 beskriver ud over selve API'et også hvilke opgaver en SIP Servlet container skal kunne udføre, hvordan den skal opføre sig i forskellige situationer, og en række andre funktionelle krav. JSR-116 stiller ingen krav til hvordan de enkelte funktioner skal implementeres.

Kravspecifikationen til den forestående implementering vil indeholde et krav om at leve op til JSR-116 i en eller anden grad. Da JSR-116 hovedsagelig koncentrerer sig om grænseflader og opførsel og ikke dikterer hvordan den egentlige implemen-

tering skal udformes, må kravspecifikationen nødvendigvis indeholde krav til de dele JSR-116 ikke dækker. Her vil især være tale om ikke-funktionelle krav.

5.2 Analyse

Analysen vil behandle eventuelle problemstillinger der står i vejen for designet af SIP Servlet containeren. Da SIP Servlet API specifikationen sammen med reference implementeringen giver en god reference til hvordan de mere tekniske problemstillinger kan håndteres, vil analysen hovedsagelig behandle problemstillinger, der er relervante i forhold til projektets problemformulering.

Da målet for projektet ikke er at demonstrere, hvordan man laver den mest effektive implementering, men blot at demonstrere, at det er muligt for enhver at lave en implementering, er det vigtigt at designet og dermed implementeringen holdes så gennemskueligt som muligt.

Da implementeringen vil blive udført af en enkelt person, og da JSR spænder over en lang række problemstillinger, der ikke kan overskues på én gang, vil analysen indledningsvis diskutere hvordan implementeringen kan opdeles og udføres over flere iterationer. Efterfølgende vil implementeringen af netværkshåndtering i containeren blive diskuteret. Da dele af SIP kan være meget komplekse er det vigtigt at diskutere hvordan netværket kan håndteres uden at komplicere implementeringen unødigt. Endelig vil konvergerede applikationer blive diskuteret, da det dels er en af de interessante typer af applikationer en SIP Servlet Container muliggør, og dels er erklæret som et vigtigt mål for JSR-116.[ea03a]

5.2.1 Fokus og proces model

For at kunne afgøre hvad der skal prioriteres i implementeringen, opdeles implementeringen i en række dele, der hver især løser en bestemt opgave i implementeringen. Eksempler på opgaver er håndtering af netværks-kommunikation, konfiguration af containeren og håndtering af Servlets. Ved håndtering af komplekse opgaver kan det være nødvendigt at splitte delene yderligere op i vertikale abstraktionslag, der øverst arbejder med højniveau- og nederst med lavniveau problemstillinger. I de øvre lag vil man typisk arbejde med *hvad* der skal gøres, mens man i de nedre i højere grad arbejder med *hvordan* de enkelte operationer udføres. Problemstillingerne i de nedre lag vil være ligeså, hvis ikke mere, komplicerede som de, der håndteres i de øvre. Dog vil problemstillingerne, der behandles i de nedre lag oftest virke mere fjerne fra de problemstillinger delen som helhed forsøger at løse.

Da de forskellige dele ikke er lige afgørende for om implementeringen vil leve op til projektets mål, inddeles de i primære og sekundære dele. De primære dele arbejder med problemstillinger, der er centrale for at containeren kan fungere og leve op til JSR-116. De sekundære dele kan i mange tilfælde være lige så vigtige for at opnå et brugbart produkt, men er ikke lige så vitale for de grundlæggende funktioner i containeren. Et eksempel er henholdsvis netværkshåndtering og konfiguration. Mens konfiguration er vigtig for at opnå et brugbart produkt, er denne funktion ikke afgørende for om containeren fungerer. Det vil f.eks. være muligt at erstatte en XML-konfigurationsfil, der kan beskrive en række forskellige Servlets,

med “hardcodede” informationer, der beskriver en enkelt Servlet, og som kan udskiftes ved hver afprøvning. Netværkshåndteringen er på den anden side meget central. En SIP Servlet container vil ikke være til meget nytte hvis den ikke kan kommunikere med et SIP netværk.

Ideelt skal størstedelen af implementeringsarbejdet foregå i de primære dele da det er her de fleste problemstillinger der interessante for projektet, bliver behandlet. Udviklingen skal ske med en procesmodel, der tillader at containeren bliver implementeret over flere iterationer således at fokuset i de første iterationer kan være på de primære dele, og derefter de sekundære. Målet med projektet er først og fremmest at gennemføre implementeringen af de primære dele.

5.2.2 Netværkshåndtering

Netværkshåndtering er én af de centrale dele i implementeringen, men samtidig også en del, der spænder over mange forskellige problemstillinger. For at sænke kompleksiteten i implementeringen vil netværkshåndteringen blive opdelt i 2 lag, hvor det nedre lag vil blive implementeret af en eksisterende komponent. Det nederste lag skal håndtere så mange af de low-level SIP-relaterede problemstillinger som muligt således at det øvre lag hovedsagelig nøjes med at arbejde med de SIP-servlet relaterede problemstillinger. Da komponenten vil komme til at spille en meget central rolle i implementeringen, er det vigtigt, at den er af høj kvalitet. Det vil i denne sammenhæng betyde, at den skal være veldokumenteret, gennemprøvet, og ideelt også standardiseret således, at den senere kan udskiftes med en anden implementering. Da SIP har eksisteret siden 1999[ea99], findes der en del forskellige netstak-implementeringer. Det er dog de færreste af disse implementeringer der er standardiseret, og mange er kommercielle produkter, der kræver indkøb af licenser.

JAIN er et tiltag under JCP, der udvikler API'er til understøttelse af åbne Javabaserede kommunikationstjenester [Mir04]. JAIN SIP(JSR-32) er én af de grundlæggende JAIN specifikationer der beskriver en SIP netstak. JSR-32 blev blandt andet udviklet af det amerikanske “National Institute of Standards and Technology”(NIST), der efterfølgende har stillet deres implementering af JSR-32 til fri afbenyttelse.

En implementering af JSR-32 vil være et godt valg til dette projekt, da JSR-32 er en del af hvad der ser ud til at blive de facto standarden for javabaseret kommunikations teknologi.

5.2.3 Understøttelse af konvergerede applikationer

Et af de overordnede mål for JSR-116 er at muliggøre konvergerede applikationer. En konvergeret applikation er en applikation der benytter mere end én type Servlet, og dermed spænder over flere protokoller. JSR-116 beskriver hvordan det vil være muligt at implementere en konvergeret HTTP og SIP applikation, der ved at benytte begge typer servlets kan udbyde tjenester, der anvender både SIP og HTTP.

Et eksempel på en sådan applikation er en online telefon-væknings tjeneste: Tjenesten lader en bruger indtaste tidspunkt for vækning på en hjemmeside. Efterfølgende ringer en SIP-Servlet op til det specificerede nummer og afspiller en standard besked. Et andet eksempel er et web-baseret omstillingsbord hvor en sekretær via

et webinterface kan omstille opkald. Når sekretæren klikker på et givent link på en webside, modtages forspørgslen af en HTTP servlet, der ved hjælp af en SIP Servlet installeret i samme applikation sætter viderestilligen igang.

Designet af SIP Servlet containeren skal indeholde overvejelser af hvordan implementeringen vil kunne kombineres med en eksisterende HTTP Servlet container. Dog vil det ikke være en prioritet at lave en faktisk implementering af denne kombination.

5.3 Endelig Kravspecifikation

Ud fra de ovenstående overvejelser og JSR-116 stilles der følgende krav til implementeringen

- Implementeringen skal følge en iterativ procesmodel, hvor de første iterationer skal fokusere på design og implementering de centrale dele i containeren.
- Første iteration skal fokusere på at få et grundlæggende design implementeret, og på at få implementeret de grundlæggende JSR-116 typer der benyttes til repræsentation af data.
- Implementeringen skal benytte sig af komponenter til løsning af problemstillinger, der ikke ligger inden for målet af projeketet. Disse komponenter skal så vidt muligt være standardiserede og frit tilgængelige
- Netværkhåndteringen skal udføres med en implementering af JSR-32. Med mindre den viser sig at have betydelige fejl eller mangler skal NIST-implementeringen benyttes
- Implementeringsafsnittet skal indeholde overvejelser om hvordan SIP Servlet Containeren kan indgå i en implementering af en konvergeret container. Der stilles ingen krav til en egentlig implementering.
- Der stilles ingen performance-krav til implementeringen.

Afsnit 6

Design

Design kapitlet vil give et overblik over interne struktur i containeren. Kapitlet beskriver de forskellige dele hvoraf containeren er opbygget, og giver ved hjælp af flow-diagrammer et indblik i hvordan de forskellige dele arbejder sammen i forskellige situationer.

JSR-116 beskriver hvordan en implementering skal opføre sig, hvilke grænseflader den skal implementere og andre problemstillinger, der er vigtige for at garantere at en SIP Servlet kan afvikles ens med alle implementeringer af JSR-116. Den specificerer dog intet om hvordan en implementering skal designes og design-afsnittet vil derfor beskrive de valg, der er gjort i designet af implementeringen.

Designet af implementeringen vil prioritere at den endelige implementering skal være så gennemskuelig og simpel som muligt. Dette gøres i henhold til kravene fra problemformuleringen og kravspecifikationen om at implementering skal demonstrere hvordan, det er muligt at lave en implementering af JSR-116, og at der ikke stilles nogen funktionelle krav til f.eks. performance. De følgende afsnit vil derfor kun indeholde overvejelser om hvilke elementer, der er nødvendige i implementeringen, hvordan de skal arbejde sammen, og ingen overvejelser om funktionelle krav.

6.1 Overordnet design

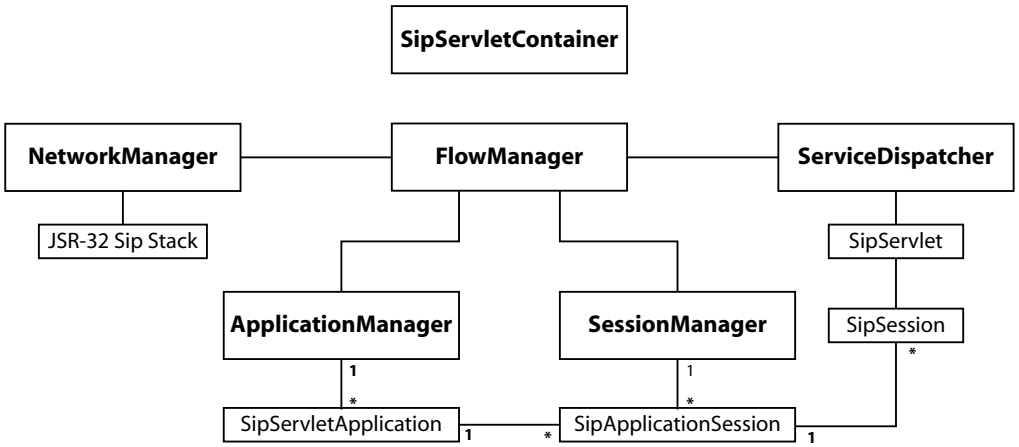
Implementeringen vil som beskrevet i analysen bestå af en række dele der hver især løser specifikke opgaver. Som vist på figur 6.1 er de enkelte dele implementeret som enkelte klasser samt en række hjælpeklasser. NetworkManager implementerer netværkshåndteringen og benytter som foreslået i analyse-afsnittet en JSR-32 implementering som et nedre abstraktions-lag.

Hver del spiller hver sin rolle i containeren og vil blive beskrevet i de efterfølgende afsnit:

6.2 SipServletContainer - bruges til at starte og stoppe containeren.

6.3 FlowManager - koordinerer flowet af data mellem de andre dele

6.4 NetworkManager - er containerens grænseflade til en JSR-32 implementering



Figur 6.1: De centrale dele(fremhævet) i containeren samt hjælpeklasser.

6.6 SessionManager vedligeholder tilstands-information omkring de igangværende dialoger

6.7 ApplicationManager administrerer de forskellige SipSevlet-applikationer

6.5 ServiceDispatcher er containerens grænseflade til de enkelte SIP Servlets når en forspørgsel eller et svar skal delegeres til en SIP Servlet.

De forskellige dele arbejder sammen for at løse de forskellige problemstillinger en JSR-116 implementering bliver stillet overfor. Dette samarbejde kræver to vigtige elementer: logik, der kan koordinere samarbejdet og klasse-instanser, der kan benyttes til at udveksle data.

Kontrol-logikken vil alt efter situationen være placeret enten i de enkelte dele, eller være implementeret i en separat del. Dette beskrives nærmere i afsnit 6.3.

Dataudveksling foregår med klasse-instanser fra tre pakker:

1. javax.sip.* indeholder klasser beskrevet i JSR-32.
2. javax.servlet.sip.* indeholder klasser beskrevet i JSR-116.
3. dk.itu.ssc.* indeholder implementerings-specifikke klasser.

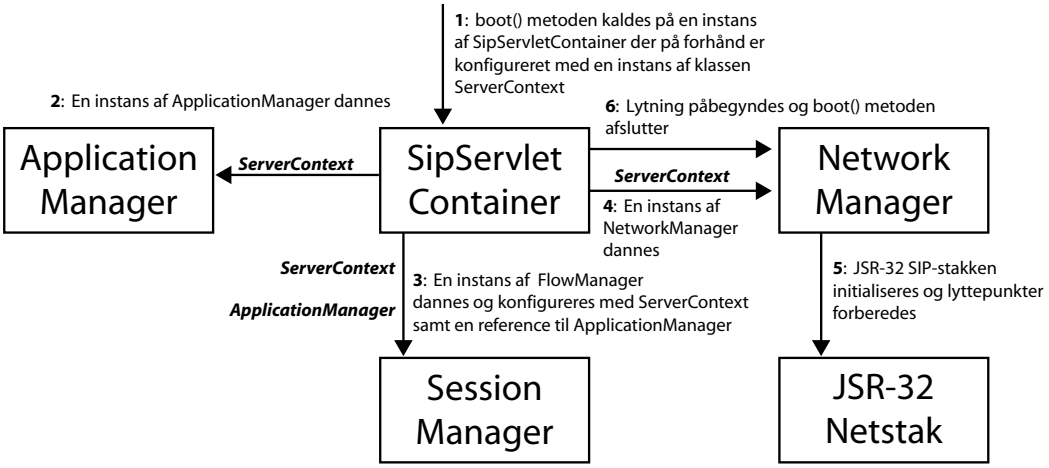
JSR-116 klasse-instanser benyttes hovedsageligt når containerne skal kommunikere med en Servlet, JSR-32 klasse-instanser benyttes til kommunikation med SIP netstakken, dk.itu.ssc.* klasse-instanser bruges internt i containeren til dataudveksling samt som mellemled i konverteringen imellem JSR-116 og JSR-32 klasse-instanser. Denne konvertering bliver beskrevet i detaljer i afsnit 6.4.1, kode-eksempel 6.1 på side 33 viser et eksempel på hvordan de forskellige typer bliver benyttet i containeren.

Listing 6.1: Simplificeret kode-eksempel, der viser hvordan de forskellige typer bruges mellem de forskellige dele. Fulde type-navne er benyttet selektivt for at fremhæve hvor de enkelte typer stammer fra.

```
dk.itu.ssc.SipNetworkManager implements javax.sip.SipListener { ...
   public void processRequest(RequestEvent requestEvent) {
      ...
      // JSR-32 klasse-instansen modtages i NetworkManager
      javax.sip.messageRequest jainRequest = requestEvent.getRequest();

      // instansen wrappes
      dk.itu.ssc.network.SipServletRequestImpl requestWrapper =
         new dk.itu.ssc.network.RequestImpl(SipServletRequestImpl);

      // og sendes videre til FlowManager
      dk.itu.ssc.FlowManager.processRequest(req);
}

public class FlowManager {
...
   public static void processRequest(SipServletRequestImpl req) {
   ...
   // FlowManager danner en instans af ServiceDispatcher via
   // ApplicationManager
   dk.itu.ssc.Dispatcher dispatcher =
      dk.itu.ssc.application.ApplicationManager.route(req);

   // wrapper-instansen sendes videre via dispatcheren
   dispatcher.forward(req, null);
   }
}

public class ServiceDispatcher{
...
javax.servlet.sip.SipServlet servlet;
...
   public void forward(ServletRequestImpl req, ServletResponseImpl res){

      // RequestImpl implementerer JSR-116 interfacet SipServletRequest
      // og kan derfor gives videre til en SipServlet instans
      servlet.service( (javax.servlet.sip.SipServletRequest) req,
             (javax.servlet.sip.SipServletResponse) res,
         );
   }
}
```

6.2 SipServletContainer

Denne del står for den overordnede opstart og nedlukning af containeren. Under opstart står SipServletContainer for initialisering og konfigurering af de enkelte dele. Konfigurationen er indeholdt i en instans af ServerContext. Figur 6.2 viser hvordan SipServletContainer-klassen koordinerer opstarten af containeren.



Figur 6.2: Container opstart

SipServletContainer tilgås via en Singleton, et designmønster, der sikrer at der til en hver tid kun kan eksistere én instans af en klasse. Instansen tilgås ved at kalde en statisk metode, og instansen kan dermed tilgås fra enhver klasse i containeren. En opstart af containeren kunne udføres på følgende måde:

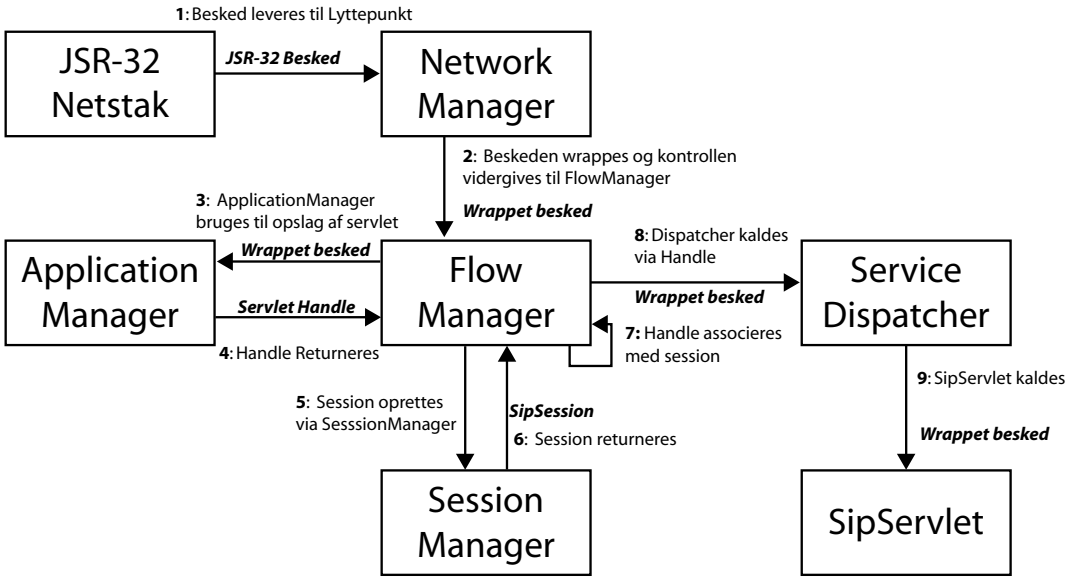
```
1    SipServletContainer.getInstance().boot();
```

6.3 FlowManager

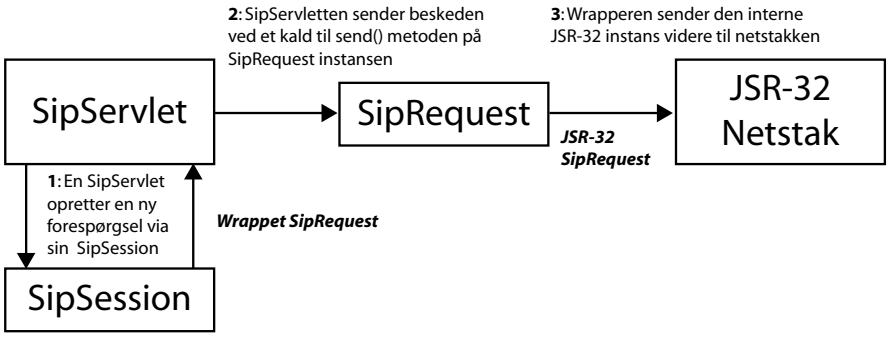
Når en besked (svar eller forespørgsel) modtages fra SIP-stakken, gennemgår den en række trin, før den endelig bevirker, at en SIP Servlet bliver kaldt. Disse trin involverer størstedelen af de forskellige dele i forskellig grad, og kommunikationen mellem de enkelte dele må derfor koordineres.

Logikken, der bestemmer i hvilken rækkefølge de forskellige trin udføres, kan enten distribueres i de enkelte dele, eller centraliseres i en dedikeret del. I den distribuerede løsning vil valget af det næste trin i flowet blive taget af den enkelte del. Fordelen ved denne metode er, at den er simpel og ikke komplicerer et potentielt simpelt flow, ved at indføre unødige ekstra trin. I den centraliserede løsning vil kontrollen skiftevis gå mellem de forskellige dele og FlowManager. Denne løsning egner sig bedst til mere komplicerede flow hvor det er ønskeligt at separere flow-kontrol, og de opgaver de forskellige dele løser.

Flowmanageren benyttes som vist på figur 6.3 ved modtagelse af beskeder fra SIP-stakken. Konceptuelt vil implementeringen benytte FlowManager til at koordinere alle flows, dog vil visse flows være så simple at de ikke indeholder et egentlig kald



Figur 6.3: Første besked



Figur 6.4: Besked afsendt fra SIP Servlet

til FlowManager. FlowManager-klassen vil f.eks. ikke blive kaldt når en SIP Servlet afsender en besked, som vist på figur 6.4. Skulle kompleksiteten i dette flow dog øges i en senere opdatering bør FlowManageren benyttes.

6.4 NetworkManager

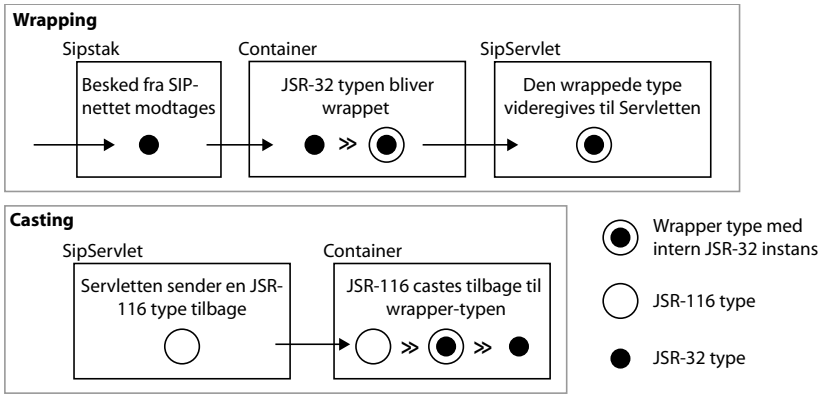
NetworkManager står for kommunikationen mellem containeren og en implementering af JSR-32. Networkmanageren registrer sig ved opstart som et lyttepunkt i SIP-stakken og vil efterfølgende modtage SIP-beskeder fra SIP-stakken. Et lyttepunkt registreres med en IP-adresse og en transport protokol, i dette tilfælde UDP eller TCP, og modtager efterfølgende alle beskeder, der matcher adresse og protokol.

NetworkManageren videregiver som, som vist på figur 6.3, alle beskeder til FlowManageren, der efterfølgende bestemmer hvilke trin der skal udføres.

Alternativt kunne man i stedet for en eksisterende implementering af JSR-32 have lavet implementering selv, eller have proprietær implementering. En proprietær

implementering kunne skræddersys til netop dette projekt, og vil derfor kunne være en del mindre kompleks at arbejde med end en JSR-32 implementering. Men, da implementering af en sådan ville have været et kæmpe projekt i sig selv, ville det være et dårligt valg. Ligeledes ville det til dette projekt være spild af tid at udarbejde en implementering af JSR-32 da en eksisterende implementering fuldt ud lever op til kravene i problemformuleringen.

6.4.1 Wrapping af JSR-32 typer



Figur 6.5: Konvertering imellem interne, JSR-32 og JSR-116 typer

Både JSR-116 og JSR-32 indeholder typer til repræsentation af de forskellige elementer i et SIP-netværk, så som adresser, forespørgsler, svar og SIP-dialoger. Alle SIP-releaterede typer i JSR-116 er repræsenteret i JSR-32, og implementeringen vil derfor benytte JSR-32 type-instanser til implementering af JSR-116 typerne. Typerne implementeres via designmønsteret Adaptor.

For hver JSR-116 type implementeres der en wrapper-klasse, der implementerer det krævede interface, samt indeholder en instans af en tilsvarende JSR-32 type. Wrapper-typen implementerer funktionaliteten i de forskellige metoder ved at delegere kaldet videre til den interne JSR-32 instans. Wrapperen forbereder og konverterer eventuelle argumenter, kalder JSR-32 instansen, og konverterer og returnerer eventuel retur-data.

Fordelen ved denne tilgang er at implementeringsarbejdet holdes på et minimum, JSR-32 typerne udfører størstedelen af arbejdet og det resterende arbejde består i at konvertere data før og efter delegeringen. Ulempen er, at implementeringen bliver afhængig af JSR-32 typernes måde at håndtere de forskellige metodekald, samt at containeren bliver nødt til at caste imellem JSR-116 og wrapper-typer. Afhængigheden er acceptabel da den er et resultat af kravet om at benytte en JSR-32 implementering.

Da al kommunikation, mellem containeren og dens SIP Servlets skal forgå via JSR-116 typer, og da containeren internt benytter sig af sine egne og JSR-32 typer internt, er det nødvendigt at kunne konvertere mellem de forskellige typer. Dette gøres med wrapping og casting som vist på figur 6.5. Når en besked videregives til en Servlet fra SIP-stakken wrappes typen inden den videregives til SIP Servletten. Liste 6.1 på side 33 viste hvordan en forespørgsel blev wrappet. Når en JSR-116 type skal konverteres tilbage til wrapper-typen, sker det via casting som vist i liste 6.2.

Listing 6.2: Metoden `setRequestURI` i typen javax.servlet.sip.SipServletRequest implementeres via en intern JSR-32 type

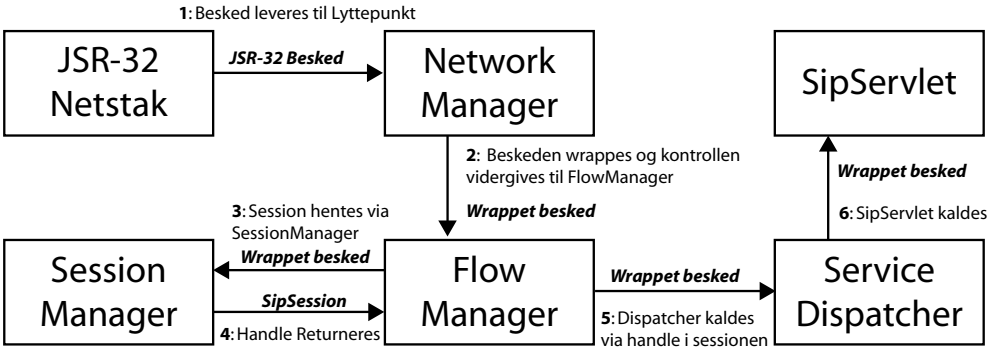
```
public class SipServletRequestImpl implements SipServletRequest{
   ...
      javax.sip.message.Request jsr32Request;
   ...
   public void setRequestURI(javax.servlet.sip.URI sipServletURI) {
      // metode-argumentet castes til wrapper-typen
      dk.itu.ssc.network.URIImpl sscURI = (dk.itu.ssc.network.URIImpl)
          sipServletURI;

      // JSR-32 typen udtrækkes fra wrapperen
      javax.sip.address.SipURI jsr32URI = sscURI.getJainURI();

      // og kan efterfølgende delegeres
      this.jsr32Request.setRequestURI(jsr32URI);
}
```

Problemet ved konverteringen fra JSR-116 typer til wrapper-typer, er at de sker ved hjælp af et "blindt" cast. Blindt fordi det ved oversættelsestidspunktet hverken er syntaktisk eller statisk muligt at afgøre om typen SIP Servletten leverer til containeren rent faktisk *er* en dk.itu.ssc.* type. Dette cast kan dog foretages uden fare under den antagelse, at SIP Servletten kun har adgang til én implementering af JSR-116. De relevante JSR-116 typer er alle interfaces, og kan derfor ikke instansieres direkte af SIP Servletten. Den eneste måde hvorpå SIP Servletten kan få adgang til disse typer er vha. forskellige metodekald, som containeren skal stille til rådighed. Hvis SIP Servletten udelukkende tilegner sig JSR-116 type-instanser via disse metoder, kan man med stor sikkerhed argumentere for, at alle JSR-116 type-instanser som SIP Servletten sender tilbage til containeren via metodekald, reelt vil være en dk.itu.ssc.* type.

Det blinde cast kunne undgås ved at udtrække al data fra instansen via metodekald defineret i JSR-116 interfacet. Disse data kunne efterfølgende bruges til at opbygge en instans af den tilsvarende JSR-32 type. Dette ville dog dels komplicere implementeringen en del, og dels ville der ikke være nogen garanti for, at det rent faktisk ville være muligt at konvertere alle typer på denne måde. Instansieringen af JSR-32 typerne kunne kræve data, der ikke er direkte tilgængelig via metodekald i JSR-116 typen.

6.5 ServiceDispatcher

ServiceDispatcher er grænsefladen imellem containeren og de enkelte Servlets. ServiceDispatcher er ansvarlig for at levere forespørgsler og svar til instanser af `SipServlet` via deres `service()` metode. Dette sker via et såkaldt ServletHandle, der indeholder information om hvordan den enkelte servlet skal kaldes. Dispatcheren benyttes direkte af SipServelets, når de ønsker at viderestille en besked fra sig selv til en anden SIP Servlet.



Figur 6.6: Efterfølgende besked

6.6 SessionManager

SessionManager har til ansvar at vedligeholde tilstands-information for igangværende SIP-dialoger i form af instanser af klasserne ApplicationSession og SipSession. En SipSession bruges til at holde de data, de involverede servlets ønsker at persistere mellem besked-udvekslinger og til at danne nye beskeder.

Der oprettes en SipSession hver gang containeren modtager eller afsender en forspørgsel der ikke i forvejen tilhører en SipSession. Dette vil f.eks. ske, når containeren modtager den første forespøgsel i et three-way handshake. SipSessionen oprettes ved modtagelsen af den første besked, og alle efterfølgende beskeder i dialogen vil tilhøre den samme SipSession.

SipSession bruges også til at holde et handle til den SIP Servlet, der er ansvarlig for at håndtere beskeder i en given dialog. Et handle til SIP Servletten oprettes vha. `ApplicationManager` ved modtagelsen af den første bedsked i dialogen, og gemmes i dennes SipSession. Dette handle bruges til håndtering af alle efterfølgende beskeder i dialogen, som vist på figur 6.6.

Alle SipSession er associeret til en instans af SipApplicationSession, der som før beskrevet bruges til at binde Servlets i konvergerede applikationer.

6.7 ApplicationManager

ApplicationManager administrerer de forskellige SIP Servlet applikationer indeholdt i SIP Servlet containeren. Ud over at holde referencer til de enkelte servlets, administrerer ApplicationManager en række SIP Servlet-mappings der bruges til at afgøre hvilke SIP Servlets der skal håndtere hvilke beskeder. Konfigurationen af de enkelte applikationer sker ved at indlæse deres tilhørende konfigurationsfil, den såkaldt DeploymentDescriptor, der beskriver hvilke SIP Servlets og tilhørende mappings applikationen indeholder.

ApplicationManager bruges af FlowManageren, når der skal findes et handle til at håndtere beskeder, der endnu ikke har fået tilknyttet en SipSession.

6.8 Opsummerende overvejelser

Designet er forsøgt udarbejdet så simpelt som muligt. Denne prioritering kan betyde at den færdige implementering viser sig utilstrækkelig i situationer hvor bl.a. høj performance er et krav. Men da dette som før nævnt hverken strider imod kravspecifikationen eller problemformuleringen, er dette acceptabelt.

Løsningen af problemet ved konvertering mellem JSR-32, JSR-116 og SSC-typer med wrapping kan ligeledes vise sig at være et problem i sjældne situationer. Da alternativet til wrapping ville være at udføre et særdeles komplekst arbejde, og dermed komplicere implementeringen (hvilket strider mere imod problemformuleringen end at containeren ikke vil fungerer optimalt i eksotiske situationer), er denne svaghed acceptabel.

På grund af det forholdsvis simple design kan centraliseret kontrol via FlowManager virke som unødig. Dette problem løses ved at tillade, at FlowManager bliver omgået i simple flows, med det forbehold, at FlowManager bliver inddraget i flowet hvis det ved en senere videreudvikling af implementeringen kompliceres. Da overskuelighed er et af målene for implementeringen, spiller FlowManager en vigtig rolle, og den evt lettere øgede kompleksitet, den skulle medbringe, accepteres derfor.

Endelig vil SipServletContainer-klassen spille en vigtig rolle i implementeringen af en evt. konvergeret HTTP- og SIP-servlet container, da den giver en eksisterende container et enkelt punkt hvorfra SIP Servlet Container implementeringen kan kontrolleres.

Afsnit 7

Implementering

De følgende afsnit vil gennemgå den centrale dele af implementeringen og give et overblik over hvordan de enkelte dele benyttes. Første afsnit giver et overordnet overblik over implementeringen, og de følgende afsnit gennemgår forskellige vitale dele af implementeringen. Målet med dette kapitel er at give læseren tilstrækkelig indblik i hvordan implementeringen fungerer, til at kunne benytte og evt. udvide implementeringen.

7.1 Overblik

Figur 7.1 på side 41 og 7.2 på side 41 viser de forskellige klasser, der bliver brugt i implementeringen, samt hvilke pakker, de tilhører. En udvidet version af figurene findes i bilag B på side 90 og 89. Klasserne er fordelt i pakker efter hvilken del af containeren de hører til:

dk.itu.ssc Generelle klasser der bruges til administration af selve containeren.

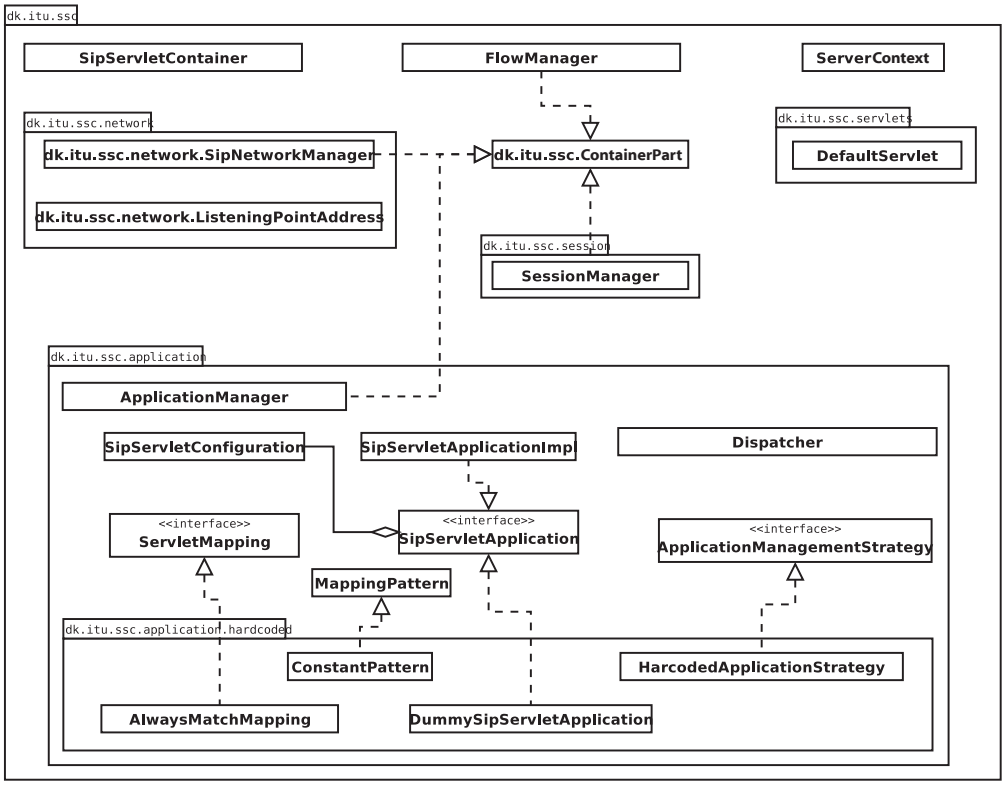
dk.itu.ssc.application ApplicationManager og klasser relevante for SIP-applikationer

dk.itu.ssc.network NetworkManager og klasser relevante for konfiguration af netværket.

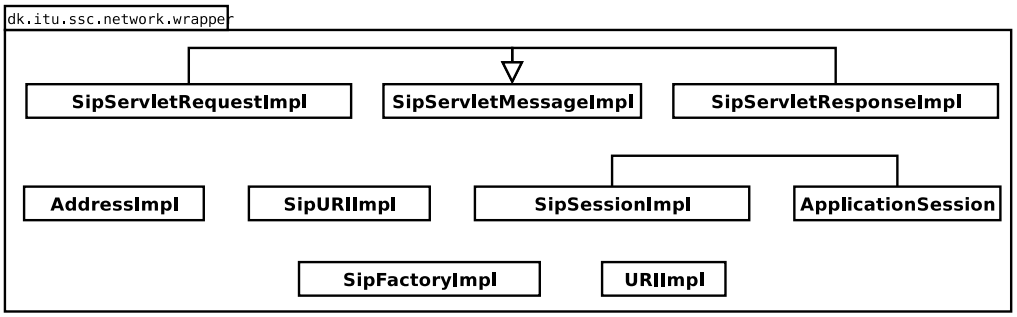
dk.itu.ssc.network.wrapper De forskellige wrapper-klasser

dk.itu.ssc.session SessionManager og klasser relevante for SIP- og applikations-sessioner.

Wrapper-klasserne implementerer de basale JSR-116 typer, og er placeret i en seperat pakke fremfor i pakker, der har med deres funktion at gøre. Således er klassen `SipSessionImpl` placeret f.eks i pakken `network.wrapper`, fremfor i pakken `session`. I en fremtidig udvidelse af containeren kunne det være ønskeligt at lave en alternativ implementering af wrapper-klasserne, og ved at placere disse i en seperat pakke, undgår man at have forskellige implementeringer af den samme wrapper-klasse i samme pakke.



Figur 7.1: Beskrivelse af klasserne i containere, klasser i pakken dk.itu.ssc.network.wrapper er vist på figur 7.2



Figur 7.2: Oversigt over wrapper-klasserne

Implementeringen benytter sig som beskrevet i design-afsnittet af en række eksisterende komponenter. Følgende skema beskriver hvilke komponenter, der benyttes, samt hvilken version af komponenten, containeren er implementeret med.

Navn	Filnavn	Version	Beskrivelse
Log4J	log4j-1.2.9.jar	1.2.9	Komponent til logging
JDOM	jdom.jar	1.0	Komponent til håndtering af XML-data
NIST JSR-32	nist-sip-1.2.jar	1.2	JSR-32 implementering lavet af The National Institute of Standards
JSR-32 API	JainSipApi1.1.jar	1.1	Standard JSR-32 API
Servlet API	servlet.jar	2.4	Standard Servlet API'et
SIP Servlet API	sipservlet.jar	1.0	Standard SIP Servlet API'et

7.2 Opstart og nedlukning af containeren

Klassen `SipServletContainer` koordinerer den overordnede opstart og nedlukning af containeren. Opstarten påbegyndes ved at kalde metoden `boot` der udfører opstarten. En instans af `SipServletContainer` kan som nævnt i design-afsnittet tilgås via en Singleton, og containeren kan således opstartes på følgende måde:

```
SipServletContainer.getInstance().boot();
```

Under opstarten instansieres følgende klasser

- `ApplicationManager`
- `FlowManager`
- `NetworkManager`

Klasserne implementerer alle interfacet `ContainerPart`, der definerer metoderne `configure(ServerContext)` og `shutdown()` der henholdsvis benyttes til konfiguration og nedlukning af de forskellige dele. De resterende dele af containeren kan til enhver tid tilgå konfigurationen via metoden `getConfiguration()`.

Nedlukningen af containeren startes ved et kald til metoden `shutdown`. Når denne metode returnerer, vil containeren være lukket ned, og alle netværksforbindelser afbrudt.

7.3 Routing af beskeder

Når en besked modtages af containeren, skal den, såfremt den er gyldig, videregives til en SIP Servlet. Hvis beskeden tilhører en eksisterende SipSession, vil den, som beskrevet i afsnit 6.6 i foregående kapitel, blive routet via et i forvejen oprettet handle. Tilhører beskeden ikke en eksisterende SipSession, bliver et handle dannet v.h.a. metoden `route(SipServletRequestImpl req)` i klassen `ApplicationManager`. Da den første besked i en dialog altid vil være en forespørgsel, accepterer `route`-metoden udelukkende forespørgsler.

Metoden `route` undersøger alle applikationer og tilhørende mappings `ApplicationManager` på det givne tidspunkt har kendskab til, og finder v.ha. disse informationer frem til hvilken SIP Servlet, der skal håndtere den forespørgslen. Metoden danner herefter en instans af Dispatcher, der bruges som handle til routing af requesten til den korrekte SIP Servlet, og senere til routing af efterfølgende beskeder i dialogen.

`ApplicationManager` implementerer routing-logikken via et Strategy-pattern. Denne tilgangsmåde gør det muligt for `ApplicationManager` at benytte forskellige strategier til at finde frem til det korrekte handle. Selve routingen udføres af en klasse, der implementerer interfacet `ApplicationManagementStrategy`, og da `ApplicationManager` udelukkende tilgår strategien via interfacet, kan routing-logikken udskiftes uden at skulle ændre på `ApplicationManager`. Valget af strategi sker ikke dynamisk under kørsel, men bliver valgt via en konfiguration før containeren bliver startet. Under test kan en instans af klassen `HarcodedApplicationStrategy` f.eks. bruges som strategi. Denne implementering af `ApplicationManagementStrategy` router alle beskeder til den samme SIP Servlet og gør det derved lettere at teste en bestemt SIP Servlet.

7.4 Logging

Logging sker via komponenten Log4J. Alle klasser der ønsker at skrive til den fælles log bruger en instans af klassen `Logger`, der dannes via følgende metodekald (eksempel fra `NetworkManager`)

```
Logger log = Logger.getLogger(SipNetworkManager.class);
```

Ved at specificere hvilken klasse, der ønsker at benytte `Logger`-instansen, er det muligt for Log4J at mærke alle beskeder sendt via instansen med klassens navn. Logbeskeder udsendes via metoderne `log.debug(String msg)` der bruges til beskeder relevante for fejlsøgning, `log.info(String msg)` der bruges til beskeder af generel interesse og `log.fatal(String msg)` der bruges i forbindelse med fejlbeskeder. Såfremt fejlen rapporteret med `log.fatal(String msg)` er så alvorlig, at containeren ikke bør fortsætte sin eksekvering, bør `shutdown()` metoden på `SipServletContainer` kaldes.

7.5 Integrering med eksisterende containere

Det er af JSR 116 erklæret som et vigtigt mål at en implementering af specifikationen kan integreres med eksisterende HTTP Servlet-containere. Der er ingen designmæssige hindringer for at SIP Servlet Container implementeringen kan integreres med en eksisterende HTTP Servlet container. Implementeringen kan kontrolleres og konfigureres udelukkende v.ha. `SipServletContaineren`-klassen. Der er dog en række problemstillinger der skal håndteres i forbindelse med integration af de to containere.

Styrende element Da der grundlæggende vil være tale om en integration af to selvstændige applikationer vil det være vigtigt at fastlægge en autoritativ enhed. Denne enhed kan enten være den ene af containerene der således vil skulle administrere den anden, eller en tredje udenforstående enhed hvis eneste opgave er at administrere de to containere.

Deling af ressoucer Begge containerere vil have brug for adgang til en række ressourcer så som netværk og filsystemer. Det skal overvejes om der på noget sted er overlappende behov der kan skabe konflikter, og hvordan de i så fald håndteres.

Overlappende datastrukturer JSR 116 beskriver at de to elementer i en konvergeret applikation(SIP og HTTP applikationerne) kommunikerer vha. en `SipApplicationSession`, der som før beskrevet er navngivet sådan som resultat af at Servlet-specifikationen ikke har taget højde for konvergerede applikationer. En Integration af de to containere vil således kræve en udvidelse af HTTP Servlet containeren, samt implementering af en funktionalitet der lader de to dele tilgå den samme instans af en `SipApplicationSession` .

Integrationen af SIP Servlet Container implementeringen og en eksisterende HTTP Servlet container vil ikke blive diskuteret i nærmere detaljer, da de specifikke tiltag der skal tages for at muliggøre integrationen afhænger af hvordan den endelige implementering af SIP Servlet Containeren ser ud.

Afsnit 8

Afprøvning

Dette afsnit vil beskrive hvordan afprøvning af implementeringen er foretaget. De første afsnit vil diskturere de forskellige problemstillinger, der er forbundet med at udføre afprøvningen, samt hvilke værktøjer, der rent praktisk benyttes til udføre afprøvningen. De resterende afsnit beskriver de forskellige testcases, deres resultater og bemærkninger til disse.

8.1 Valg af overordnet teststrategi

Overordnet skal der i forbindelse med afprøvningen af en implementering tages stilling til hvorvidt der skal udføres en strukturel afprøvning, funktionel afprøvning eller evt begge dele. Strukturel afprøvning er også kendt som whitebox eller intern afprøvning og funktionel afprøvning er også kendt som blackbox eller ekstern afprøvning.

I strukturel afprøvning tages der udgangspunkt i den interne struktur i programmet. Strukturel afprøvning har til opgave at eksekvere programmet med en række af særligt udvalgte input, der tilsammen er repræsentative for alle input programmet kan modtage. Således vil et program efter en succesfuld strukturel afprøvning have gennemgået alle dets mulige tilstande, og alle dele af progammet vil have været eksekveret minimum én gang.

Ved en ekstern afprøvning afprøves et program gennem dets eksterne grænseflader. En ekstern afprøvning har til opgave at forsøge at afgøre, om et givet program har en eller flere påkrævede funktionalliteter. Disse krav kan have forskellige karakterer, og er typisk beskrevet i en kravspecifikation til et givet program. Den enkelte eksterne afprøvning tager udgangspunkt i et krav og forsøger at påvise eller afvise om programmet lever op til kravet igennem et eller flere trin. Hvert trin påvirker programmet ved f.eks. at eksekvere en metode, og undersøger herefter om programmet opfører sig som forventet ved at læse output eller på anden vis at undersøge programmets tilstand.

Da der i modsætning til strukturen i et programmeringssprog ikke er nogen formel beskrivelse af hvordan et programs funktionalitet udtrykkes, er det ikke muligt endeligt at afgøre, om et program lever op til dets kravspecifikation udelukkende via ekstern afprøvning. Ekstern afprøvning kan dog være et meget nyttigt værktøj til at øge tiltroen til at et program lever op til dets kravspecifikation. Samtidig

har ekstern afprøvning den fordel, at det kun er nødvendigt at have adgang til et programs eksterne grænseflade for at kunne afprøve det.

Implementeringen vil udelukkende blive afprøvet v.h.a. ekstern/funktionel afprøvning. Resultatet af disse afprøvninger kan efterfølgende bruges til at afgøre, om dele af implementeringen skal afprøves grundigere, og om der evt skal laves intern afprøvning af disse dele.

8.2 Valg af testværktøjer

Med reference-implementeringen af JSR-116 følger, som før beskrevet, et testværktøj, det såkaldte TCK, designet til at afgøre hvorvidt en implementering af JSR-116 er udført korrekt. TCK'et er designet ud fra kravene i JSR-116, og er en ekstern afprøvning.

TCK'et består af en række test-cases, der beskriver hvordan en eller flere eksterne agenter skal interagere med tilhørende SIP Servlet applikationer installeret i containeren, der er under afprøvning. Agenterne opfører sig alt efter testens karakter som klienter eller servere. De er implementeret v.h.a. JSR-116 referenceimplementeringerne og har de samme mangler som denne, heriblandt manglende understøttelse af UDP som transport-protokol, og af modtagelse af SIP-beskeder, hvis linjer er ombrudt.

Fordelen ved at bruge TCK'et som test-værktøj er, at det er designet til at afprøve en JSR-116 implementering. TCK'et indeholder derfor en lang række afprøvninger, der er relevante for en JSR-116 implementering, og det er ikke nødvendigt at definere de forskellige test-cases manuelt. Et generelt testværktøj kan omvendt have en fordel i forbindelse med mere simple afprøvninger hvor man ikke ønsker at gennemløbe en komplet testcase fra TCK'et eller ønsker at afprøve en implementeringshåndtering af UDP.

Ulemperne ved at benytte TCK'et er, at dets agenter har de førnævnte mangler, og man derfor kan risikere, at en afprøvning fejler på trods af en fuldt fungerende implementering. Men det er for dette projekt særligt en ulempe, at TCK'et hovedsagelig er lavet for at afprøve en færdig implementering af JSR-116. De forskellige test-cases udfører en lang række afprøvninger af en masse funktionaliteter, hvoraf kun er et fåtal er relevante for containeren i dens nuværende tilstand.

Til afprøvning af dette projects implementering er valgt at benytte samme model som TCK'et, dvs med en eller flere test-agenter, der kommunikerer med en test-applikation. Da TCK'ets agenter i modsætning til containerne ikke understøtter UDP, vil afprøvnings-værktøjet SIPp i stedet blive benyttet. SIPp udmærker sig ved at være veldokumenteret og understøtter beskrivelse af call-flows via et XML-baseret sprog.

8.3 Design af testcases

Afprøvningen vil blive udformet som en række test-cases, der hver især afprøver funktionaliteten af de i analyse og design-afsnittene udvalgte dele. De følgende

afsnit vil beskrive hver testcase, og resultatet af dens udførsel. De forskellige testcases har til formål tilsammen at afprøve alle de implementerede dele af containeren. Følgende giver et kort overblik over hvilke dele, der er under test, samt hvilke funktionaliteter, der vil blive forsøgt afprøvet:

A: **ApplicationManager** Står for konfiguration og administration af de enkelte SIP Servlet-applikationer

- A.1 Indlæsning af applikations-konfiguration
- A.2 Initialisering af SIP Servlets
- A.3 Routing af beskeder til SIP Servlets

B: **FlowManager** koordinerer flowet mellem de enkelte dele

- B.1 Koordination af de andre del

C: **NetworkManager** står for kommunikation med Sip-stakken, heriblandt opsætning af lyttepunkter og modtagelse af beskeder

- C.1 Tilføjelse af lyttepunkter
- C.2 Modtagelse af beskeder
- C.3 Afsendelse af beskeder
- C.4 Udtrækning af dialog-id

D: **ServiceDispatcher** videregiver beskeder til de enkelte SIP Servlets

- D.1 Delegering af beskeder til SIP Servlets
- D.2 Forward af beskeder mellem SIP Servlets

E: **SessionManager** står for oprettelse af sessioner, samt at kunne hente dem frem, når en efterfølgende besked bliver behandlet af containeren.

- E.1 Oprettelse af session
- E.2 Opretholdelse af sessioner undervejs i en dialog
- E.3 Fremfindelse af sessioner

F: **SipServletContainer** administration af selve containeren

- F.1 Konfiguration af de enkelte dele ved opstart
- F.2 Signal om nedlukning til de enkelte dele

8.4 Testcases

8.4.1 Testcase 1, Opstart og nedlukning

Testen har til formål at undersøge om containeren kan startes korrekt op og lukkes korrekt ned. Ved opstart og nedlukning skal de forskellige dele af containeren henholdsvis konfigureres og notificeres om nedlukningen.

Fremgangsmåde

1. Start containeren op
2. Luk containeren ned

Suceskriterium

- Hver del skal have meldt konfiguration OK
- Hver del skal have meldt shutdown OK

Dele under test

- SipServletContainer: F.1, F.2

8.4.2 Testcase 2, Test af sessioner

Testen har til opgave at kontrollere om en værdi kan gemmes i en session, og senere hentes frem igen. På container-siden består testen af en servlet, der ved `doInvite` gemmer noget i sessionen, og ved `doBye` kontrollerer om den korrekte værdi findes i sessionen. Servletten gemmer remote-id'et, der er unikt for sessionen, og derfor samtidig kan bruges til at kontrollere, om sessions-håndteringen fungerer korrekt under samtidige requests. Skulle værdien ikke findes i sessionen ved kaldet til `doBye` , kastes en exception.

Testcases udføres for UDP såvel som TCP

Fremgangsmåde

1. start containeren
2. start en agent der laver et standard UAS testcase

Suceskriterium

- Dialogen oprettes og nedlægges uden fejl

Dele under test

- NetworkManager: C.1, C.3, C.2, C.4
- SessionManager: E.1, E.2, E.3
- ServiceDispatcher: D.1
- Flowmanager: B.1
- ApplicationManager: A.3

8.4.3 Testcase 3, Test af applikations-håndtering

Undersøger om ApplicationManager kan repræsentere en applikation korrekt ved først at indlæse en applikation, og derefter prøve at udtrække den tilsvarende data og se om den er korrekt. Under konfigurationen skal ApplicationManager kalde metoden `init()` på hver servlet, den aktiverer.

Fremgangsmåde

1. Få AM til at indlæse en deployment-descriptor
2. Forsøg at udlæse strukturen via de forskellige metoder

Successkriterier

- At de udtrukne data er korrekt
- At servletten i logfilen melder at dens init-metoden blev kaldt

Dele under test

- ApplicationManager: A.1, A.2

8.5 Resultater

8.5.1 Testcase 1

Afprøvningen forløb uden problemer. Input og output kan ses i bilag C.1 på side 91.

8.5.2 Testcase 2

Afprøvningen forløb uden problemer, men ved omkring hver 10. kørsel fejlede testen ved at en exception blev kastet fra `NetworkManager`. Der var ingen umiddelbar forskel på forholdende de succesfulde og fejlende afprøvninger blev afviklet under. Input og output kan ses i bilag C.2 på side 92.

8.5.3 Testcase 3

Ikke alle data i deployment-descriptoren var repræsenteret i `SipServletApplication` instansen. Følgende data blev udlæst korrekt:

- `<servlet>`
- `<servlet-class>`
- `<servlet-name>`

- `<init-param>`
- `<param-name>`
- `<param-value>`

Input kan ses i bilag C.3 på side 98.

8.6 Observationer og bemærkninger

Testcase 1 og 2 fejler af og til pga. en exception kastet i `NetworkManager`. En nærmere undersøgelse har vist, at problemet opstår, når `NetworkManager` forsøger at udtrække en unik identifikation for dialogen (et såkaldt dialog-id), og tilsyneladende tilfældigt fejler af og til. Eventuelle årsager til problemet kan være, at dialog-id'et bliver udtrukket på den forkerte måde, eller samtidigheds-problemer, da tråden, der skal forberede dialog-id'et i netstakken på kalds-tidspunktet, ikke har nået at gøre data klar.

Da målet for første iteration har været at implementere det grundlæggende design, har det faktum at testcasen rent faktisk bestås i de fleste kørsler dog større betydning end at den af og til fejler. Grunden er at dette viser at den grundlæggende arkitektur fungerer.

`ServiceDispatcher`'ens evne til at kunne forwarde beskeder mellem SIP Servlets er ikke afprøvet.

Der har ikke været indikationer af problemer med konverteringen mellem JSR-32 og JSR-116 typer.

Det har vist sig, at implementeringen kan bestå den første testcase i TCK'et men fejler alle andre. Negativt set betyder dette, at implementeringen er langt fra at være en gyldig JSR-116 implementering, men positivt set viser det, at implementeringen rent faktisk kan bestå en testcase og at en del grundlæggende funktionaliteter som sessions-håndtering og flow virker korrekt, og at implementeringen så at sige er på rette spor.

Sipp har vist sig at være et meget nyttigt test-værktøj, og kan stærkt anbefales.

På trods af en række mindre problemer under afprøvningen, vurderes det, at implementeringen lever op til kravene stillet i kravspecifikationen. Der mangler dog stadig en del arbejde før implementeringen kan leve op til JSR-116.

Afsnit 9

Konklusion og evaluering af implementeringen

Formålet med Del II har været at beskrive hvordan en implementering af JSR-116 kunne designes og implementeres. Målet med selve implementeringen har været at vise, at man selv med relativt få midler kan lave en implementering af JSR-116, og dermed tage det første skridt til at kunne udvikle tjenester til fremtidens kommunikations netværk.

Implementeringen har benyttet sig af komponenter til implementering af netværksfunktionalitet, indlæsning af XML-data og logging.

Afprøvningen af containeren har vist, at implementeringen lever op til kravspecifikationen udformet i begyndelsen af del II, men har også i tilkøb vist, at implementeringen i sin nuværende stand rent faktisk er i stand til at bestå en testcase fra JSR-116 TCK'et på trods af, at dette ikke har været et mål. De grundlæggende JSR-116 typer er implementeret ved hjælp af wrapper-klasser, og har under afprøvningen ikke udvist fejl. Dog fejler visse testcases tilsyneladende uprovokeret og tilfældigt.

Såfremt en række problemstillinger der hovedsageligt vedrører arbejds-fordeling og deling af ressourcer håndteres, forventes der ikke at være noget til hinder for at SIP Servlet container implmenteringen kan integereres med en eksisterende HTTP Servlet container.

Konklusionen er derfor, at implementeringen opfylder kravspecifikationen, samt at den, på trods af endnu ikke at kunne leve op til JSR-116, har vist, at det er muligt at lave en færdig implementering selv med få ressourcer.

Del III

Design af et SIP Servlet applikations-framework

Afsnit 10

Introduktion

Del III vil udarbejde et design af et framework, der bygger på SIP Servlets, og har som mål at lette udviklingen af SIP-baserede teletjenester. Som forberedelse på designet af frameworket vil de første afsnit diskutere fordele og ulemper ved frameworks generelt, og herefter nogle af de eksisterende web- og teleframeworks. Endelig vil de sidste afsnit udforme designet af frameworket.

10.1 Frameworks, platforme og API'er

Med "framework" vil der i de følgende afsnit menes en række af klasser, interfaces, konfigurationsfiler samt regler, der tilsammen definerer en struktur hvori en applikation kan udvikles. Et framework kan basere sig på eksisterende teknologier som HTTP Servlets, eller kan være implementeret fra bunden. Kendetegnende for et framework er, at frameworket i sig selv ikke kan noget før centrale dele er blevet implementeret af en applikations-udvikler.

En applikations-platform vil typisk være implementeret med henblik på at generalisere og simplificere underliggende lag. Ved at bruge en applikations-platform forsøger en udvikler at sikre sig, at applikationen vil blive ved med at fungere upåvirket af ændringer i de underliggende lag.

Et API er som regel en standardisering af en eller flere grænseflader. Et API implementerer i sig selv ikke nogen funktionalitet, men beskriver hvordan f.eks. to applikationer kan kommunikere. SIP Servlet specifikationen er et eksempel på et API.

Forskellen mellem frameworks, applikations-platforme og API'er kan i nogle tilfælde være minimal og svær at bestemme. En applikations-platform kan f.eks. vælge at tilbyde ekstra funktionaliteter, men bliver ikke nødvendigvis et framework af den grund. Et API kan inkludere standard-implementeringer af problem-stillinger, og dermed delvist gøre det ud for en applikations-platform, osv.

10.2 Motiverende faktorer

Fordelen ved et framework er, at applikations-programmøren kan fokusere på de forretnings-relaterede problemer, og mindre på de tekniske problemstillinger der

ikke er relevante for kernefunktionaliteten i en applikation. Et framework kan f.eks. lade programmøren implementere sin forretningslogik i komponenter, som frameworket efterfølgende binder sammen. Frameworks kan også tilbyde forsimplet tilgang til nogle af de ellers komplicerede ressourcer, som applikationen tilgår, så som fil-, database- og operativ-systemer. Programmerings-modellen, en programmør skal benytte for at implementere en given applikation, er oftere bestemt af de underliggende tekniske problemstillinger, end de forretnings-relaterede. Et framework kan i disse situationer tilbyde en alternativ programmerings-model, der passer bedre til de konkrete problemstillinger programmøren skal løse.

I bedste fald kan et velvalgt framework hjælpe en applikation godt i gang men omvendt kan det i værste fald forværre nogle af de problemstillinger, det prøver at løse. Viser frameworket at have uforudsete begrænsninger eller fejl, risikerer man at skulle håndtere problemstillinger, der ikke oprindeligt var planlagt. Hvis det ikke er muligt at rette fejlen fra applikations-siden, kan det være nødvendigt at vente på en opdatering af frameworket. I værste fald kan man blive nødt til at starte forfra på projektet, eller leve med fejlen hvis gen-implementering ikke er en mulighed. Et forkert valg af framework kan påføre applikationen akavede programmeringsmodeller eller unødig kompleksitet. Vælger man f.eks. at bruge et framework, der er beregnet til store enterpriseapplikationer, til en meget simpel applikation, kan man i sidste ende risikere at bruge mere tid på at opfylde krav fra frameworket end fra kunden.

Afsnit 11

Eksisterende platforme og rammeværktøjer

De følgende afsnit vil give en summarisk gennemgang af en række frameworks, der er relevante for designet af et framework baseret på SIP Servlets. Der findes et lang række forskellige frameworks på markedet, og valget af netop disse frameworks vil derfor blive begrundet.

Følgende tele-applikations frameworks vil blive beskrevet

- 11.1: Parlay/OSA og Parlay/X]
- 11.2.1: JAIN SIP
- 11.2.2: JAIN SLEE
- 11.2.3: JAIN SIP Servlets

Parlay-teknologierne er valgt på baggrund af popularitet på markedet, JAIN-teknologierne pga. SIP Servlets tilhørsforhold til disse.

Følgende frameworks til understøttelse af web-applikationer vil blive beskrevet:

- 11.3: Struts
- 11.4: JavaServerFaces
- 11.5: Spring Webflow

Struts er udvalgt pga. sin popularitet på markedet og JavaServerFaces dels på grund af at være en ny standard for MVC applikationer og dels på grund af af sin alternative programmerings-model. Endelig vil Spring WebFlow blive beskrevet for et give et eksempel på endnu bud på en alternativ programmerings-model. Alle baserer sig på HTTP Servlets og håndterer, på trods af at have den underliggende protokol til forskel, mange af de samme problemstillinger man kan forvente at skulle håndtere i en SIP Servlet applikation. Heriblandt adskillelse af forretnings- og præsentations-logik og bortabstrahering af uønskede egenskaber ved den underliggende Servlet-platform osv.

11.1 Parlay/OSA og Parlay/X

Parlay/OMA er et API, udviklet af "The Parlay Group", der er et samarbejde mellem 65 forskellige IT- og televirksomheder[Loz03]. Parlay/OSA APIerne muliggør implementering af netværks-uafhængige teleapplikationer og kan derfor blandt andet sikre investeringer i udvikling af nye teletjenester i en overgangsperiode til et nyt netværk. Parlay/OSA indeholder API'er til en række forskellige elementer en advanceret teleapplikation typisk vil have brug for, så som kald-kontrol (f.eks. viderestilling), adgang til lokations-information, adgangskontrol og platformsuafhængige datatyper. Parlay/OSA API'erne tilgås via CORBA[Loz03], og kan derfor benyttes fra en Java-applikation. En teleudbyder kan selv vælge på hvilket niveau den ønsker at benytte Parlay; Parlay kan indgå som alt fra en traditionel enhed i et IN-netværk(arkitektur der benyttes til implementering af tjenester i traditionelle telenet) til en fuld erstatning for IN[Loz05].

Parlay/X er en specifikation af hvordan dele af en Parlay/OSA applikation kan tilgås som en webtjeneste. Parlay/X API'et er en stærkt forsimplet udgave af Parlay/OSA API'erne, der gør en tjeneste-udbyder i stand til at tilbyde sine kunder adgang til sine tjenester og intern data uden kræve kendskab til det omfattende Parlay/OSA API.

Parlay/OSA og Parlay/X implementerer en høj abstraktion, der bevirker at man kan opnå netværks-uafhængige applikationer, der indeholder et minimum af netværksrelaterede problemstillinger. Parlay/OSA er blevet designet med henblik på at håndtere voldsomt komplicerede problemstillinger og er derfor også en meget stor specifikation. Parlay/OSA egner sig derfor bedst til mellemstore til store teleapplikationer.

11.2 JAIN

JAIN står for "Java API for Integrated Networks" og er ligesom Parlay/OSA et fælles tiltag mellem en lang række firmaer, der til dagligt er beskæftiget med Tele/IT verdenen [Mir04]. JAIN projektet består af en række ekspertgrupper, der udarbejder forskellige specifikationer til brug for kommunikationsverdenen. JAIN-projektet vil flytte udviklingen af teletjenester fra proprietære platforme over på åbne Java-baserede platforme. Ved at standardisere de grænseflader, teleapplikationer bliver implementeret op imod, bliver det muligt at flytte en applikation, udviklet på én platform over på en anden. Ved samtidig at lade alle standarderne være offentligt tilgængelige, muliggør man forskellige implementeringer af platformene. Alle standarder bliver fastlagt via "Java Community Process"(JCP), et åbent tiltag, hvor alle har mulighed for at bidrage til udviklingen af Java teknologier.

JAIN-projektet søger at levere[Mir04]:

Portabillitet af tjenester ved at vedtage, at alle specifikationer via JCP sikres åbne standarder, der kan implementeres af enhver. Dette muliggør platformsuafhængighed for applikationer, der følger JAIN-standarderne.

Netværksuafhængighed. JAIN-specifikationerne bliver udviklet med netværksuafhængighed i øjemed. Dette skal sikre, at et maksimum af de ydelser en given tjeneste muliggør, er tilgængeligt for kunden.

Lettere tilgang for udviklere. Ved at tilbyde lettilgængelige API'er til avancerede kommunikations-teknologier er det håbet, at JAIN-projektet vil lette udviklingen af fremtidens kommunikations-applikationer for nutidens Java-udviklere.

De forskellige specifikationer, udarbejdet af ekspertgrupperne, spiller forskellige roller. Specifikationerne spænder over så forskellige emner som Instant Messaging og ENUM (et system, der blandt andet kan bruges til at lede opkald mellem forskellige IP-telefoni udbydere). Dog har de alle kommunikation, samt de 3 ovennævnte mål, som fælles mål.

SIP Servlets er en af de JAIN-certificerede teknologier, der forventes at blive brugt på de centrale servere i et SIP-netværk.

I det følgende vil SIP Servlets blive beskrevet sammen med JAIN SIP og JAIN SLEE, der er to andre JAIN underprojekter, som er meget relevante for designet af rammeværktøjet.

11.2.1 JAIN SIP

JAIN SIP, der er blevet beskrevet summarisk tidligere, var den første SIP specifikation, der blev standardiseret gennem JCP[O'D03]. JAIN SIP giver en lavniveau tilgang til en implementering af en SIP-netstak. JAIN SIP giver en udvikler mulighed for at skifte netstak-implementering uden at skulle ændre i sin applikation. JAIN SIP giver fuld kontrol over stort set alle aspekter af SIP, og er tiltænkt projekter, hvor ydelse og finkornet kontrol er en høj prioritet. Det er bl.a. dette, der gør JAIN SIP ideel til brug i en SIP Servlet container.

I forhold til SIP Servlets giver Jain SIP langt mere kontrol på bekostning af at man skal håndtere mange SIP-relaterede problemer.

11.2.2 JAIN SLEE

SLEE står for "Service Logic Execution Enviroment" og definerer et standardiseret og kontrolleret miljø hvori advancerede kommunikations-tjenester kan udvikles[Lim03] [ea03b]. SLEE er en komponentmodel designet ud fra erfaringer og idéer fra J2EE specifikationerne, og mange af koncepterne fra SLEE så som containere, transaktions styring og management vil derfor virke bekendte for J2EE udviklere. Ligheden med J2EE muliggør også interoperabilitet mellem en SLEE og en J2EE applikationsserver via Enterprise JavaBeans™. SLEE specifikationen inkluderer blandt andet en såkaldt "Resource Adapter", der ligesom Parlay/OSA giver en generel tilgang til det underliggende netværk og v.h.a. forskellige implementeringer af disse giver platformsuafhængighed. Hvor Parlay/OSA hovedsagelig er et API, går SLEE skridtet videre og definerer en række færdigeimplementerede funktionaliteter en SLEE kompatibel applikations-server skal tilbyde SLEE applikationen. Applikationer bliver i SLEE udviklet som en række komponenter af forskellige typer, der kommunikerer via et hændelses-orienteret paradigme. En komponent spænder fra enheder, der binder andre komponenter sammen, til simple byggeblokke, der implementerer den egentlige funktionalitet.

Hvor SIP Servlets og JAIN SIP udelukkende giver tilgang til et SIP-netværk, er SIP blot en af de mange netværks-typer, en SLEE applikations server kan give adgang

til.

Ligesom det er tilfældet med Parlay/OSA giver JAIN SLEE et højt abstraktionsnivau, der både kan give mange fordele og i værste fald visse ulemper. Den rige funktionalitet, der bliver stillet til rådighed for udvikleren, betyder samtidig også, at der er meget at holde styr på og forstå. JAIN SLEE er derfor ligesom Parlay/OSA bedst egnet til større applikationer.

11.2.3 SIP Servlets

SIP Servlets ligger kompleksitetsmæssigt imellem JAIN SIP og JAIN SLEE. Mange af SIP Servlets egenskaber er allerede gennemgået, og vil derfor kun blive beskrevet summarisk her.

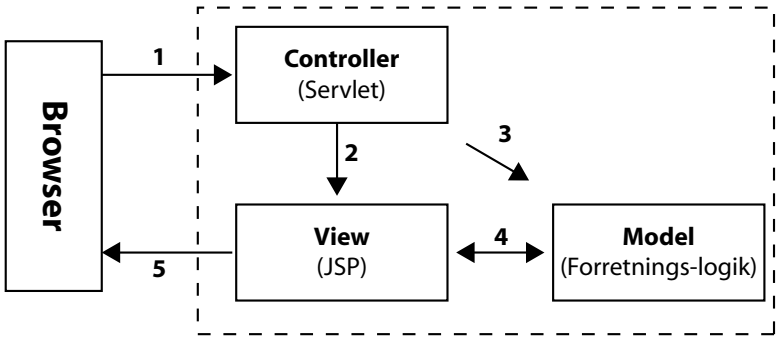
- Servlets er designet til at kunne tilgås flertrådet og til at bibeholde et minimum af tilstands-information, dette betyder at de potentielt kan give god ydelse selv under store belastninger.
- SIP Servlets er udelukkende designet til at kommunikere med et SIP-netværk, men kan kombineres med Web-applikationer vha. en kombineret SIP/HTTP-Servlet container.
- Servlet konceptet er velkendt pga. den store udbredelse af HTTP Servlets, og en udvikler, der har erfaring med HTTP Servlets, vil derfor fra begyndelsen føle sig mere hjemme end med f.eks. JAIN SIP eller JAIN SLEE.
- SIP Servlets er på én gang en simpel og kraftfuld tilgang til SIP-baserede applikationer. Simpel fordi specifikationen ikke indeholder mange punkter der ikke er direkte relateret til SIP, og kraftfuld fordi den kun fjerner de mest omstændige detaljer, så som ACK forespørgsler fra udviklerens kontrol, og lader de fleste andre SIP funktionaliteter stå til hans rådighed.

Hvor Parlay/OSA og JAIN SLEE egner sig bedst til større applikationer vil SIP Servlets primært henvende sig til små mellemstore applikationer, der hovedsageligt baserer sig på SIP.

11.3 Struts

Web-frameworket Struts, udviklet af The Apache Software Foundation, er et af de mere successfulde og velkendte forsøg på at lave et framework til understøttelse af web-applikationer. Struts følger designmønstret Model-View-Controller(MVC), hvori en applikation opdeles i dele, der håndterer henholdsvis repræsentation, visning og manipulation af data. Struts følger den såkaldte “Model 2”(vist på figur 11.1) implementering af MVC, der er blevet beskrevet af SUN i forbindelse med udvikling af JSP. I model 2 fungerer en HTTP Servlet som en central controller, der kommunikerer med en model og delegerer requests til Views implementeret med JSP.

Struts implementerer sin controller v.h.a. en særlig `ActionServlet`, der konfigureres med et antal `ActionMappings`, der hver især peger på en instans af en `Action` klasse. `ActionMapping`-instanser dannes af Struts ved at indlæse en særlig konfigurationsfil ved opstart, og de forskellige instanser af `Action`-klasser, der implementerer den egentlige controller-logik, implementeres af brugeren af frameworket.



Figur 11.1: Model 2 implementering af en Model-View-Controller arkitektur

Viewet implementeres ved afslutningsvis at videregive kontrollen til en JSP-side eller anden view-teknologi.

Ud over en MVC-arkitektur tilbyder Struts en lang række andre funktionaliteter, der er nyttige til implementering af web-applikationer, f.eks. objektificering af HTTP-argumenter, validering af input-data fra HTML-formularer, JSP biblioteker, der letter opbygningen af HTML-dokumenter, og meget andet.

Struts udmærker sig ved at tilbyde en solid struktur til implementering af web-applikationer, der både er simpel og påvirker implementering af applikationen mindst mulig. De eneste krav Struts stiller er, at man implementerer de dele af ens applikation, der skal kunne tage imod en forespørgsel, som en Action-klasse, der ikke kræver andet end at programmøren overskriver en enkelt metode. Struts understøtter således også en række andre View-teknologier end JSP, og alternative teknologier kan også let implementeres.

Simpliciteten i Struts medfører dog samtidig, at Struts i sig selv ikke er nok til at understøtte de fleste web-applikationer. Det vil som regel være nødvendigt at kombinere Struts med en række andre teknologier for at få et "komplet" framework, hvilket i bedste fald kan være en fordel, og i værste fald en tidsrøvende opgave.

11.4 JavaServerFaces

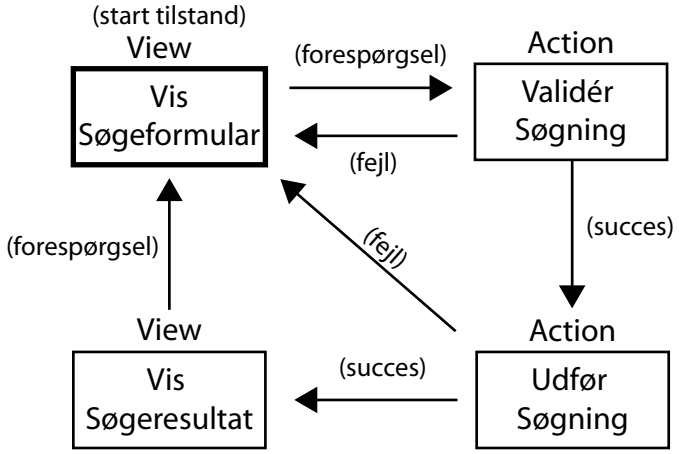
JavaServerFaces(JSF) er et MVC web-framework, der blandt andet tilbyder en komponentmodel samt opbygning af Views vha. generelle brugergrænseflade-komponenter. JSF er en J2EE standard og er standardiseret via JCP. JSF er som så mange JCP-projekter udarbejdet af en lang række store virksomheder, og har til mål at tilfredsstille den mangel, der har været på et standardiseret MVC framework. JSF bruger som udgangspunkt JavaBeans, JSP og Servlets til at implementere henholdsvis Model, View og Controller, men tillader ligesom Struts, at man bruger alternative View-teknologier.

JSF lader udvikler arbejde med sin web-applikation i et eventbaseret paradigme. En programmør kan således implementere metoder, der vil blive eksekveret når en bruger trykker på en knap, udfylder et felt, eller på anden måde interagerer med websiden. Det eventorienterede paradigme har, ud over at være velkendt for applikations-programmører, den fordel at udvikleren selv vælger præcis hvilke komponenter han ønsker skal have tilknyttet ekstra funktionalitet, og kan lade

de resterende komponenter opføre sig normalt.

Komponentmodellen og de standardiserede brugergrænsefladekomponenter opfordre i høj grad til genbrug, og i kraft af at være en J2EE standard vil JSF være tilgængelig i alle J2EE kompatible web-containere.

11.5 Spring og Spring WebFlow



Figur 11.2: Flow mellem Action og View elementer i et WebFlow

Spring er et J2EE applikations-framework, der har til mål at gøre det lettere at implementere J2EE applikationer. Spring tilbyder håndtering af en række J2EE problemstillinger, som administration af transaktioner, abstraktion af JDBC samt et MVC web-framework.

Spring benytter sig i høj grad af "Inversion of Control" (IoC)[Fow04], et princip, hvor de ressourcer en komponent er afhængig af, bliver indskudt i komponenten umiddelbart før dens logik afvikles, fremfor at komponenten selv skal tilvejebringe ressourcerne. Dette princip er også kendt som "Dependency Injection".

Spring inkluderer desuden en letvægtscontainer, der kan sammenbinde løst koblede komponenter. Containeren benytter sig blandt andet af IoC til at sikre, at komponenterne ikke bindes til containeren, men kan fungere uden for containeren. Dette gør blandt andet de enkelte komponenter lettere at afprøve, da en afprøvning blot laver den samme Dependency Injection, som containeren ellers ville have gjort umiddelbart før afprøvningen foretages.

Spring WebFlow er en viderebygning af Springs model 2 MVC implementering. Spring WebFlow lader applikations-programmøren arbejde med flowet mellem de enkelte sidevisninger og model-manipulationer separat fra forretnings- og præsentationslogik. Flowet beskrives i en separat konfiguration, og definerer de forskellige trin og overgange i et givet flow. Et flow beskriver hvilke tilstande applikationen kan gå igennem, og hvad der udløser overgangen fra én tilstand til en anden. Figur 11.2 viser de tilstande en simpel web-applikation der lader en bruger søge i en database, kan gennemgå. Flowet er således en implementering af et tilstands-diagram, hvor hver tilstand internt er en "Action", der kan manipulere med modellen, eller et "View", der præsenterer data fra modellen. Hver view-tilstand resulterer i en

side-visning, mens kontrollen kan gå mellem en række forskellige Action-tilstande før den endeligt gives til et view.

11.6 Opsummering

Navn	Kompleksitet	Velegnet til	Netværksabstraktion
Parlay/OSA	høj	implementering af større applikationer	ja
JAIN SIP	mellem	low-level SIP applikationer	nej
SIP Servlets	lav	mindre høj- og mellemniveau SIP applikationer	nej
JAIN SLEE	høj	større komplekse telekommunkations-applikationer	ja
Struts	lav	mellemstore til store webapplikationer	nej
JSF	mellem	mellemstore til store webapplikationer	nej
Spring WebFlow	mellem	mellemstore til store applikationer med fokus på komponenter	nej

Afsnit 12

Diskussion

Før design af frameworket kan påbegyndes, må der først udarbejdes mål for hvilke egenskaber, frameworket skal besidde. Dette afsnit har til formål gennem en diskussion at definere disse mål. Dette vil blandt andet ske ved at udvælge de typer af applikationer, frameworket skal henvende sig til, samt ved at reflektere over nogle af de observationer der blev gjort omkring eksisterende frameworks i forrige afsnit.

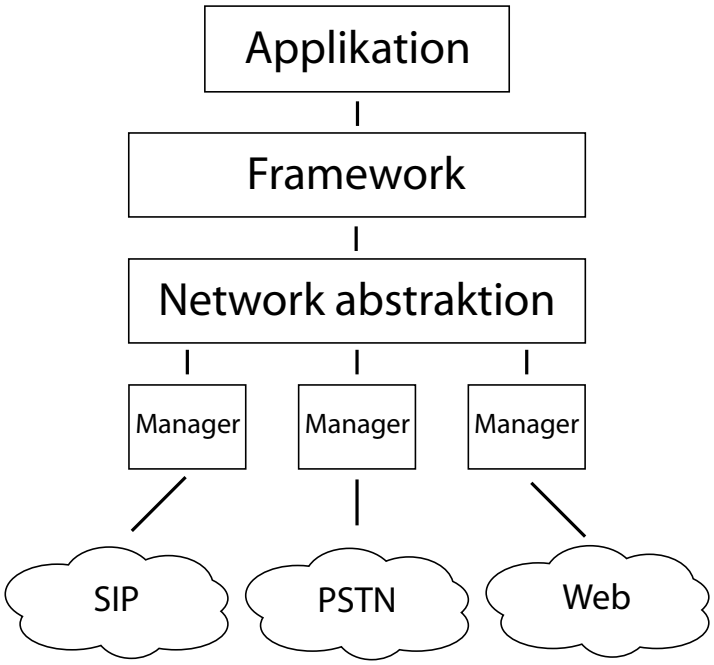
12.1 Indledende diskussion

Først og fremmest er det vigtigt at gøre sig klart hvilke typer applikationer frameworket skal henvende sig til. SIP Servlets ligger rent kompleksitetsmæssigt, som før beskrevet, et sted imellem JAIN SIP og JAIN SLEE. I design af et framework baseret på SIP Servlets vil det derfor være vigtigt ikke at prøve at genopfinde JAIN SLEE ved at tilbyde "for" avancerede features og samtidig huske, at SIP Servlets er tænkt som en let tilgang til SIP-baserede tjenester, og derfor ikke fokusere for meget på low-level features der ville være bedre håndteret af JAIN SIP. Frameworket vil hovedsageligt fokusere på undestøttelse af SIP *Server* applikationer, dvs applikationer hvis eksekvering påbegyndes af en forespørgsel.

Interessant ved SIP Servlets er, som før nævnt, at de henvender sig til de eksisterende J2EE miljøer. Man kan forestille sig, at SIP Servlets kunne spille en vigtig rolle i integration mellem eksisterende J2EE applikationer, og SIP-baserede kommunikations netværk. SIP Servlets ville i en sådan situation fungere som et tyndt mellemled, hvis hovedopgave på den en side ville være at kommunikere med eksisterende J2EE applikationer og på den anden side at viderebringe kommunikation til en eller flere enheder i et SIP-netværk. I en sådan applikation ville det være ønskeligt at de mere komplicerede SIP detaljer blev håndteret automatisk, og udvikleren dermed kan fokusere mere på integration af forretningslogikken.

I fremtiden er det højst sandsynligt, at udviklingen vil gå fra at handle om integration af eksisterende systemer, til udvikling af nye systemer, der fra start er designet til at interagere med et SIP-netværk. SIP Servlets vil være langt mindre egnet til denne type af applikation, da SIP Servlets som sådan ikke tilbyder mange andre features end en abstrahering af SIP. Større systemer som f.eks. JAIN SLEE tilbyder en lang række funktionaliteter som indbygget netværks-uafhængighed og en avanceret komponentmodel, der, såfremt teknologierne bliver benyttet korrekt, gør den

mere egnet til udviklinger af komplekse applikationer fra bunden end SIP Servlets.

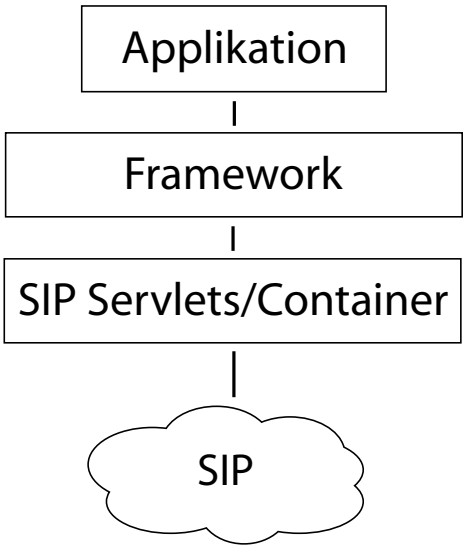


Figur 12.1: Højniveau framework der abstraherer netværket væk

Frameworket vil derfor hovedsaglig fokusere på at understøtte letvægts SIP-applikationer, hvor man ønsker at have mere fokus på forretningslogik end på SIP-releaterede problemstillinger. Et af målene med frameworket bør derfor være at bygge videre på den bortabstrahering SIP-netværket som SIP-Servlets til en hvis grad allerede tilbyder. Med høj grad af abstraktion vil frameworket fuldstændig bortabstrahere det underliggende netværk, og vil kunne bruges til at implementere netværksuafhængige applikationer. Det høje abstraktionsniveau kan opnås ved at implementere et overordnet design som det afbilledet på figur 12.1. En SIP Servlet container ville i et sådant framework være ansvarlig for at implementere kommunikationen mellem det øverste lag i frameworket, der interagerer med applikationen, og et SIP netværk. Det ville være et stort arbejde at implementere dette niveau af netværksabstraktion, og da en SIP Servlet container under alle omstændigheder kun ville kunne indgå i den nederste del af et sådan framework, ville et framework bygget på SIP Servlets være uegnet til at løse denne opgave. Eksisterende frameworks som Parlay/OSA og JAIN SLEE implementerer netværks-abstraktions på dette niveau.

Frameworket bør i stedet placere sig som vist på figur 12.2 og fokusere på at bortabstrahere elementer i SIP Servlet API'et, der findes irrelevante for applikationen. Frameworket bør ikke gøre noget særligt for at skjule, at applikationen kommunikerer med et SIP netværk.

Ud over abstraheringen af SIP bør frameworket også understøtte komponentbaseret udvikling. SIP Servlets er i kraft af at være en specialisering af Servlets en komponent-type. Komponent-modellen er dog ikke så "ren", som den kunne være, da programmøren i kraft bl.a. do* metoderne bliver nødt til at tage stilling til SIP-relaterede problemstillinger. I stedet burde det være muligt for programmøren at opdele sin forretnings- og sip-logik i separate komponenter, der efterfølgende



Figur 12.2: Highlevel SIP-based framework

kunne kædes sammen.

Ved at lade udvikleren implementere sin logik som komponenter, opfordres der også til genbrug. Såfremt den enkelte komponent er udformet generelt nok, kan den genbruges fra projekt til projekt. Det er dog stadig vigtigt at holde komponentmodellen så simpel som mulig, da det ikke er et mål at erstatte JAIN SLEE.

Som antallet af forskellige eksisterende frameworks og deres feature-lister viser, er der et utal af problemstillinger man kan vælge at takle i et framework. Den videre diskussion vil dog holde sig til de problemstillinger der netop er blevet diskuteret: Abstraktion af SIP Servlet API'et samt øget understøttelse af komponenter.

12.2 Abstraktion af SIP Servlet API'et

Et af de steder SIP Servlets tvinger en udvikler til at beskæftige sig med potentielt unødige SIP-relaterede problemer er de forskellige "do*" metoder som `doINVITE`, og `doOPTIONS`. Ved at lade frameworket håndtere disse metoder, undgår man at en udvikler skal sætte sig ind en række potentielt irrelevante metoder i SIP, og dermed hurtigere kan komme i gang med den egentlige implementering. Web-frameworket Struts tilbyder en lignende funktionalitet ved at lade alle forespørgsler blive håndteret af en "Controller Servlet", der delegerer den enkelte forespørgsel til en "Action"-klasse implementeret af udvikleren. På denne måde undgår udvikleren at skulle bekymre sig om de forskellige do* metoder i HTTP Servletten, og kan i stedet fokusere på forretningslogikken, der skal udføres på den enkelte side.

Under en typisk SIP dialog bliver der benyttet flere forskellige metoder end man normalt ville bruge i kommunikation med HTTP. I HTTP vil `GET`-metoden som oftest være den eneste, der bliver benyttet, mens en SIP applikation, der ønsker at kunne gå igennem et handshake og efterfølgende kunne nedlægge dialogen må som minimum benytte metoderne `INVITE`, `ACK` og `BYE`. Langt de fleste HTTP Servlets kan dermed nøjes med at implementere `doGet`, mens en SIP Servlet som minimum må forventes at implementere metoderne `doInvite`, `doAck` og `doBye` samt metoder

til håndtering af de forskellige fejl-svar, der kan modtages undervejs. Dette er en naturlig konsekvens af, at HTTP er en forholdsvis simpel klient/server protokol baseret på TCP hvor en kommunikation består af én forespørgsel og ét svar. SIP er derimod en noget mere avanceret peer-to-peer protokol, baseret på en upålidelig transport-protol (UDP), og bliver blandt andet nødt til selv at kunne kompensere for tabte beskeder. Den øgede kompleksitet i SIP betyder, at det vil være utilstrækkeligt at håndtere logikken der ellers ville være indkapslet i en SIP Servlet, med et kald til en enkelt metode i en `Action`-klasse.

Selvom kommunikation med SIP er mere kompliceret end med HTTP, kan en SIP Server applikation falde i en af følgende fire roller beskrevet i SIP specifikationen (RFC 3261)[ea02]:

Proxy - en applikation der tager imod en forespørgsel, og efterfølgende agerer stedfortræder for klienten.

Redirect - en applikation der modtager en forespørgsel, og efterfølgende viderestiller klienten med et `3xx`-svar.

Registrar - en applikation der kan modtage registrerings-information fra en klient.

UAS - generel kategori for en applikation der tager imod forespørgsler og som ikke falder i en af de ovenstående kategorier

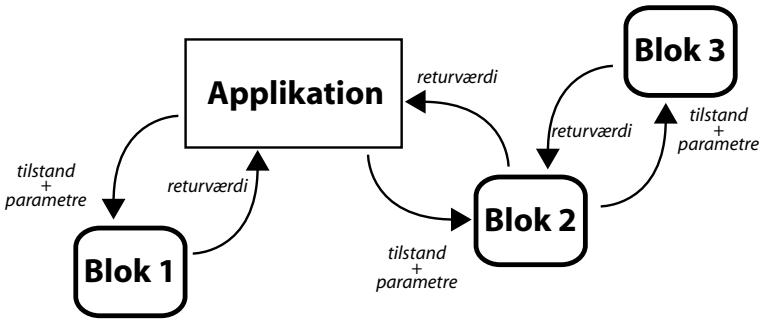
I stedet for at tilbyde en enkelt `Action`-klasse, kan SIP Servlet frameworket tilbyde generelle implementeringer af de 4 ovenstående roller i form af "standard-scenarier". Frameworket ville tilbyde standard-implementering af de forskellige trin i scenarierne, der efter behov kan overskrives af applikations-programmøren. Ved at åbne mulighed for at applikations-programmøren selv kan definere scenarier, kan frameworket samtidig lade udvikleren fokusere på flowet i applikationen, og forretningslogikken separat. Dette princip er også benyttet i webframeworket Spring Webflow.

Ud over selve forretnings-logikken vil standard-scenarierne også skulle konfigureres med en række data som f.eks. adressen på den server en Redirect-server skal videresende en klient til.

12.3 Øget understøttelse af komponenter

Komponenter er som tidligere beskrevet indkapslet logik, der kan tilgås via standardiserede grænseflader. SIP Servlets indkapsler funktionalitet og tilgås gennem SIP Servlet API'et, og kan derfor betragtes som en komponent. Men, da SIP Servlet API'et, som forrige afsnit diskuterede, kræver at applikations-udvikleren tager stilling til SIP-specfikke problemer i kraft af do* metoderne, er SIP Servlets ikke egnede som komponenter til indkapsling af forretnings-logik, der ikke beskæftiger sig med SIP. Komponentmodellen frameworket tilbyder bør derfor være mere generel end den SIP Servlet API'et tilbyder, være uafhængig af SIP. Det foregående afsnit beskrev hvordan man ved hjælp af standard-scenarier kan lette udviklingen af applikationer ved at lade applikations-udvikleren fokusere mere på den rolle, applikationen skal spille og dermed på de problemstillinger, der er specifikke for rollen. Men brugen af standard-scenarier afhjælper i sig selv ikke problematikken med adskillelse af forretnings- og SIP-relateret- logik.

Det er *ikke* et mål for frameworket totalt at eliminere SIP-logik i applikationen, da manipulation med SIP er en af SIP Servlets, og dermed også de applikationer der vil basere sig på SIP Servlets, hovedopgave. Frameworket skal i stedet tilbyde en model hvormed man så pænt som muligt kan opdele sin applikation i komponenter, der har hvert sit virkeområde.



Figur 12.3: Applikation der benytter byggeblokke

Frameworket vil tilbyde, at man implementerer sin logik i enkelte komponenter kaldt bygge-blokke. Hver bygge-blok indkapsler en mængde logik implementeret af applikations-programmøren, og kan modtage parametre samt tilstandsinformation, og efterfølgende afgive en retur-værdi. Under sin eksekvering kan bygge-blokken have en eller flere side-effekter, der udfører den ønskede funktionalitet (se figur 12.3.

I forbindelse med konfiguration af et givent scenarium beskrives hvilke byggeblokke scenariet skal kunne benytte sig af. Hver bygge-blok, der skal kunne benyttes, tildeles et navn, og tildeles eventuelle konfigurations-værdier.

En bygge-blok kan indkapsle funktionalitet er så som opslag af brugernavn og password, udregning af adresse til given bruger, formatering af data og meget andet.

Ideelt skal en applikation kunne bygges op ved at kombinere en række forskellige bygge-blokke med en minimal mængde logik, der kobler de forskellige blokke sammen. En blok kan tage en anden blok som argument, og kan eksekvere hinanden. En blok, der kan undersøge, om et brugernavn og password passer sammen kan f.eks. kombineres med en blok der kan tilgå en bruger-database. Hvordan komponenten konkret modtager input og afgiver output beskrives nærmere i designafsnittet på side 69.

Ved at fastlægge grænsefladen til de generelle komponenter i frameworket, åbner man mulighed for, at frameworket selv kan tilbyde en række bygge-blokke, der kan udføre standard-opgaver, eller fungere som en standardiseret grænseflade til ressourcer, der ellers ikke kunne tilgås via API'et.

12.4 Tilstands information

Servlet specifikationen tilbyder, at den tilstandsinformation, en Servlet har brug for at lagre undervejs i en kommunikation, lagres i et `Session`-objekt. Data lagres i et `Map` som `Object`-instanser indekseret af `String`-instanser. Problemet ved at lagre

data på denne måde, er at type-sikkerheden mistes, og instanser lagret i sessionen må konverteres tilbage til deres oprindelige typer, før de kan benyttes.

Scenarierne og byggeblokkene, der benyttes i forbindelse med dem, vil have samme behov for at lagre tilstandsinformation som de, SIP Servlets applikationen ellers ville være implementeret med. Fremfor at den underliggende `Session`-instans, som frameworket har adgang til i kraft af at være en overbygning af SIP Servlets, bør frameworket tilbyde en mere type-sikker løsning. Det forventes, at applikationsprogrammøren på forhånd ved, hvilke datatyper, der skal lagres undervejs i et scenarium, og det derfor er muligt på forhånd at definere en passende klasse, der kan benyttes til at lagre tilstands-information. Frameworket vil efterfølgende kunne danne en instans af klassen umiddelbart før scenariet, der skal benytte den, påbegyndes, indskyde eventuel konfigurationsinformation, og overbringe en reference til instansen til de forskellige dele af scenariet, der har brug for at lagre tilstandsinformation.

De enkelte byggeblokke kan ligeledes benytte tilstandsobjektet til at lagre deres tilstand. Ved at undgå lagring af tilstandsinformation i den enkelte byggeblok, kan frameworket lade alle scenarier tilgå den samme instans af byggeblokken, fremfor at skulle danne en ny instans for hvert scenarium.

Endelig vil centraliseringen af tilstandsinformationen også lette afprøvningen af de enkelte byggeblokke udenfor frameworket, da en testcase blot behøver at danne en passende instans af tilstands-klassen, fremfor at skulle simulere tilstedeværelse af hele frameworket.

12.5 Lokationstjeneste

RFC 3261 beskriver på linje med de fire standardelementer Proxy, UAS, Registrar og Redirect også en lokationstjeneste, som en abstrakt tjeneste elementerne bruger til lagring af information om en klients placering på SIP-nettet. Specifikationen beskriver ikke hvordan en sådan tjeneste skal implementeres, men blot at den på baggrund af en SIP URI, kan oplyse en eller flere alternative destinations-adresser.

Da en lokations-tjeneste er et centralt behov for standard-scenarierne, bør frameworket tilbyde en implementering af en sådan. Ved samtidig at tilbyde en standardiseret grænseflade til en sådan, åbner frameworket mulighed for at applikationsprogrammøren selv implementerer en lokationstjeneste, der efterfølgende kan benyttes af standardimplementeringen af de forskellige scenarier.

12.6 Opsummering af krav til frameworket

Frameworket der designes i efterfølgende afsnit vil tilbyde følgende funktionaliteter:

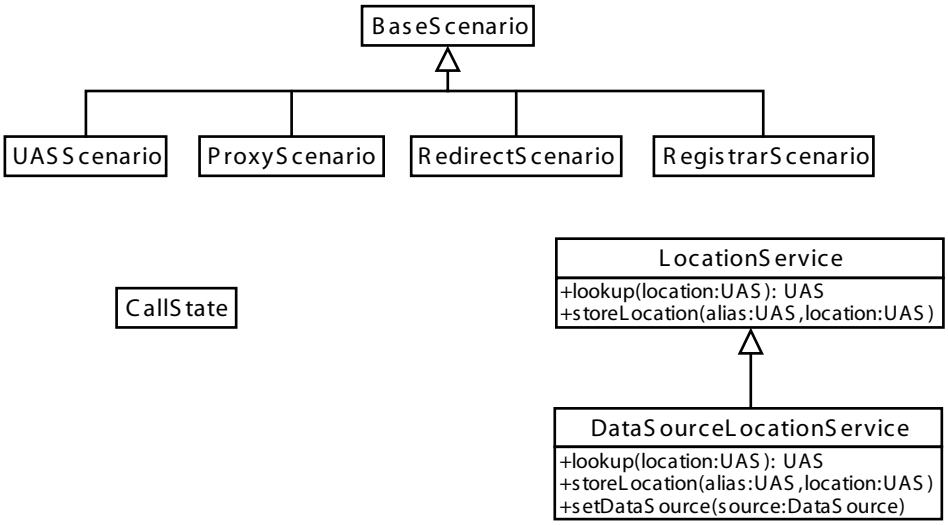
- En abstraktion af SIP Servlet API'et i form af en række standardiserede scenarier
- Mulighed for at applikations-udvikleren kan implementere den centrale logik i disse scenarier
- Fokus på komponentbaseret udvikling

- Typesikker repræsentation af tilstandsinformation
- En implementering af en lokationstjeneste, samt en standardiseret grænseflade til denne

Afsnit 13

Design

13.1 Overordnet design



Figur 13.1: Frameworketes overordnede design

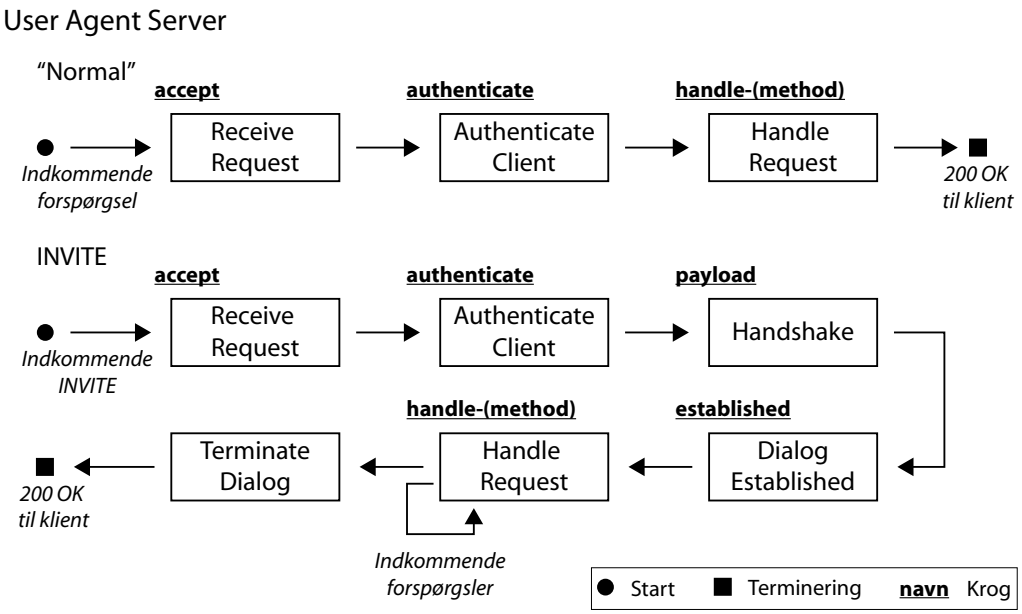
13.2 Standardscenarier

De følgende afsnit vil beskrive designet af de 4 standardscenarier: User Agent Server, Proxy, Registrar og Redirect. Hvert scenarium består af et foruddefineret flow, der bestemmer i hvilken rækkefølge en række trin bliver udført. Undervejs i flowet er der defineret en række punkter, de såkaldte “kroge”, hvorpå applikations-programmøren kan vedhæfte logik. Der er på forhånd lavet en implementering af hver krog, således at applikations-programmøren kun behøver at lave en implementering af denne, såfremt han er utilfreds med standardimplementeringen. Implementering af krogen vedhæftes ved at implementere logikken i en metode og efterfølgende angive klassen og metoden i en konfigurationsfil. Alt efter hvilken krog, der er tale om, vil metoden skulle have en bestemt signatur, og vil skulle returnere en bestemt type. Som minimum vil metoden skulle tage en instans af CallState som argument.

Beskrivelsen af hvert scenarium vil gennemgå dets flow, beskrive de forskellige kroge, deres standardimplementering, kravene til en ny implementering samt indeholde et flow-diagram, der illustrerer de enkelte trin i scenariet.

Krogene til de enkelte scenarier beskrives kun kort i de følgende afsnit. Den fulde beskrivelse kan finde i bilag E.

13.2.1 User Agent Server (UAS)



Figur 13.2: Flowet i User Agent Server scenariet

Dette scenarium er tiltænkt implementering af en generel SIP Server. Da beskrivelsen af en UAS er meget generel, tilbyder frameworket kun begrænset funktionalitet. Scenariet lader applikationsprogrammøren, implementere en server-applikation der kan modtage en forespørgsel og sende et svar.

Forespørgslerne kan inddeles i to grupper: én der kan oprette en dialog, og én der ikke kan. Såfremt forespørgslen ikke kan oprette en dialog, vil flowet være som vist øverst på figur 13.2 side 70: forespørgslen modtages, der genereres en svar, der efterfølgende sendes.

Hvis forespørgslen er af en type, der kan oprette en dialog (ifølge RFC 3261 kan kun `INVITE` oprette en dialog), vil flowet være som vist nederst på figur 13.2: scenariet indledes med et handshake, efterfølgende kan der nu modtages *flere* forespørgsler, og endelig nedlægges dialogen. Frameworket tilbyder at håndtere handshaket, såvel som at nedlæggelse dialogen.

Såfremt krogen `handle_invite` ikke implementeres vil standardimplementeringen blive. Ønsker applikations-programmøren stadig at have kontrol over, hvad der overgives af sessions-information i det indledende handshake, kan dette håndteres ved at implementere krogen `payload`. Såfremt applikationsprogrammøren ikke har implementeret krogen `handle_bye`, vil en standard-implementering blive benyttet. Ønsker applikationen selv at afslutte dialogen, gøres dette ved at kalde den statiske metode `terminate` på klassen `UASScenario` med `CallState` instansen som

argument. Når dialogen nedlægges, vil krogen `terminated` blive kaldt, uafhængig af om applikations-programmøren har valgt at implementere `handle-bye`. Denne krog kan f.eks. bruges til at frigive ressourcer.

Uanset om forespørgslen kan oprette en dialog eller ej, implementeres behandlingen af den enkelte forespørgsel af krogene handle-"metodenavn". Håndteringen minder dermed om måden forespørgsler håndteres af SIP Servlets, dog med den forskel at krogen ikke som SIP Servlets tager et svar som argument, men nøjes med en `SipServletRequest` samt en `CallState` instans.

Kroge

accept - afgør om beskeden skal modtages, tager en forespørgsel som argument og returnerer afgørelsen som en boolesk værdi.

authenticate - implementeres hvis brugeren skal logge ind, modtage logininformation og returnere en boolesk værdi, der angiver om informationen er godtaget.

payload - implementeres hvis det indledende handshake skal videregive information til klienten. Returnerer den ønskede krop af `200 OK` svaret.

established - kaldes når handshake er færdigt (ack modtaget)

terminated - kaldes når dialoge nedlægges

handle-invite - kaldes når frameworket modtager en `INVITE` forespørgsel

handle-option - kaldes når frameworket modtager en `OPTION` forespørgsel

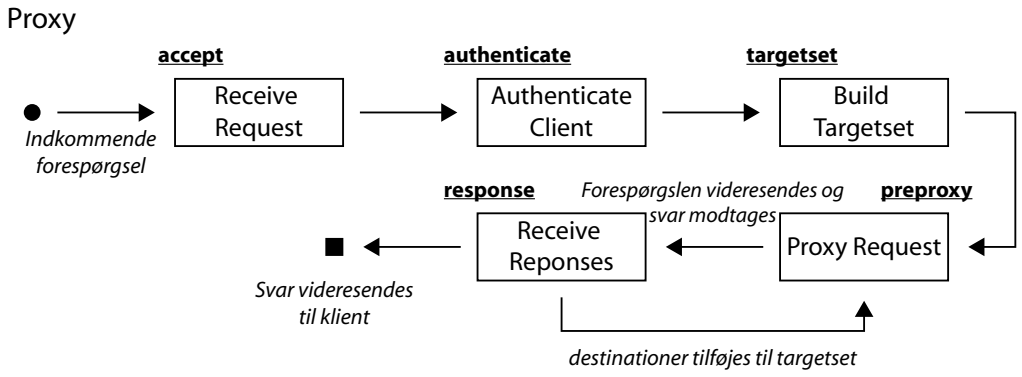
handle-register - kaldes når frameworket modtager en `REGISTER` forespørgsel

handle-ack - kaldes når frameworket modtager en `ACK` forespørgsel

handle-cancel - kaldes når frameworket modtager en `CANCEL` forespørgsel

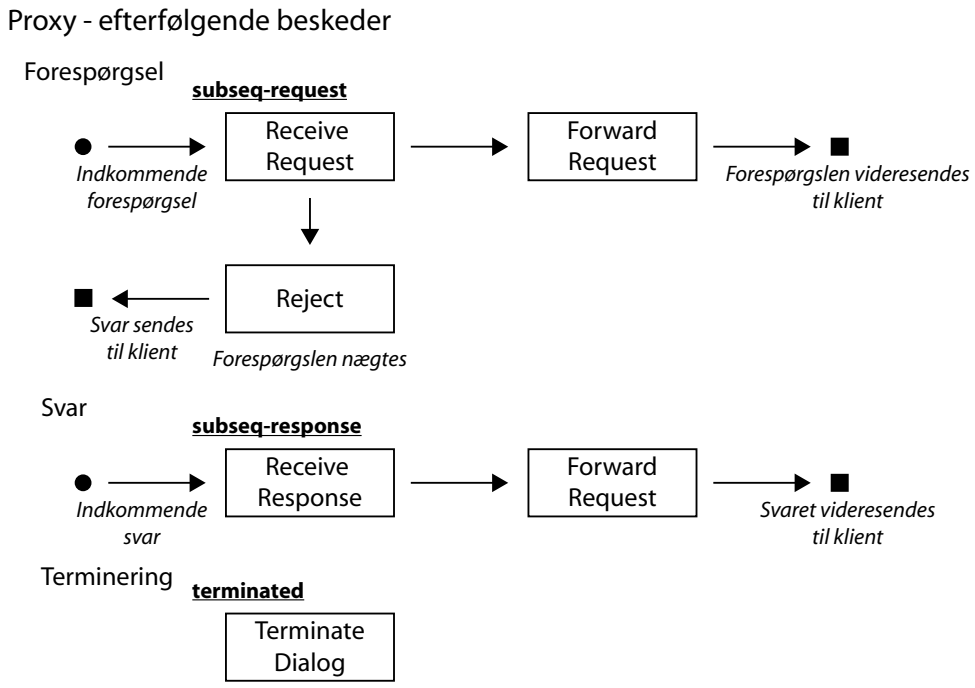
handle-bye - kaldes når frameworket modtager en `BYE` forespørgsel

13.2.2 Proxy



Figur 13.3: Flowet i Proxy scenaret

Proxy-scenariet benyttes til at implementere en applikation, der skal kunne placere sig mellem to parter i en kommunikation. En proxy modtager en forespørgsel,



Figur 13.4: Flowet for efterfølgende beskeder i Proxy scenariet

sender den videre, modtager svaret på forespørgslen, og sender det tilbage til den oprindelige klient. Ved modtagelse af forespørgsler, der ikke er en del af en dialog (INVITE inklusiv), kan proxien vælge at videresende forespørgslen til en eller flere alternative destinationer. Disse destinationer samles i et "targetset", og forespørgslen sendes nu sekventielt eller parallelt til samtlige destinationer i sættet. Når proxyen har modtaget et svar, den kan stille sig tilfreds med, videresendes svaret til den oprindelige klient. Flowet for dette scenarium er afbilledet på figur 13.3, side 71, hvorpå man kan se, at det også er muligt at tilføje flere destinationer til target-sættet. Der er dog ikke alle typer svar, det på denne måde er tilladt at ignorere. En nærmere beskrivelse kan finde i SIP Servlet specifikationen.

Hvis forespørgslen proxien videresender bevirker, at der bliver oprettet en dialog, vil proxien også være mellemled for de efterfølgende beskeder i dialogen. Da endepunkterne i dialogen er faste fra det øjeblik dialogen er etableret, er det dog ikke tilladt proxien at ændre på destinationen for disse beskeder. Proxien har dog mulighed for at nægte forespørgsler ved selv at besvare dem i stedet for at sende dem videre. Figur 13.4 på side 72 viser flow-diagrammet for efterfølgende beskeder.

Der findes forskellige typer proxier. En proxy kan vælge mellem at være state-full eller state-less og mellem at benytte sig af "Record routing" eller lade være. Dette har indflydelse på, om proxyen vil modtage efterfølgende beskeder, og om den lagre tilstands-information om dialogen. For en nærmere beskrivelse henvises der til RFC 3261, og JSR 116. Frameworkets proxy-scenarium bruges til at implementere en statefull, record-routing proxy.

Kroge

Følgende kroge er defineret for beskeder uden for dialog:

accept - afgør om beskeden skal modtages, tager en forespørgsel som argument og returnerer afgørelsen som en boolesk værdi.

authenticate - implementeres hvis brugeren skal logge ind, modtage logininformation og returnere en boolesk værdi, der angiver om informationen er godtaget.

targetset - opbygger et target-sæt i form af en liste af URI'er

preproxy - kaldes umiddelbart før forespørgslen videresendes til destinationerne i target-sættet

response - kaldes når et svar modtages. Hvis applikationen ikke er tilfreds med svaret kan den tilføje mere til target-sættet

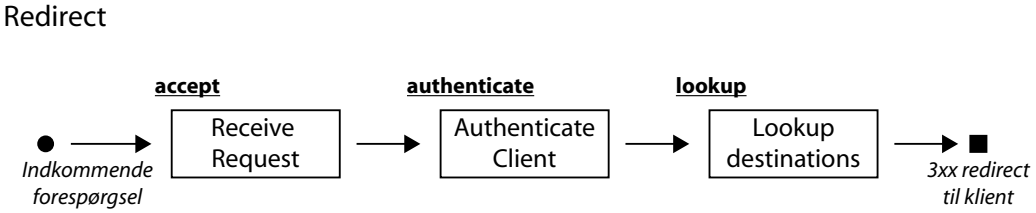
Følgende kroge er defineret for efterfølgende beskeder i en dialog:

subseq-request - angiver om en efterfølgende forespørgsel skal accepteres. Såfremt den ikke accepteres, sender krogen selv et svar.

subseq-response - giver applikationen mulighed for at undersøge svaret på forspørgslen. Det er ikke muligt at manipulere svaret.

terminated - kaldes når dialogen nedlægges

13.2.3 Redirect



Figur 13.5: Flowet i Redirect scenariet

Redirect-scenariet bruges til at implementere applikationer, der videresender forespørgsler uden selv at placere sig mellem de to parter. Forespørgslen videresendes ved at sende et 3xx svar tilbage til klienten indeholdende de nye destinationer, hvorefter klienten kontakter disse uden at involvere redirect-serveren yderligere. Destinationerne slås op i et lokations-tjeneste, der ud fra en URI kan returnere én eller flere alternative destinationer i form af andre URI'er. Figur 13.5 på side 73 viser flowdiagramet for dette scenarium.

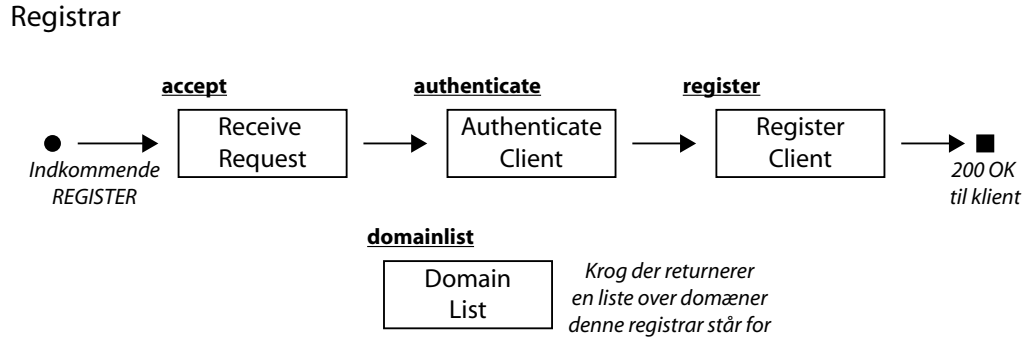
Kroge

accept - afgør om beskeden skal modtages, tager en forespørgsel som argument og returnerer afgørelsen som en boolesk værdi.

authenticate - implementeres hvis brugeren skal logge ind, modtage logininformation og returnere en boolesk værdi der angiver om informationen er godtaget.

lookup søger efter lokationsinformation, og benytter sig evt. af en lokationstjeneste. Krogen returnerer en liste af URI'er, der indsættes i det efterfølgende 3xx svar.

13.2.4 Registrar (registreringsserver)



Figur 13.6: Flowet i Registrar

En registreringsserver modtager `REGISTER` forespørgsler, der indeholder information om på hvilken URI en bruger med en given alias-adresse ønsker at blive kontaktet. Registrerings-serveren lagrer brugerens alias-adresse samt den ønskede adresse i et lokations-tjeneste. Denne tjeneste kan efterfølgende bruges af proxy- og redirect-servere til at videresende forespørgsler til alias-adresse til den korrekte adresse. Flowet i dette scenarium kan ses på figur 13.6 side 74.

En registrerings-server accepterer forespørgsler rettet imod et eller flere domæner. Disse domæner konfigureres i scenariet vha. en krog.

Kroge

domainlist returnerer en liste af domæner denne registrar håndterer

accept - afgør om beskeden skal modtages, tager en forespørgsel som argument og returnerer afgørelsen som en boolesk værdi. Denne krog kan evt. konsultere domæne-listen.

authenticate - implementeres hvis brugeren skal logge ind, modtager logininformation, og returnere en boolesk værdi, der angiver om informationen er godtaget.

register lagrer lokations-information.

13.3 Lokations tjeneste

Frameworket tilbyder en standardiseret grænseflade til implementering af en lokationstjeneste. Derudover indeholder frameworket en simpel implementering af en sådan lokationstjeneste, der benytter en DataSource, specificeret af applikationsprogrammøren, til at lagre sin lokations-information. Strukturen i klassen er afbilledet på figur 13.1. Ved at benytte den standardiserede grænseflade kan applika-

tionsprogrammøren bruge sin egen implementering i kombination med standardimplementeringerne af de forskellige kroge. F.eks. benytter standardimplementeringen hooken "lookup" i scenariet "Redirect" en lokations-tjeneste til at afgøre hvilke URI'er, der efterfølgende skal omstilles til. Såfremt applikations-programmøren udelukkende ønsker, at destinations-URI'erne skal findes ved at lave et simpelt opslag i lokations-tjenesten, kan programmøren ved at implementere standardgrænsefladen til sin lokations-tjeneste nøjes med standard implementeringen af krogen. Dette er demonstreret i eksemplet vist i afsnit 13.7.

For at benytte en standardiseret lokations-tjeneste i et scenarium skal en instans defineres i konfigurationsfilen. Se afsnit 13.6 for en beskrivelse af syntaksen.

13.4 CallState

Klassen `CallState` og specialiseringer af denne benyttes af de forskellige konfigurationer af standard-scenarier til lagring af tilstands-information og konfiguration. Klassen instansieres af frameworket umiddelbart inden den første krog i et scenarium bliver kaldt, og bliver bibeholdt indtil scenariet afsluttes. Såfremt scenariet involverer en dialog, slettes instansen først, når dialogen nedlægges.

`CallState` er implementeret som en JavaBean™, og specialiseringer af denne skal følge JavaBean™retnings-linjerne.

Ved at benytte en specialisering af `CallState` til at opbevare referencer og anden data undervejs i en dialog, opnår en applikations-programmør ikke alene typesikkerhed, men også at frameworket kan bidrage til tilvejebringelsen af de forskellige nødvendige data. Frameworket kan således konfigureres til at befolke `CallState`-specialiseringen med referencer til forskellige ressourcer som f.eks. en byggeblok, umiddelbart inden den første krog kaldes. Et eksempel på en opsætning, der benytter sig af denne funktionalitet kan ses i afsnit 13.7.

13.5 Byggeblokke

En byggeblok i frameworket er en vilkårlig klasse, der indkapsler logik, der skal bruges af en krog i et standardscenarium. Klassen implementeres som en JavaBean™, og kan modtage konfiguration fra frameworket via set-metoder. Instanser til de enkelte blokke gemmes i en `CallState`-instans, der indgår som metodeargument til den enkelte krog. Det er op til krogen om byggeblokken selv har brug for at få adgang til CallState-instansen ved eksekvering af denne, men da blokken kan tilgås flertrådet skal en sådan instans i så fald gives som argument til en forretningsmetode i blokken.

13.6 Konfiguration

Frameworket konfigureres ved hjælp af en XML-konfigurationsfil. Konfigurationen inddeles følgende sektioner der tilsammen konfigurere de forskellige byggeblokke, scenarier, lokationstjenester og sammenkoblingen af disse:

building-blocks beskrivelse og konfiguration af byggeblokke

location-service beskrivelse og konfiguration af lokationstjenester

scenarios beskrivelse af de enkelte scenarier, samt beskrivelse af hvilke byggeblokke og andre ressourcer scenariet benytter sig af

scenario-mappings beskriveles af de kriterier der afgører om et givet scenarium skal benyttes til håndtering af en indkommende forspørgsel.

Strukturen i konfigurations-filen beskrevet i udvidet Backus-Naur form, og kan findes i bilag D på side 100.

13.7 Eksempler

LeetCorp er et lille firma med fire ansatte, der ønsker at indtræde i IP-telefoni markedet. Deres plan er at få markeds-dominans ved at tilbyde en række revolutionerende telekommunkations-tjenester så som viderestilling og telefonsvarere.

De to programmører firmaet råder over har en del erfaring med HTTP Servlets i forbindelse med firmaets succesfulde poker-dating site. LeetCorp har derfor valgt at implementere deres næste pengemaskine ved hjælp af SIP Servlets. De to programmører må dog erkende at SIP ikke lige er deres stærkeste side, og de vælger derfor at bruge et særdeles succesfuld framework de har fundet på nettet.

LeetCorp vælger at begynde deres IP-telefoni eventyr med de to tjenester "SIP Alias'o'Magic" og "Hyper VoiceMail Deluxe", der er henholdsvis en viderestillings tjeneste, og en telefonsvarer. Da LeetCorp gerne skal kunne høste frugten af deres hårde arbejde, implementerer de også applikationen "Mega Logger II" der kan registrere deres kunders brug af de to tjenester.

De følgende afsnit vil beskrive hvordan frameworket kan benyttes til at implementere de tre tjenester.

13.7.1 SIP Alias'o'Magic

Denne tjeneste tilbyder at lade en bruger oprette en alias-adresse, der viderestiller til brugerens aktuelle adresse. Tjenesten implementeres med de to scenarier Registrar og Redirect.

Til lagring af adresser benyttes standard-implementeringen af lokationstjenesten. De to scenarier konfigureres til at benytte den samme lokationstjeneste, og standard-implementeringen af kroge benyttes så vidt muligt til at håndtere funktionaliteten i tjenesten. Integration med kundedatabasen der indeholder alle kunderes brugernavn og password implementeres i en byggeblok.

Registrar-scenariet benyttes når brugeren skal registrere sine adresser. Indledningsvis skal brugeren logge ind med et brugernavn og password, dette håndteres ved at implementere krogen `authenticate`, krogen validerer brugeren ved hjælp af den ovennævnte byggeblok.

Redirect-scenariet benyttes når en klient forsøger at kontakte en kunde på dennes alias-adresse. Der er ikke implementeret nogen kroge i dette scenarium da standardimplementeringen af `lookup` er tilstrækkelig.

Konfigurationsfilen der er benyttet til implementering af denne tjeneste kan findes i bilag F.1 på side 112.

13.7.2 Hyper VoiceMail Deluxe

Denne revolutionerende tjeneste lader en bruger aflytte talebeskeder andre brugere har indtalt. Tjenesten implementeres med scenariet UAS. Applikationen håndtere kun SIP signaleringen, optagelse og afspilning håndteres af en medie-server indkøbt til formålet.

Krogen `authenticate` er implementeret på samme måde som i SIP Alias'o'Magic applikationen. Scenariet er ligeledes konfigureret med byggeblokken der implementere adgangen til kundedatabasen. Da brugeren skal føre en samtale med medie-serveren, oprettes der en dialog v.ha. en `INVITE` forespørgsel. Instruktioner om at kontakte medie-serveren videregives til klienten ved at implementere krogen `payload`. Krogen `established` implementeres med logik der notificere medieserveren om at en session brugeren skal påbegyndes. Krogen `implementeres` med logik der notificerer medie-serveren om at brugeren har lagt på.

Scenariet lagrer medieserverens adresse, samt id'et på brugerens medie-session i et specialiseret `CallState`. Denne information bruges når sessionen imellem brugeren og medie-serveren skal oprettes og nedlægges.

Konfigurationsfilen der er benyttet til implementering af denne tjeneste kan findes i bilag F.2 på side 113.

13.7.3 Mega Logger II

Denne tjeneste logger information om forskellige kunders forbrug af de to beskrevne tjenester. Tjenesten implementeres med scenariet Proxy.

Applikationen modtager alle forspørgler rettet imod "SIP Alias'o'Magic" og "Hyper VoiceMail Deluxe". Disse videresendes uden at blive modificeret, men ved nedlæggelsen af en dialog imellem en tjeneste og en kunde logger "Mega Logger II" hvor langt tid tjenesten har været benyttet, og af hvilken kunde.

"Mega Logger II" benytter en specialisering af CallState til at lagre brugernavn og start-tidspunkt, samt en byggeblok til logging. Krogen `preproxy` udtrækker informationen og lagre den i objektet, mens en implementering af krogen terminated benytter byggeblokken til at logge forbrugs-data.

Konfigurationsfilen der er benyttet til implementering af denne tjeneste kan findes i bilag F.3 på side 114.

13.8 Afsluttende bemærkninger

Valget af XML som konfigurations-format er ikke revolutionerende. Der er mange holdninger til XMLs egnethed som konfigurations-format til særligt typesikre sprog, da man med et generelt tekstformat mister noget af den sikkerhed, et typesikkert sprog ellers vil give. Samtidig kan store XML-dokumenter være uoverskuelige, og

det er ikke muligt at bruge konstruktioner, som man ellers bruger i programmeringssprog så som funktioner og inkludering af filer. Dette kan dog implementeres ved udvidelse af konfigurations-formatet, men man vil i så fald genopfinde funktionaliteter, man på forhånd har adgang til i sit f.eks. Java.

Designet har som før beskrevet hovedsageligt været forkuseret på at designe et frameworket til implementering af SIP *server* applikationer. En oplagt udvidelse af frameworket vil derfor være en række scenarier der understøtter implementeringen af SIP klienter. Dette vil f.eks. kunne bruges i en automatiseret telefonvæknings-tjeneste hvor applikationen har brug for at kunne foretage et opkald.

Del IV

Konklusion

Afsnit 14

Opsummerende Diskussion

SIP er til tider en meget kompleks men samtidig også meget kraftfuld og fleksibel protokol. Kompleksiteten betyder blandt andet, at de forskellige scenarier ved første øjekast kan virke mere komplicerede end de mere simple metoder i SIP Servlet API'et. Man skal dog huske, at frameworket ved at tilbyde standardimplementering af kroge og nogle mere faste rammer at implementere applikationer indenfor, reducerer mængden af beslutninger, en programmør skal tage, før implementering af den egentlige forretningslogik kan begynde. Forudsætningen for det fordelagtige i dette er, at standardscenarierne passer til den type applikation, man forsøger at implementere.

Da det grundlæggende frameworkdesign er lavet på baggrund af dels en analyse af eksisterende frameworks, og dels af standardiserede begreber fra SIP specifikationen, forventes det, at videreudvikling af frameworket nærmere vil betyde udvidelse af de eksisterende elementer end et egentligt re-design.

Containeren er i skrivende stund ikke fuldt implementeret. Første iteration har, som før beskrevet, fokuseret på design og implementering af de centrale dele. Næste iteration bør adressere nogle af de problemer, der er blevet afdækket under afprøvningen. Gennemførelse af første iteration har dog vist, at det forekommer at være en overkommelig opgave at lave implementering af JSR-116. Særligt brugen af den frit tilgængelige JSR-32 implementering har muliggjort udviklingen.

Afsnit 15

Konklusion

Specialet har vist, at det vil være muligt for ressourcesvage aktører at lave en implementering af JSR-116. Implementeringen, der er udviklet i forbindelse med projektet, har vist potentiale, men er endnu ikke færdigudviklet.

Frameworket er blevet designet på baggrund af en række overvejelser, der tilsammen bør gøre det særdeles anvendeligt for SIP Servlet programmører.

Frameworket er designet til at fungere med en generel JSR-116 implementering og er derfor ikke afhængig af den implementering, der er lavet i forbindelse med dette projekt.

Afsnit 16

Perspektivering

En af de største drivende faktorer for IP-telefoni er de lavere priser. Man kan forvente, at økonomien, vil være det væsentligste konkurrencemonent i nære fremtid. Men når priserne har stabiliseret sig er det tænkeligt, at konkurrencen vil være på et andet område, nemlig udbuddet af tjenester.

Selvom markedet for teletjenester i fremtiden vil være mere åbent i kraft af brugen af åbne SIP-netværk, vil det for især mindre aktører være lige så vigtigt, at de grundlæggende værktøjer til implementering af disse tjenester er frit tilgængelige. Det er derfor vigtigt for konkurrencen på fremtidens telemarked, at der bliver gjort tiltag til at udvikle implementeringer af grundlæggende teknologier som SIP Servlet containere og JAIN SLEE applikations-servere. Dette projekt er et eksempel på et sådan tiltag, men er i den henseende ikke unikt:

- Projektet "Jiplet" har udarbejdet et alternativ til SIP Servlets. En foreløbig implementering af en Jiplet container kan hentes på `http://www.cafesip.org/projects/jiplet/`
- MobiCents er en fuldt fungerende OpenSource implementering af en JAIN SLEE applikations-server. MobiCents er implementeret som en udvidelse af OpenSource J2EE applikations-serveren JBOSS (`http://jboss.org`), og understøtter konvergerede JAIN SLEE applikationer. MobiCents kan hentes på `http://mobicents.dev.java.net/`.

Det vil ligeledes være vigtigt, at der konstant fokuseres på forbedring af de forskellige teknologier, og hvordan forskellige programmeringsmodeller, der viser sig mere anvendelige til løsning af problemstillinger, understøttes. Dette kan gøres ved at deltage i udvikling af de forskellige standarder, men kan som regel lettere gøres ved at udarbejde frameworks i ligehed med det, der er designet i dette projekt.

Den øgede konkurrence og tilblivelsen af en række nye aktører vil utvivlsomt accelerere udviklingen af nye innovative tjenester. I første omgang vil fokus dog nok være på reducering af omkostninger. Således vil mange af de første tiltag, man vil se, nok nærmere være effektivisering og automatisering af eksisterende tjenester og funktioner så som konference-systemer og omstillingsborde.

Del V

Bilag

Følgende bilag er inkluderet i rapporten:

Bilag A

Kilder og figurer

A.1 Refererede

[Dan04] Mads Danquah. Telefoni over internettet - en introduktion til tekniske og juridiske problemstillinger. http://mads.danquah.dk/projekter/9sem/voip_intro.pdf, 2004.

[ea99] J. Rosenberg et al. Sip: Session initiation protocol. http://web.archive.org/web/20021201185138/http://www.ietf.org/rfc/rfc2543.txt, 1999.

[ea02] J. Rosenberg et al. Sip: Session initiation protocol. http://web.archive.org/web/20041127014944/http://www.ietf.org/rfc/rfc3261.txt, 2002.

[ea03a] Anders Kristensen et al. Jsr 116: Sip servlet api. http://web.archive.org/web/20040604010039/http://jcp.org/en/jsr/detail?id=116, 2003.

[ea03b] Swee Lim et al. Jain slee tutorial. http://web.archive.org/web/20040728132051/http://java.sun.com/products/jain/JAIN-SLEE-Tutorial.pdf, 2003.

[Fow04] Martin Fowler. Inversion of control containers and the dependency injection pattern. http://web.archive.org/web/20041125093913/http://www.martinfowler.com/articles/injection.html, 2004.

[Lim03] Open Clound Limited. A slee for all seasons. http://www.opencloud.com/slee/downloads/asfas.pdf, 2003.

[Loz03] Zygmunt Lozinski. Parlay/osa - a new way to create wireless services. http://web.archive.org/web/20041108103926/http://parlay.org/docs/2003_06_01_Parlay_for_IEC_Wireless.pdf, 2003.

[Loz05] Zygmunt Lozinski. Parlay/osa and the intelligent network. http://www.parlay.org/imwp/idms/popups/pop_download.asp?ContentID=5906, 2005.

[Man04] Kito Mann. Javaserverfaces in action, 2004.

[Mic] Microsoft. Microsoft netmeeting homepage. http://www.microsoft.com/windows/netmeeting/.

[Mir04] Sun Mircosystems. Jslee and the jain initiative. http://web.archive.org/web/20041125033114/http://java.sun.com/products/jain/, 2004.

[MOPS02] Michele Motsko, Patricia Oberndorf, Ellen-Jane Pairo, and James Smith. Rules of thumb for the use of cots products. CMU/SEI-2002-TR-032, 2002.

[O'D03] Phelim O'Doherty. White paper - sip and the java platforms. http://web.archive.org/web/20041102045914/http://java.sun.com/products/jain/SIP-and-Java.html, 2003.

[Ole03] Robert Olesen. System architecture and component reuse, 2003.

[PC00] Paul J. Perrone and Vankata Changanti. Building java enterprise systems with j2ee, 2000.

[Sky] Skype. The skype homepage. http://skype.com/.

A.2 Supplerende

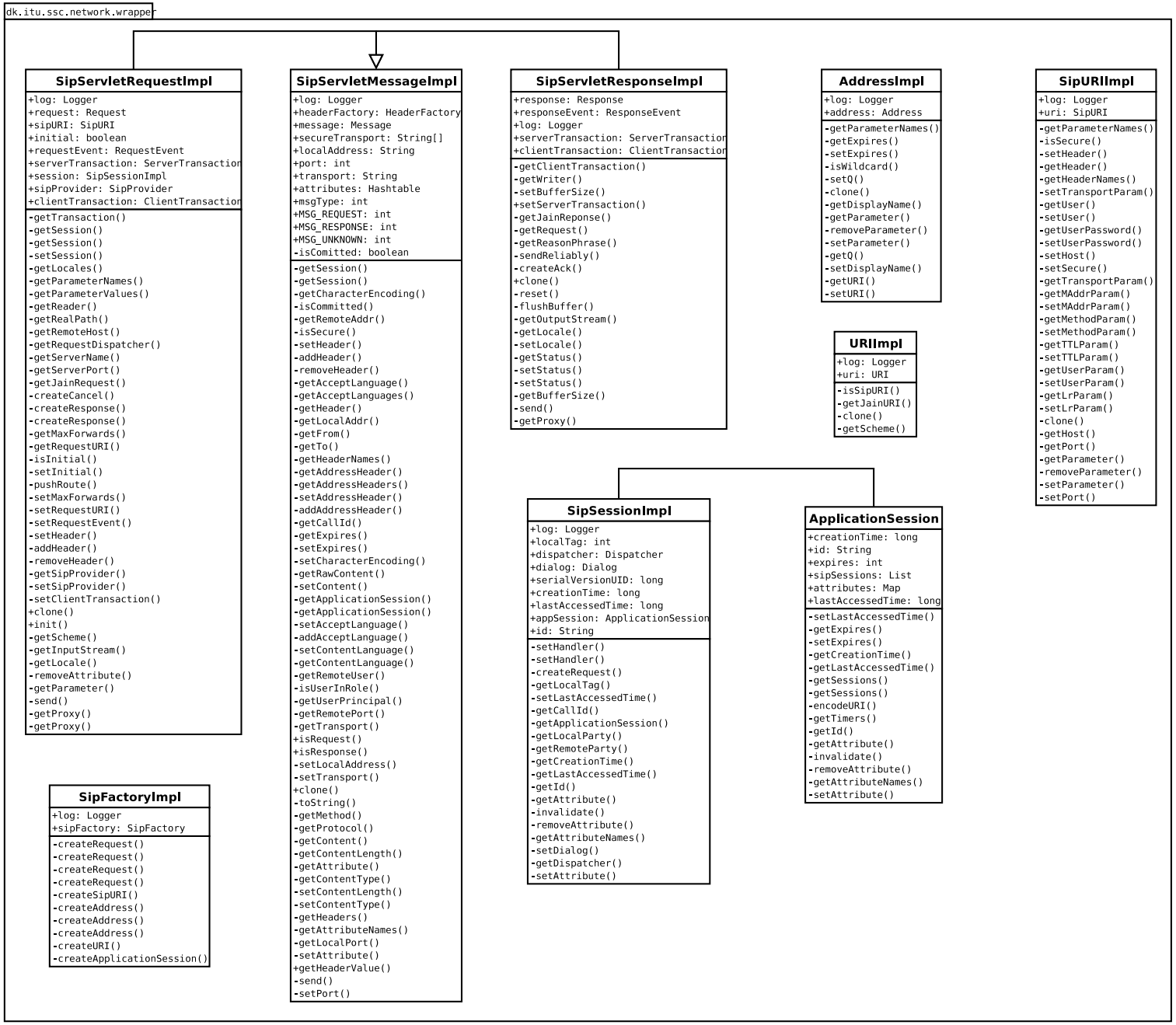
[Gro02] The Parlay Group. Parlay x web services white paper. http://web.archive.org/web/20031116225033/http://www.parlay.org/specs/library/ParlayX-WhitePaper-1.0.pdf, 2002.

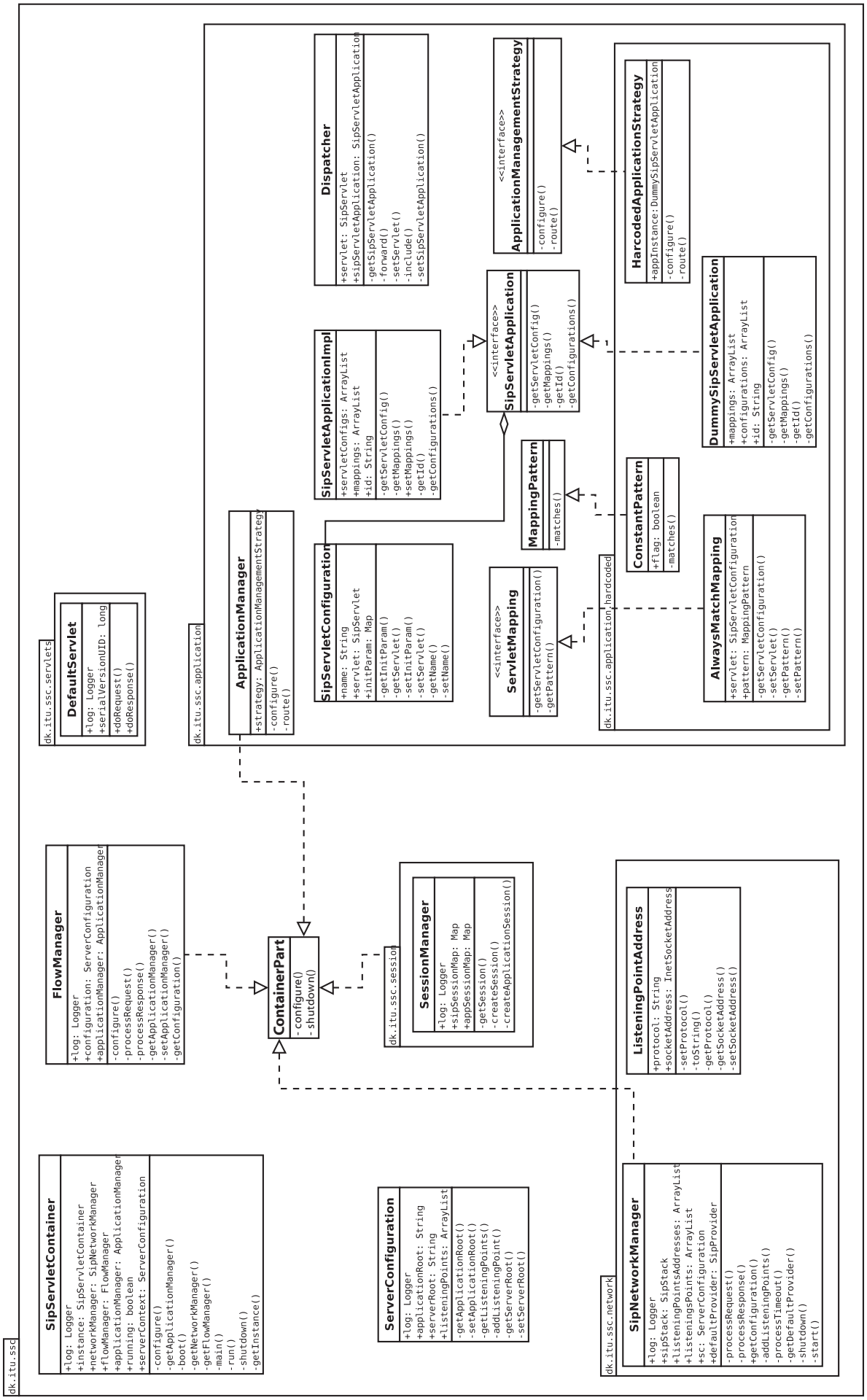
[SJ01] Henry Sinnreich and Ala B. Johnston. Internet communication using sip, 2001.

Figurer

Bilag B

Klassediagrammer


dk.itu.ssc.network.wrapper
SipServletRequestImpl
+log: Logger
+request: Request
+sipURI: SipURI
+initial: boolean
+requestEvent: RequestEvent
+serverTransaction: ServerTransaction
+session: SipSessionImpl
+sipProvider: SipProvider
+clientTransaction: ClientTransaction
-getTransaction()
-getSession()
-getSession()
-setSession()
-getLocales()
-getParameterNames()
-getParameterValues()
-getReader()
-getRealPath()
-getRemoteHost()
-getRequestDispatcher()
-getServerName()
-getServerPort()
-getJainRequest()
-createCancel()
-createResponse()
-createResponse()
-getMaxForwards()
-getRequestURI()
-isInitial()
-setInitial()
-pushRoute()
-setMaxForwards()
-setRequestURI()
-setRequestEvent()
-setHeader()
-addHeader()
-removeHeader()
-getSipProvider()
-setSipProvider()
-setClientTransaction()
+clone()
+init()
-getScheme()
-getInputStream()
-getLocale()
-removeAttribute()
-getParameter()
-send()
-getProxy()
-getProxy()
SipServletMessageImpl
+log: Logger
+headerFactory: HeaderFactory
+message: Message
+secureTransport: String[]
+localAddress: String
+port: int
+transport: String
+attributes: Hashtable
+msgType: int
+MSG_REQUEST: int
+MSG_RESPONSE: int
+MSG_UNKNOWN: int
-isComitted: boolean
-getSession()
-getSession()
-getCharacterEncoding()
-isCommitted()
-getRemoteAddr()
-isSecure()
-setHeader()
-addHeader()
-removeHeader()
-getAcceptLanguage()
-getAcceptLanguages()
-getHeader()
-getLocalAddr()
-getFrom()
-getTo()
-getHeaderNames()
-getAddressHeader()
-getAddressHeaders()
-setAddressHeader()
-addAddressHeader()
-getCallId()
-getExpires()
-setExpires()
-setCharacterEncoding()
-getRawContent()
-setContent()
-getApplicationSession()
-getApplicationSession()
-setAcceptLanguage()
-addAcceptLanguage()
-setContentLanguage()
-getContentLanguage()
-getRemoteUser()
-isUserInRole()
-getUserPrincipal()
-getRemotePort()
-getTransport()
+isRequest()
+isResponse()
-setLocalAddress()
-setTransport()
+clone()
-toString()
-getMethod()
-getProtocol()
-getContent()
-getContentLength()
-getAttribute()
-getContentType()
-setContentLength()
-setContentType()
-getHeaders()
-getAttributeNames()
-getLocalPort()
-setAttribute()
+getHeaderValue()
-send()
-setPort()
SipServletResponseImpl
+response: Response
+responseEvent: ResponseEvent
+log: Logger
+serverTransaction: ServerTransaction
+clientTransaction: ClientTransaction
-getClientTransaction()
-getWriter()
-setBufferSize()
+setServerTransaction()
-getJainReponse()
-getRequest()
-getReasonPhrase()
-sendReliably()
-createAck()
+clone()
-reset()
-flushBuffer()
-getOutputStream()
-getLocale()
-setLocale()
-getStatus()
-setStatus()
-setStatus()
-getBufferSize()
-send()
-getProxy()
AddressImpl
+log: Logger
+address: Address
-getParameterNames()
-getExpires()
-setExpires()
-isWildcard()
-setQ()
-clone()
-getDisplayName()
-getParameter()
-removeParameter()
-setParameter()
-getQ()
-setDisplayName()
-getURI()
-setURI()
SipURIImpl
+log: Logger
+uri: SipURI
-getParameterNames()
-isSecure()
-setHeader()
-getHeader()
-getHeaderNames()
-setTransportParam()
-getUser()
-setUser()
-getUserPassword()
-setUserPassword()
-setHost()
-setSecure()
-getTransportParam()
-getMAddrParam()
-setMAddrParam()
-getMethodParam()
-setMethodParam()
-getTTLParam()
-setTTLParam()
-getUserParam()
-setUserParam()
-getLrParam()
-setLrParam()
-clone()
-getHost()
-getPort()
-getParameter()
-removeParameter()
-setParameter()
-setPort()
URIImpl
+log: Logger
+uri: URI
-isSipURI()
-getJainURI()
-clone()
-getScheme()
SipSessionImpl
+log: Logger
+localTag: int
+dispatcher: Dispatcher
+dialog: Dialog
+serialVersionUID: long
+creationTime: long
+lastAccessedTime: long
+appSession: ApplicationSession
+id: String
-setHandler()
-setHandler()
-createRequest()
-getLocalTag()
-setLastAccessedTime()
-getCallId()
-getApplicationSession()
-getLocalParty()
-getRemoteParty()
-getCreationTime()
-getLastAccessedTime()
-getId()
-getAttribute()
-invalidate()
-removeAttribute()
-getAttributeNames()
-setDialog()
-getDispatcher()
-setAttribute()
ApplicationSession
+creationTime: long
+id: String
+expires: int
+sipSessions: List
+attributes: Map
+lastAccessedTime: long
-setLastAccessedTime()
-getExpires()
-setExpires()
-getCreationTime()
-getLastAccessedTime()
-getSessions()
-getSessions()
-encodeURI()
-getTimers()
-getId()
-getAttribute()
-invalidate()
-removeAttribute()
-getAttributeNames()
-setAttribute()
SipFactoryImpl
+log: Logger
+sipFactory: SipFactory
-createRequest()
-createRequest()
-createRequest()
-createRequest()
-createSipURI()
-createAddress()
-createAddress()
-createAddress()
-createURI()
-createApplicationSession()


dk.itu.ssc
SipServletContainer
+log: Logger
+instance: SipServletContainer
+networkManager: SipNetworkManager
+flowManager: FlowManager
+applicationManager: ApplicationManager
+running: boolean
+serverContext: ServerConfiguration
-configure()
-getApplicationManager()
-boot()
-getNetworkManager()
-getFlowManager()
-main()
-run()
-shutdown()
-getInstance()
FlowManager
+log: Logger
+configuration: ServerConfiguration
+applicationManager: ApplicationManager
-configure()
-processRequest()
-processResponse()
-getApplicationManager()
-setApplicationManager()
-getConfiguration()
ContainerPart
-configure()
-shutdown()
ServerConfiguration
+log: Logger
+applicationRoot: String
+serverRoot: String
+listeningPoints: ArrayList
-getApplicationRoot()
-setApplicationRoot()
-getListeningPoints()
-addListeningPoint()
-getServerRoot()
-setServerRoot()
dk.itu.ssc.session
SessionManager
+log: Logger
+sipSessionMap: Map
+appSessionMap: Map
-getSession()
-createSession()
-createApplicationSession()
dk.itu.ssc.network
SipNetworkManager
+log: Logger
+sipStack: SipStack
+listeningPointsAddresses: ArrayList
+listeningsPoints: ArrayList
+sc: ServerConfiguration
+defaultProvider: SipProvider
-processRequest()
-processResponse()
+getConfiguration()
-addListeningPoints()
-processTimeout()
-getDefaultProvider()
-shutdown()
-start()
ListeningPointAddress
+protocol: String
+socketAddress: InetSocketAddress
-setProtocol()
-toString()
-getProtocol()
-getSocketAddress()
-setSocketAddress()
dk.itu.ssc.servlets
DefaultServlet
+log: Logger
+serialVersionUID: long
+doRequest()
+doResponse()
dk.itu.ssc.application
ApplicationManager
+strategy: ApplicationManagementStrategy
-configure()
-route()
SipServletConfiguration
+name: String
+servlet: SipServlet
+initParam: Map
-getInitParam()
-getServlet()
-setInitParam()
-setServlet()
-getName()
-setName()
SipServletApplicationImpl
+servletConfigs: ArrayList
+mappings: ArrayList
+id: String
-getServletConfig()
-getMappings()
+setMappings()
-getId()
-getConfigurations()
Dispatcher
+servlet: SipServlet
+sipServletApplication: SipServletApplication
-getSipServletApplication()
-forward()
-setServlet()
-include()
-setSipServletApplication()
<<interface>>
SipServletApplication
-getServletConfig()
-getMappings()
-getId()
-getConfigurations()
<<interface>>
ApplicationManagementStrategy
-configure()
-route()
<<interface>>
ServletMapping
-getServletConfiguration()
-getPattern()
MappingPattern
-matches()
dk.itu.ssc.application.hardcoded
ConstantPattern
+flag: boolean
-matches()
HarcodedApplicationStrategy
+appInstance: DummySipServletApplication
-configure()
-route()
DummySipServletApplication
+mappings: ArrayList
+configurations: ArrayList
+id: String
-getServletConfig()
-getMappings()
-getId()
-getConfigurations()
AlwaysMatchMapping
+servlet: SipServletConfiguration
+pattern: MappingPattern
-getServletConfiguration()
-setServlet()
-getPattern()
-setPattern()

Bilag C

Afprøvning

C.1 Testcase 1

Containeren blev konfigureret med følgende Deployment Descriptor:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE sip-app
  PUBLIC "-//Java Community Process//DTD SIP Application 1.0//EN"
  "http://www.jcp.org/dtd/sip-app_1_0.dtd">
<sip-app>
   <display-name>DisplayName</display-name>
   <servlet>
      <servlet-name>TestServlet</servlet-name>
      <servlet-class>dk.itu.ssc.tests.TestServlet</servlet-class>
      <init-param>
         <param-name>key</param-name>
         <param-value>value</param-value>
      </init-param>
   </servlet>
</sip-app>
```

Containeren gav under afprøvningen følgende output

```
DEBUG SipServletContainer: Basic configuration: {applicationmanager.
    hardcoded.dd=D:\workspace\eclipse\ssc_code\src\dk\itu\ssc\tests\sip.
    xml}
INFO SipServletContainer: Booting container
DEBUG SipServletContainer: Configuring container
DEBUG ServerContext: Adding listeningpoint /127.0.0.1:5060 / udp
DEBUG ServerContext: Adding listeningpoint /127.0.0.1:5060 / tcp
DEBUG SipXmlEntityResolver: resolved entity "-//Java Community Process//
    DTD SIP Application 1.0//EN" to "/javax/servlet/sip/resources/sip-
    app_1_0.dtd"
DEBUG DDReader: Added servlet parameter key->value
INFO SessionManager: Creating new ApplicationSesssion for application dk
    .itu.ssc.application.SipServletApplicationImpl@1a8c4e7 with id SipApp
    -a7tjx0
INFO HarcodedApplicationStrategy: Created tempoary directory C:\DOCUME
    ~1\MADSDA~1\LOCALS~1\Temp\sipapp2500
INFO HarcodedApplicationStrategy: Test: C:\DOCUME~1\MADSDA~1\LOCALS~1\
    Temp\sipapp2500
DEBUG HarcodedApplicationStrategy: Configured class dk.itu.ssc.tests.
    TestServlet
INFO TestServlet: init
```

```
INFO ApplicationManager: ApplicationManager configured
INFO FlowManager: FlowManager configured
DEBUG SipNetworkManager: Initializing netstack with address 127.0.0.1
DEBUG SipNetworkManager: sipStack = gov.nist.javax.sip.
    SipStackImpl@471e30
DEBUG SipNetworkManager: Created listningpoint: gov.nist.javax.sip.
    ListeningPointImpl@e3b895 for address /127.0.0.1:5060/udp
DEBUG SipNetworkManager: provider: gov.nist.javax.sip.
    SipProviderImpl@50d89c
DEBUG SipNetworkManager: Adding Listeningpoint: /127.0.0.1:5060/udp
    provider gov.nist.javax.sip.SipProviderImpl@50d89c
DEBUG SipNetworkManager: Created listningpoint: gov.nist.javax.sip.
    ListeningPointImpl@1f6f0bf for address /127.0.0.1:5060/tcp
DEBUG SipNetworkManager: provider: gov.nist.javax.sip.
    SipProviderImpl@137c60d
DEBUG SipNetworkManager: Adding Listeningpoint: /127.0.0.1:5060/tcp
    provider gov.nist.javax.sip.SipProviderImpl@137c60d
INFO SipServletContainer: Container boot completed
INFO SipServletContainer: Shutting down container
INFO ApplicationManager: shutdown complete
INFO FlowManager: shutdown complete
INFO SipNetworkManager: shutdown complete
```

C.2 Testcase 2

Containeren blev konfigureret med følgende Deployment Descriptor:

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE sip-app
  PUBLIC "-//Java Community Process//DTD SIP Application 1.0//EN"
  "http://www.jcp.org/dtd/sip-app_1_0.dtd">

<sip-app>
  <display-name>DisplayName</display-name>

  <servlet>
    <servlet-name>TestServlet</servlet-name>
    <servlet-class>dk.itu.ssc.tests.Testcase2Servlet</servlet-class>
  </servlet>
</sip-app>
```

Containeren afgav under afprøvningen følgende output

```
DEBUG SipServletContainer: Basic configuration: {applicationmanager.
    hardcoded.dd=D:\workspace\eclipse\ssc_code\src\dk\itu\ssc\tests\
    testcase2dd.xml}
INFO SipServletContainer: Booting container
DEBUG SipServletContainer: Configuring container
DEBUG ServerContext: Adding listeningpoint /127.0.0.1:5060 / udp
DEBUG ServerContext: Adding listeningpoint /127.0.0.1:5060 / tcp
DEBUG SipXmlEntityResolver: resolved entity "-//Java Community Process//
    DTD SIP Application 1.0//EN" to "/javax/servlet/sip/resources/sip-
    app_1_0.dtd"
INFO SessionManager: Creating new ApplicationSesssion for application dk
    .itu.ssc.application.SipServletApplicationImpl@1a8c4e7 with id SipApp
    -3hplmt
INFO HarcodedApplicationStrategy: Created tempoary directory C:\DOCUME
    ~1\MADSDA~1\LOCALS~1\Temp\sipapp8472
```

```
INFO HarcodedApplicationStrategy: Test: C:\DOCUME~1\MADSDA~1\LOCALS~1\
    Temp\sipapp8472
DEBUG HarcodedApplicationStrategy: Configured class dk.itu.ssc.tests.
    Testcase2Servlet
INFO Testcase2Servlet: init
INFO ApplicationManager: ApplicationManager configured
INFO FlowManager: FlowManager configured
DEBUG SipNetworkManager: Initializing netstack with address 127.0.0.1
DEBUG SipNetworkManager: sipStack = gov.nist.javax.sip.
    SipStackImpl@471e30
DEBUG SipNetworkManager: Created listningpoint: gov.nist.javax.sip.
    ListeningPointImpl@e3b895 for address /127.0.0.1:5060/udp
DEBUG SipNetworkManager: provider: gov.nist.javax.sip.
    SipProviderImpl@50d89c
DEBUG SipNetworkManager: Adding Listeningpoint: /127.0.0.1:5060/udp
    provider gov.nist.javax.sip.SipProviderImpl@50d89c
DEBUG SipNetworkManager: Created listningpoint: gov.nist.javax.sip.
    ListeningPointImpl@1f6f0bf for address /127.0.0.1:5060/tcp
DEBUG SipNetworkManager: provider: gov.nist.javax.sip.
    SipProviderImpl@137c60d
DEBUG SipNetworkManager: Adding Listeningpoint: /127.0.0.1:5060/tcp
    provider gov.nist.javax.sip.SipProviderImpl@137c60d
INFO SipServletContainer: Container boot completed
DEBUG SipNetworkManager: Request with metod "INVITE" received at SSC
    with server transaction id null
DEBUG SipNetworkManager: No related dialog
DEBUG SipNetworkManager:
 request:
INVITE sip:service@127.0.0.1:5060 SIP/2.0
Via: SIP/2.0/UDP 213.173.243.219:5061;received=localhost;rport=5061
From: "sipp" <sip:sipp@213.173.243.219:5061>;tag=1
To: "sut" <sip:service@127.0.0.1:5060>
Call-ID: 1.1188.213.173.243.219@sipp.call.id
CSeq: 1 INVITE
Contact: <sip:sipp@213.173.243.219:5061>
Max-Forwards: 70
Subject: Performance Test
Content-Type: application/sdp
Content-Length: 136

v=0
o=user1 53655765 2353687637 IN IP4 127.0.0.1
s=-
c=IN IP4   127.0.0.1
t=0 0
m=audio 10000 RTP/AVP 0
a=rtpmap:0 PCMU/8000

DEBUG SipServletMessageImpl: Creating new message instance
DEBUG SipServletMessageImpl: Headers:
DEBUG SipServletMessageImpl: - Content-Length
DEBUG SipServletMessageImpl: - Via
DEBUG SipServletMessageImpl: - From
DEBUG SipServletMessageImpl: - To
DEBUG SipServletMessageImpl: - Call-ID
DEBUG SipServletMessageImpl: - CSeq
DEBUG SipServletMessageImpl: - Contact
DEBUG SipServletMessageImpl: - Max-Forwards
DEBUG SipServletMessageImpl: - Subject
DEBUG SipServletMessageImpl: - Content-Type
DEBUG SessionManager: Looked for session with associated with dialogID "
    1.1188.213.173.243.219@sipp.call.id:service@127.0.0.1:5060:sipp@213
    .173.243.219:5061:1" but found nothing
```

```
INFO SessionManager: Creating a new session with id null
INFO Testcase2Servlet: Count: 1
INFO Testcase2Servlet: INVITE recived
DEBUG Testcase2Servlet: ------------STAGE 1----------------GOT
    INVITE SENDING OK
DEBUG Testcase2Servlet: Core JAIN SIP message:
-------start--------
INVITE sip:service@127.0.0.1:5060 SIP/2.0
Via: SIP/2.0/UDP 213.173.243.219:5061;received=localhost;rport=5061
From: "sipp" <sip:sipp@213.173.243.219:5061>;tag=1
To: "sut" <sip:service@127.0.0.1:5060>
Call-ID: 1.1188.213.173.243.219@sipp.call.id
CSeq: 1 INVITE
Contact: <sip:sipp@213.173.243.219:5061>
Max-Forwards: 70
Subject: Performance Test
Content-Type: application/sdp
Content-Length: 136

v=0
o=user1 53655765 2353687637 IN IP4 127.0.0.1
s=-
c=IN IP4   127.0.0.1
t=0 0
m=audio 10000 RTP/AVP 0
a=rtpmap:0 PCMU/8000

-------end--------

DEBUG SipSessionImpl: Generating new local tag: bdg1px
DEBUG SipServletMessageImpl: Creating new message instance
DEBUG SipServletMessageImpl: Headers:
DEBUG SipServletMessageImpl: - Content-Length
DEBUG SipServletMessageImpl: - Via
DEBUG SipServletMessageImpl: - From
DEBUG SipServletMessageImpl: - To
DEBUG SipServletMessageImpl: - Call-ID
DEBUG SipServletMessageImpl: - CSeq
DEBUG SipServletMessageImpl: - Max-Forwards
DEBUG SipServletMessageImpl: - Contact
DEBUG Testcase2Servlet: Generated reponse:
Core JAIN SIP message:
-------start--------
SIP/2.0 200
Via: SIP/2.0/UDP 213.173.243.219:5061;received=localhost;rport=5061
From: "sipp" <sip:sipp@213.173.243.219:5061>;tag=1
To: "sut" <sip:service@127.0.0.1:5060>;tag=bdg1px
Call-ID: 1.1188.213.173.243.219@sipp.call.id
CSeq: 1 INVITE
Max-Forwards: 70
Contact: <sip:127.0.0.1:5060>
Content-Length: 0


-------end--------

DEBUG SipServletResponseImpl: Sending reponse:
SIP/2.0 200
Via: SIP/2.0/UDP 213.173.243.219:5061;received=localhost;rport=5061
From: "sipp" <sip:sipp@213.173.243.219:5061>;tag=1
To: "sut" <sip:service@127.0.0.1:5060>;tag=bdg1px
Call-ID: 1.1188.213.173.243.219@sipp.call.id
CSeq: 1 INVITE
```

```
Max-Forwards: 70
Contact: <sip:127.0.0.1:5060>
Content-Length: 0


INFO SessionManager: Storing session with id 1.1188.213.173.243.219@sipp
    .call.id:service@127.0.0.1:5060:bdg1px:sipp@213.173.243.219:5061:1
DEBUG Testcase2Servlet: ------------STAGE 2----------------SENDT
     OK WAITING FOR ACK
DEBUG SipNetworkManager: Request with metod "ACK" received at SSC with
    server transaction id gov.nist.javax.sip.stack.
    SIPServerTransaction@b759d917
DEBUG SipNetworkManager: dialog id:1.1188.213.173.243.219@sipp.call.
    id:service@127.0.0.1:5060:bdg1px:sipp@213.173.243.219:5061:1
DEBUG SipNetworkManager:
 request:
ACK sip:service@127.0.0.1:5060 SIP/2.0
Via: SIP/2.0/UDP 213.173.243.219:5061;received=localhost;rport=5061
From: "sipp" <sip:sipp@213.173.243.219:5061>;tag=1
To: "sut" <sip:service@127.0.0.1:5060>;tag=bdg1px
Call-ID: 1.1188.213.173.243.219@sipp.call.id
CSeq: 1 ACK
Contact: <sip:sipp@213.173.243.219:5061>
Max-Forwards: 70
Subject: Performance Test
Content-Length: 0


DEBUG SipServletMessageImpl: Creating new message instance
DEBUG SipServletMessageImpl: Headers:
DEBUG SipServletMessageImpl: - Content-Length
DEBUG SipServletMessageImpl: - Via
DEBUG SipServletMessageImpl: - From
DEBUG SipServletMessageImpl: - To
DEBUG SipServletMessageImpl: - Call-ID
DEBUG SipServletMessageImpl: - CSeq
DEBUG SipServletMessageImpl: - Contact
DEBUG SipServletMessageImpl: - Max-Forwards
DEBUG SipServletMessageImpl: - Subject
DEBUG SessionManager: Found session associated with dialogID "
    1.1188.213.173.243.219@sipp.call.id:service@127.0.0.1
    :5060:bdg1px:sipp@213.173.243.219:5061:1"
INFO Testcase2Servlet: ACK Recived
DEBUG Testcase2Servlet: ------------STAGE 3----------------GOT
    ACK
DEBUG SipNetworkManager: Request with metod "BYE" received at SSC with
    server transaction id gov.nist.javax.sip.stack.
    SIPServerTransaction@9cd2e28a
DEBUG SipNetworkManager: dialog id:1.1188.213.173.243.219@sipp.call.
    id:service@127.0.0.1:5060:bdg1px:sipp@213.173.243.219:5061:1
DEBUG SipNetworkManager:
 request:
BYE sip:service@127.0.0.1:5060 SIP/2.0
Via: SIP/2.0/UDP 213.173.243.219:5061;received=localhost;rport=5061
From: "sipp" <sip:sipp@213.173.243.219:5061>;tag=1
To: "sut" <sip:service@127.0.0.1:5060>;tag=bdg1px
Call-ID: 1.1188.213.173.243.219@sipp.call.id
CSeq: 2 BYE
Contact: <sip:sipp@213.173.243.219:5061>
Max-Forwards: 70
Subject: Performance Test
Content-Length: 0
```

```
175 DEBUG SipServletMessageImpl: Creating new message instance
    DEBUG SipServletMessageImpl: Headers:
    DEBUG SipServletMessageImpl: - Content-Length
178 DEBUG SipServletMessageImpl: - Via
    DEBUG SipServletMessageImpl: - From
    DEBUG SipServletMessageImpl: - To
181 DEBUG SipServletMessageImpl: - Call-ID
    DEBUG SipServletMessageImpl: - CSeq
    DEBUG SipServletMessageImpl: - Contact
184 DEBUG SipServletMessageImpl: - Max-Forwards
    DEBUG SipServletMessageImpl: - Subject
    DEBUG SessionManager: Found session associated with dialogID "
        1.1188.213.173.243.219@sipp.call.id:service@127.0.0.1
        :5060:bdg1px:sipp@213.173.243.219:5061:1"
187 DEBUG Testcase2Servlet: -------------STAGE 4----------------GOT
        BYE CONFIRMING
    DEBUG SipServletMessageImpl: Creating new message instance
    DEBUG SipServletMessageImpl: Headers:
190 DEBUG SipServletMessageImpl: - Content-Length
    DEBUG SipServletMessageImpl: - Via
    DEBUG SipServletMessageImpl: - From
193 DEBUG SipServletMessageImpl: - To
    DEBUG SipServletMessageImpl: - Call-ID
    DEBUG SipServletMessageImpl: - CSeq
196 DEBUG SipServletMessageImpl: - Max-Forwards
    DEBUG SipServletMessageImpl: - Contact
    DEBUG SipServletResponseImpl: Sending reponse:
199 SIP/2.0 200
    Via: SIP/2.0/UDP 213.173.243.219:5061;received=localhost;rport=5061
    From: "sipp" <sip:sipp@213.173.243.219:5061>;tag=1
202 To: "sut" <sip:service@127.0.0.1:5060>;tag=bdg1px
    Call-ID: 1.1188.213.173.243.219@sipp.call.id
    CSeq: 2 BYE
205 Max-Forwards: 70
    Contact: <sip:127.0.0.1:5060>
    Content-Length: 0
208

    DEBUG Testcase2Servlet: -------------STAGE 5----------------BYE
        CONFIRMED - 1 COMPLETED
211 INFO SipServletContainer: Shutting down container
    INFO ApplicationManager: shutdown complete
    INFO FlowManager: shutdown complete
214 INFO SipNetworkManager: shutdown complete
```

SIPp blev startet med følgende kommando

```
1 sipp -m 1 -sn uac 127.0.0.1
```

Sipp afgav under afprøvningen følgende output

testcase2_sipp_output.txt Page 1/1

```
Resolving 127.0.0.1...
------------------------------ Scenario Screen -------- [1-4]: Change Screen --
  Call-rate(length)     Port   Total-time  Total-calls  Remote-host
  10.0(0 ms)/1.000s   5061       0.37 s            1  127.0.0.1:5060(UDP)

  Call limit reached (-m 1), 0.378 s period  3 ms scheduler resolution
  0 concurrent calls (limit 30)          Peak was 1 calls, after 0 s
  0 out-of-call msg (discarded)
  1 open sockets

                              Messages  Retrans   Timeout   Unexpected-Msg
      INVITE ---------->      1         0         0
         100 <----------      0         0                   0
         180 <----------      0         0                   0
         200 <---------- E-RTD 1         0                   0
         ACK ---------->      1         0
             [     0 ms]
         BYE ---------->      1         0         0
         200 <----------      1         0                   0

------------------------------ Test Terminated --------------------------------


------------------------------ Statistics Screen ------- [1-4]: Change Screen --
  Start Time              | 2005-10-30 16:30:51

  Last Reset Time         | 2005-10-30 16:30:51

  Current Time            | 2005-10-30 16:30:52

-------------------------+---------------------------+--------------------------

  Counter Name            | Periodic value            | Cumulative value
-------------------------+---------------------------+--------------------------

  Elapsed Time            | 00:00:00:417              | 00:00:00:417

  Call Rate               |    2.398 cps              |    2.398 cps

-------------------------+---------------------------+--------------------------

  Incoming call created   |        0                  |        0

  OutGoing call created   |        1                  |        1

  Total Call created      |                           |        1

  Current Call            |        0                  |

-------------------------+---------------------------+--------------------------

  Successful call         |        1                  |        1

  Failed call             |        0                  |        0

-------------------------+---------------------------+--------------------------

  Response Time           | 00:00:00:230              | 00:00:00:230

  Call Length             | 00:00:00:276              | 00:00:00:276

------------------------------ Test Terminated --------------------------------
```

1/1

C.3 Testcase 3

ApplicationManager blev konfigureret med følgende Deployment Descriptor

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE sip-app
  PUBLIC "-//Java Community Process//DTD SIP Application 1.0//EN"
  "http://www.jcp.org/dtd/sip-app_1_0.dtd">


<sip-app>
  <display-name>Deployment Descriptor test</display-name>

  <context-param>
    <param-name>pet</param-name>
    <param-value>cat</param-value>
  </context-param>

  <listener>
    <listener-class>not.supported</listener-class>
  </listener>

  <servlet>
    <servlet-name>TestServlet</servlet-name>
    <servlet-class>dk.itu.ssc.tests.Testcase3Servlet</servlet-class>
  <init-param>
     <param-name>key1</param-name>
     <param-value>value1</param-value>
  </init-param>
     <init-param>
     <param-name>key2</param-name>
     <param-value>value2</param-value>
    </init-param>
  </servlet>

  <servlet-mapping>
    <servlet-name>TestServlet</servlet-name>
    <pattern>
      <and>
        <equal>
          <var>request.method</var>
          <value>INVITE</value>
        </equal>
        <equal>
          <var>request.uri.user</var>
          <value>uas-passive</value>
        </equal>
      </and>
    </pattern>
  </servlet-mapping>

  <session-config>
    <session-timeout>1</session-timeout>
  </session-config>
</sip-app>
```

Bilag D

Framework-konfiguration beskrevet på udvidet Backus-Naur form

config.ebnf


```
configuration     := '<configuration>'
                        building-blocks?
                        location-services?
                        (
                           scenarios
                           scenario-mappings
                        )?
                     '</configuration>'

scenarios         := '<scenarios>' scenario+ '</scenarios>'

scenario          := '<scenario>'
                         name
                         class
                         '<hooks>' hook+ '</hooks>'
                         '<location-service>' reference '</location-service>'
                         buildingblockref
                         callstate
                      '</scenario>'

buildingblockref  := '<buildingblock>' reference '</buildingblock>'

callstate         := '<call-state>'
                        class
                        param*
                     '</call-state>'

hook              := '<hook>'
                        name
                        '<class>' classname '</class>'
                        '<method>' litterals '</method>'
                     '</hook>'

building-blocks   := building-block+

building-block    :=  '<block>'
                         name
                         class
                         param*
                      '</block>'

location-services := location-service+

location-service  := '<location-service>'
                        name
                        class
                        param*
                     '</location-service>'

scenario-mappings := '<scenario-mappings>'
                        scenario-mapping+
                     '</scenario-mappings>'

scenario-mapping  := '<scenario-mapping>'
                        reference
                        name
                        '<matching-rule>' rule '<matching-rule'>
                     '</scenario-mapping>'

rule              := [JSR-116 matching rule]

param             := '<param>'
                        '<param-name>' litterals '</param-name>'
                        '<param-value>' litterals '</param-value>'
                     '</param>'

classname         := [fully classified classname]

class             := '<class>' classname '</class>'

litterals         := [A-Za-z0-9]+

reference         := '<ref>' litterals '</ref>'

name              := '<name>' litterals '</name>'
```

Bilag E

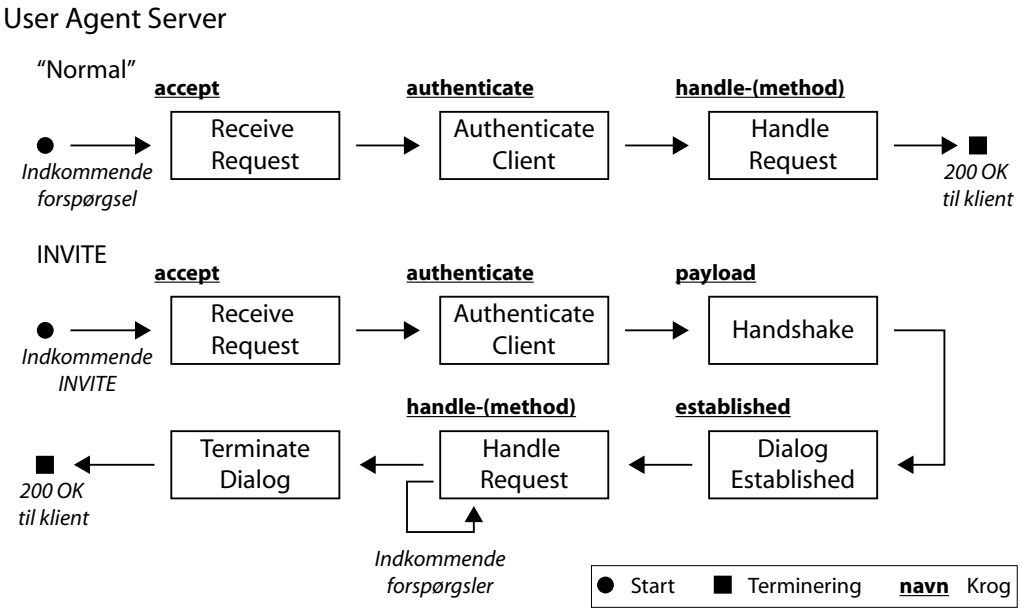
Dokumentation af kroge

De følgende afsnit vil beskrive de forskellige scenarier samt de tilhørende kroge i detaljer. For hver krog beskrives:

- Formålet med krogen
- Signaturen en metode, der benyttes til at implementere denne krog, skal have
- Parametrene krogen skal modtage
- En eventuel returværdi
- En eventuel standardimplementering. En standard implementering kan også være en “tom krog”, hvilket betyder, at krogen vil blive sprunget over såfremt der ikke er vedhæftet en implementering.

E.1 User Agent Server

Flowdiagram



Figur E.1: Flowet i User Agent Server scenariet

Kroge

accept

Formål	at afgøre om forespørgslen skal modtages
Metodesignatur	(CallState callstate, Request request) : Boolean
Parametre	1. callstate: et tilstandsobjekt 2. request : forespørgslen
Returværdi	boolesk værdi der angiver om forespørgslen accepteres
Standard-implementering	acceptere alle forespørgsler

authenticate

Formål	at afgøre om en klient må tilgå tjenesten
Metodesignatur	(CallState callstate, String username, String password) : Boolean
Parametre	1. callstate: et tilstandsobjekt 2. username: brugernavn 3. password: kodeord
Returværdi	boolesk værdi der angiver om login-informationen accepteres
Standard-implementering	returnerer “sand” for alle forespørgsler.

handle-(requestname)

Nedenstående beskrivelse dækker følgende kroge

- handle-invite
- handle-option
- handle-register
- handle-ack
- handle-cancel

Formål	at håndtere forskellige typer af forespørgsler.
Metodesignatur	(CallState callstate, Request request) : Response
Parametre	1. callstate: et tilstandsobjekt 2. request : forespørgslen
Returværdi	et svar-objekt
Standard-implementering	returnerer svaret "500 Unimplemented method"

payload

Formål	at producere et evt indhold til det indledende handshake
Metodesignatur	(CallState callstate) : String
Parametre	1. callstate: et tilstandsobjekt
Returværdi	data der skal indgå i kroppen på 200 OK- svaret på den indledende INVITE-forespørgsel
Standard-implementering	returnere en tom streng

established

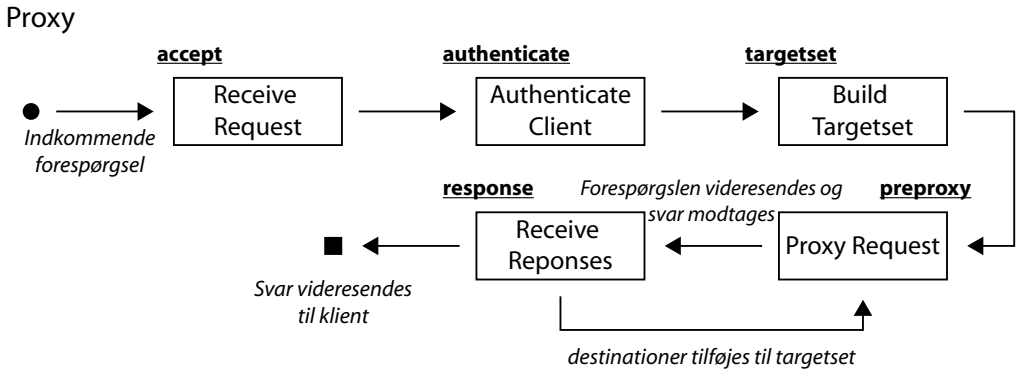
Formål	at implementere eventuel forretningslogik, der skal udføres ved etablering af en dialog
Metodesignatur	(CallState callstate) : void
Parametre	1. callstate: et tilstandsobjekt
Returværdi	ingen
Standard-implementering	tom krog

terminated

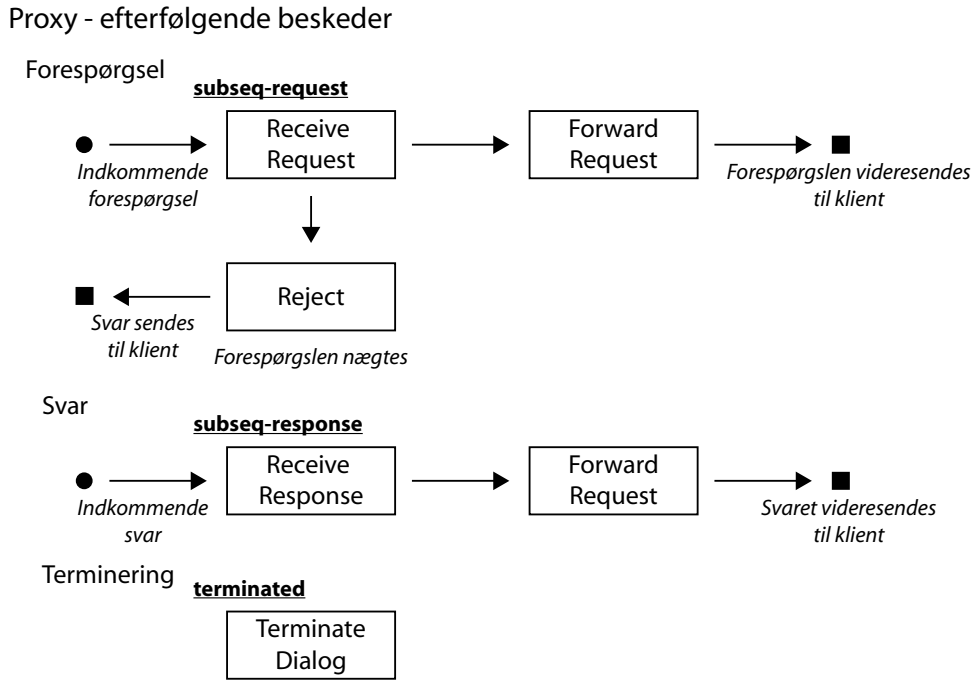
Formål	at implementere eventuel forretningslogik, der skal udføres når dialogen nedlægges
Metodesignatur	(CallState callstate) : void
Parametre	1. callstate: et tilstandsobjekt
Returværdi	ingen
Standard-implementering	tom krog

E.2 Proxy

Flowdiagram



Figur E.2: Flowet i Proxy scenariet



Figur E.3: Flowet for efterfølgende beskeder i Proxy scenariet

Kroge

accept

Formål	at afgøre om forespørgslen skal modtages
Metodesignatur	(CallState callstate, Request request) : Boolean
Parametre	1. callstate: et tilstandsobjekt 2. request : forespørgslen
Returværdi	boolesk værdi der angiver om forespørgslen accepteres
Standard-implementering	acceptere alle forespørgsler

authenticate

Formål	at afgøre om en klient må tilgå tjenesten
Metodesignatur	(CallState callstate, String username, String password) : Boolean
Parametre	1. callstate: et tilstandsobjekt 2. username: brugernavn 3. password: kodeord
Returværdi	boolesk værdi der angiver om login-informationen accepteres
Standard-implementering	returnerer “sand” for alle forespørgsler.

targetset

Formål	at producere en liste over destinationer for en indkommende forespørgsel
Metodesignatur	(CallState callstate, Request request) : Set<URI>
Parametre	1. callstate: et tilstandsobjekt 2. request : forespørgslen
Returværdi	et Set indeholdende en eller flere URI-instanser, der tilsammen udgør target-sættet
Standard-implementering	indsætter den oprindelige destination som det eneste element i target-sættet

preproxy

Formål	at implementere eventuel forretningslogik, der skal udføres før forespørgslen videresendes. Kan f.eks. benyttes hvis man ikke implementerer krogen **targetset**
Metodesignatur	(CallState callstate, Request request) :
Parametre	1. callstate: et tilstandsobjekt 2. request : forespørgslen
Returværdi	ingen
Standard-implementering	tom krog

response

Formål	at modtage svarene på den videresendte forespørgsel
Metodesignatur	(CallState callstate, Response response, Set<URI> targetset) : Set<URI>
Parametre	1. callstate: et tilstandsobjekt 2. response: det indkommende svar 3. targetset: et Set der indeholder eventuelle destinationer, der endnu ikke er modtaget svar fra
Returværdi	target-sættet, såfremt krogen ønsker at tilføje flere destinationer indsættes de i Set instansen, hvis ikke returneres det blot uændret. Såfremt sættet er tomt vil dette afslutte scenariet.
Standard-implementering	returnerer targetsættet uændret.

Følgende kroge er defineret for efterfølgende beskeder i et proxy-scenarium, der påbegynder en dialog.

subseq-request

Formål	at modtage en efterfølgende forespørgsel
Metodesignatur	(CallState callstate, Request request) : Message
Parametre	1. callstate: et tilstandsobjekt 2. request: forespørgslen
Returværdi	forspørgslen eller et svar på denne. Såfremt krogen returnere et svar videresendes forespørgslen *ikke*
Standard-implementering	tom krog, forespørgslen videresendes

subseq-response

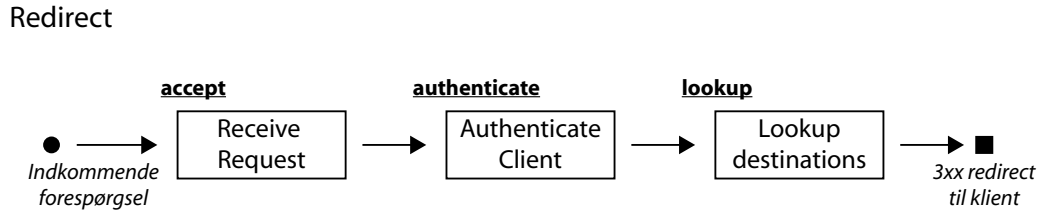
Formål	at modtage efterfølgende svar. Krogen har ikke mulighed for at manipulere svaret
Metodesignatur	(CallState callstate, Response response) : void
Parametre	1. callstate: et tilstandsobjekt 2. response: det efteføl gende svar
Returværdi	ingen
Standard-implementering	tom krog

terminated

Formål	implementerer logik der skal afvikles når dialogen nedlægges
Metodesignatur	(CallState callstate) : void
Parametre	1. callstate: et tilstandsobjekt
Returværdi	ingen
Standard-implementering	tom krog

E.3 Redirect

Flowdiagram



Figur E.4: Flowet for Redirect scenariet

Kroge

accept

Formål	at afgøre om forespørgslen skal modtages
Metodesignatur	(CallState callstate, Request request) : Boolean
Parametre	1. callstate: et tilstandsobjekt 2. request : forespørgslen
Returværdi	boolesk værdi der angiver om forespørgslen accepteres
Standard-implementering	acceptere alle forespørgsler

authenticate

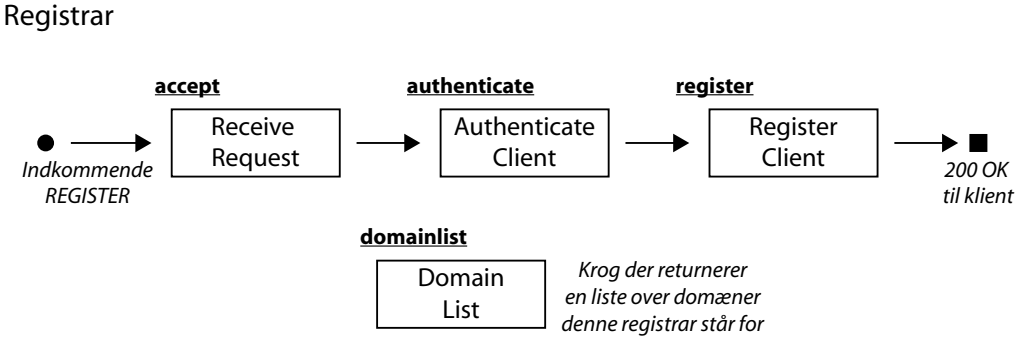
Formål	at afgøre om en klient må tilgå tjenesten
Metodesignatur	(CallState callstate, String username, String password) : Boolean
Parametre	1. callstate: et tilstandsobjekt 2. username: brugernavn 3. password: kodeord
Returværdi	boolesk værdi der angiver om login-informationen accepteres
Standard-implementering	returnerer “sand” for alle forespørgsler.

lookup

Formål	at viderestille en forespørgsel
Metodesignatur	(CallState callstate, Request request) : Set<URI>
Parametre	1. callstate: et tilstandsobjekt 2. request: forespørgslen der skal videresendes
Returværdi	et Set indeholdende en eller flere nye destinationer for forespørgslen, såfremt sættet er tom returneres svaret "404 not found"
Standard-implementering	

E.4 Registrar

Flowdiagram



Figur E.5: Flowet i Registrar Scenariet

Kroge

accept

Formål	at afgøre om forespørgslen skal modtages
Metodesignatur	(CallState callstate, Request request) : Boolean
Parametre	1. callstate: et tilstandsobjekt 2. request : forespørgslen
Returværdi	boolesk værdi der angiver om forespørgslen accepteres
Standard-implementering	acceptere alle forespørgsler

authenticate

Formål	at afgøre om en klient må tilgå tjenesten
Metodesignatur	(CallState callstate, String username, String password) : Boolean
Parametre	1. callstate: et tilstandsobjekt 2. username: brugernavn 3. password: kodeord
Returværdi	boolesk værdi der angiver om login-informationen accepteres
Standard-implementering	returnerer “sand” for alle forespørgsler.

register

Formål	at registrere alternative adresser for en klients alias-adrese
Metodesignatur	(CallState callstate, URI alias, URI destination) : Response
Parametre	1. callstate: et tilstandsobjekt 2. alias: alias-adresse 3. destination: ny destination
Returværdi	et svar, såfremt krogen blot ønsker at returnere 200 OK kan null returneres
Standard-implementering	alias og destination indsættes i en lokations-tjeneste såfremt den er registreret med navnet “locationService” i tilstandsobjektet. Hvis en lokations-tjeneste ikke er defineret, returneres svaret “500 Configuration error”

domainlist

Formål	at returnere en liste over hvilke domæne-navne denne registrar er ansvarlig for
Metodesignatur	(CallState callstate) : Set<String>
Parametre	1. callstate: et tilstandsobjekt
Returværdi	et Set indeholdende domainenavne
Standard-implementering	returnere et Set indeholdende navnet “*” der vil medføre at alle forespørgsler modtages

Bilag F

Eksempel-konfigurationsfiler

F.1 SIP Alias'o'Magic

config_alias_o_magic.xml Page 1/2

```
<!-- Konfiguration af SIP Alias'o'Magic, en tjeneste der viderestiller -->
<!-- rettet imod et alias til en eller flere "rigtige" destinationer  -->
<configuration>
    <!-- Konfiguration af byggeblokke -->
    <building-blocks>
      <!-- En byggenblok der kan validere login-information -->
      <block>
        <name>UserLookup</name>
        <class>my.package.UserLookup</class>
        <param>
          <key>passwordfile</key>
          <value>/etc/passwd</value>
        </param>
      </block>
    </building-blocks>

    <location-services>
      <location-service>
        <name>LocationService</name>
        <class>framework.DataSourceLocationService</class>
        <param>
          <key>data-source</key>
          <value>ContainerManagededataSource</value>
        </param>
      </location-service>
    </location-services>

    <!-- Konfiguration af standardscenarier -->
    <scenarios>
      <!-- Konfiguration af et Registrar scenarium -->
      <scenario>
        <name>MyRegistrar</name>
        <class>framework.scenarios.RegistrarScenario</class>
        <hooks>
          <!-- Krog der implementere brugervalideringslogik -->
          <hook>
            <name>authenticate</name>
            <class>my.package.MyRegistrar</class>
            <method>authorizeUser</method>
          </hook>

          <!-- Standardimplementeringen af krogen "register" benyttes -->
          <!-- til lagring af bruger-data                             -->
        </hooks>

        <!-- Dette scenario benytter samme lokationstjeneste som -->
        <!-- MyRedirector-scenariet                              -->
        <location-service>
          <ref>LocationService</ref>
        </location-service>

        <!-- Scenariet konfigureres med en byggeblok -->
        <buildingblock>
          <ref>UserLookup</ref>
        </buildingblock>
      </scenario>

      <!-- Konfiguration af et Redirect scenarium -->
      <scenario>
        <name>MyRedirect</name>
        <class>framework.scenarios.RedirectScenario</class>

        <!-- Standardimplementeringen af krogen "lookup" benyttes -->
```

config_alias_o_magic.xml Page 2/2

```
        <!-- Dette scenario benytter samme lokationstjeneste som -->
        <!-- MyRegistrar-scenariet                               -->
        <location-service>
          <ref>LocationService</ref>
        </location-service>
      </scenario>
    </scenarios>

    <!-- Beskrivelse hvornår de forskellige instanser skal bruges -->
    <scenario-mappings>
      <!-- mapping af viderestillings-tjenesten -->
      <scenario-mapping>
        <ref>MyRedirector</ref>
        <name>Redirecting service</name>
        <matching-rule>
          <and>
            <!-- besvarer alle forspørgsler til @leetcorp.com -->
            <equal>
              <var>request.to.uri.host</var>
              <value>leetcorp.com</value>
            </equal>

            <!-- kræver at forespørgslen indeholder et brugernavn  -->
            <!-- (delen foran @)                                    -->

            <exists>
              <var>request.to.uri.user</var>
            </exists>
          </and>
        </matching-rule>
      </scenario-mapping>

      <!-- mapping af registrerings-tjenesten -->
      <scenario-mapping>
        <ref>MyRegistrar</ref>
        <name>Registration Service</name>
        <matching-rule>
          <and>
            <!-- besvarer alle forspørgsler til @leetcorp.com -->
            <equal>
              <var>request.to.uri.host</var>
              <value>leetcorp.com</value>
            </equal>

            <!-- kræver at forespørglsen ikke indeholder et -->
            <!-- brugernavn (delen foran @)                 -->
            <not>
              <exists>
                <var>request.to.uri.user</var>
              </exists>
            </not>
          </and>
        </matching-rule>
      </scenario-mapping>
    </scenario-mappings>
</configuration>
```

F.2 Hyper VoiceMail Deluxe

config_voicemail_deluxe.xml Page 1/2

```
<!-- Konfiguration af Hyper VoiceMail Deluxe, en tjenste der lader en  -->
<!-- bruger aflytte talebeskeder lagt af klienter af anden vej       -->
<configuration>
    <!-- Konfiguration af byggeblokke -->
    <building-blocks>
      <!-- En byggenblok der kan validere login-information -->
      <block>
        <name>UserLookup</name>
        <class>my.package.UserLookup</class>
        <param>
          <key>passwordfile</key>
          <value>/etc/passwd</value>
        </param>
      </block>
    </building-blocks>

    <!-- Konfiguration af standardscenarier -->
    <scenarios>
      <!-- Konfiguration af et Proxy scenarium -->
      <scenario>
        <name>MyProxy</name>
        <class>framework.scenarios.ProxyScenario</class>
        <hooks>
          <!-- Krog der implementere brugervalideringslogik -->
          <hook>
            <name>authenticate</name>
            <class>my.package.MyRegistrar</class>
            <method>authorizeUser</method>
          </hook>

          <!-- Returnerer medieserverens kontaktinformation  -->
          <hook>
            <name>payload</name>
            <class>my.package.MediaServeRemoteControl</class>
            <method>generateSDP</method>
          </hook>

          <!-- Opretter en session på medie-serveren -->
          <hook>
            <name>established</name>
            <class>my.package.MediaServeRemoteControl</class>
            <method>initSession</method>
          </hook>

          <!-- Nedlægger brugerens session med medieserveren  -->
          <!-- ved hjælp af id'et i SessionIDCallState        -->
          <hook>
            <name>terminated</name>
            <class>my.package.MediaServeRemoteControl</class>
            <method>tairDownSession</method>
          </hook>
        </hooks>

        <!-- Scenariet konfigureres med en byggeblok -->
        <buildingblock>
          <ref>UserLookup</ref>
        </buildingblock>

        <!-- CallState objekt der indeholder et sessionsid  samt -->
        <!-- ip-adressen medie-serveren kan kontaktes            -->
        <call-state>
          <class>my.package.SessionIDCallState</class>
        <param>
```

config_voicemail_deluxe.xml Page 2/2

```
            <param-name>ServeAddress</param-name>
            <param-value>10.1.33.7</param-value>
          </param>
        </call-state>
      </scenario>
    </scenarios>

    <!-- Beskrivelse hvornår de forskellige instanser skal bruges -->
    <scenario-mappings>
      <!-- mapping af registrerings-tjenesten -->
      <scenario-mapping>
        <ref>MyProxy</ref>
        <name>Hyper VoiceMail Deluxe</name>
        <matching-rule>
          <and>
            <!-- besvarer alle forspørgsler til -->
            <!-- @voicemail.leetcorp.com       -->
            <equal>
              <var>request.to.uri.host</var>
              <value>voicemail.leetcorp.com</value>
            </equal>

            <!-- kræver at forespørglsen indeholder et brugernavn -->
            <!-- (delen foran @)                                  -->
            <exists>
              <var>request.to.uri.user</var>
            </exists>
          </and>
        </matching-rule>
      </scenario-mapping>
    </scenario-mappings>
</configuration>
```

F.3 Mega Logger II

config_megaloggerII.xml Page 1/1

```
<!-- Konfiguration af Mega Logger II, en tjeneste der logger kunders       -->
<!-- forbrug af tjensterne  SIP Alias'o'Magic" og "Hyper VoiceMail Deluxe" -->
<configuration>
    <!-- Konfiguration af byggeblokke -->
    <building-blocks>
        <!-- En byggenblok der kan logge forbrugsdata -->
        <block>
            <name>Logger</name>
            <class>my.package.Logger</class>
            <param>
                <key>logfile</key>
                <value>/var/log/megalogger.log</value>
            </param>
        </block>
    </building-blocks>

    <!-- Konfiguration af standardscenarier -->
    <scenarios>
        <!-- Konfiguration af et Proxy scenarium -->
        <scenario>
            <name>MyProxy</name>
            <class>framework.scenarios.ProxyScenario</class>
            <hooks>
                <!-- Krog der udtrækker starttidspunkt og brugernavn  -->
                <hook>
                    <name>preproxy</name>
                    <class>my.package.MegaLogger</class>
                    <method>start</method>
                </hook>

        <!-- krog der logger forbrugs-data vha en byggeblok -->
                <hook>
                    <name>terminated</name>
                    <class>my.package.MegaLogger</class>
                    <method>end</method>
                </hook>
            </hooks>

            <!-- Scenariet konfigureres med en byggeblok -->
            <buildingblock>
                <ref>Logger</ref>
            </buildingblock>

        <!-- CallState objekt der kan indeholder start-tidspunkt -->
        <!-- samt brugernavn -->
        <call-state>
            <class>my.package.MegaLoggerCallState</class>
        </call-state>
        </scenario>
    </scenarios>

    <!-- Beskrivelse hvornår de forskellige instanser skal bruges -->
    <scenario-mappings>
        <!-- mapping af viderestillings-tjenesten -->
        <scenario-mapping>
            <ref>MyProxy</ref>
            <name>Mega Logger II</name>

        <!-- Proxien modtager alle beskeder -->
            <matching-rule>*</matching-rule>
        </scenario-mapping>
    </scenario-mappings>
</configuration>
```

1/1

Bilag G

Kildekode

log4j.properties


```
log4j.appender.sysout=org.apache.log4j.ConsoleAppender
log4j.appender.sysout.layout=org.apache.log4j.PatternLayout
log4j.appender.sysout.layout.ConversionPattern=%-6p %c{1}: %m%n

log4j.rootCategory=DEBUG, sysout
```

ContainerPart.java


```
package dk.itu.ssc;

public interface ContainerPart {

    public void configure(ServerContext config);

    /**
     * Invoked when the container shuts down
     */
    public void shutdown();

}
```

FlowManager.java


```
package dk.itu.ssc;

import java.io.IOException;

import javax.servlet.ServletException;
import javax.sip.Dialog;

import org.apache.log4j.Logger;

import dk.itu.ssc.application.ApplicationManager;
import dk.itu.ssc.application.Dispatcher;
import dk.itu.ssc.network.wrapper.SipServletRequestImpl;
import dk.itu.ssc.network.wrapper.SipServletResponseImpl;
import dk.itu.ssc.network.wrapper.SipSessionImpl;
import dk.itu.ssc.session.SessionManager;

/**
 * Manages the steps a request or response goes through from it is taken off the
 * netstack to the target servlet returns
 *
 * @author Mads Danquah
 */
public class FlowManager implements ContainerPart {

    private static final Logger log = Logger.getLogger(FlowManager.class);

    private static ServerContext configuration = null;

    private static ApplicationManager applicationManager;

    public void configure(ServerContext sc) {
        configuration = sc;
        log.info("FlowManager configured");
    }

    public void processRequest(SipServletRequestImpl req) {
        Dispatcher dispatcher = null;

        Dialog dialog = req.getDialog();

        SipSessionImpl session = null;
        if (dialog != null) {
            // lookup the session
            session = SessionManager.getSession(dialog);
        }

        // lookup the application and servlet
        if (session == null) {
            // use the ApplicationManager to get a handle
            dispatcher = applicationManager.route(req);

            // initial request
            req.setInitial(true);

            // create the session
            session = SessionManager.createSession(req, dispatcher
                    .getSipServletApplication(), true);
            // store the handler in the session for later retrival
            session.setHandler(dispatcher);

            // store the dialog
            session.setDialog(dialog);
        } else {
            dispatcher = session.getDispatcher();
        }

        req.setSession(session);

        try {
            dispatcher.forward(req, null);
        } catch (ServletException e) {
            log.error("Could not dispatch", e);

        } catch (IOException e) {
            log.error("Could not dispatch", e);
        }

    }

    public void processResponse(SipServletResponseImpl res) {
        // We' an UAC
        // lookup the session
        SipSessionImpl session = SessionManager.getSession(res
                .getClientTransaction().getDialog());

        if (session == null) {
            log
```

FlowManager.java


```
                    .fatal("Could not locate session for a response ( A response shold always be within a session)");
            SipServletContainer.getInstance().shutdown();
        }
        // get the dispatacher from the session
        Dispatcher dispatcher = session.getDispatcher();

        res.setSession(session);

        // invoke the servlet
        try {
            dispatcher.forward(null, res);
        } catch (ServletException e) {
            log.error("Could not dispatch", e);

        } catch (IOException e) {
            log.error("Could not dispatch", e);
        }
    }

    /**
     * @return Returns the applicationManager.
     */
    public ApplicationManager getApplicationManager() {
        return applicationManager;
    }

    /**
     * @param applicationManager
     *            The applicationManager to set.
     */
    public void setApplicationManager(ApplicationManager applicationManager) {
        FlowManager.applicationManager = applicationManager;
    }

    /**
     * @return Returns the configuration.
     */
    public static ServerContext getConfiguration() {
        return configuration;
    }

    public void shutdown() {
        // do nothing for now
        log.info("shutdown complete");
    }

}
```

ServerContext.java


```
package dk.itu.ssc;

import java.net.InetAddress;
import java.net.InetSocketAddress;
import java.net.UnknownHostException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;

import org.apache.log4j.Logger;

import dk.itu.ssc.exceptions.UnknownProtocolException;
import dk.itu.ssc.network.ListeningPointAddress;

public class ServerContext {
    static private Logger log = Logger.getLogger(ServerContext.class);

    public Map<String, String> configuration;

    protected String applicationRoot;

    protected String serverRoot;

    protected ArrayList<ListeningPointAddress> listeningPoints;

    public ServerContext() {
        this.configuration = new HashMap<String, String>();
        this.listeningPoints = new ArrayList<ListeningPointAddress>();
    }

    public String getApplicationRoot() {
        return applicationRoot;
    }

    public void setApplicationRoot(String applicationRoot) {
        this.applicationRoot = applicationRoot;
    }

    public ArrayList<ListeningPointAddress> getListeningPoints() {
        return listeningPoints;
    }

    public void addListeningPoint(String protocol, String address, int port)
            throws UnknownHostException, UnknownProtocolException {
        InetSocketAddress isa = new InetSocketAddress(InetAddress
                .getByName(address), port);
        if (!(protocol.equalsIgnoreCase("udp") || protocol
                .equalsIgnoreCase("tcp"))) {
            throw new UnknownProtocolException("\"" + protocol
                    + "\" is an unsupported protocol");
        }
        log.debug("Adding listeningpoint " + isa + "/" + protocol);
        this.listeningPoints.add(new ListeningPointAddress(protocol, isa));
    }

    public String getServerRoot() {
        return serverRoot;
    }

    public void setServerRoot(String serverRoot) {
        this.serverRoot = serverRoot;
    }

}
```

SipServletContainer.java


```
package dk.itu.ssc;

import java.net.UnknownHostException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

import org.apache.log4j.Logger;

import dk.itu.ssc.application.ApplicationManager;
import dk.itu.ssc.application.hardcoded.HarcodedApplicationStrategy;
import dk.itu.ssc.exceptions.ConfigurationErrorException;
import dk.itu.ssc.exceptions.SipNetworkException;
import dk.itu.ssc.exceptions.UnknownProtocolException;
import dk.itu.ssc.network.SipNetworkManager;

// TODO: Lav flow-beskrivelse her

public class SipServletContainer implements Runnable {

    static Logger log = Logger.getLogger(SipServletContainer.class);

    private static SipServletContainer instance;

    SipNetworkManager networkManager = null;

    FlowManager flowManager = null;

    ApplicationManager applicationManager = null;

    private boolean running = false;

    private ServerContext serverContext;

    private List<ContainerPart> containerParts;


    /**
     * Singleton that ensures no-more than one instance of the container can
     * exist at one time
     *
     * @return
     */
    public static SipServletContainer getInstance() {
        if (SipServletContainer.instance == null) {
            SipServletContainer.instance = new SipServletContainer();
        }

        return SipServletContainer.instance;
    }

    private SipServletContainer() {
        this.serverContext = new ServerContext();
        this.containerParts = new ArrayList<ContainerPart>();
    }

    public void configure(Map<String, String> configuration) {
        serverContext.configuration = configuration;
        log.debug("Basic configuration: " + configuration);
    }

    public void configure() throws ConfigurationErrorException {
        // TODO: Read configurations from serverconfig.xml
        log.debug("Configuring container");

        // Configure manually for now
        serverContext.setApplicationRoot("/");
        serverContext.setServerRoot("/");
        try {
            serverContext.addListeningPoint("udp", "127.0.0.1", 5060);
            serverContext.addListeningPoint("tcp", "127.0.0.1", 5060);
        } catch (UnknownHostException e) {
            throw new ConfigurationErrorException("Invalid host", e);
        } catch (UnknownProtocolException e) {
            throw new ConfigurationErrorException("Unknown protocol", e);
        }
    }

    public void boot() {
        log.info("Booting container");

        // retrive Container configurations and initialise ServerContext
        try {
            this.configure();
```

SipServletContainer.java


```
        } catch (ConfigurationErrorException e) {
            e.printStackTrace();
            log.fatal("Could not configure network: " + e.getMessage());
            System.exit(1);
        }

        // should discover all sip applications and initialize them by reading
        // the DD
        applicationManager = new ApplicationManager(
                new HarcodedApplicationStrategy());
        applicationManager.configure(serverContext);
        containerParts.add(applicationManager);

        flowManager = new FlowManager();
        // Initialize the flowmanager
        flowManager.configure(serverContext);
        flowManager.setApplicationManager(applicationManager);
        containerParts.add(flowManager);

        // create the netstack and add listening points
        // The manager interfaces with an SIP-JAIN compatible stack
        try {
            networkManager = new SipNetworkManager(serverContext);
        } catch (ConfigurationErrorException e) {
            e.printStackTrace();
            log.fatal("Could not configure network: " + e.getMessage());
            System.exit(1);
        }

        // start listening
        try {
            networkManager.start();
        } catch (SipNetworkException e) {
            e.printStackTrace();
            log.fatal("Could not start network: " + e.getMessage());
            shutdown(true);
        }
        containerParts.add(networkManager);

        this.running = true;
        log.info("Container boot completed");
        // furter actions are triggered by messages delivered to the listening
        // points registred by the network manager
    }

    /**
     * Starts the container from the commandline
     *
     * @param args
     */
    public static void main(String[] args) {

        SipServletContainer ssc = SipServletContainer.getInstance();
        log.info("args: " + args.length);
        if (args.length > 0) {
            Map<String, String> configuration = new HashMap<String, String>();
            // get configurations
            Pattern pattern = Pattern.compile("^(.*)=(.*)$");
            for (int i = 0; i < args.length; i++) {
                String arg = args[i];
                Matcher m = pattern.matcher(arg);
                if (m.matches()) {
                    configuration.put(m.group(1), m.group(2));
                }
            }
            ssc.configure(configuration);
            log.info("Booting with configuration" + configuration);
        }

        ssc.run();
    }

    /*
     * (non-Javadoc), required run-method
     *
     * @see java.lang.Runnable#run()
     */
    public synchronized void run() {
        if (this.running == false) {
            this.boot();
        } else {
            log.error("The container is already running!");
        }

    }

    /**
```

SipServletContainer.java


```
     * @return Returns the NetworkManager.
     */
    public SipNetworkManager getNetworkManager() {
        return networkManager;
    }

    /**
     * @return Returns the applicationManager.
     */
    public ApplicationManager getApplicationManager() {
        return applicationManager;
    }

    /**
     * @return Returns the flowManager.
     */
    public FlowManager getFlowManager() {
        return flowManager;
    }

    public void shutdown(boolean... failure) {
        log.info("Shutting down container");
        // shutdown the registered parts

        for (ContainerPart part : containerParts) {
            part.shutdown();
        }

        if (failure.length == 0 || failure[0]) {
            throw new RuntimeException(
                    "Internal container error, please consult the errorlog for explaination");

        } else {
            System.exit(1);
        }
    }

}
```

DialogState.java

Page 1/1

```
package dk.itu.ssc.session;

public class DialogState {

        /*
         * Dialog state
         * - local/remote uris
         * - local/remote tags
         * - local/remote sequence numbers
         * - routeset
         * - remote target URI
         * - Secureflag
         */


    /*
     * A dialog contains certain pieces of state needed for further message
     * transmissions within the dialog. This state consists of the dialog ID, a
     * local sequence number (used to order requests from the UA to its peer), a
     * remote sequence number (used to order requests from its peer to the UA),
     * a local URI, a remote URI, remote target, a boolean flag called "secure",
     * and a route set, which is an ordered list of URIs. The route set is the
     * list of servers that need to be traversed to send a request to the peer.
     * A dialog can also be in the "early" state, which occurs when it is
     * created with a provisional response, and then transition to the
     * "confirmed" state when a 2xx final response arrives. For other responses,
     * or if no response arrives at all on that dialog, the early dialog
     * terminates
     */


}
```

SessionManager.java

Page 1/2

```
package dk.itu.ssc.session;

import java.util.HashMap;
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;

import javax.sip.Dialog;
import javax.sip.header.CallIdHeader;
import javax.sip.header.FromHeader;
import javax.sip.header.Header;
import javax.sip.header.HeaderFactory;
import javax.sip.header.ToHeader;
import javax.sip.message.Message;
import javax.sip.message.Request;

import org.apache.log4j.Logger;

import dk.itu.ssc.SipServletContainer;
import dk.itu.ssc.application.SipServletApplication;
import dk.itu.ssc.application.SipServletApplicationImpl;
import dk.itu.ssc.network.wrapper.AddressImpl;
import dk.itu.ssc.network.wrapper.ApplicationSession;
import dk.itu.ssc.network.wrapper.SipServletRequestImpl;
import dk.itu.ssc.network.wrapper.SipSessionImpl;

public class SessionManager {

    private static final Logger log = Logger.getLogger(SessionManager.class);

    /**
     * Contains a map of all active SipSessions aka "pseudo dialogs", the
     * sessions are indexed by their dialog-id
     */
    private static Map<String, SipSessionImpl> sipSessionMap;

    /**
     * Contains a map of all active application sessions, the map is indexed by
     * the unique id of the appliaction
     */
    private static Map<String, ApplicationSession> appSessionMap;

    static {
        sipSessionMap = new ConcurrentHashMap<String, SipSessionImpl>();
        appSessionMap = new ConcurrentHashMap<String, ApplicationSession>();
    }

    /**
     * Retrives the SipSession for a given message in a given Dialog
     *
     * @param msg
     * @param dialog
     * @param create
     * @return
     */
    public synchronized static SipSessionImpl getSession(Dialog dialog) {
        // get the session
        String dialogId = dialog.getDialogId();
        SipSessionImpl sess = sipSessionMap.get(dialogId);
        if (sess == null) {
            log.debug("Looked for session with associated with dialogID \""
                    + dialogId + "\" but found nothing");
        } else {
            log.debug("Found session  associated with dialogID \"" + dialogId
                    + "\"");
        }
        return sess;

    }

    /**
     * Creates a new SipSession
     *
     * @param msg
     * @param trans
     * @param ssa
     * @return a newly created SipSession
     */
    public synchronized static SipSessionImpl createSession(
            SipServletRequestImpl request, SipServletApplication ssa,
            boolean uas) {
        // get the ApplicationSession

        ApplicationSession appSession = appSessionMap.get(ssa.getId());

        if (appSession == null) {
            log
                    .fatal("Could not locate ApplicationSession for application "
```

SessionManager.java


```
                            + ssa
                            + "with ID: "
                            + ssa.getId()
                            + "An ApplicationSession should have been created when the Application where loaded");
            SipServletContainer.getInstance().shutdown();
        }

        // extract essencial data from the message
        String callId = ((CallIdHeader) request.getJainRequest().getHeader(
                CallIdHeader.NAME)).getCallId();

        // mandatory headers so we know they exists
        AddressImpl toAddr = new AddressImpl(((ToHeader) request
                .getJainRequest().getHeader(ToHeader.NAME)).getAddress());
        ;
        AddressImpl fromAddr = new AddressImpl(((FromHeader) request
                .getJainRequest().getHeader(FromHeader.NAME)).getAddress());
        ;

        SipSessionImpl sess = null;

        // create a new session
        if (uas) {
            // localParty = to, remoteParty = from
            sess = new SipSessionImpl(appSession, callId, toAddr, fromAddr);
        } else {
            // localParty = from, remoteParty = to
            sess = new SipSessionImpl(appSession, callId, fromAddr, toAddr);
        }

        // return the new session
        log.info("Creating a new session with id " + sess.getId());
        return sess;

    }

    public synchronized static ApplicationSession createApplicationSession(
            SipServletApplicationImpl ssa) {
        // create new instance
        ApplicationSession appSession = new ApplicationSession(ssa);
        log.info("Creating new ApplicationSesssion for application " + ssa
                + " with id " + ssa.getId());
        appSessionMap.put(ssa.getId(), appSession);

        return appSession;
    }

    public synchronized static void store(SipSessionImpl session) {
        log.info("Storing session with id " + session.getId());
        sipSessionMap.put(session.getId(), session);
    }

}
```

TestServlet.java


```
package dk.itu.ssc.tests;

import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.sip.SipServlet;
import javax.servlet.sip.SipServletRequest;
import javax.servlet.sip.SipServletResponse;
import javax.servlet.sip.SipSession;
import javax.servlet.sip.SipURI;

import org.apache.log4j.Logger;

/**
 * Servlet used to test the implementation of the container This servlet is not
 * a part of the TCK
 *
 * The servlet will accept an Invite, go though the threeway handshake, and then
 * close the session with a BYE-request This will require the basic
 * Request/Reponse code to work
 *
 * @author Mads Danquah
 *
 */
public class TestServlet extends SipServlet {

    Logger log = Logger.getLogger(TestServlet.class);

    /*
     * 0 = initial 1 = got invite 2 = got ack1 3 = send bye 4 = got ok
     */

    int currentStage = 0;

    int iteration = 0;

    /**
     * Update when the structure of the class changes
     */
    private static final long serialVersionUID = 1L;

    @Override
    protected void doInvite(SipServletRequest req) throws ServletException,
            IOException {
        log.info("INVITE received");
        // store something in the session
        req.getSession().setAttribute("testkey", "testval");
        goToStage(1);
        log.debug(req);
        SipServletResponse res = req.createResponse(200);
        log.debug("Generating reponse: \n" + res);
        res.send();
        goToStage(2);
    }

    private void goToStage(int nextStage) {
        if (nextStage == (currentStage + 1)) {
            currentStage++;
            printStage();
        } else {
            log.fatal("Invalid stage " + nextStage + ", expected "
                    + (currentStage + 1));
        }
    }

    private void printStage() {
        String message = "";
        switch (currentStage) {
        case 1:
            message = "got invite sending ok";
            break;

        case 2:
            message = "sendt ok waiting for ack";
            break;

        case 3:
            message = "got ACK, waiting 2 sec and sending bye";
            break;

        case 4:
            message = "Sendt bye, waiting for ok";
            break;

        case 5:
            message = "Got OK, resetting stage (iteration " + iteration + ")";
            currentStage = 0;
```

TestServlet.java


```
            iteration++;
            break;
        default:
            break;
        }

        log.debug("----------STAGE " + (currentStage) + "--------------"
                + message.toUpperCase());

    }

@Override
    protected void doAck(SipServletRequest req) throws ServletException,
            IOException {
        log.info("ACK Received");
        goToStage(3);
        try {
            Thread.sleep(2000);
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

        log.info("BYE'ing");
        SipSession sess = req.getSession();
        SipServletRequest request = sess.createRequest("BYE");

        // cast the uri to a sip-uri
        SipURI sr = (SipURI) request.getRequestURI();

        sr.setTransportParam("tcp");


        log.info("Prepared request: " + request);
        request.send();
        goToStage(4);
        // check wether the attribute is still containend in the session
        String attrib = (String)req.getSession().getAttribute("testkey");
        if(attrib == null){
            log.error("Could not retrive attribute \"testkey\" from the session");
        }else{
            log.info("Retrived value " + attrib);
            if(!attrib.equals("testval")){
                log.info("should have been \"testval\"");
            }
        }
    }   // REPONSES

    @Override
    protected void doProvisionalResponse(SipServletResponse res)
            throws ServletException, IOException {
        log.info("Provisional reponse (1xx) received");
    }

    @Override
    protected void doSuccessResponse(SipServletResponse res)
            throws ServletException, IOException {
        goToStage(5);
        log.info("Success reponse (2xx) received");
    }

    @Override
    protected void doRedirectResponse(SipServletResponse res)
            throws ServletException, IOException {
        log.info("Redirect response (3xx) received");
    }

    @Override
    protected void doErrorResponse(SipServletResponse res)
            throws ServletException, IOException {
        log.info("Error response (4xx, 5xx, 6xx) received");
    }
}
```

Testcase1.java


```
package dk.itu.ssc.tests;

import java.util.HashMap;
import java.util.Map;

import dk.itu.ssc.SipServletContainer;
import junit.framework.TestCase;

public class Testcase1 extends TestCase {

    private SipServletContainer ssc;

    @Override
    protected void setUp() throws Exception {
        // setup the container
        this.ssc = SipServletContainer.getInstance();
        Map<String, String> configuration = new HashMap<String, String>();
        configuration.put("applicationmanager.hardcoded.dd", "D:\\workspace\\eclipse\\ssc_code\\src\\dk\\itu\\ssc\\tests\\testcase1dd.xml");
        ssc.configure(configuration);
        ssc.run();
    }

    public void testHandshake() {
        try {
            Thread.sleep(2000);
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    @Override
    protected void tearDown() throws Exception {
        ssc.shutdown();
    }

}
```

Testcase2.java


```
package dk.itu.ssc.tests;

import java.util.HashMap;
import java.util.Map;

import dk.itu.ssc.SipServletContainer;
import junit.framework.TestCase;

public class Testcase2 extends TestCase {

    private SipServletContainer ssc;

    @Override
    protected void setUp() throws Exception {
        // setup the container
        this.ssc = SipServletContainer.getInstance();
        Map<String, String> configuration = new HashMap<String, String>();
        configuration.put("applicationmanager.hardcoded.dd", "D:\\workspace\\eclipse\\ssc_code\\src\\dk\\itu\\ssc\\tests\\testcase2dd.xml");
        ssc.configure(configuration);
        ssc.run();
    }

    public void testHandshake() {
        try {
            Thread.sleep(40000);
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    @Override
    protected void tearDown() throws Exception {
        ssc.shutdown();
    }

}
```

Testcase2Servlet.java


```
package dk.itu.ssc.tests;

import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.sip.SipServlet;
import javax.servlet.sip.SipServletRequest;
import javax.servlet.sip.SipServletResponse;
import javax.servlet.sip.SipSession;

import org.apache.log4j.Logger;

/**
 * Simple server that goes through a handshake, and later accepts a bye
 * @author Mads Danquah
 *
 */
public class Testcase2Servlet extends SipServlet {

    Logger log = Logger.getLogger(Testcase2Servlet.class);

    static int count = 0;
    static int completed = 0;

    /*
     * 0 = initial 1 = got invite 2 = got ack1 3 = send bye 4 = got ok
     */

    /**
     * Update when the structure of the class changes
     */
    private static final long serialVersionUID = 1L;

    @Override
    protected void doInvite(SipServletRequest req) throws ServletException,
            IOException {
        count++;
        log.info("Count: " + count);
        log.info("INVITE recived");

        // store the stage in the session
        SipSession session = req.getSession();
        req.getSession().setAttribute("stage", 0);
        goToStage(1, session);

        log.debug(req);
        SipServletResponse res = req.createResponse(200);
        log.debug("Generated reponse: \n" + res);
        res.send();
        goToStage(2, session);
    }

    private void goToStage(Integer next, SipSession session) {
        Integer current = (Integer)session.getAttribute("stage");
        if (next == (current + 1)) {
            current++;
            session.setAttribute("stage", current);
            printStage(current);
        } else {
            log.fatal("============ Invalid stage " + next + ", expected "
                    + (current + 1) + "========================");
            System.exit(1);
        }
    }

    private void printStage(Integer current) {
        String message = "";
        switch (current) {
        case 1:
            message = "got invite sending ok";
            break;

        case 2:
            message = "sendt ok waiting for ack";
            break;

        case 3:
            message = "got ACK";
            break;

        case 4:
            message = "Got bye confirming";
            break;

        case 5:
            message = "Bye confirmed";
            completed++;
```

Testcase2Servlet.java


```
            message += " - " + completed + " completed";
            break;
        default:
            break;
        }

        log.debug("------------STAGE " + (current) + "---------------"
                + message.toUpperCase());

    }

    @Override
    protected void doAck(SipServletRequest req) throws ServletException,
            IOException {
        log.info("ACK Recived");
        goToStage(3, req.getSession());
    } // REPONSES

    @Override
    protected void doBye(SipServletRequest req) throws ServletException,
            IOException {
        goToStage(4, req.getSession());
        SipServletResponse res = req.createResponse(200);
        res.send();
        goToStage(5, req.getSession());
    }

    @Override
    protected void doProvisionalResponse(SipServletResponse res)
            throws ServletException, IOException {
        log.info("Provisional reponse (1xx) recived");
    }

    @Override
    protected void doSuccessResponse(SipServletResponse res)
            throws ServletException, IOException {
        goToStage(5, res.getSession());
        log.info("Success reponse (2xx) recived");
    }

    @Override
    protected void doRedirectResponse(SipServletResponse res)
            throws ServletException, IOException {
        log.info("Redirect response (3xx) recived");
    }

    @Override
    protected void doErrorResponse(SipServletResponse res)
            throws ServletException, IOException {
        log.info("Error response (4xx, 5xx, 6xx) recived");
    }
}
```

Testcase3Servlet.java


```
package dk.itu.ssc.tests;

import javax.servlet.sip.SipServlet;

/**
 * Dummy servlet
 *
 * @author Mads Danquah
 *
 */
public class Testcase3Servlet extends SipServlet {

}
```

testcase1dd.xml

Page 1/1

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE sip-app
   PUBLIC "-//Java Community Process//DTD SIP Application 1.0//EN"
   "http://www.jcp.org/dtd/sip-app_1_0.dtd">

<sip-app>
  <display-name>DisplayName</display-name>
</sip-app>
```

testcase2dd.xml

Page 1/1

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE sip-app
   PUBLIC "-//Java Community Process//DTD SIP Application 1.0//EN"
   "http://www.jcp.org/dtd/sip-app_1_0.dtd">

<sip-app>
  <display-name>DisplayName</display-name>

  <servlet>
    <servlet-name>TestServlet</servlet-name>
    <servlet-class>dk.itu.ssc.tests.Testcase2Servlet</servlet-class>
  </servlet>
</sip-app>
```

testcase3dd.xml

Page 1/1

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE sip-app
   PUBLIC "-//Java Community Process//DTD SIP Application 1.0//EN"
   "http://www.jcp.org/dtd/sip-app_1_0.dtd">


<sip-app>
  <display-name>Deployment Descriptor test</display-name>

  <context-param>
    <param-name>pet</param-name>
    <param-value>cat</param-value>
  </context-param>

  <listener>
    <listener-class>not.supported</listener-class>
  </listener>

  <servlet>
    <servlet-name>TestServlet</servlet-name>
    <servlet-class>dk.itu.ssc.tests.Testcase3Servlet</servlet-class>
     <init-param>
         <param-name>key1</param-name>
         <param-value>value1</param-value>
     </init-param>
     <init-param>

         <param-name>key2</param-name>
         <param-value>value2</param-value>
    </init-param>
  </servlet>

  <servlet-mapping>
    <servlet-name>TestServlet</servlet-name>
    <pattern>
      <and>
        <equal>
          <var>request.method</var>
          <value>INVITE</value>
        </equal>
        <equal>
          <var>request.uri.user</var>
          <value>uas-passive</value>
        </equal>
      </and>
    </pattern>
  </servlet-mapping>

  <session-config>
    <session-timeout>1</session-timeout>
  </session-config>
</sip-app>
```

EnumAdaptor.java

Page 1/1

```
package dk.itu.ssc.util;

import java.util.Enumeration;
import java.util.Iterator;

/**
 * Adapts an Iterator to implement Enumeration
 *
 * @author Mads Danquah
 *
 * @param <T>
 */
public class EnumAdaptor<T> implements Enumeration<T> {

    Iterator<T> iterator;

    public EnumAdaptor(Iterator<T> i) {
        iterator = i;
    }

    public boolean hasMoreElements() {
        return iterator.hasNext();
    }

    public T nextElement() {
        return iterator.next();
    }
}
```

IDGenerator.java


```java
package dk.itu.ssc.util;

import java.util.Random;

public class IDGenerator {
    private static Random random = new Random();

    public static String generate() {
        return Integer.toString(Math.abs(Math.abs(random.nextInt())),
                Character.MAX_RADIX);
    }

}
```

DefaultServlet.java


```java
package dk.itu.ssc.servlet;

import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.sip.SipServlet;
import javax.servlet.sip.SipServletRequest;
import javax.servlet.sip.SipServletResponse;

import org.apache.log4j.Logger;

public class DefaultServlet extends SipServlet {

    Logger log = Logger.getLogger(DefaultServlet.class);

    /**
     * Update when the structure of the class changes
     */
    private static final long serialVersionUID = 1L;

    @Override
    protected void doRequest(SipServletRequest req) throws ServletException,
            IOException {
        log.info("Default servlet invoked");
        if (!(req.getMethod().equals("CANCEL") && req.getMethod().equals("ACK"))) {
            // do nothing
        } else {
            // respond with a 5xx
            SipServletResponse res = req.createResponse(500,
                    "Unknown application");
            res.send();
        }

        super.doRequest(req);
    }

    @Override
    protected void doResponse(SipServletResponse res) throws ServletException,
            IOException {
        // TODO Auto-generated method stub
        log.info("Defautl servlet invoked");
        super.doResponse(res);
    }
}
```

ServletConfigImpl.java


```
package dk.itu.ssc.servlet;

import java.util.Enumeration;
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;

import javax.servlet.ServletConfig;
import javax.servlet.ServletContext;

import dk.itu.ssc.util.EnumAdaptor;

public class ServletConfigImpl implements ServletConfig {

    ServletContext servletContext;

    String servletName;

    Map<String, String> initParameters;

    public ServletConfigImpl(ServletContext context) {
        initParameters = new ConcurrentHashMap<String, String>();
        servletContext = context;
    }

    public String getInitParameter(String key) {
        return initParameters.get(key);
    }

    public Enumeration getInitParameterNames() {
        return new EnumAdaptor<String>(initParameters.keySet().iterator());
    }

    /**
     * @return Returns the servletContext.
     */
    public ServletContext getServletContext() {
        return servletContext;
    }

    /**
     * @return Returns the servletName.
     */
    public String getServletName() {
        return servletName;
    }

    /**
     * @param servletName
     *            The servletName to set.
     */
    public void setServletName(String servletName) {
        this.servletName = servletName;
    }

}
```

ServletContextImpl.java


```
package dk.itu.ssc.servlet;

import java.io.InputStream;
import java.net.MalformedURLException;
import java.net.URL;
import java.util.Enumeration;
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;

import javax.servlet.RequestDispatcher;
import javax.servlet.Servlet;
import javax.servlet.ServletContext;
import javax.servlet.ServletException;

import org.apache.log4j.Logger;

import dk.itu.ssc.util.EnumAdaptor;

public class ServletContextImpl implements ServletContext {

    private Logger log = null;

    private Map<String, Object> attributes;

    public ServletContextImpl(String servletName) {
        this.log = Logger.getLogger(servletName);
        attributes = new ConcurrentHashMap<String, Object>();
    }

    public Object getAttribute(String key) {
        return attributes.get(key);
    }

    public Enumeration getAttributeNames() {
        return new EnumAdaptor<String>(attributes.keySet().iterator());
    }

    public ServletContext getContext(String arg0) {
        // have no meaning for SipServlets
        return null;
    }

    public String getInitParameter(String arg0) {
        // TODO Auto-generated method stub
        return null;
    }

    public Enumeration getInitParameterNames() {
        // TODO Auto-generated method stub
        return null;
    }

    public int getMajorVersion() {
        // TODO Auto-generated method stub
        return 0;
    }

    public String getMimeType(String arg0) {
        // TODO Auto-generated method stub
        return null;
    }

    public int getMinorVersion() {
        // TODO Auto-generated method stub
        return 0;
    }

    public RequestDispatcher getNamedDispatcher(String arg0) {
        // TODO Auto-generated method stub
        return null;
    }

    public String getRealPath(String arg0) {
        // have no meaning for SipServlets
        return null;
    }

    public RequestDispatcher getRequestDispatcher(String arg0) {
        // have no meaning for SipServlets
        return null;
    }

    public URL getResource(String arg0) throws MalformedURLException {
        // TODO Auto-generated method stub
        return null;
    }
```

ServletContextImpl.java


```
    public InputStream getResourceAsStream(String name) {
        // TODO Auto-generated method stub
        return null;
    }

    public String getServerInfo() {
        // TODO Auto-generated method stub
        return null;
    }

    public Servlet getServlet(String arg0) throws ServletException {
        // TODO Auto-generated method stub
        return null;
    }

    public Enumeration getServletNames() {
        // TODO Auto-generated method stub
        return null;
    }

    public Enumeration getServlets() {
        // TODO Auto-generated method stub
        return null;
    }

    public void log(Exception e, String msg) {
        log.error(msg, e);

    }

    public void log(String msg) {
        log.info(msg);

    }

    public void log(String msg, Throwable t) {
        log.error(msg, t);

    }

    public void removeAttribute(String key) {
        attributes.remove(key);

    }

    public void setAttribute(String key, Object val) {
        attributes.put(key, val);

    }

}
```

ListeningPointAddress.java


```
package dk.itu.ssc.network;

import java.net.InetSocketAddress;

/**
 * Used to hold listeningpoints
 *
 * @author Mads Danquah
 */
public class ListeningPointAddress {
    protected String protocol;

    protected InetSocketAddress socketAddress;

    public ListeningPointAddress(String proto, InetSocketAddress isa) {
        this.protocol = proto;
        this.socketAddress = isa;
    }

    public String getProtocol() {
        return protocol;
    }

    public void setProtocol(String protocol) {
        this.protocol = protocol;
    }

    public InetSocketAddress getSocketAddress() {
        return socketAddress;
    }

    public void setSocketAddress(InetSocketAddress socketAddress) {
        this.socketAddress = socketAddress;
    }

    @Override
    public String toString() {
        return this.socketAddress + "/" + this.protocol;
    }
}
```

SipNetworkManager.java Page 1/5

```
package dk.itu.ssc.network;

import java.util.ArrayList;
import java.util.Properties;
import java.util.TooManyListenersException;

import javax.sip.ClientTransaction;
import javax.sip.Dialog;
import javax.sip.InvalidArgumentException;
import javax.sip.ListeningPoint;
import javax.sip.ObjectInUseException;
import javax.sip.PeerUnavailableException;
import javax.sip.RequestEvent;
import javax.sip.ResponseEvent;
import javax.sip.ServerTransaction;
import javax.sip.SipException;
import javax.sip.SipFactory;
import javax.sip.SipListener;
import javax.sip.SipProvider;
import javax.sip.SipStack;
import javax.sip.Transaction;
import javax.sip.TransactionAlreadyExistsException;
import javax.sip.TransactionUnavailableException;
import javax.sip.TransportNotSupportedException;
import javax.sip.message.Request;
import javax.sip.message.Response;

import org.apache.log4j.Logger;

import sun.net.spi.DefaultProxySelector;

import dk.itu.ssc.ContainerPart;
import dk.itu.ssc.ServerContext;
import dk.itu.ssc.SipServletContainer;
import dk.itu.ssc.exceptions.ConfigurationErrorException;
import dk.itu.ssc.exceptions.SipNetworkException;
import dk.itu.ssc.network.wrapper.SipServletRequestImpl;
import dk.itu.ssc.network.wrapper.SipServletResponseImpl;
import dk.itu.ssc.network.wrapper.SipSessionImpl;

public class SipNetworkManager implements SipListener, ContainerPart {

    static private Logger log = Logger.getLogger(SipNetworkManager.class);

    private SipStack sipStack;

    private ArrayList<ListeningPointAddress> listeningPointsAddresses;

    private ArrayList<ListeningPoint> listeningsPoints;

    ServerContext sc;

    private SipProvider defaultProvider;

    public SipNetworkManager(ServerContext sc)
            throws ConfigurationErrorException {
        this.sc = sc;

        this.listeningsPoints = new ArrayList<ListeningPoint>();

        // extract listening points
        listeningPointsAddresses = sc.getListeningPoints();

        // get address (needed for the inital stack setup)
        String address = "";
        if (listeningPointsAddresses.isEmpty()) {
            throw new ConfigurationErrorException("No listeningpoints defined");
        } else {
            // We'll only use the first address for now
            // multiple addresses would require multiple sip-stacks

            address = listeningPointsAddresses.get(0).getSocketAddress()
                    .getAddress().getHostAddress();
        }

        log.debug("Initializing netstack with address " + address);
        // Prepare the netstack
        SipFactory sipFactory = null;
        sipStack = null;
        sipFactory = SipFactory.getInstance();

        // should be wrapped in some handler in order to allow for alternative
        // implementations. Maybe simply a class with an static initialiser that
        // sat up the property instance?
        sipFactory.setPathName("gov.nist");

        // create the stack
```

SipNetworkManager.java Page 2/5

```
        try {
            sipStack = sipFactory.createSipStack(getConfiguration(address));
            log.debug("sipStack = " + sipStack);
        } catch (PeerUnavailableException e) {
            // could not find
            // gov.nist.jain.protocol.ip.sip.SipStackImpl
            // in the classpath
            e.printStackTrace();
            log.error("Could not create SipStack", e);
            if (e.getCause() != null)
                e.getCause().printStackTrace();
            System.exit(0);
        }
    }

    /**
     * Builds the configuration for the SipStack
     *
     * @param address
     *            The address this SipStack should attach to
     * @return
     */
    private Properties getConfiguration(String address) {

        // hardcoded for now but could be retrived from an external source
        Properties properties = new Properties();
        properties.setProperty("javax.sip.IP_ADDRESS", address);
        properties.setProperty("javax.sip.RETRANSMISSION_FILTER", "true");
        properties.setProperty("javax.sip.STACK_NAME", "SSC");

        // You need 16 for logging traces. 32 for debug + traces.
        // Your code will limp at 32 but it is best for debugging.
        properties.setProperty("gov.nist.javax.sip.TRACE_LEVEL", "32");
        properties.setProperty("gov.nist.javax.sip.DEBUG_LOG", "ssc_debug.txt");
        properties.setProperty("gov.nist.javax.sip.SERVER_LOG", "ssc_log.txt");
        // Guard against starvation.
        properties.setProperty("gov.nist.javax.sip.READ_TIMEOUT", "1000");

        properties.setProperty("gov.nist.javax.sip.CACHE_SERVER_CONNECTIONS",
                "false");

        return properties;
    }

    /**
     * Start listning for incoming messages
     *
     * @throws SipNetworkException
     */
    public void start() throws SipNetworkException {
        addListeningPoints();
        // caclulate contact

    }

    /**
     * Shuts down the network-manager
     */
    public void shutdown() {

        for (ListeningPoint point : this.listeningsPoints) {
            try {
                sipStack.deleteListeningPoint(point);
            } catch (ObjectInUseException e) {
                log.error("Could not delete listeningpoint " + point, e);
            }
        }

        log.info("shutdown complete");
    }

    /**
     * Adds a listningpoint.
     *
     * The listeningpoint will not become active before start() is called
     *
     * @throws SipNetworkException
     */
    public void addListeningPoints() throws SipNetworkException {

        ListeningPoint lp;

        for (ListeningPointAddress lpa : listeningPointsAddresses) {

            try {
                // extract the address and create the listeningspoint
                lp = sipStack.createListeningPoint(lpa.getSocketAddress()
```

```
                    .getPort(), lpa.getProtocol());
            log.debug("Created listningpoint: " + lp + " for address "
                    + lpa);
            // for later removal in shutdown()
            this.listeningsPoints.add(lp);

            // add the listeningpoint
            SipProvider sipProvider = sipStack.createSipProvider(lp);

            log.debug("provider: " + sipProvider);

            // first listeningpoint is the default
            if (this.defaultProvider == null)
                this.defaultProvider = sipProvider;

            // this.sipProvideres.add(sipProvider);
            log.debug("Adding Listeningpoint: " + lpa + " provider "
                    + sipProvider);
            sipProvider.addSipListener(this);
        } catch (TransportNotSupportedException e) {
            throw new SipNetworkException("Unsupported transport", e);
        } catch (InvalidArgumentException e) {
            throw new SipNetworkException("Invalid argument", e);
        } catch (TooManyListenersException e) {
            throw new SipNetworkException("Could not add listener", e);
        } catch (ObjectInUseException e) {
            e.printStackTrace();
            throw new SipNetworkException(e);
        }

    }

}

// ******************
// CALL-BACK METHODS
// ******************

/*
 * (non-Javadoc)
 *
 * @see javax.sip.SipListener#processRequest(javax.sip.RequestEvent)
 */
public void processRequest(RequestEvent requestEvent) {
    // extract request data
    Request request = requestEvent.getRequest();
    ServerTransaction serverTransaction = requestEvent
            .getServerTransaction();
    SipProvider sipProvider = (SipProvider) requestEvent.getSource();

    Dialog dialog = null;

    // JAIN or NIST specific?
    SipProvider provider = (SipProvider) requestEvent.getSource();
    log.debug("Request with metod \"" + request.getMethod()
            + "\" received at " + sipStack.getStackName()
            + " with server transaction id " + serverTransaction);

    if (serverTransaction != null && serverTransaction.getDialog() != null) {
        dialog = serverTransaction.getDialog();
        log.debug("dialog id:" + dialog.getDialogId());
    } else {
        log.debug("No related dialog");
    }
    log.debug("\n request:\n" + request);

    // create a new server-transaction for dialog-creating requests (INVITE
    // for now)
    if (serverTransaction == null) {
        if (request.getMethod().equals(Request.INVITE)) {
            try {
                serverTransaction = sipProvider
                        .getNewServerTransaction(request);
                dialog = serverTransaction.getDialog();
            } catch (TransactionAlreadyExistsException e) {
                log
                        .fatal(
                                "Attempted to create a server-transaction when one already existed:",
                                e);
                // until we get cleaner errorHandleing
                SipServletContainer.getInstance().shutdown();
            } catch (TransactionUnavailableException e) {
                log.fatal("Could not create ServerTransaction:", e);
                // until we get cleaner errorHandleing
                SipServletContainer.getInstance().shutdown();
            }
        } else {
```



```
            log
                    .info("Missing ServerTransaction for non-dialog creating request, probably unimportant");
            try {
                log.info("Attempting to create");
                serverTransaction = sipProvider
                        .getNewServerTransaction(request);
                log.info("success:" + serverTransaction);
            } catch (TransactionAlreadyExistsException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            } catch (TransactionUnavailableException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
    }

    // instantiate new ServletSipRequest and pass it on to the FlowManager
    SipServletRequestImpl req = new SipServletRequestImpl(request,
            serverTransaction);

    // information about there the request where recived
    req.setLocalAddress(sipStack.getIPAddress());
    req.setTransport(provider.getListeningPoint().getTransport());
    req.setPort(provider.getListeningPoint().getPort());
    req.setSipProvider(provider);

    // store the dialog (could be null)
    req.setDialog(dialog);

    SipServletContainer.getInstance().getFlowManager().processRequest(req);
}

/*
 * (non-Javadoc)
 *
 * @see javax.sip.SipListener#processResponse(javax.sip.ResponseEvent)
 */
public void processResponse(ResponseEvent responseReceivedEvent) {
    log.debug("Got a response");
    Response response = (Response) responseReceivedEvent.getResponse();
    ClientTransaction clientTransaction = responseReceivedEvent
            .getClientTransaction();

    log.debug("Response with code:" + response.getStatusCode()
            + " received with client transaction id: " + clientTransaction
            + " and dialog id:"
            + clientTransaction.getDialog().getDialogId() + ":\n"
            + response);

    // instantiate new SipServletRequestImpl and pass it on to the
    // FlowManager
    SipServletResponseImpl resWrapper = new SipServletResponseImpl(
            response, clientTransaction);

    SipServletContainer.getInstance().getFlowManager().processResponse(
            resWrapper);

}

/*
 * (non-Javadoc)
 *
 * @see javax.sip.SipListener#processTimeout(javax.sip.TimeoutEvent)
 */
public void processTimeout(javax.sip.TimeoutEvent timeoutEvent) {
    Transaction transaction;
    if (timeoutEvent.isServerTransaction()) {
        transaction = timeoutEvent.getServerTransaction();
    } else {
        transaction = timeoutEvent.getClientTransaction();
    }
    log.debug("state = " + transaction.getState());
    log.debug("dialog = " + transaction.getDialog());
    log.debug("dialogState = " + transaction.getDialog().getState());
    log.debug("Transaction Time out");
}

/**
 * @return Returns the defaultProvider.
 */
public SipProvider getDefaultProvider() {
    return defaultProvider;
}

/**
 * Used by SipResponseImpl to send responses
```

SipNetworkManager.java


```
     *
     * @param response
     * @param impl
     */
    public void sendResponse(SipServletResponseImpl response) {
        // attempt to get the server transaction
        ServerTransaction st = response.getServerTransaction();

        try {

            if (st != null) {
                // send using the transaction
                st.sendResponse(response.getJainReponse());

                // Examine wether this will create a dialog
                if (st.getRequest().getMethod().equals(Request.INVITE)) {

                    // if so, make sure to store the session as we now can
                    // create a
                    // complete dialogid with both local and remote tags
                    response.getSession().setDialog(
                            response.getServerTransaction().getDialog());
                    response.getSession().store();
                }
            } else {
                // send stateless
                defaultProvider.sendResponse(response.getJainReponse());

            }

        } catch (SipException e) {
            log.fatal("Could not send response " + response);
            SipServletContainer.getInstance().shutdown(true);
        }
    }

    public void configure(ServerContext config) {
        log.info("NetworkManager configured");
    }
}
```

AddressImpl.java


```
package dk.itu.ssc.network.wrapper;

import java.text.ParseException;
import java.util.Iterator;

import javax.servlet.sip.Address;
import javax.servlet.sip.URI;
import javax.sip.PeerUnavailableException;
import javax.sip.SipFactory;
import javax.sip.address.SipURI;
import javax.sip.address.TelURL;

import org.apache.log4j.Logger;

/**
 * Done
 *
 * @version 0.1
 * @author Mads Danquah
 */
public class AddressImpl implements Address {

    private static final Logger log = Logger.getLogger(AddressImpl.class);

    protected javax.sip.address.Address address;

    public AddressImpl(javax.sip.address.Address address) {
        this.address = address;
    }

    public String getDisplayName() {

        return this.address.getDisplayName();
    }

    public void setDisplayName(String name) {
        try {
            this.address.setDisplayName(name);
        } catch (ParseException e) {
            log.debug("Refusing to set invalid displayname: " + name);
        }
    }

    public URI getURI() {
        javax.sip.address.URI uri = address.getURI();
        if (uri instanceof SipURI) {
            System.out.println(((SipURI)uri).getTransportParam());
            return new SipURIImpl((SipURI) uri);
        } else if (uri instanceof TelURL) {
            return new TelURLImpl((TelURL) uri);
        } else {
            log.fatal("Uknown Address type??");
            return null;
        }
    }

    public void setURI(URI uri) {
        // update internal jain-sip uri instance
        address.setURI((javax.sip.address.URI) new URIImpl(uri));
    }

    public String getParameter(String key) {
        javax.sip.address.URI uri = address.getURI();
        if (uri instanceof SipURI) {
            return ((SipURI) uri).getParameter(key);
        } else if (uri instanceof TelURL) {
            return ((TelURL) uri).getParameter(key);
        } else {
            log.fatal("Uknown Address type??");
            return null;
        }
    }

    public void setParameter(String key, String value) {
        javax.sip.address.URI uri = address.getURI();
        try {
            if (uri instanceof SipURI) {
                ((SipURI) uri).setParameter(key, value);
            } else if (uri instanceof TelURL) {
                ((TelURL) uri).setParameter(key, value);
            } else {
                log.fatal("Uknown Address type??");
            }
        } catch (ParseException e) {
            log.fatal("Could not parse parameter:", e);
        }
    }
```

```
    public void removeParameter(String key) {
        javax.sip.address.URI uri = address.getURI();
        if (uri instanceof SipURI) {
            ((SipURI) uri).removeParameter(key);
        } else if (uri instanceof TelURL) {
            ((TelURL) uri).removeParameter(key);
        } else {
            log.fatal("Uknown Address type??");
        }
    }

    public Iterator getParameterNames() {
        javax.sip.address.URI uri = address.getURI();
        if (uri instanceof SipURI) {
            return ((SipURI) uri).getParameterNames();
        } else if (uri instanceof TelURL) {
            return ((TelURL) uri).getParameterNames();
        } else {
            log.fatal("Uknown Address type??");
            return null;
        }
    }

    public boolean isWildcard() {
        return address.isWildcard();
    }

    public float getQ() {
        String q = getParameter("q");
        if (q != null) {
            return Float.parseFloat(q);
        }
        return -1.0F;
    }

    public void setQ(float q) {
        if (q == -1.0F) {
            removeParameter("q");
        } else {
            setParameter("q", String.valueOf(q));
        }
    }

    public int getExpires() {
        String expire = getParameter("expires");
        if (expire != null) {
            return Integer.parseInt(expire);
        }
        return -1;
    }

    public void setExpires(int seconds) {
        if (seconds < 0) {
            removeParameter("expires");
        } else {
            setParameter("expires", Integer.toString(seconds));
        }
    }

    /*
     * Deep copy the address instance
     *
     * @see java.lang.Object#clone()
     */
    public Object clone() {
        AddressImpl clone = null;
        try {
            clone = (AddressImpl) super.clone();
            // copy internal address instance
            javax.sip.address.Address addressClone = SipFactory.getInstance()
                    .createAddressFactory().createAddress(
                            address.getDisplayName(),
                            (javax.sip.address.URI) address.getURI().clone());

            clone.address = addressClone;

        } catch (CloneNotSupportedException e) {
            log.fatal("Could not clone address", e);

        } catch (PeerUnavailableException e) {
            // guess this should never happen as the address instance should be
            // valid
            log.fatal("Invalid address, could not clone", e);

        } catch (ParseException e) {
            // guess this should never happen as the address instance should be
```



```
            // valid
            log.fatal("Invalid address, could not clone", e);
        }

        return clone;
    }
}
```

ApplicationSession.java Page 1/2

```
package dk.itu.ssc.network.wrapper;

import java.util.ArrayList;
import java.util.Collection;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;

import javax.servlet.sip.SipApplicationSession;
import javax.servlet.sip.SipURI;
import javax.servlet.sip.TimerListener;
import javax.servlet.sip.URI;

import dk.itu.ssc.application.SipServletApplicationImpl;
import dk.itu.ssc.util.IDGenerator;

public class ApplicationSession implements SipApplicationSession {

    private long creationTime;

    private String id = IDGenerator.generate();

    private int expires;

    private List<SipSessionImpl> sipSessions;

    private Map<String, Object> attributes;

    private long lastAccessedTime;

    private SipServletApplicationImpl app;

    public ApplicationSession(SipServletApplicationImpl app) {
        this.app = app;
        creationTime = System.currentTimeMillis();
        sipSessions = new ArrayList<SipSessionImpl>();
        attributes = new ConcurrentHashMap<String, Object>();
        this.timers = new LinkedList<ServletTimerImpl>();
    }

    public long getCreationTime() {
        return creationTime;
    }

    public long getLastAccessedTime() {
        return lastAccessedTime;
    }

    public void setLastAccessedTime(long time){
        lastAccessedTime = time;
    }

    public String getId() {
        return id;
    }

    public int setExpires(int expires) {
        // could inforce some kind of policy here
        this.expires = expires;
        return expires;
    }

    public int getExpires(){
        return expires;
    }

    public void invalidate() {
        // TODO Auto-generated method stub
    }

    public Iterator getSessions() {
        return sipSessions.iterator();
    }

    public Iterator getSessions(String protocol) {
        if (protocol.equals("SIP")) {
            return sipSessions.iterator();
        } else {
            // return empty iterator
            return new ArrayList(0).iterator();
        }
    }

    public void encodeURI(URI uri) {
        if (!uri.isSipURI()) {
```

ApplicationSession.java Page 2/2

```
            throw new IllegalArgumentException("non-SipURI: " + uri);
        }
        ((SipURI) uri).setParameter("appsessionid", getId());
    }

    public Object getAttribute(String key) {
        return attributes.get(key);
    }

    public Iterator getAttributeNames() {
        return attributes.keySet().iterator();
    }

    public void setAttribute(String key, Object value) {
        attributes.put(key, value);
    }

    public void removeAttribute(String key) {
        attributes.remove(key);

    }

    List<ServletTimerImpl> timers;

    public Collection getTimers() {
        throw new RuntimeException("Missing implementation");
    }

    public void addTimer(ServletTimerImpl timer) {
        timers.add(timer);

    }

    public void removeTimer(ServletTimerImpl timer) {
        timers.remove(timer);

    }

    public void timeout(ServletTimerImpl timer) {
         // missing implementation for now
    }

}
```

JainAdaptorUtil.java


```java
package dk.itu.ssc.network.wrapper;

import gov.nist.javax.sip.header.Contact;

import java.util.HashMap;
import java.util.Map;

import javax.sip.header.CSeqHeader;
import javax.sip.header.CallIdHeader;
import javax.sip.header.FromHeader;
import javax.sip.header.Header;
import javax.sip.header.RecordRouteHeader;
import javax.sip.header.RouteHeader;
import javax.sip.header.ToHeader;
import javax.sip.header.ViaHeader;

import org.apache.log4j.Logger;

public class JainAdaptorUtil {

    static Logger log = Logger.getLogger(JainAdaptorUtil.class);
    protected static Map<String, String> systemHeaders;

    static {
        // Static initializor, setup data
        systemHeaders = new HashMap<String, String>();

        // The check for systemheaders is mostly done case-insensitivly
        systemHeaders.put(CallIdHeader.NAME.toLowerCase(), CallIdHeader.NAME);
        systemHeaders.put(FromHeader.NAME.toLowerCase(), FromHeader.NAME);
        systemHeaders.put(ToHeader.NAME.toLowerCase(), ToHeader.NAME);
        systemHeaders.put(CSeqHeader.NAME.toLowerCase(), CSeqHeader.NAME);
        systemHeaders.put(ViaHeader.NAME.toLowerCase(), ViaHeader.NAME);
        systemHeaders.put(RecordRouteHeader.NAME.toLowerCase(),
                RecordRouteHeader.NAME);
        systemHeaders.put(RouteHeader.NAME.toLowerCase(), RouteHeader.NAME);
    }

    /**
     * Verifies wether an Header instance represents a system header
     *
     * @param header
     * @return true if the instance is a system header
     */
    public static boolean isSystemHeader(Header header, boolean contactOk) {
        return isSystemHeader(header.getName(), contactOk);
    }

    /**
     * Verifies wether a String matches the name of a system header (case
     * insensitive)
     *
     * @param name
     * @return true if the String matches
     */
    public static boolean isSystemHeader(String name, boolean contactOk) {
        log.info("Testing header " + name + " contactOk=" + contactOk);
        return systemHeaders.containsKey(name.toLowerCase())
                || (!contactOk && Contact.NAME.toLowerCase().equals(name.toLowerCase()));
    }
}
```

ServletTimerImpl.java


```java
package dk.itu.ssc.network.wrapper;

import java.io.Serializable;
import java.util.Timer;
import java.util.TimerTask;

import javax.servlet.sip.ServletTimer;
import javax.servlet.sip.SipApplicationSession;


public class ServletTimerImpl implements ServletTimer {

    private static final Timer timer = new Timer(true);

    ApplicationSession applicationSession = null;

    Serializable info;

    private long delay;

    private boolean isPersistent;

    private boolean isRepeating;

    private dk.itu.ssc.network.wrapper.ServletTimerImpl.ExpiryTask expiryTask;

    class ExpiryTask extends TimerTask {
        public void run() {
            timeout();
        }
    }

    public ServletTimerImpl(SipApplicationSession appSession, long delay,
            boolean isPersistent, Serializable info) {
        this.applicationSession = (ApplicationSession)appSession;
        this.delay = delay;
        this.isPersistent = isPersistent;
        this.info = info;
        this.isRepeating = false;
    }

    public ServletTimerImpl(SipApplicationSession appSession, long delay,
            long period, boolean fixedDelay, boolean isPersistent,
            Serializable info) {
        this.applicationSession = (ApplicationSession)appSession;
        this.delay = delay;
        this.isPersistent = isPersistent;
        this.info = info;
        this.isRepeating = true;
        this.expiryTask = new ExpiryTask();

        this.applicationSession.addTimer(this);
        if (fixedDelay) {
            timer.schedule(expiryTask, delay, period);
        } else {
            timer.scheduleAtFixedRate(expiryTask, delay, period);
        }
    }

    /**
     * @param applicationSession
     *            The applicationSession to set.
     */
    public void setApplicationSession(SipApplicationSession applicationSession) {
        this.applicationSession = (ApplicationSession)applicationSession;
    }

    /**
     * @param info
     *            The info to set.
     */
    public void setInfo(Serializable info) {
        this.info = info;
    }

    public SipApplicationSession getApplicationSession() {
        // TODO Auto-generated method stub
        return null;
    }

    public Serializable getInfo() {
        // TODO Auto-generated method stub
        return null;
    }

    public long scheduledExecutionTime() {
        return expiryTask.scheduledExecutionTime();
```

ServletTimerImpl.java


```
    }

    public void cancel() {
        expiryTask.cancel();
        applicationSession.removeTimer(this);
    }

    /**
     * The timer expired -- notify the application and remove the timer from the
     * appsession unless it's a repeating timer.
     */
    private void timeout() {
        if (!isRepeating) {
            applicationSession.removeTimer(this);
        }
        applicationSession.timeout(this);
    }
}
```

SipFactoryImpl.java


```
package dk.itu.ssc.network.wrapper;

import java.text.ParseException;

import javax.servlet.sip.Address;
import javax.servlet.sip.ServletParseException;
import javax.servlet.sip.SipApplicationSession;
import javax.servlet.sip.SipFactory;
import javax.servlet.sip.SipServletRequest;
import javax.servlet.sip.SipURI;
import javax.servlet.sip.URI;
import javax.sip.PeerUnavailableException;
import javax.sip.message.Request;

import org.apache.log4j.Logger;

public class SipFactoryImpl implements SipFactory {

    Logger log = Logger.getLogger(SipFactoryImpl.class);

    javax.sip.SipFactory sipFactory;

    static private SipFactoryImpl instance = null;

    public static SipFactoryImpl getInstance(){
        if(instance == null){
            instance = new SipFactoryImpl();
        }

        return instance;
    }

    private SipFactoryImpl() {
        sipFactory = javax.sip.SipFactory.getInstance();
    }

    public URI createURI(String uri) throws ServletParseException {
        // create the JAIN-uri
        javax.sip.address.URI jainURI = null;
        try {
            jainURI = sipFactory.createAddressFactory().createURI(uri);
        } catch (PeerUnavailableException e) {
            log.error("Could not create URI", e);
            // the best we can do
            throw new ServletParseException("Could not create URI");
        } catch (ParseException e) {
            log.error("Could not create URI", e);
            // the best we can do
            throw new ServletParseException("Could not create URI");
        }

        // wrap URI
        URI servletURI = new URIImpl(jainURI);
        return servletURI;
    }

    public SipURI createSipURI(String user, String host) {
        javax.sip.address.SipURI jainSipURI = null;

        try {
            jainSipURI = sipFactory.createAddressFactory().createSipURI(user,
                    host);
        } catch (PeerUnavailableException e) {
            log.error("Could not create URI", e);
        } catch (ParseException e) {
            log.error("Could not create URI", e);
        }

        // the best we can do when we're not allowed to thrown an Exception
        if (jainSipURI != null) {
            return new SipURIImpl(jainSipURI);
        } else {
            return null;
        }
    }

    public Address createAddress(String address) throws ServletParseException {
        javax.sip.address.Address jainAddress = null;
        try {
            jainAddress = sipFactory.createAddressFactory().createAddress(
                    address);
        } catch (PeerUnavailableException e) {
            log.error("Could not create URI", e);
            // the best we can do
            throw new ServletParseException("Could not create Address");
        } catch (ParseException e) {
            log.error("Could not create URI", e);
```

SipFactoryImpl.java


```
            // the best we can do
            throw new ServletParseException("Could not create Address");
        }

        if (jainAddress != null) {
            return new AddressImpl(jainAddress);
        } else {
            return null;
        }
    }

    public Address createAddress(URI uri) {
        javax.sip.address.Address jainAddress = null;
        // blind cast
        javax.sip.address.URI jainURI = ((URIImpl) uri).getJainURI();

        try {
            jainAddress = sipFactory.createAddressFactory().createAddress(
                    jainURI);
        } catch (PeerUnavailableException e) {
            log.error("Could not create the Address", e);
        }

        if (jainAddress != null) {
            return new AddressImpl(jainAddress);
        } else {
            return null;
        }
    }

    public Address createAddress(URI uri, String displayName) {
        javax.sip.address.Address jainAddress = null;
        // blind cast
        javax.sip.address.URI jainURI = ((URIImpl) uri).getJainURI();

        try {
            jainAddress = sipFactory.createAddressFactory().createAddress(
                    displayName, jainURI);
        } catch (PeerUnavailableException e) {
            log.error("Could not create the Address", e);
        } catch (ParseException e) {
            log.error("Could not create the Address", e);
        }

        if (jainAddress != null) {
            return new AddressImpl(jainAddress);
        } else {
            return null;
        }
    }

    public SipServletRequest createRequest(SipApplicationSession appSession,
            String method, Address from, Address to) {
        if (method.equals(Request.ACK) || method.equals(Request.CANCEL)) {
            throw new java.lang.IllegalArgumentException("" + method
                    + " requests can only be created by the container");
        }

        // Call-ID and CSeq headers, as well as Contact header if the method is not REGISTER.
        // This method makes a copy of the from and to arguments and associates them with the new SipSession


        return null;
    }

    public SipServletRequest createRequest(SipApplicationSession appSession,
            String method, URI from, URI to) {

        // delegate to the "Address"-override
        return createRequest(appSession, method, createAddress((URI)from.clone()),
                createAddress((URI)to.clone()));
    }

    public SipServletRequest createRequest(SipApplicationSession appSession,
            String method, String from, String to) throws ServletParseException {

        // delegate "Address"-override
        return createRequest(appSession, method, createAddress(from),
                createAddress(to));

    }

    public SipServletRequest createRequest(SipServletRequest request, boolean sameCallId) {
        // TODO Auto-generated method stub
        return null;
    }
```

SipFactoryImpl.java


```
    public SipApplicationSession createApplicationSession() {
        // TODO Auto-generated method stub
        return null;
    }

}
```

SipServletMessageImpl.java


```
package dk.itu.ssc.network.wrapper;

import java.io.IOException;
import java.io.UnsupportedEncodingException;
import java.security.Principal;
import java.text.ParseException;
import java.util.Enumeration;
import java.util.Hashtable;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;
import java.util.ListIterator;
import java.util.Locale;

import javax.servlet.sip.Address;
import javax.servlet.sip.ServletParseException;
import javax.servlet.sip.SipApplicationSession;
import javax.servlet.sip.SipServletMessage;
import javax.servlet.sip.SipSession;
import javax.sip.Dialog;
import javax.sip.InvalidArgumentException;
import javax.sip.ListeningPoint;
import javax.sip.PeerUnavailableException;
import javax.sip.SipFactory;
import javax.sip.header.AcceptLanguageHeader;
import javax.sip.header.CSeqHeader;
import javax.sip.header.CallIdHeader;
import javax.sip.header.ContentEncodingHeader;
import javax.sip.header.ContentLanguageHeader;
import javax.sip.header.ContentLengthHeader;
import javax.sip.header.ContentTypeHeader;
import javax.sip.header.ExpiresHeader;
import javax.sip.header.FromHeader;
import javax.sip.header.Header;
import javax.sip.header.HeaderAddress;
import javax.sip.header.HeaderFactory;
import javax.sip.header.ToHeader;
import javax.sip.message.Message;
import javax.sip.message.Request;
import javax.sip.message.Response;

import org.apache.log4j.Logger;

/**
 * An implementation of SipServletMessage that acts as an of an JAIN-SIP's
 * Message instance
 *
 * @author Mads Danquah
 */
public abstract class SipServletMessageImpl implements SipServletMessage,
        Cloneable {

    private Dialog dialog;

    private static final Logger log = Logger
            .getLogger(SipServletMessageImpl.class);

    private HeaderFactory headerFactory = null;

    private Message message = null;

    private final static String[] secureTransport = { ListeningPoint.SCTP,
            ListeningPoint.TLS };

    private String localAddress;

    protected int port;

    private String transport;

    // Attribute map
    private Hashtable<String, Object> attributes = new Hashtable<String, Object>();

    // due to the somewhat mixed request/response nature of a
    // javax.servlet.sip.SipServletMessage it will help us to know whether this
    // is a request or response
    protected int msgType = 3;

    protected final static int MSG_REQUEST = 1;

    protected final static int MSG_RESPONSE = 2;

    protected final static int MSG_UNKNOWN = 3;

    public boolean isComitted;

    private boolean isRequest() {
```

SipServletMessageImpl.java


```
        return this.msgType == MSG_REQUEST;
    }

    private boolean isResponse() {
        return this.msgType == MSG_RESPONSE;
    }

    public SipServletMessageImpl(Message msg) {
        message = msg;

        log.debug("Creating new message instance");
        log.debug("Headers:");
        Iterator<String> i = message.getHeaderNames();
        while (i.hasNext()) {
            String header = i.next();
            log.debug("- " + header);
        }
        try {
            this.headerFactory = SipFactory.getInstance().createHeaderFactory();
        } catch (PeerUnavailableException e) {
            log.fatal("Could not get HeaderFactory", e);
        }
    }

    public Address getFrom() {
        FromHeader from = (FromHeader) message.getHeader(FromHeader.NAME);

        if (from != null) {
            return new AddressImpl(from.getAddress());
        } else {
            return null;
        }
    }

    public Address getTo() {
        ToHeader to = (ToHeader) message.getHeader(ToHeader.NAME);

        if (to != null) {
            return new AddressImpl(to.getAddress());
        } else {
            return null;
        }
    }

    public String getMethod() {

        if (isRequest()) {
            // "normal" method
            return ((Request) message).getMethod();

        } else if (isResponse()) {
            // get from cseq
            // (response->CSeqHeader->getMethod())
            return ((CSeqHeader) ((Response) message)
                    .getHeader(CSeqHeader.NAME)).getMethod();
        } else {
            log.fatal("Unknown message type??");
            return null;
        }
    }

    public String getProtocol() {
        return message.getSIPVersion();
    }

    public String getHeader(String name) {
        return getHeaderValue(message.getHeader(name));
    }

    public ListIterator<String> getHeaders(String name) {
        // we'll have to go through every element and extract the String
        // representation of the Header

        ListIterator<Header> lit = (ListIterator<Header>) message
                .getHeaders(name);

        List<String> headers = new LinkedList<String>();

        while (lit.hasNext()) {
            Header jainHeader = lit.next();
            // again we rely on the implementation of toString to give an
            // accurate String representation
            headers.add(jainHeader.toString());
        }

        return headers.listIterator();
    }
```

```
public Iterator getHeaderNames() {
    // simmilar implementation
    return message.getHeaderNames();
}

public void setHeader(String name, String value)
        throws IllegalArgumentException {

    // first create an JAIN Header instance
    Header newHeader;
    try {
        newHeader = SipFactory.getInstance().createHeaderFactory()
                .createHeader(name, value);
        // make sure we've not just created a system header
        // TODO Special handling of Contact headers
        if (JainAdaptorUtil.isSystemHeader(newHeader, contactOk())) {
            throw new IllegalArgumentException(
                    "Attempt to manipulate a system-header");
        }

    } catch (PeerUnavailableException e) {
        throw new IllegalArgumentException(e);
    } catch (ParseException e) {
        throw new IllegalArgumentException("Invalid name or value:", e);
    }

    // then set the header if everything is ok
    message.setHeader(newHeader);

}

public void addHeader(String name, String value)
        throws IllegalArgumentException {
    Header newHeader;
    try {
        newHeader = SipFactory.getInstance().createHeaderFactory()
                .createHeader(name, value);
        // make sure we've not just created a system header
        // TODO Special handling of Contact headers
        if (JainAdaptorUtil.isSystemHeader(newHeader, contactOk())) {
            throw new IllegalArgumentException(
                    "Attempt to manipulate a system-header");
        }

    } catch (PeerUnavailableException e) {
        throw new IllegalArgumentException(e);
    } catch (ParseException e) {
        throw new IllegalArgumentException("Invalid name or value:", e);
    }

    // then set the header if everything is ok
    message.addHeader(newHeader);
}

public void removeHeader(String headerName) throws IllegalArgumentException {
    if (JainAdaptorUtil.isSystemHeader(headerName, contactOk())) {
        throw new IllegalArgumentException(
                "Attempt to manipulate a system-header");
    }

    message.removeHeader(headerName);
}

abstract boolean contactOk();

public Address getAddressHeader(String headerName)
        throws ServletParseException {
    // attemt to dig out the header

    Header header = message.getHeader(headerName);
    if (header == null) {
        return null;
    } else if (header instanceof HeaderAddress) {
        // we have a header containing addresses
        return new AddressImpl(((HeaderAddress) header).getAddress());

    } else {
        throw new ServletParseException("A '" + headerName
                + "'-header cannot contain an address");
    }

}

public ListIterator getAddressHeaders(String name)
        throws ServletParseException {
    ListIterator<Header> lit = (ListIterator<Header>) message
```



```
            .getHeaders(name);

    List<Address> headers = new LinkedList<Address>();

    while (lit.hasNext()) {
        Header header = lit.next();

        if (header == null) {
            return null;
        } else if (header instanceof HeaderAddress) {
            // we have a header containing addresses
            headers.add(new AddressImpl(((HeaderAddress) header)
                    .getAddress()));

        } else {
            throw new ServletParseException("A '" + name
                    + "'-header cannot contain an address");
        }

    }

    return headers.listIterator();
}

public void setAddressHeader(String name, Address value) {
    // first create an JAIN Header instance
    Header header;
    try {
        header = SipFactory.getInstance().createHeaderFactory()
                .createHeader(name, value.toString());
        // make sure we've not just created a system header
        // TODO Special handling of Contact headers
        if (JainAdaptorUtil.isSystemHeader(header, contactOk())) {
            throw new IllegalArgumentException(
                    "Attempt to manipulate a system-header");
        } else if (!(header instanceof HeaderAddress)) {
            // make sure this is actually an Address-capable header
            throw new IllegalArgumentException("A '" + name
                    + "'-header cannot contain an address");
        }
        // all is good

    } catch (PeerUnavailableException e) {
        throw new IllegalArgumentException(e);
    } catch (ParseException e) {
        throw new IllegalArgumentException("Invalid name or value:", e);
    }

    // then set the header if everything is ok
    message.setHeader(header);
}

public void addAddressHeader(String header, Address arg1, boolean arg2) {
    if (JainAdaptorUtil.isSystemHeader(header, contactOk())) {
        throw new IllegalArgumentException(
                "Attempt to manipulate a system-header");
    }
    // serious problem here, JAIN does not have support for controlling
    // wether a header should be added or prepended to the message

    // ... but the NIST implementation _do_ have support .....
}

public String getCallId() {
    CallIdHeader header = (CallIdHeader) message
            .getHeader(CallIdHeader.NAME);

    if (header != null) {
        return ((CallIdHeader) header).getCallId();
    } else {
        return null;
    }
}

public int getExpires() {
    ExpiresHeader header = (ExpiresHeader) message
            .getHeader(ExpiresHeader.NAME);

    if (header != null) {
        return header.getExpires();
    } else {
        return -1;
    }
}

public void setExpires(int seconds) {
    this.setHeader("Expires", String.valueOf(seconds));
```

```
    }

    public String getCharacterEncoding() {
        ContentEncodingHeader encoding = message.getContentEncoding();
        if (encoding != null) {
            return encoding.getEncoding();
        } else {
            return null;
        }
    }

    public void setCharacterEncoding(String encName)
            throws UnsupportedEncodingException {
        ContentEncodingHeader encoding = message.getContentEncoding();
        if (encoding != null) {
            // set existing header
            try {
                encoding.setEncoding(encName);
            } catch (ParseException e) {
                log.error("Could not set encoding" + encName, e);
                return;
            }
        } else {
            // add new header
            try {
                encoding = SipFactory.getInstance().createHeaderFactory()
                        .createContentEncodingHeader(encName);
            } catch (PeerUnavailableException e) {
                log.error("Could not set encoding to " + encName, e);
                return;
            } catch (ParseException e) {
                log.error("Could not set encoding" + encName, e);
                return;
            }
            // set the header
            message.setHeader(encoding);
        }
    }

    public int getContentLength() {
        ContentLengthHeader header = message.getContentLength();
        if (header != null) {
            return header.getContentLength();
        } else {
            return -1;
        }
    }

    public String getContentType() {
        ContentTypeHeader header = (ContentTypeHeader) message
                .getHeader(ContentTypeHeader.NAME);

        if (header != null) {
            return header.getContentType();
        } else {
            return null;
        }
    }

    public byte[] getRawContent() throws IOException {
        return message.getRawContent();
    }

    public Object getContent() throws IOException, UnsupportedEncodingException {
        return message.getContent();
    }

    public void setContent(Object content, String type)
            throws UnsupportedEncodingException {

        // parse type
        String supertype;
        String subtype;

        supertype = type.substring(0, type.indexOf("/"));
        subtype = type.substring(type.indexOf("/") + 1);

        ContentTypeHeader contentHeader = null;
        try {
            contentHeader = headerFactory.createContentTypeHeader(supertype,
                    subtype);
        } catch (ParseException e) {
            log.error("Could not create new ContentTypeHeader", e);
            return;
        }

        try {
```



```
            message.setContent(content, contentHeader);
        } catch (ParseException e) {
            log.error("Could not set content", e);
            return;
        }
    }

    public void setContentLength(int len) {
        ContentLengthHeader header = null;
        try {
            headerFactory.createContentLengthHeader(len);
        } catch (InvalidArgumentException e) {
            log.error("Could not create new Content-Length header", e);
            return;
        }
        message.setContentLength(header);
    }

    public void setContentType(String type) {
        // parse type
        String supertype;
        String subtype;

        supertype = type.substring(0, type.indexOf("/"));
        subtype = type.substring(type.indexOf("/") + 1);

        // create header
        ContentTypeHeader ctHeader = null;
        try {
            ctHeader = headerFactory
                    .createContentTypeHeader(supertype, subtype);
        } catch (ParseException e) {
            log.error("Invalid content type", e);
            return;
        }

        // set the header
        message.setHeader(ctHeader);
    }

    public Object getAttribute(String name) {
        return attributes.get(name);
    }

    public Enumeration getAttributeNames() {
        return attributes.keys();
    }

    public void setAttribute(String key, Object value) {
        attributes.put(key, value);
    }

    public abstract SipSession getSession();

    public abstract SipSession getSession(boolean create);

    public abstract SipApplicationSession getApplicationSession();

    public abstract SipApplicationSession getApplicationSession(boolean create);

    public Locale getAcceptLanguage() {
        AcceptLanguageHeader aHeader = (AcceptLanguageHeader) message
                .getHeader(AcceptLanguageHeader.NAME);

        if (aHeader != null) {
            return aHeader.getAcceptLanguage();
        } else {
            return null;
        }
    }

    public Iterator<Locale> getAcceptLanguages() {
        List<Locale> languages = new LinkedList<Locale>();

        Iterator iter = message.getHeaders(AcceptLanguageHeader.NAME);
        while (iter.hasNext()) {
            AcceptLanguageHeader aHeader = (AcceptLanguageHeader) iter.next();
            languages.add(aHeader.getAcceptLanguage());

        }
        return languages.iterator();
    }

    public void setAcceptLanguage(Locale locale) {
        AcceptLanguageHeader aHeader = headerFactory
                .createAcceptLanguageHeader(locale);
```

```
        message.setHeader(aHeader);
    }

    public void addAcceptLanguage(Locale locale) {
        AcceptLanguageHeader aHeader = headerFactory
                .createAcceptLanguageHeader(locale);

        // should have lower q-value, but adding it like this should do it
        message.addHeader(aHeader);
    }

    public void setContentLanguage(Locale locale) {
        ContentLanguageHeader clHeader = headerFactory
                .createContentLanguageHeader(locale);
        message.setContentLanguage(clHeader);

    }

    public Locale getContentLanguage() {
        ContentLanguageHeader clHeader = message.getContentLanguage();
        if (clHeader != null) {
            return clHeader.getContentLanguage();
        } else {
            return null;
        }
    }

    // should be implemented by Request and Reponse as a send requires access to
    // a Client or Server transaction
    public abstract void send() throws IOException;

    public boolean isSecure() {
        for (int i = 0; i < secureTransport.length; i++) {
            if (secureTransport[i].equals(transport)) {
                return true;
            }
        }
        return false;
    }

    public boolean isCommitted() {
        return isComitted;
    }

    // Security not implemented (allowed according to JSR-116)
    public String getRemoteUser() {
        return null;
    }

    // Security not implemented (allowed according to JSR-116)
    public boolean isUserInRole(String arg0) {
        return false;
    }

    // Security not implemented (allowed according to JSR-116)
    public Principal getUserPrincipal() {
        return null;
    }

    public String getLocalAddr() {
        return localAddress;
    }

    public int getLocalPort() {
        return port;
    }

    public String getRemoteAddr() {
        // should be handled by request/response implementation
        return null;
    }

    public int getRemotePort() {
        // should be handled by request/response implementation
        return -1;
    }

    public String getTransport() {
        return transport;
    }

    private String getHeaderValue(Header header) {
        // TODO: come up a cleaner way to do this
        // this could give us problems if the JAIN-SIP implementations toString
        // does not give us an exact textual representation of the header when
        // we invoke toString
        // Version 1.2 of NIST's implementation implements toString by invoking
```



```
        // an internal encode() method which returns an usable String

        String encodedHdr = null;
        try {
            encodedHdr = header.toString();
        } catch (Exception ex) {
            return null;
        }
        StringBuffer buffer = new StringBuffer(encodedHdr);
        while (buffer.length() > 0 && buffer.charAt(0) != ':') {
            buffer.deleteCharAt(0);
        }
        if (buffer.length() > 0)
            buffer.deleteCharAt(0);
        return buffer.toString().trim();
    }

    public void setLocalAddress(String address) {
        localAddress = address;
    }

    public void setPort(int port) {
        this.port = port;
    }

    public void setTransport(String transport) {
        this.transport = transport;
    }

    /*
     * Copy all internal mutable state
     *
     * @see java.lang.Object#clone()
     */
    @Override
    protected Object clone() throws CloneNotSupportedException {
        SipServletMessageImpl clone = (SipServletMessageImpl) super.clone();

        // copy mutable state
        clone.message = (Message) message.clone();

        return clone;
    }

    @Override
    public String toString() {
        String output = "Core JAIN SIP message:\n------start-------\n"
                + message.toString() + "\n------end--------\n";

        return output;
    }

    /**
     * @return Returns the dialog.
     */
    public Dialog getDialog() {
        return dialog;
    }

    /**
     * @param dialog
     *            The dialog to set.
     */
    public void setDialog(Dialog dialog) {
        this.dialog = dialog;
    }

}
```

SipServletRequestImpl.java


```
package dk.itu.ssc.network.wrapper;

import gov.nist.javax.sip.message.SIPRequest;

import java.io.BufferedReader;
import java.io.IOException;
import java.text.ParseException;
import java.util.Enumeration;
import java.util.Locale;

import javax.servlet.RequestDispatcher;
import javax.servlet.ServletInputStream;
import javax.servlet.sip.Proxy;
import javax.servlet.sip.SipApplicationSession;
import javax.servlet.sip.SipServletRequest;
import javax.servlet.sip.SipServletResponse;
import javax.servlet.sip.SipSession;
import javax.servlet.sip.SipURI;
import javax.servlet.sip.TooManyHopsException;
import javax.servlet.sip.URI;
import javax.sip.ClientTransaction;
import javax.sip.InvalidArgumentException;
import javax.sip.PeerUnavailableException;
import javax.sip.RequestEvent;
import javax.sip.ServerTransaction;
import javax.sip.SipException;
import javax.sip.SipFactory;
import javax.sip.SipProvider;
import javax.sip.address.Address;
import javax.sip.header.ContactHeader;
import javax.sip.header.Header;
import javax.sip.header.MaxForwardsHeader;
import javax.sip.header.RouteHeader;
import javax.sip.header.ToHeader;
import javax.sip.message.Request;
import javax.sip.message.Response;

import org.apache.log4j.Logger;
import org.sipservlet.SipRequest;

import dk.itu.ssc.SipServletContainer;
import dk.itu.ssc.util.EnumAdaptor;

/**
 * @author Mads Danquah
 */
public class SipServletRequestImpl extends SipServletMessageImpl implements
        SipServletRequest, Cloneable {

    private static final Logger log = Logger
            .getLogger(SipServletRequestImpl.class);

    private Request request;

    javax.sip.address.SipURI sipURI = null;

    private boolean initial;

    private RequestEvent requestEvent = null;

    private ServerTransaction serverTransaction;

    private SipSessionImpl session;

    private SipProvider sipProvider;

    private ClientTransaction clientTransaction;

    public SipServletRequestImpl(Request req) {
        super(req);
        super.msgType = SipServletMessageImpl.MSG_REQUEST;
        this.request = req;
        init();
    }

    public SipServletRequestImpl(Request req, ServerTransaction st) {
        super(req);
        super.msgType = SipServletMessageImpl.MSG_REQUEST;
        this.request = req;
        this.serverTransaction = st;
        init();
    }

    public Request getJainRequest() {
        return request;
    }
```

SipServletRequestImpl.java


```
    private void init() {
        if (request.getRequestURI().isSipURI()) {
            sipURI = (javax.sip.address.SipURI) request.getRequestURI();
        } else {
            log.fatal("The RequestURL of this SipRequest is not an SIP-URL!");
            throw new RuntimeException("Unsupported URL");
        }
    }

    /*
     * (non-Javadoc)
     *
     * @see javax.servlet.ServletRequest#getLocale()
     */
    public Locale getLocale() {
        return getAcceptLanguage();
    }

    /*
     * (non-Javadoc)
     *
     * @see javax.servlet.ServletRequest#getLocales()
     */
    public Enumeration<Locale> getLocales() {
        Enumeration<Locale> wrapper = new EnumAdaptor<Locale>(
                getAcceptLanguages());

        return wrapper;
    }

    /*
     * (non-Javadoc)
     *
     * @see javax.servlet.ServletRequest#getParameter(java.lang.String)
     */
    public String getParameter(String param) {
        // get url

        return sipURI.getParameter(param);
    }

    /*
     * (non-Javadoc)
     *
     * @see javax.servlet.ServletRequest#getParameterNames()
     */
    public Enumeration getParameterNames() {
        return new EnumAdaptor<String>(sipURI.getParameterNames());
    }

    /*
     * (non-Javadoc)
     *
     * @see javax.servlet.ServletRequest#getParameterValues(java.lang.String)
     */
    public String[] getParameterValues(String name) {
        // a parameter is unique
        String value = getParameter(name);
        if (value != null) {
            return new String[] { value };
        } else {
            return null;
        }
    }

    /*
     * (non-Javadoc)
     *
     * @see javax.servlet.ServletRequest#getRealPath(java.lang.String)
     */
    public String getRealPath(String path) {
        // makes no sense in a SipServlet-context
        return path;
    }

    /*
     * (non-Javadoc)
     *
     * @see javax.servlet.ServletRequest#getRemoteHost()
     */
    public String getRemoteHost() {
        return null;
    }
```

```
    /*
     * (non-Javadoc)
     *
     * @see javax.servlet.ServletRequest#getRequestDispatcher(java.lang.String)
     */
    public RequestDispatcher getRequestDispatcher(String arg0) {
        // TODO Await implementation of a RequestDispatcher
        return null;
    }

    /*
     * (non-Javadoc)
     *
     * @see javax.servlet.ServletRequest#getScheme()
     */
    public String getScheme() {
        return sipURI.getScheme();
    }

    /*
     * (non-Javadoc)
     *
     * @see javax.servlet.ServletRequest#getServerName()
     */
    public String getServerName() {
        // only relervant for HTTPServlets hosted in containers that can have
        // multiple hostnames
        return null;
    }

    /*
     * (non-Javadoc)
     *
     * @see javax.servlet.ServletRequest#getServerPort()
     */
    public int getServerPort() {
        // only relervant for HTTPServlets hosted in containers that can have
        // multiple interfaces
        return -1;
    }

    /*
     * (non-Javadoc)
     *
     * @see javax.servlet.ServletRequest#removeAttribute(java.lang.String)
     */
    public void removeAttribute(String name) {
        sipURI.removeParameter(name);

    }

    /*
     * (non-Javadoc)
     *
     * @see javax.servlet.sip.SipServletRequest#createCancel()
     */
    public SipServletRequest createCancel() throws IllegalStateException {
        // we can only cancel if we orginally where recived by the
        // network-manager and instansiated with an requestEvent
        if (requestEvent == null) {
            throw new IllegalStateException(
                    "Only Requests recived by the container can generate new Requests");
        }

        // Wrap up a new Jain-Request and return it
        try {
            return new SipServletRequestImpl(requestEvent
                    .getServerTransaction().getDialog().createRequest(
                            Request.CANCEL));
        } catch (SipException e) {
            log.error("Could not create CANCEL", e);
        }
        return null;
    }

    /*
     * (non-Javadoc)
     *
     * @see javax.servlet.sip.SipServletRequest#createResponse(int,
     *      java.lang.String)
     */
    public SipServletResponse createResponse(int statusCode, String reason) {
        // create the response
        Response reponse;
        try {
            reponse = SipFactory.getInstance().createMessageFactory()
                    .createResponse(statusCode, request);
```



```
            // setup the response
            ToHeader toHeader = (ToHeader) reponse.getHeader(ToHeader.NAME);

            // Contact header is optional for informal (1xx) reponses
            if (statusCode >= 200) {
                // A contact header is mandatory in response to INVITE's and
                // optional in
                // response to OPTIONS and REGISTER requests. An user agent
                // server sending a
                // Success Response to an INIVTE must insert a ContactHeader in
                // the Response
                // indicating the SIP address under which it is reachable most
                // directly for
                // future SIP Requests.
                boolean skipContact = false;

                // success 2xx
                // contact header is mandatory in response to INVITE's and
                // optional in response to OPTIONS and REGISTER requests.
                if (statusCode < 300) {
                    // optional for OPTIONS and REGISTER
                    skipContact = (request.getMethod() == Request.OPTIONS || request
                            .getMethod() == Request.REGISTER);
                }

                // redirect 3xx
                // A contact header is optional in response to INVITE's,
                // OPTIONS, REGISTER and BYE requests.
                if (statusCode < 400) {
                    skipContact = (request.getMethod() == Request.INVITE
                            || request.getMethod() == Request.OPTIONS
                            || request.getMethod() == Request.REGISTER || request
                            .getMethod() == Request.BYE);
                }

                if (!skipContact) {
                    // create the contact header
                    // generate contact
                    javax.sip.address.SipURI uri = SipFactory.getInstance()
                            .createAddressFactory().createSipURI(null,
                                    getLocalAddr());
                    uri.setPort(this.port);
                    Address contact = SipFactory.getInstance()
                            .createAddressFactory().createAddress(uri);
                    ContactHeader contactHeader = SipFactory.getInstance()
                            .createHeaderFactory().createContactHeader(contact);
                    reponse.addHeader(contactHeader);
                }
            }

            // examine wether we should add a tag (reponse on INVITE)
            if (request.getMethod().equals(Request.INVITE) && statusCode >= 200
                    && statusCode < 300) {

                // generate a new tag and store the session
                toHeader.setTag(this.session.createTag());

                // store the session
            }

            // set the reason if any
            // if null JAIN (The NIST implementation anyway) will set one for us
            reponse.setReasonPhrase(reason);

            // create the wrapper
            SipServletResponseImpl reponseWrapper = new SipServletResponseImpl(
                    reponse);

            // pass on some essencial data
            reponseWrapper.setServerTransaction(this.serverTransaction);
            reponseWrapper.setSession(this.session);

            return reponseWrapper;
        } catch (PeerUnavailableException e) {
            log.error("Could not create response", e);
        } catch (ParseException e) {
            log.error("Could not create response", e);
        }
        return null;
    }

    /*
     * (non-Javadoc)
     *
     * @see javax.servlet.sip.SipServletRequest#createResponse(int)
     */
```

SipServletRequestImpl.java


```
public SipServletResponse createResponse(int statusCode) {
    return createResponse(statusCode, "");

}

/*
 * (non-Javadoc)
 *
 * @see javax.servlet.sip.SipServletRequest#getInputStream()
 */
public ServletInputStream getInputStream() throws IOException {
    // makes no sense for a SipRequest
    return null;
}

/*
 * (non-Javadoc)
 *
 * @see javax.servlet.sip.SipServletRequest#getMaxForwards()
 */
public int getMaxForwards() {
    MaxForwardsHeader mfHeader = (MaxForwardsHeader) request
            .getHeader(MaxForwardsHeader.NAME);
    if (mfHeader != null) {
        return mfHeader.getMaxForwards();
    } else {
        return -1;
    }
}

/*
 * (non-Javadoc)
 *
 * @see javax.servlet.sip.SipServletRequest#getProxy()
 */
public Proxy getProxy() throws TooManyHopsException {
    /*
     * if(this.requestEvent == null){ // The Request instance must have been
     * created locally == we'r not a // server and cannot proxy throw new
     * IllegalStateException("Not a server"); } // ok, frist check that
     * we're allowed to forward any furter if(getMaxForwards() == 0){ throw
     * new TooManyHopsException(); }else if (getMaxForwards() == -1) { //
     * create a new maxForwards header with the default value
     * setMaxForwards(70); }
     */

    // wait until we actually decide to implement proxying
    throw new RuntimeException("not implemented");
}

/*
 * (non-Javadoc)
 *
 * @see javax.servlet.sip.SipServletRequest#getProxy(boolean)
 */
public Proxy getProxy(boolean arg0) throws TooManyHopsException {
    throw new RuntimeException("not implemented");
}

/*
 * (non-Javadoc)
 *
 * @see javax.servlet.sip.SipServletRequest#getReader()
 */
public BufferedReader getReader() throws IOException {
    // does not make sense for SipServlets
    return null;
}

/*
 * (non-Javadoc)
 *
 * @see javax.servlet.sip.SipServletRequest#getRequestURI()
 */
public URI getRequestURI() {
    // Adapt jainUri to a Servlet-URI
    javax.sip.address.URI uri = request.getRequestURI();
    if (uri.isSipURI()) {
        return new SipURIImpl((javax.sip.address.SipURI) uri);
    } else {
        return new URIImpl(request.getRequestURI());
    }

}

/*
 * (non-Javadoc)
```

SipServletRequestImpl.java


```
     *
     * @see javax.servlet.sip.SipServletRequest#isInitial()
     */
    public boolean isInitial() {
        return initial;
    }

    /**
     * Make this request initial
     *
     * @param initial
     */
    public void setInitial(boolean initial) {
        this.initial = initial;
    }

    /*
     * (non-Javadoc)
     *
     * @see javax.servlet.sip.SipServletRequest#pushRoute(javax.servlet.sip.SipURI)
     */
    public void pushRoute(SipURI uri) {
        SipFactory sf = SipFactory.getInstance();
        // get route header
        RouteHeader rh = (RouteHeader) request.getHeader(RouteHeader.NAME);

        Address a = null;
        try {
            RouteHeader newRouteHeader = SipFactory.getInstance()
                    .createHeaderFactory().createRouteHeader(a);
            a = sf.createAddressFactory().createAddress(
                    ((URIImpl) uri).getJainURI());
            if (rh == null) {
                // no route header yet, add
                request.addHeader(newRouteHeader);

            } else {
                request.addHeader(newRouteHeader);
            }

        } catch (PeerUnavailableException e) {
            log.error("COuld not create Address instance", e);
        }
    }

    /*
     * (non-Javadoc)
     *
     * @see javax.servlet.sip.SipServletRequest#setMaxForwards(int)
     */
    public void setMaxForwards(int max) {
        try {
            // create new MaxForwards header
            Header mfHeader;
            mfHeader = SipFactory.getInstance().createHeaderFactory()
                    .createMaxForwardsHeader(max);
            request.setHeader(mfHeader);
        } catch (PeerUnavailableException e) {
            log.error("Could not set MaxForwards", e);
        } catch (InvalidArgumentException e) {
            log.error("Could not set MaxForwards", e);
        }
    }

    /*
     * (non-Javadoc)
     *
     * @see javax.servlet.sip.SipServletRequest#setRequestURI(javax.servlet.sip.URI)
     */
    public void setRequestURI(URI uri) {

        // blind cast to URIImpl
        request.setRequestURI(((URIImpl) uri).getJainURI());

    }

    public void setRequestEvent(RequestEvent requestEvent) {
        this.requestEvent = requestEvent;

    }

    @Override
    protected Object clone() throws CloneNotSupportedException {
        SipServletRequestImpl clone = (SipServletRequestImpl) super.clone();
        clone.request = (Request) this.request.clone();

        return clone;
```

SipServletRequestImpl.java


```
    }

    /**
     * @return Returns the session.
     */
    public SipSessionImpl getSession() {
        return session;
    }

    /**
     * @param session
     *            The session to set.
     */
    public void setSession(SipSessionImpl session) {
        this.session = session;
    }

    @Override
    public void send() throws IOException {
        try {
            // store the session
            // SessionManager.store(this);
            log.debug("1%%%%%%%%%%%%%% "
                    + clientTransaction.getDialog().getDialogId());
            this.clientTransaction.sendRequest();
            log.debug("2%%%%%%%%%%%%%% "
                    + clientTransaction.getDialog().getDialogId());
            // update last accessed time
            session.setLastAccessedTime(System.currentTimeMillis());
        } catch (SipException e) {
            log.fatal("Could not send request", e);
            SipServletContainer.getInstance().shutdown();
        }
    }

    /**
     * @return Returns the sipProvider.
     */
    public SipProvider getSipProvider() {
        return sipProvider;
    }

    public void setSipProvider(SipProvider provider) {
        this.sipProvider = provider;

    }

    public void setClientTransaction(ClientTransaction ct) {
        this.clientTransaction = ct;

    }

    @Override
    public SipSession getSession(boolean create) {
        return session;
    }

    @Override
    public SipApplicationSession getApplicationSession() {
        return session.getApplicationSession();
    }

    @Override
    public SipApplicationSession getApplicationSession(boolean create) {
        // TODO: Should not create the application-session by default?
        return session.getApplicationSession();
    }

    @Override
    boolean contactOk() {
        return request.getMethod().toLowerCase().equals(SIPRequest.REGISTER);
    }


}
```

SipServletResponseImpl.java


```
package dk.itu.ssc.network.wrapper;

import gov.nist.javax.sip.message.SIPRequest;

import java.io.IOException;
import java.io.PrintWriter;
import java.text.ParseException;
import java.util.Locale;

import javax.servlet.ServletOutputStream;
import javax.servlet.sip.Proxy;
import javax.servlet.sip.Rel100Exception;
import javax.servlet.sip.SipApplicationSession;
import javax.servlet.sip.SipServletRequest;
import javax.servlet.sip.SipServletResponse;
import javax.servlet.sip.SipSession;
import javax.sip.ClientTransaction;
import javax.sip.ResponseEvent;
import javax.sip.ServerTransaction;
import javax.sip.SipException;
import javax.sip.message.Request;
import javax.sip.message.Response;

import org.apache.log4j.Logger;

import dk.itu.ssc.SipServletContainer;
import dk.itu.ssc.network.SipNetworkManager;

public class SipServletResponseImpl extends SipServletMessageImpl implements
        SipServletResponse, Cloneable {

    Response response = null;

    ResponseEvent responseEvent = null;

    Logger log = Logger.getLogger(SipServletResponseImpl.class);

    private ServerTransaction serverTransaction;

    private ClientTransaction clientTransaction = null;

    private SipSessionImpl session = null;

    public Response getJainReponse() {
        return response;
    }

    public SipServletResponseImpl(Response res) {
        super(res);
        super.msgType = SipServletMessageImpl.MSG_RESPONSE;
        response = res;
    }

    public SipServletResponseImpl(Response res,
            ClientTransaction clientTransaction) {
        super(res);
        super.msgType = SipServletMessageImpl.MSG_RESPONSE;
        response = res;
        this.clientTransaction = clientTransaction;
    }

    public SipServletRequest getRequest() {
        if (this.responseEvent != null) {
            return new SipServletRequestImpl(responseEvent
                    .getClientTransaction().getRequest());
        } else {
            return null;
        }
    }

    public int getStatus() {
        return response.getStatusCode();
    }

    public void setStatus(int status) {
        try {
            response.setStatusCode(status);
        } catch (ParseException e) {
            log.error("Could not set status", e);
        }
    }

    public void setStatus(int status, String reason) {
        try {
            response.setStatusCode(status);
            response.setReasonPhrase(reason);
        } catch (ParseException e) {
```

SipServletResponseImpl.java


```
            log.error("Could not set status", e);
        }
    }

    public String getReasonPhrase() {
        return response.getReasonPhrase();
    }

    public ServletOutputStream getOutputStream() throws IOException {
        return null;
    }

    public PrintWriter getWriter() throws IOException {
        return null;
    }

    public Proxy getProxy() {
        // wait until we deside to actually support Proxying
        throw new RuntimeException("Not implemented");
    }

    public void sendReliably() throws Rel100Exception {
        throw new Rel100Exception(Rel100Exception.NOT_SUPPORTED);
    }

    public SipServletRequest createAck() {
        if (!"INVITE".equals(getMethod())) {
            throw new IllegalStateException("ACKs are for INVITEs only");
        }
        if (getStatus() >= 300) {
            throw new IllegalStateException(
                    "ACKs for non-2xx's are generated automatically by the container");
        }

        // ok, get the ACK
        try {
            return new SipServletRequestImpl(responseEvent
                    .getClientTransaction().createAck());
        } catch (SipException e) {
            log.error("Could not create ACK", e);
        }
        return null;

    }

    public void setLocale(Locale loc) {
        setContentLanguage(loc);
    }

    // Buffer-related methods that makes no sense for SipReponses
    public void flushBuffer() throws IOException {
    }

    public int getBufferSize() {
        return 0;
    }

    public Locale getLocale() {
        return getContentLanguage();
    }

    public void reset() {
    }

    public void setBufferSize(int arg0) {
    }

    @Override
    protected Object clone() throws CloneNotSupportedException {
        SipServletResponseImpl clone = (SipServletResponseImpl) super.clone();
        clone.response = (Response) this.response.clone();

        return clone;
    }

    protected void setServerTransaction(ServerTransaction st) {
        this.serverTransaction = st;
    }

    @Override
    public void send() throws IOException {
        log.debug("Sending reponse: \n" + response);
        SipServletContainer.getInstance().getNetworkManager()
                .sendResponse(this);
    }

    /**
```

SipServletResponseImpl.java


```
     * @return Returns the clientTransaction.
     */
    public ClientTransaction getClientTransaction() {
        return clientTransaction;
    }

    public void setSession(SipSessionImpl session) {
        this.session = session;
    }

    @Override
    public SipSessionImpl getSession() {
        return this.session;
    }

    @Override
    public SipSession getSession(boolean create) {
        return this.session;
    }

    @Override
    public SipApplicationSession getApplicationSession() {
        return this.session.getApplicationSession();
    }

    @Override
    public SipApplicationSession getApplicationSession(boolean create) {
        // TODO Should only create app-sessions when nessacary
        return this.session.getApplicationSession();
    }

    /**
     * @return Returns the serverTransaction.
     */
    public ServerTransaction getServerTransaction() {
        return serverTransaction;
    }

    @Override
    boolean contactOk() {
        return (getStatus() >= 300 && getStatus() < 400) || getStatus() == 485;
    }
}
```

SipSessionImpl.java

Page 1/3

```
package dk.itu.ssc.network.wrapper;

import java.util.Collections;
import java.util.Enumeration;
import java.util.HashMap;
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;

import javax.servlet.ServletException;
import javax.servlet.sip.Address;
import javax.servlet.sip.SipApplicationSession;
import javax.servlet.sip.SipServletRequest;
import javax.servlet.sip.SipSession;
import javax.sip.ClientTransaction;
import javax.sip.Dialog;
import javax.sip.SipException;
import javax.sip.SipProvider;
import javax.sip.TransactionUnavailableException;
import javax.sip.header.CallIdHeader;
import javax.sip.message.Message;
import javax.sip.message.Request;

import org.apache.log4j.Logger;

import dk.itu.ssc.SipServletContainer;
import dk.itu.ssc.application.Dispatcher;
import dk.itu.ssc.session.SessionManager;
import dk.itu.ssc.util.EnumAdaptor;
import dk.itu.ssc.util.IDGenerator;

public class SipSessionImpl implements SipSession {
    private Logger log = Logger.getLogger(SipSessionImpl.class);

    private String localTag = null;

    protected Dispatcher dispatcher;

    private Map<String, Object> attributes;

    private static final long serialVersionUID = 3690193252523194417L;

    private long creationTime;

    private long lastAccessedTime;

    private ApplicationSession appSession;

    private String id = null;

    private String callId;
    private Address localParty;
    private Address remoteParty;

    private Dialog dialog;

    public SipSessionImpl(ApplicationSession appSession, String callId, Address localParty, Address remotePart
y) {
        this.appSession = appSession;
        // note creation time
        this.creationTime = System.currentTimeMillis();
        this.attributes = new ConcurrentHashMap<String, Object>();

    }

    public long getCreationTime() {
        return creationTime;
    }

    public String getId() {
        return id;
    }

    public long getLastAccessedTime() {
        return lastAccessedTime;
    }

    public void setLastAccessedTime(long last) {
        this.lastAccessedTime = last;
        appSession.setLastAccessedTime(last);
    }

    public void invalidate() {
        // remove the session from the sessionmap

        // remove the session from the application session

    }
```

SipSessionImpl.java

Page 2/3

```
    public SipApplicationSession getApplicationSession() {
        return appSession;
    }

    public SipServletRequest createRequest(String methodName) {
        // for now we assume dialog exist.
        // It should probably be created manually for new requests outside a
        // dialog

        Request byeRequest;
        try {
            byeRequest = dialog.createRequest(Request.BYE);
        } catch (SipException e) {
            log.fatal("Could not create request", e);
            SipServletContainer.getInstance().shutdown();
            return null;
        }

        // get new sip-provider
        SipProvider provider = SipServletContainer.getInstance()
                .getNetworkManager().getDefaultProvider();

        ClientTransaction ct = null;
        try {
            ct = provider.getNewClientTransaction(byeRequest);
        } catch (TransactionUnavailableException e) {
            log
                    .fatal(
                            "Could not create request (Could not create new ClientTransaction)",
                            e);
            SipServletContainer.getInstance().shutdown();
            return null;
        }
        SipServletRequestImpl requestWrapper = new SipServletRequestImpl(
                byeRequest);

        // set client transaction and session
        requestWrapper.setClientTransaction(ct);
        requestWrapper.setSession(this);

        return requestWrapper;
    }

    public void setHandler(String handler) throws ServletException {

    }

    public void setHandler(Dispatcher d) {
        this.dispatcher = d;
    }

    public Dispatcher getDispatcher() {
        return dispatcher;
    }

    public Object getAttribute(String key) {
        return this.attributes.get(key);
    }

    public Enumeration getAttributeNames() {
        return new EnumAdaptor<String>(this.attributes.keySet().iterator());
    }

    public void setAttribute(String key, Object val) {
        this.attributes.put(key, val);
    }

    public void removeAttribute(String key) {
        this.attributes.remove(key);

    }

    public String getLocalTag() {
        return this.localTag;
    }

    public String createTag() {
        this.localTag = IDGenerator.generate();
        log.debug("Generating new local tag:" + localTag);

        return this.localTag;
    }

    public void store() {
        this.id = dialog.getDialogId();
        SessionManager.store(this);
```

SipSessionImpl.java


```
    }

    public void setDialog(Dialog dialog) {
        this.dialog = dialog;

    }

    /**
     * @return Returns the callId.
     */
    public String getCallId() {
        return callId;
    }

    /**
     * @return Returns the localParty.
     */
    public Address getLocalParty() {
        return localParty;
    }

    /**
     * @param callId The callId to set.
     */
    public void setCallId(String callId) {
        this.callId = callId;
    }

    /**
     * @param localParty The localParty to set.
     */
    public void setLocalParty(Address localParty) {
        this.localParty = localParty;
    }

    /**
     * @param localTag The localTag to set.
     */
    public void setLocalTag(String localTag) {
        this.localTag = localTag;
    }

    /**
     * @return Returns the remoteParty.
     */
    public Address getRemoteParty() {
        return remoteParty;
    }

    /**
     * @param remoteParty The remoteParty to set.
     */
    public void setRemoteParty(Address remoteParty) {
        this.remoteParty = remoteParty;
    }
}
```

SipURIImpl.java


```
package dk.itu.ssc.network.wrapper;

import java.text.ParseException;
import java.util.Iterator;

import javax.servlet.sip.SipURI;
import javax.sip.InvalidArgumentException;

import org.apache.log4j.Logger;

public class SipURIImpl extends URIImpl implements SipURI {

    static private Logger log = Logger.getLogger(SipURIImpl.class);

    javax.sip.address.SipURI uri;

    public SipURIImpl(javax.sip.address.SipURI jainUri) {
        super(jainUri);
        uri = jainUri;
    }

    public String getUser() {
        return uri.getUser();
    }

    public void setUser(String user) {
        try {
            uri.setUser(user);
        } catch (ParseException e) {
            log.error("Could not set user", e);
        }
    }

    public String getUserPassword() {
        return (uri.getUserPassword());
    }

    public void setUserPassword(String password) {
        try {
            uri.setUserPassword(password);
        } catch (ParseException e) {
            log.error("Could not set UserPassword", e);
        }

    }

    public String getHost() {
        return (uri.getHost());
    }

    public void setHost(String host) {
        try {
            uri.setHost(host);
        } catch (ParseException e) {
            log.error("Could not set host", e);
        }
    }

    public int getPort() {
        return uri.getPort();
    }

    public void setPort(int port) {
        uri.setPort(port);

    }

    public boolean isSecure() {
        return uri.isSecure();
    }

    public void setSecure(boolean secure) {
        uri.setSecure(secure);
    }

    public String getParameter(String name) {
        // hack to be compatable with the RI
        if(name.equals("transport") && uri.getParameter(name) != null){
            return uri.getParameter(name).toLowerCase();
        }
        return uri.getParameter(name);
    }

    public void setParameter(String name, String value) {
        try {
            uri.setParameter(name, value);
        } catch (ParseException e) {
```

```
            log.error("Could not set parameter" + name, e);
        }
    }

    public void removeParameter(String name) {
        uri.removeParameter(name);

    }

    public Iterator getParameterNames() {
        return uri.getParameterNames();
    }

    public String getTransportParam() {
        return getParameter("transport");
    }

    public void setTransportParam(String transport) {
        try {
            uri.setTransportParam(transport);
        } catch (ParseException e) {
            log.error("Could not set TransportParam", e);
        }

    }

    public String getMAddrParam() {
        return uri.getMAddrParam();
    }

    public void setMAddrParam(String mAddr) {
        try {
            uri.setMAddrParam(mAddr);
        } catch (ParseException e) {
            log.error("Could not set MAddrParam", e);
        }
    }

    public String getMethodParam() {
        return uri.getMethodParam();
    }

    public void setMethodParam(String method) {
        try {
            uri.setMethodParam(method);
        } catch (ParseException e) {
            log.error("Could not set MethodParam", e);
        }
    }

    public int getTTLParam() {
        return uri.getTTLParam();
    }

    public void setTTLParam(int ttl) {
        try {
            uri.setTTLParam(ttl);
        } catch (InvalidArgumentException e) {
            log.error("Could not set TTLParam", e);
        }
    }

    public String getUserParam() {
        return uri.getUserParam();
    }

    public void setUserParam(String userParam) {
        try {
            uri.setUserParam(userParam);
        } catch (ParseException e) {
            log.error("Could not set UserParam", e);
        }
    }

    public boolean getLrParam() {
        return uri.hasLrParam();
    }

    public void setLrParam(boolean value) {
        // The JAIN implementation of this only allows to set the value, but not
        // to unset it
        if (value == false && uri.hasLrParam()) {
            log
                    .error("Unsupported action: cannot unset LrParm when its been set set");
            // TODO Document this!!
            throw new RuntimeException("Missing implementation");
        }
```



```
        uri.setLrParam();
    }

    public String getHeader(String name) {
        return uri.getHeader(name);
    }

    public void setHeader(String name, String value) {
        try {
            uri.setHeader(name, value);
        } catch (ParseException e) {
            log.error("Could not set header " + name, e);
        }
    }

    public Iterator getHeaderNames() {
        return uri.getHeaderNames();
    }

    @Override
    public Object clone() {
        SipURIImpl clone = (SipURIImpl) super.clone();
        clone.uri = (javax.sip.address.SipURI) this.uri.clone();
        return clone;

    }
}
```

TelURLImpl.java


```
package dk.itu.ssc.network.wrapper;

import java.util.Iterator;

import javax.servlet.sip.TelURL;

import org.apache.log4j.Logger;

public class TelURLImpl implements TelURL {

    private Logger log = Logger.getLogger(TelURLImpl.class);

    javax.sip.address.TelURL jainURL;

    public TelURLImpl(javax.sip.address.TelURL jainURL) {
        this.jainURL = jainURL;
    }

    public String getPhoneNumber() {
        return jainURL.getPhoneNumber();
    }

    public boolean isGlobal() {
        return jainURL.isGlobal();
    }

    public String getParameter(String key) {
        return jainURL.getParameter(key);
    }

    public Iterator getParameterNames() {
        return jainURL.getParameterNames();
    }

    public String getScheme() {
        return jainURL.getScheme();
    }

    public boolean isSipURI() {
        return jainURL.isSipURI();
    }

    public Object clone() {
        try {
            // clone Object
            TelURLImpl clon = (TelURLImpl) super.clone();

            // clone internal delegate
            clon.jainURL =  (javax.sip.address.TelURL) jainURL.clone();

            return clon;
        } catch (CloneNotSupportedException e) {
            // should not happen as the parent is Object
            log.fatal("Could not clone URI", e);
        }
        return null;
    }

}
```

TimerServiceImpl.java


```
package dk.itu.ssc.network.wrapper;

import java.io.Serializable;

import javax.servlet.sip.ServletTimer;
import javax.servlet.sip.SipApplicationSession;
import javax.servlet.sip.TimerService;

public class TimerServiceImpl implements TimerService {

    public ServletTimer createTimer(SipApplicationSession appSession, long delay,
            boolean isPersistent, Serializable info) {
        return new ServletTimerImpl(appSession, delay, isPersistent, info);
    }

    public ServletTimer createTimer(SipApplicationSession appSession, long delay,
            long period, boolean fixedDelay, boolean isPersistent, Serializable info) {
        return new ServletTimerImpl(appSession, delay, period, fixedDelay,
                isPersistent, info);
    }

}
```

URIImpl.java


```java
package dk.itu.ssc.network.wrapper;

import java.text.ParseException;

import javax.servlet.sip.URI;
import javax.sip.PeerUnavailableException;
import javax.sip.SipFactory;

import org.apache.log4j.Logger;

/**
 * Done
 * @author Mads Danquah
 *
 */
public class URIImpl implements URI {

    private static final Logger log = Logger.getLogger(URIImpl.class);

    protected javax.sip.address.URI uri = null;

    public URIImpl(javax.sip.address.URI jainURI) {
        this.uri = jainURI;
    }

    public javax.sip.address.URI getJainURI(){
        return uri;
    }

    public URIImpl(URI servletURI) {
        // as this is just a wrapper around javax.sip.address.URI we need a new
        // instance to wrap
        try {
            this.uri = SipFactory.getInstance().createAddressFactory()
                    .createURI(servletURI.toString());
        } catch (PeerUnavailableException e) {
            log.fatal("Could not instansiate new URI", e);
        } catch (ParseException e) {
            log.fatal("Could not instansiate new URI", e);
        }
    }

    public String getScheme() {
        return uri.getScheme();
    }

    public boolean isSipURI() {
        return uri.isSipURI();
    }

    public Object clone() {
        try {
            // clone Object
            URIImpl clon = (URIImpl)super.clone();

            // clone internal delegate
            clon.uri = (javax.sip.address.URI)uri.clone();

            return clon;
        } catch (CloneNotSupportedException e) {
            // should not happen as the parent is Object
            log.fatal("Could not clone URI", e);
        }
        return null;
    }
}
```

ConfigurationErrorException.java


```java
package dk.itu.ssc.exceptions;

public class ConfigurationErrorException extends Exception {

    /**
     * Update when the structure of the class changes
     */
    private static final long serialVersionUID = 1L;

    public ConfigurationErrorException() {
        super();
        // TODO Auto-generated constructor stub
    }

    public ConfigurationErrorException(String arg0, Throwable arg1) {
        super(arg0, arg1);
        // TODO Auto-generated constructor stub
    }

    public ConfigurationErrorException(String arg0) {
        super(arg0);
        // TODO Auto-generated constructor stub
    }

    public ConfigurationErrorException(Throwable arg0) {
        super(arg0);
        // TODO Auto-generated constructor stub
    }

}
```

IlligalStateException.java

Page 1/1

```java
package dk.itu.ssc.exceptions;

public class IlligalStateException extends Exception {

    /**
     * Update when the structure of the class changes
     */
    private static final long serialVersionUID = 1L;

    public IlligalStateException() {
        super();
        // TODO Auto-generated constructor stub
    }

    public IlligalStateException(String arg0, Throwable arg1) {
        super(arg0, arg1);
        // TODO Auto-generated constructor stub
    }

    public IlligalStateException(String arg0) {
        super(arg0);
        // TODO Auto-generated constructor stub
    }

    public IlligalStateException(Throwable arg0) {
        super(arg0);
        // TODO Auto-generated constructor stub
    }

}
```

SipNetworkException.java

Page 1/1

```java
package dk.itu.ssc.exceptions;

public class SipNetworkException extends Exception {

    /**
     * Update when the structure of the class changes
     */
    private static final long serialVersionUID = 1L;

    public SipNetworkException() {
        super();
        // TODO Auto-generated constructor stub
    }

    public SipNetworkException(String arg0, Throwable arg1) {
        super(arg0, arg1);
        // TODO Auto-generated constructor stub
    }

    public SipNetworkException(String arg0) {
        super(arg0);
        // TODO Auto-generated constructor stub
    }

    public SipNetworkException(Throwable arg0) {
        super(arg0);
        // TODO Auto-generated constructor stub
    }

}
```

UnknownProtocolException.java

Page 1/1

```java
package dk.itu.ssc.exceptions;

public class UnknownProtocolException extends Exception {

    /**
     * Update when the structure of the class changes
     */
    private static final long serialVersionUID = 1L;

    public UnknownProtocolException() {
        super();
        // TODO Auto-generated constructor stub
    }

    public UnknownProtocolException(String arg0, Throwable arg1) {
        super(arg0, arg1);
        // TODO Auto-generated constructor stub
    }

    public UnknownProtocolException(String arg0) {
        super(arg0);
        // TODO Auto-generated constructor stub
    }

    public UnknownProtocolException(Throwable arg0) {
        super(arg0);
        // TODO Auto-generated constructor stub
    }

}
```

CallState.java

Page 1/1

```java
package dk.itu.ssc.framework;

import javax.servlet.sip.SipSession;
import javax.servlet.sip.URI;

public class CallState {

    URI remote;

    URI local;

    SipSession session;

    Configuration configuration;

    /**
     * Recives the basic configuration from the framework
     *
     * @param conf
     */
    public void configure(Configuration conf) {
        this.configuration = conf;
    }

    protected Configuration getConfig() {
        return configuration;
    }

    /**
     * @return Returns the local.
     */
    public URI getLocal() {
        return local;
    }

    /**
     * @return Returns the remote.
     */
    public URI getRemote() {
        return remote;
    }

    /**
     * @return Returns the session.
     */
    public SipSession getSession() {
        return session;
    }

    /**
     * @param local
     *            The local to set.
     */
    public void setLocal(URI local) {
        this.local = local;
    }

    /**
     * @param remote
     *            The remote to set.
     */
    public void setRemote(URI remote) {
        this.remote = remote;
    }

    /**
     * @param session
     *            The session to set.
     */
    public void setSession(SipSession session) {
        this.session = session;
    }

}
```

Configuration.java

Page 1/1

```
package dk.itu.ssc.framework;

public class Configuration {

}
```

DataSourceLocationService.java

Page 1/1

```
package dk.itu.ssc.framework;

import javax.servlet.sip.URI;
import javax.sql.DataSource;

public class DataSourceLocationService implements LocationService {

    /**
     * Recieves the datasource that should be used to store the
     * location-information
     *
     * @param datasource
     */
    public void setDataSource(DataSource datasource) {

    }

    /*
     * (non-Javadoc)
     *
     * @see dk.itu.ssc.framework.LocationService#storeLocation(javax.servlet.sip.URI,
     *      javax.servlet.sip.URI)
     */
    public void storeLocation(URI alias, URI real) {

    }

    /*
     * (non-Javadoc)
     *
     * @see dk.itu.ssc.framework.LocationService#lookup(javax.servlet.sip.URI)
     */
    public void lookup(URI alias) {
        // TODO Auto-generated method stub

    }

}
```

LocationService.java

Page 1/1

```
package dk.itu.ssc.framework;

import javax.servlet.sip.URI;

public interface LocationService {
    public void storeLocation(URI alias, URI real);
    public void lookup(URI alias);
}
```

BaseScenario.java

Page 1/1

```
package dk.itu.ssc.framework.scenario;

import dk.itu.ssc.framework.Configuration;

public class BaseScenario {
    public void getConfiguration(Configuration conf) {

    }
}
```

ProxyScenario.java


```
package dk.itu.ssc.framework.scenario;

public class ProxyScenario extends BaseScenario {

}
```

RedirectScenario.java


```
package dk.itu.ssc.framework.scenario;

public class RedirectScenario extends BaseScenario {

}
```

RegistrarScenario.java

Page 1/1

```
package dk.itu.ssc.framework.scenario;

import dk.itu.ssc.framework.Configuration;

public class RegistrarScenario extends BaseScenario {


}
```

UASScenario.java

Page 1/1

```
package dk.itu.ssc.framework.scenario;

public class UASScenario extends BaseScenario {

}
```

ApplicationManagementStrategy.java


```
package dk.itu.ssc.application;

import dk.itu.ssc.ServerContext;
import dk.itu.ssc.network.wrapper.SipServletRequestImpl;

public interface ApplicationManagementStrategy {

    Dispatcher route(SipServletRequestImpl request);

    void configure(ServerContext sc);

}
```

ApplicationManager.java


```
package dk.itu.ssc.application;

import org.apache.log4j.Logger;

import dk.itu.ssc.ContainerPart;
import dk.itu.ssc.ServerContext;
import dk.itu.ssc.network.wrapper.SipServletRequestImpl;

public class ApplicationManager implements ContainerPart {

    ApplicationManagementStrategy strategy;
    Logger log = Logger.getLogger(ApplicationManager.class);

    public ApplicationManager(ApplicationManagementStrategy strategy) {
        this.strategy = strategy;
    }

    public Dispatcher route(SipServletRequestImpl request) {
        return strategy.route(request);
    }

    /**
     * Should register and configure the ApplicationManager and all
     * SipApplications. A SipApplicationSession should be created for each
     * application
     *
     * @param sc
     */
    public void configure(ServerContext sc) {
        strategy.configure(sc);
        log.info("ApplicationManager configured");
    }

    public void shutdown() {
        log.info("shutdown complete");
        // do nothing for now
    }
}
```

DDReader.java Page 1/3

```
package dk.itu.ssc.application;

import java.io.File;
import java.io.IOException;
import java.io.InputStream;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;
import java.util.Map;

import javax.servlet.sip.SipServlet;

import org.apache.log4j.Logger;
import org.jdom.Document;
import org.jdom.Element;
import org.jdom.JDOMException;
import org.jdom.input.SAXBuilder;

import dk.itu.ssc.application.hardcoded.AlwaysMatchMapping;
import dk.itu.ssc.exceptions.ConfigurationErrorException;

public class DDReader {
    Logger log = Logger.getLogger(DDReader.class);


    public SipServletApplicationImpl read(InputStream configStream)
    throws ConfigurationErrorException {

        SAXBuilder parser = new SAXBuilder();
        parser.setEntityResolver(new SipXmlEntityResolver());
        parser.setValidation(true);

        Document doc = null;
        try {
            doc = parser.build(configStream);
        } catch (JDOMException e) {
            log.error("Could not parse DeploymentDescriptor", e);
            return null;
        } catch (IOException e) {
            log.error("Could not parse DeploymentDescriptor", e);
            return null;
        }

        return interpet(doc);
    }

    public SipServletApplicationImpl read(File configFile)
    throws ConfigurationErrorException {

        SAXBuilder parser = new SAXBuilder();
        parser.setEntityResolver(new SipXmlEntityResolver());
        parser.setValidation(true);

        Document doc = null;
        try {
            doc = parser.build(configFile);
        } catch (JDOMException e) {
            log.error("Could not parse DeploymentDescriptor", e);
            return null;
        } catch (IOException e) {
            log.error("Could not parse DeploymentDescriptor", e);
            return null;
        }

        return interpet(doc);
    }



    private SipServletApplicationImpl interpet(Document doc) throws ConfigurationErrorException{
        // configure the application
        SipServletApplicationImpl sipApp = new SipServletApplicationImpl();

        Element root = doc.getRootElement();

        if (!root.getName().equals("sip-app")) {
            log.error("missing sip-app root");
            throw new ConfigurationErrorException(
                    "root-element of a deployment descriptor should be \"sipp-app\"");
        }

        // get servlets if any
        SipServletConfiguration firstServletConfig = null; // used by the
        // always-match-mapping
        List servletElements = root.getChildren("servlet");
```

DDReader.java Page 2/3

```
        Iterator i = servletElements.iterator();
        List<SipServletConfiguration> servletConfigs = new ArrayList<SipServletConfiguration>();
        while (i.hasNext()) {
            Element servletElement = (Element) i.next();

            Element servletClass = servletElement.getChild("servlet-class");
            if (servletClass == null) {
                throw new ConfigurationErrorException(
                        "Could not find servlet-class in servlet element!");
            }
            String className = servletClass.getText();
            // attempt to load the servlet
            SipServlet sipServlet = null;
            try {
                sipServlet = SipServlet.class.cast(Class.forName(className)
                        .newInstance());
            } catch (ClassCastException e) {
                log.error("Could not cast servlet-class", e);
                throw new ConfigurationErrorException(
                        "Servlet-class instance could not be casted to a SipServlet");
            } catch (ClassNotFoundException e) {
                log.error("Could not find servlet-class", e);
                throw new ConfigurationErrorException(
                        "Could not find Servlet class");

            } catch (InstantiationException e) {
                log.error("Could not instansiate servlet class", e);
                throw new ConfigurationErrorException(
                        "Could not instansiate servlet class");
            } catch (IllegalAccessException e) {
                log.error("Could not instansiate servlet class", e);
                throw new ConfigurationErrorException(
                        "Could not instansiate servlet class");
            }

            // create a new servlet configuration and add it to the lists
            SipServletConfiguration servletConfiguration = new SipServletConfiguration();
            servletConfigs.add(servletConfiguration);

            // used by the always-match-mapping
            if (firstServletConfig != null) {
                firstServletConfig = servletConfiguration;
            }

            // populate the configuration
            servletConfiguration.setServlet(sipServlet);

            servletConfiguration.setName(servletElement
                    .getChild("servlet-name").getText());

            // get init params if any
            List initParamList = servletElement.getChildren("init-param");
            Map<String, Object> servletParams = new HashMap<String, Object>();


            for (Iterator pi = initParamList.iterator(); pi.hasNext();) {
                Element initParam = (Element)pi.next();

                List initParamElements = initParam.getChildren();
                for (Iterator iter = initParamElements.iterator(); iter
                        .hasNext();) {
                    Element key = (Element) iter.next();
                    if (!key.getName().equals("param-name")) {
                        throw new ConfigurationErrorException(
                                "expected a param-name element, got "
                                        + key.getName());
                    }

                    if (!iter.hasNext()) {
                        log
                                .error("Could not read init-params: name must be followed by a value");
                        throw new ConfigurationErrorException(
                                "Could not read init-params: name must be followed by a value");
                    }

                    Element val = (Element) iter.next();
                    if (!val.getName().equals("param-value")) {
                        throw new ConfigurationErrorException(
                                "expected a param-value element, got "
                                        + val.getName());
                    }

                    // all is good
                    if (key.getText().length() > 0
                            && val.getText().length() > 0) {
                        servletParams.put(key.getText(), val.getText());
                        log.debug("Added servlet parameter " + key.getText()
```

DDReader.java


```
                            + "~>" + val.getText());
                    }
                }
            }
            servletConfiguration.setInitParam(servletParams);
        }

        sipApp.setServletConfigs(servletConfigs);

        // mappings?
        List<ServletMapping> servletMappings = new ArrayList<ServletMapping>();
        if (firstServletConfig != null) {
            ServletMapping mapping = new AlwaysMatchMapping(firstServletConfig);
            servletMappings.add(mapping);
        }

        sipApp.setMappings(servletMappings);

        return sipApp;
    }
}
```

Dispatcher.java


```
package dk.itu.ssc.application;

import java.io.IOException;

import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.sip.SipServlet;


public class Dispatcher implements RequestDispatcher {

    SipServlet servlet;
    SipServletApplication sipServletApplication;


    public Dispatcher() {
        // TODO Auto-generated constructor stub
    }

    public Dispatcher(SipServlet target, SipServletApplication application) {
        this.servlet = target;
        this.sipServletApplication = application;
    }

    public void setServlet(SipServlet servlet) {
        this.servlet = servlet;
    }

    public void forward(ServletRequest req, ServletResponse res)
            throws ServletException, IOException {
        servlet.service(req, res);
    }

    public void include(ServletRequest req, ServletResponse res)
            throws ServletException, IOException {
        throw new ServletException("SipServlets cannot be included");
    }

    /**
     * @return Returns the sipServletApplication.
     */
    public SipServletApplication getSipServletApplication() {
        return sipServletApplication;
    }

    /**
     * @param sipServletApplication The sipServletApplication to set.
     */
    public void setSipServletApplication(SipServletApplication sipServletApplication) {
        this.sipServletApplication = sipServletApplication;
    }

}
```

MappingPattern.java


```java
package dk.itu.ssc.application;

import javax.sip.message.Request;

public interface MappingPattern {
    public boolean matches(Request request);
}
```

ServletMapping.java


```java
package dk.itu.ssc.application;

public interface ServletMapping {

    /**
     * @return Returns the pattern.
     */
    public abstract MappingPattern getPattern();

    /**
     * @return Returns the servlet.
     */
    public abstract SipServletConfiguration getServletConfiguration();

}
```

SipServletApplication.java


```java
package dk.itu.ssc.application;

import java.util.Iterator;

public interface SipServletApplication {

    /**
     * Returns an Iterator that iterates all mappings in the
     * SipServletApplication
     *
     * @return The Iterator
     */
    public abstract Iterator<ServletMapping> getMappings();

    /**
     * Returns an iterator that iterates trough each SipServlet configuration
     *
     * @return The iterator
     */
    public abstract Iterator<SipServletConfiguration> getConfigurations();

    /**
     * Retrives the configuration for a specific Servlet
     *
     * @param name
     *            the SipServlets name
     * @return the configuration
     */
    public abstract SipServletConfiguration getServletConfig(String name);

    /**
     * Returns an identifier that must be unique across all active applications
     *
     * @return the ID
     */
    public abstract String getId();

}
```

SipServletApplicationImpl.java


```java
package dk.itu.ssc.application;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import dk.itu.ssc.util.IDGenerator;

public class SipServletApplicationImpl implements SipServletApplication {

    List<SipServletConfiguration> servletConfigs;

    List<ServletMapping> mappings;

    String id;

    public SipServletApplicationImpl() {
        this.id = "SipApp-" + IDGenerator.generate();
    }

    /*
     * (non-Javadoc)
     *
     * @see dk.itu.ssc.SipServletApplication#getMappings()
     */
    public Iterator<ServletMapping> getMappings() {
        return mappings.iterator();
    }

    /**
     * @param mappings
     *            The mappings to set.
     */
    protected void setMappings(List<ServletMapping> mappings) {
        this.mappings = mappings;
    }

    /*
     * (non-Javadoc)
     *
     * @see dk.itu.ssc.application.SipServletApplication#getServletConfig(java.lang.String)
     */
    public SipServletConfiguration getServletConfig(String name) {
        // go through the configurations and attempt to find the servlet

        for (SipServletConfiguration config : servletConfigs) {
            if (config.getName() != null && config.getName().equals(name)) {
                return config;
            }
        }

        return null;
    }

    /*
     * (non-Javadoc)
     *
     * @see dk.itu.ssc.application.SipServletApplication#getConfigurations()
     */
    public Iterator<SipServletConfiguration> getConfigurations() {
        return servletConfigs.iterator();
    }



    /**
     * @param servletConfigs The servletConfigs to set.
     */
    public void setServletConfigs(List<SipServletConfiguration> servletConfigurations) {
        this.servletConfigs = servletConfigurations;
    }

    /*
     * (non-Javadoc)
     *
     * @see dk.itu.ssc.application.SipServletApplication#getId()
     */
    public String getId() {
        return this.id;
    }

}
```

SipServletConfiguration.java


```
package dk.itu.ssc.application;

import java.util.HashMap;
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;

import javax.servlet.sip.SipServlet;

public class SipServletConfiguration {
    private String name;

    private SipServlet servlet;

    private Map<String, Object> initParam;

    public SipServletConfiguration() {
        initParam = new ConcurrentHashMap<String, Object>();
    }

    /**
     * @return Returns the initParam.
     */
    public Map getInitParam() {
        return initParam;
    }

    /**
     * @return Returns the name.
     */
    public String getName() {
        return name;
    }

    /**
     * @return Returns the servlet.
     */
    public SipServlet getServlet() {
        return servlet;
    }

    /**
     * @param initParam
     *            The initParam to set.
     */
    public void setInitParam(Map<String, Object> initParam) {
        this.initParam = initParam;
    }

    /**
     * @param name
     *            The name to set.
     */
    public void setName(String name) {
        this.name = name;
    }

    /**
     * @param servlet
     *            The servlet to set.
     */
    public void setServlet(SipServlet servlet) {
        this.servlet = servlet;
    }

}
```

SipXmlEntityResolver.java


```
package dk.itu.ssc.application;

import java.io.InputStream;

import org.apache.log4j.Logger;
import org.xml.sax.EntityResolver;
import org.xml.sax.InputSource;

class SipXmlEntityResolver implements EntityResolver {
    private static final Logger log = Logger
            .getLogger(SipXmlEntityResolver.class);

    /**
     * SIP Servlet DTD public ID. Appears in sip.xml files and must be resolved.
     */
    private static final String SIP_XML_PUBLIC_ID = "-//Java Community Process//DTD SIP Application 1.0//EN";

    /** Name of the SIP servlet DTD resource in the CLASSPATH. */
    private static String SIP_XML_RES = "/javax/servlet/sip/resources/sip-app_1_0.dtd";

    public InputSource resolveEntity(String publicId, String systemId) {
        if (publicId.equals(SIP_XML_PUBLIC_ID)) {
            try {
                InputStream in = SipXmlEntityResolver.class
                        .getResourceAsStream(SIP_XML_RES);
                if (in != null) {
                    log.debug("resolved entity \"" + publicId + "\" to \""
                            + SIP_XML_RES + "\"");
                    return new InputSource(in);
                } else {
                    log.error("sip-app_1_0.dtd resource not available");
                    return null;
                }
            } catch (Exception ex) {
            }
        }
        return null;
    }
}
```

AlwaysMatchMapping.java

Page 1/1

```
package dk.itu.ssc.application.hardcoded;

import dk.itu.ssc.application.MappingPattern;
import dk.itu.ssc.application.ServletMapping;
import dk.itu.ssc.application.SipServletConfiguration;


public class AlwaysMatchMapping implements ServletMapping {

    SipServletConfiguration servlet;

    MappingPattern pattern;

    public AlwaysMatchMapping(SipServletConfiguration servlet) {
        this.servlet = servlet;
        this.pattern = new ConstantPattern(true);
    }

    public MappingPattern getPattern() {
        return pattern;
    }

    public SipServletConfiguration getServletConfiguration() {
        return servlet;
    }

    /**
     * @param pattern
     *            The pattern to set.
     */
    public void setPattern(MappingPattern pattern) {
        this.pattern = pattern;
    }

    /**
     * @param servlet The servlet to set.
     */
    public void setServlet(SipServletConfiguration servlet) {
        this.servlet = servlet;
    }

}
```

ApplicationConfiguration.properties

Page 1/1

```
application.name = "Test Application 1"
application.servlet.testservlet = dk.itu.ssc.application.servlet.DefaultServlet
application.servlet.testservlet.param.muller = "fisk"
application.servlet.testservlet.param.and = "giraf"
```

ConstantPattern.java


```
package dk.itu.ssc.application.hardcoded;

import javax.sip.message.Request;

import dk.itu.ssc.application.MappingPattern;

public class ConstantPattern implements MappingPattern {

    private boolean flag;

    public ConstantPattern(boolean flag) {
        this.flag = flag;
    }

    public boolean matches(Request request) {
        return flag;
    }

}
```

DummySipServletApplication.java


```
/**
 *
 */
package dk.itu.ssc.application.hardcoded;

import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.Properties;

import javax.servlet.ServletException;
import javax.servlet.sip.SipServlet;

import org.apache.log4j.Logger;

import dk.itu.ssc.SipServletContainer;
import dk.itu.ssc.application.ServletMapping;
import dk.itu.ssc.application.SipServletApplication;
import dk.itu.ssc.application.SipServletConfiguration;
import dk.itu.ssc.servlet.ServletConfigImpl;
import dk.itu.ssc.servlet.ServletContextImpl;
import dk.itu.ssc.tests.TCKTest;

public class DummySipServletApplication implements SipServletApplication {

    ArrayList<ServletMapping> mappings;

    ArrayList<SipServletConfiguration> configurations;

    String id = "dummy";

    Logger log = Logger.getLogger(DummySipServletApplication.class);

    public DummySipServletApplication() {
        configurations = new ArrayList<SipServletConfiguration>();
        mappings = new ArrayList<ServletMapping>();

        // setup the dummy servlet
        SipServlet test = new TCKTest();
        //SipServlet test = new Context();

        // create a servlet context
        ServletContextImpl context = new ServletContextImpl(test.getClass().getName());
        ServletConfigImpl config = new ServletConfigImpl(context);

        try {
            test.init(config);
        } catch (ServletException e) {
            log.fatal("Could not configure servlet " + test);
            SipServletContainer.getInstance().shutdown(true);
        }


        // create a configuration for it
        SipServletConfiguration configuration = new SipServletConfiguration();
        configuration.setName("Test Servlet");
        configuration.setServlet(test);

        // add the configuration to the configurations and mapping lists
        configurations.add(configuration);
        mappings.add(new AlwaysMatchMapping(configuration));
    }

    public Iterator<ServletMapping> getMappings() {
        return mappings.iterator();
    }

    public SipServletConfiguration getServletConfig(String name) {
        // we always return the dummy-servlet
        return configurations.get(0);
    }

    public Iterator<SipServletConfiguration> getConfigurations() {
        return configurations.iterator();
    }

    public String getId() {
        return id;
    }
}
```

HarcodedApplicationStrategy.java Page 1/2

```
package dk.itu.ssc.application.hardcoded;

import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Iterator;

import javax.servlet.ServletException;
import javax.servlet.sip.SipServlet;

import org.apache.log4j.Logger;

import dk.itu.ssc.ServerContext;
import dk.itu.ssc.SipServletContainer;
import dk.itu.ssc.application.ApplicationManagementStrategy;
import dk.itu.ssc.application.DDReader;
import dk.itu.ssc.application.Dispatcher;
import dk.itu.ssc.application.SipServletApplicationImpl;
import dk.itu.ssc.application.SipServletConfiguration;
import dk.itu.ssc.exceptions.ConfigurationErrorException;
import dk.itu.ssc.network.wrapper.SipFactoryImpl;
import dk.itu.ssc.network.wrapper.SipServletRequestImpl;
import dk.itu.ssc.network.wrapper.TimerServiceImpl;
import dk.itu.ssc.servlet.ServletConfigImpl;
import dk.itu.ssc.servlet.ServletContextImpl;
import dk.itu.ssc.session.SessionManager;

/**
 * A dummy-application manager that will always route to the same application
 * containing the default Servlet
 *
 * @author Mads Danquah
 *
 */
public class HarcodedApplicationStrategy implements
        ApplicationManagementStrategy {
    Logger log = Logger.getLogger(HarcodedApplicationStrategy.class);

    SipServletApplicationImpl appInstance = null;

    public Dispatcher route(SipServletRequestImpl request) {
        // get the first (and only) servlet and wrap it in a dispatcher
        SipServletConfiguration servletconfig = appInstance.getConfigurations()
                .next();

        return new Dispatcher(servletconfig.getServlet(), appInstance);
    }

    public void configure(ServerContext sc) {
        // get the configuration
        String ddFilename = sc.configuration
                .get("applicationmanager.hardcoded.dd");
        if (ddFilename == null) {
            log
                    .error("Could not find deploymentDescriptor in configuration!, DD should be defined by applicationmanager.hardcoded.dd");
            SipServletContainer.getInstance().shutdown(true);
        }

        // read the DD
        File dd = new File(ddFilename);
        // parse the configuration
        DDReader ddReader = new DDReader();
        try {
            this.appInstance = ddReader.read(dd);
        } catch (ConfigurationErrorException e) {
            log.error("Could not configure application", e);
            SipServletContainer.getInstance().shutdown(true);
        }

        // initialize Servlets
        Iterator<SipServletConfiguration> servletConfigs = this.appInstance
                .getConfigurations();

        // create a session for the application
        SessionManager.createApplicationSession(appInstance);

        while (servletConfigs.hasNext()) {
            SipServletConfiguration servletConfig = (SipServletConfiguration) servletConfigs
                    .next();

            SipServlet servlet = servletConfig.getServlet();
            String servletName = servlet.getClass().getName();
            ServletContextImpl context = new ServletContextImpl(servletName);
            ServletConfigImpl config = new ServletConfigImpl(context);

            // create tempoary directory
```

HarcodedApplicationStrategy.java Page 2/2

```
            try {
                File tmpFile = File.createTempFile("sipapp", "");
                tmpFile.delete();
                if (!tmpFile.mkdir()) {
                    log.error("Could not create tempoary directory at "
                            + tmpFile);
                    SipServletContainer.getInstance().shutdown(true);
                }

                log.info("Created tempoary directory " + tmpFile);
                context.setAttribute("javax.servlet.context.tempdir", tmpFile);
                log.info("Test: " + context
                        .getAttribute("javax.servlet.context.tempdir"));
            } catch (IOException e1) {
                log.error("Could not create tempoary directory", e1);
                SipServletContainer.getInstance().shutdown(true);
            }

            File tempDir = (File) context
                    .getAttribute("javax.servlet.context.tempdir");
            if (tempDir == null) {
                log.error(
                        "no tempdir under \"javax.servlet.context.tempdir\"");
                SipServletContainer.getInstance().shutdown(true);
            }

            context.setAttribute("javax.servlet.sip.SipFactory", SipFactoryImpl.getInstance());
            context.setAttribute("javax.servlet.sip.TimerService", new TimerServiceImpl());
                    context.setAttribute("javax.servlet.sip.supported", Collections.unmodifiableList(new ArrayList<
String>()));


            log.debug("Configured " + servlet.getClass().toString());

            try {
                servlet.init(config);
            } catch (ServletException e) {
                log.fatal("Could not configure servlet " + servlet, e);
                SipServletContainer.getInstance().shutdown(true);
            }
        }

    }
}
```