

UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Štefan Šimec

Konvergenca spletnih storitev in telekomunikacijskih storitev

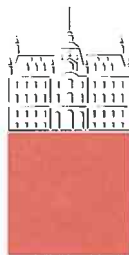
DIPLOMSKO DELO
NA UNIVERZITETNEM ŠTUDIJU

Mentor: dr. Andrej Brodnik

Ljubljana, 2013

Rezultati diplomskega dela so intelektualna lastnina Fakultete za računalništvo in informatiko Univerze v Ljubljani. Za objavljanje ali izkoriščanje rezultatov diplomskega dela je potrebno pisno soglasje Fakultete za računalništvo in informatiko ter mentorja.

Besedilo je oblikovano z urejevalnikom besedil $\text{\LaTeX}$.



Št. naloge: 01875/2012

Datum: 03.10.2012

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **ŠTEFAN ŠIMEC**

Naslov: **KONVERGENCA SPLETNIH STORITEV IN TELEKOMUNIKACIJSKIH STORITEV**

THE CONVERGENCE OF WEB SERVICES AND TELECOMMUNICATION SERVICES

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Nista dva svetova - svet je en sam. V prejšnjem stoletju sta se svet medmrežja in svet telekomunikacij razvijala relativno neodvisno eden od drugega. Šele koncem devetdesetih letih s sprejetjem protokola IP kot vseprisotnega mrežnega protokola sta se svetova pričela približevati. Ob tem je vsak od svetov razvijal svoje storitve ter jih ponujal svojim uporabnikom.

V diplomski nalogi preučite možnost vzajemnega nujenja storitev (konvergenco) med svetom IP in telekomunikacijskim svetom. Predlagajte arhitekturo, ki bo omogočala preprosto dodajanje vzajemno nujenih storitev. Arhitektura naj bo razširljiva, modularna in neodvisna od osnovne platforme. Arhitekturo implementirajte in evalvirajte na primeru storitve IP telefonije.

Mentor:

doc. dr. Andrej Brodnik



Dekan:

prof. dr. Nikolaj Zimic

IZJAVA O AVTORSTVU

diplomskega dela

Spodaj podpisani Štefan Šimec,

z vpisno številko 63060230,

sem avtor diplomskega dela z naslovom: Konvergenca spletnih storitev in
telekomunikacijskih storitev

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom dr. Andreja Brodnika
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 15.4.2013

Podpis avtorja:

Zahvala

Zahvalil bi se mentorju dr. Andreju Brodniku za mentorstvo, pomoč in lektoriranje diplomskega dela. Zahvalil bi se tudi mag. Petru Brajaku iz podjetja Medius za idejno in materialno podporo.

Staršem

Kazalo

Povzetek	1
Abstract	2
1 Uvod	3
2 Predstavitev konvergenčnih protokolov, arhitektur in orodij	5
2.1 Platforma za zagotavljanje storitev	5
2.2 Storitveno usmerjena arhitektura	7
2.3 Session Initiation Protocol	9
2.3.1 Kaj je SIP?	9
2.3.2 Kako deluje?	10
2.4 Okolja in orodja	17
2.4.1 Java™	17
2.4.2 JBoss AS	17
2.4.3 Mobicents SIP Servlets	17
2.4.4 Seam	20
2.5 Metrike	20
2.5.1 Uporabnost	21
2.5.2 Modularnost	22
2.5.3 Prenosljivost	22
3 WebPhone	23
3.1 Ideja	23
3.1.1 B2BUA	23
3.1.2 DTMF	24
3.2 Arhitektura	26
3.3 Delovanje	27
3.3.1 Klicanje	27
3.3.2 Obvestila	32

3.3.3	Razred <i>EventHandler</i>	32
4	Evalvacija	42
4.1	Razširitev – povezava v telefonsko omrežje	42
4.1.1	Asterisk	42
4.1.2	Izvedba	44
4.2	Razširitev – nova funkcionalnost	45
4.3	Prenosljivost	47
4.3.1	EAR	47
4.3.2	Namestitev na drug strežnik	47
4.3.3	Sprememba programskega okolja	47
4.3.4	Sprememba operacijskega sistema ali strojne konfiguracije	48
5	Zaključek	50

Seznam uporabljenih kratic in simbolov

AJAX (*ang.: Asynchronous JavaScript and XML*) – skupina tehnik za asinhrono komunikacijo odjemalec-strežnik

B2BUA (*ang.: Back-to-Back User Agent*) – vmesni komunikacijski člen, ki razdeli povezavo v dva ločena dela

BPM (*ang.: Business Process Management*) – poslovni pristop, ki promovira prilagajanje poslovnega procesa potrebam strank

CRLF (*ang.: Carriage Return Line Feed*) – ukaz za prehod v novo vrstico

CRM (*ang.: Customer Relationship Management*) – sistem za upravljanje odnosov s strankami

DFT (*Diskretna Fourierjeva transformacija*) – matematična transformacija, ki pretvori vzorce signala v frekvenčni prostor

DNS (*ang.: Domain Name System*) – porazdeljen hierarhični sistem za prevažanje imen naprav v IP naslove

DSL (*ang.: Digital Subscriber Line/Loop*) – tehnologije za zagotavljanje povezave preko telefonske linije

DTMF (*ang.: Dual-Tone Multi-Frequency Signaling*) – telefonska signalizacija s pomočjo sinteze dveh tonov v slušnem spektru

EAR (*ang.: Enterprise Archive*) – datotečni arhiv, ki ga Java EE uporablja za pakiranje namestitvenih modulov

EJB (*ang.: Enterprise JavaBeans*) – arhitektura za razvoj modularnih poslovnih aplikacij v Javi

- ERP** (*ang.: Enterprise Resource Planning*) – sistem za avtomatizacijo vodenja poslovanja
- FFT** (*ang.: Fast Fourier Transform*) – algoritem za izračun DFT-ja in inverzne transformacije
- FTTH** (*ang.: Fiber to the Home*) – omrežje, ki uporablja optična vlakna od ponudnika do stranke
- GSM** (*ang.: Global System for Mobile Communications*) – družina komunikacijskih protokolov za mobilne telefone
- HTML** (*ang.: HyperText Markup Language*) – jezik za izdelavo spletnih strani, ki jih prikaže spletni brskalnik
- HTTP** (*ang.: HyperText Transfer Protocol*) – aplikacijski protokol za komunikacijo med odjemalcem in strežnikom na svetovnem spletu
- IETF** (*ang.: Internet Engineering Task Force*) – neodvisna organizacija za razvoj internetnih standardov
- IM** (*ang.: Instant Messaging*) – oblika komunikacije, ki omogoča hitro izmenjavo tekstovnih sporočil
- IP** (*ang.: Internet Protocol*) – komunikacijski protokol za usmerjevanje podatkovnih paketov v internetu
- JAR** (*ang.: Java Archive*) – arhiv, ki združuje javanske razrede in pripadajoče metapodatke
- JIT** (*ang.: Just in Time*) – vrsta prevajalnikov, ki vnaprej prevajajo dele bajtne kode, ko se ta potrebuje
- JPA** (*ang.: Java Persistence API*) – okolje v Javi za upravljanje s podatki v obliki javanskih objektov
- JSF** (*ang.: JavaServer Faces*) – platforma za izdelavo uporabniških vmesnikov javanskih spletnih aplikacij
- JSP** (*ang.: JavaServer Pages*) – tehnologija za izdelavo dinamičnih HTML strani s pomočjo Jave
- JSON** (*ang.: JavaScript Object Notation*) – standard za izmenjevanje linearnih sporočil, ki predstavljajo objekte

- JVM** (*ang.: Java Virtual Machine*) – navidezni stroj, ki omogoča izvajanje javanske bajtne kode
- LTE** (*ang.: Long Term Evolution*) – četrta generacija protokolov za mobilne telefone
- MGCP** (*ang.: Media Gateway Control Protocol*) – protokol za upravljanje z vrati v IP in telefonskih omrežjih
- PSTN** (*ang.: Public Switched Telephone Network*) – svetovno telefonsko omrežje
- REST** (*ang.: Representational State Transfer*) – arhitekturni standard za sisteme spletnih storitev
- RFC** (*ang.: Request for Comments*) – memorandum, izdan s strani IETF, ki opisuje inovacijo na področju interneta. RFC lahko IETF pozneje sprejme kot internetni standard.
- RPC** (*ang.: Remote Procedure Call*) – protokol za izvajalne klice v druge naslovne prostore
- RTP** (*ang.: Real-time Transport Protocol*) – paketni format za prenos zvoka in videa preko IP omrežja, pri čemer ima uporabnik občutek prejema v realnem času
- SDP** (*ang.: Service Delivery Platform*) – množica programskih komponent, ki združeno zagotavljajo določene telekomunikacijske storitve
- SDP** (*ang.: Session Description Protocol*) – format za opisovanje parametrov za pretok medijske vsebine
- SIP** (*ang.: Session Initiation Protocol*) – signalizacijski protokol za upravljanje s komunikacijskimi sejami v IP omrežju
- SMTP** (*ang.: Simple Mail Transfer Protocol*) – protokol za prenos elektronske pošte preko IP omrežij
- SOA** (*ang.: Service-Oriented Architecture*) – množica metodologij za načrtovanje in razvoj programske opreme v obliki med seboj povezljivih storitev
- SOAP** (*ang.: Simple Object Access Protocol*) – protokol za izmenjavo podatkov med spletnimi storitvami

SS7 (*ang.: Signalling System No. 7*) – nabor signalizacijskih protokolov, ki se uporabljajo v telefonskih omrežjih

TTS (*ang.: Text-to-Speech*) – sintetizator govora iz besedila

UDP (*ang.: User Datagram Protokol*) – enostaven protokol za prenos podatkov preko IP omrežja, brez rokovanja in zagotavljanja prejetja paketov

UMTS (*ang.: Universal Mobile Telecommunications System*) – tretja generacija protokolov za mobilne telefone

URI (*ang.: Uniform Resource Identifier*) – niz znakov, ki enolično identificira določen vir

USSD (*ang.: Unstructured Supplementary Service Data*) – protokol za povezovanje GSM telefonov in GSM operaterjev

VoIP (*ang.: Voice over IP*) – tehnologije in metodologije prenosa govora preko IP omrežja

WAR (*ang.: Web Application Archive*) – JAR arhiv, ki vsebuje JSP-je, servlete, XML datoteke, ter statične spletne vsebine

Wi-Fi - sistem, ki nudi IP omrežje preko radijskih valov

WSDL - (*ang.: Web Services Description Language*) – jezik, ki opisuje funkcionalnost spletne storitve

XML - (*ang.: Extensive Markup Language*) – jezik za binarni zapis sporočil, ki omogoča označevanje elementov

XMPP - (*ang.: Extensible Messaging and Presence Protocol*) – komunikacijski protokol, ki za izmenjavo sporočil uporablja XML.

Kazalo slik

2.1	Implementacija brez SDP.	6
2.2	Implementacija s SDP.	7
2.3	SOA komunikacija.	8
2.4	Položaj SIP-a v TCP/IP hierarhiji.	10
2.5	Omrežne SIP komponente	11
2.6	Primer vzpostavitve SIP klica.	12
2.7	Primer vzpostavitve SIP klica preko SIP posrednikov.	18
2.8	Mobicents SIP Servlets znotraj VoIP platforme	19
3.1	Vzpostavitev klica s pomočjo B2BUA.	25
3.2	Storitvena arhitektura WebPhone aplikacije.	27
3.3	Shema komunikacije elementov WebPhone aplikacije.	28
3.4	Spletni uporabniški vmesnik.	29
3.5	SIP komunikacija pri navadnem klicu.	30
3.6	SIP komunikacija pri konferenčnem klicu.	31
3.7	Komunikacija pri snemanju obvestil.	33
4.1	Arhitektura SIP sistema z uporabo <i>Asterisk</i> centrale	43
4.2	Nov uporabniški vmesnik.	46

Kazalo izvornih kod

2.1	Primer SIP INVITE sporočila.	13
2.2	Primer odgovora s kodo 180.	14
2.3	Primer odgovora s kodo 200.	15
2.4	Primer SIP ACK sporočila.	15
2.5	Primer SIP BYE sporočila.	16
2.6	Primer SIP OK sporočila.	16
2.7	Zajem in proženje dogodkov v Seam okolju.	21
3.1	Primer INVITE zahtevka WebPhone aplikacije	32
3.2	Deklaracije in vhodni parametri razreda <i>EventHandler</i>	34
3.3	Sprejem povabila	35
3.4	Procesiranje odgovora	35
3.5	Pošiljanje povabila v klic	36
3.6	Pošiljanje povabila v konferenco	37
3.7	Določanje izvora odgovora	37
3.8	Sprejem odgovora prvega uporabnika v klicu	38
3.9	Sprejem odgovora drugega uporabnika v klicu	39
3.10	Sprejem odgovora na povabilo v konferenco	40
3.11	Povezava v konferenco in predvajanje obvestil	40
3.12	Detekcija DTMF signalov	41
4.1	Izsek datoteke <i>sip.conf</i>	44
4.2	Izsek datoteke <i>extensions.conf</i>	45
4.3	Spremembe v <i>EventHandler.java</i>	49

Povzetek

V telekomunikacijskem svetu se vse močnejše pojavlja potreba po konvergenčnih rešitvah, ki združujejo različne tipe storitev. Do nedavnega so bile na voljo le monolitne rešitve, ki so bile zaradi tega posledično drage za operaterje. Danes imamo na voljo številna orodja, ki nam olajšajo izdelavo lastnih rešitev. Mnoga med njimi so prosto dostopna.

Kot eden izmed problemov je predstavljen problem konvergence IP telefonije in svetovnega spleta. Cilj diplomske naloge je prikaz konvergenčne rešitve v obliki aplikacije - *WebPhone*, ki uporabnikom omogoča telefonske in spletne storitve, in za izdelavo uporabiti le odprto-kodna orodja in knjižnice. Poleg tega, omogoča rešitev tudi dodatne razširitve v obliki novih storitev.

Velik poudarek je na protokolu SIP, ki omogoča upravljanje z VoIP sejami. V uvodnem delu je s pomočjo enostavnih primerov predstavljeno delovanje SIP-a, v nadaljevanju pa njegova praktična uporaba v aplikaciji. Aplikacija je bila izdelana v programskem jeziku *Java*, s pomočjo orodij *Mobicents*, *Seam* in *JBoss AS*.

Kriteriji, ki smo jih upoštevali pri evalvaciji so uporabnost, modularnost in prenosljivost. Uporabnost smo preverili z izgradnjo povezave v telekomunikacijsko omrežje, modularnost pa z dograditvijo nove funkcionalnosti. Prenosljivost se je izkazala za spremenljivo glede na vrsto prenosa.

V splošnem se je izkazalo, da je s pomočjo izbire pravih protokolov, orodij in arhitektur, razvoj konvergenčne aplikacije razmeroma enostaven.

Ključne besede:

Konvergenca, SIP, VoIP, splet, Java

Abstract

There has been an increasing demand for converged solutions in telecommunication sector. Up until recently only monolithic solutions were available which were consequently expensive. There are several tools at our disposal today, which permit us to develop converged solutions in-house. Many among them are available for free.

In thesis is addressed a problem of convergence of IP telephony and worldwide-web. Our goal is to present a converged solution in form of a web application - *WebPhone*. Application provides telephone and web services to its users and has been developed using exclusively open source tools and libraries. Furthermore, the solution permits further extensions in form of additional services.

A substantial part of the thesis is concentrated on SIP. SIP is a protocol used for handling VoIP sessions. Theoretical background of SIP is described at the beginning of the thesis using simple examples and its practical use inside the application in the following chapters. *WebPhone* application was written in Java using tools *Mobicents*, *Seam* and *JBoss AS*.

During evaluation we took note of usability, modularity and portability. Usability was verified by integrating the application into telephone network and modularity by incorporating a new functionality into the application. Portability was shown to be varying depending on its type.

In general we succeeded to demonstrate the possibility of developing a converged application using the right protocols, tools and libraries.

Key words:

Convergence, SIP, VoIP, worldwide-web, Java

Poglavje 1

Uvod

V devetdesetih letih prejšnjega stoletja se je vedno močneje pojavljala potreba po standardizaciji internetne telefonije (VoIP). Pojavili so se že prvi programi, kot na primer prosto dostopni Linux program *MTALK* ([17]) in prvi komercialni program *Internet Phone 4* ([15]). Telekomunikacijski operaterji so v takšnih programih videli konkurenčno grožnjo in so internetni telefoniji sprva ostro nasprotovali. Ko je bil leta 1999 standardiziran protokol za vzpostavitev sej (SIP) ([11]), je to pomenilo začetek razmaha ponudnikov VoIP. Tudi telefonska omrežja so začela prehajati iz vodovno komutiranega omrežja v IP omrežja. Temu je botrovalo tudi uvajanje digitalnih optičnih povezav. Tako danes večino svetovnega telefonskega omrežja predstavlja IP omrežje.

Danes še vedno gledamo na dejavnosti, kot so telefoniranje, gledanje televizije, brskanje po internetu, kot na povsem ločene storitve, kljub temu, da se podatki prenašajo po istih linijah, preko istih omrežij in z uporabo istih protokolov.

Pojem *konvergenca*, v tehnološkem kontekstu, je prvi uporabil Blackman ([1]), ki je konvergenco definiral kot trend v razvoju tehnoloških storitev in struktur. Pozneje je pojem postal bolj specifičen. Gate ([8]) in Collins ([3]) pojmujeta konvergenco kot združevanje telekomunikacije, računalništva in televizijskega oddajanja. Nekateri, kot na primer Mueller ([18]), konvergenco razumejo celo kot prevzem vseh medijskih storitev s strani računalništva.

Telekomunikacijska podjetja so v devetdesetih letih poleg telefonskih storitev začela ponujati tudi internetne in televizijske storitve. Zaradi težavnosti integracij vedno novih storitev in stroškov vzdrževanja se je pojavila ideja o enotni konvergenčni platformi za vse storitve, ki bi zmanjšala stroške in povečala zadovoljstvo uporabnikov. Problem je predstavljalo tudi dejstvo, da so bile storitvene arhitekture v večini razvite s strani telekomunikacijskih pod-

jetij, oziroma za njih specifično s strani zunanjih razvijalcev. To je pomenilo, da so bile rešitve od podjetja do podjetja različne, kar je še dodatno oteževalo standardizacijo.

Danes imamo na voljo številne protokole, kot je SIP, ter orodja in okolja, ki olajšajo implementacijo konvergenčnih rešitev. Med njimi so mnoga odprto-kodna oziroma prosto dostopna. Široko uveljavljena je tudi storitveno usmerjena arhitektura.

Cilj diplomske naloge je prikazati, kako lahko s pomočjo odprto-kodnih orodij in okolij izdelamo konvergenčno aplikacijo, ki združuje telefonske in spletne storitve. Aplikacija omogoča povezovanje VoIP telefonov preko spletnega vmesnika, ter izvajanje akcij aplikacije s pomočjo telefonskega klica.

Morda se zdi takšna aplikacija nesmiselna, vendar, tudi v današnjem svetu pametnih telefonov in interneta, na vsakem koraku obstajajo primeri, ko bi lahko bila uporabna. Veliko internetnih uporabnikov, na primer, ne zaupa spletnim transakcijam in bi se mogoče počutili varneje, če bi številke kreditnih kartic vnašali prek telefona. Primer uporabe bi lahko bil sistem obveščanja za starejše občane, kjer bi lahko zdravnik preko spletnega vmesnika nastavil časovnik, ki bi ob določenih urah poklical pacienta in ga opozoril, da mora vzeti določeno zdravilo. Drug primer uporabe je spletni telefon, kjer bi lahko uporabnik vzpostavil klic z drugim uporabnikom preko aplikacije, in s tem ohranil svojo anonimnost.

Jedro diplomske naloge predstavlja protokol SIP in njegova implementacija. V drugem poglavju je predstavljen SIP, ter arhitekture, orodja in okolja, ki so uporabljeni pri izdelavi aplikacije. Na koncu poglavja so na kratko predstavljene tudi metrike, ki so uporabljene v evalvaciji aplikacije. V tretjem poglavju je opisana aplikacija, temu pa sledita evalvacija in zaključek.

Poglavje 2

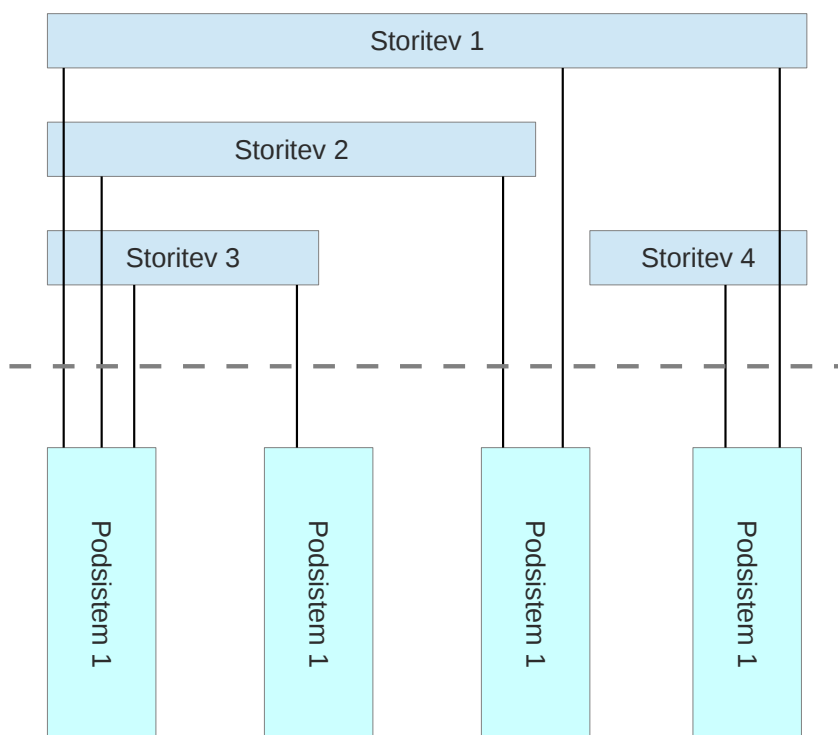
Predstavitev konvergenčnih protokolov, arhitektur in orodij

2.1 Platforma za zagotavljanje storitev

Platforma za zagotavljanje storitev (ang.: *Service Delivery Platform*, v nadaljevanju: SDP) si lahko predstavljamo kot nabor komponent, ki so večinoma, vendar ne nujno, programske, in skupaj gradijo arhitekturo za zagotavljanje storitev. Osnovna naloga SDP je, da igra vlogo vmesnika med podsistemi, ki zagotavljajo prenos podatkov do naročnikov, in storitvami (gl. sliko 2.2). Primeri storitev: telefonski klici, SMS, MMS, elektronska pošta, internet, televizija, video na zahtevo, itd. Podsistemi v telekomunikacijskem okolju so lahko na primer DSL, FTTH, GSM, GPRS, Wi-Fi, UMTS, LTE, itd. SDP omogoča dodajanje novih storitev oz. spremembe že obstoječih, brez potrebe spreminjanja podsistema in obratno. Slika 2.1 prikazuje arhitekturo brez uporabe SDP-ja, kjer je za vsako storitev potrebno poskrbeti za integracijo s podsistemi, ki naj bi podpirali to storitev.

Slika 2.2 prikazuje uporabo SDP arhitekture, kjer je dostop do storitve standardiziran. Poleg lažje implementacije novih storitev, SDP omogoča boljšo optimizacijo uporabe podsistemov, enostavnejše povezovanje z zunanjimi sistemi in lažji nadzor nad delovanjem.

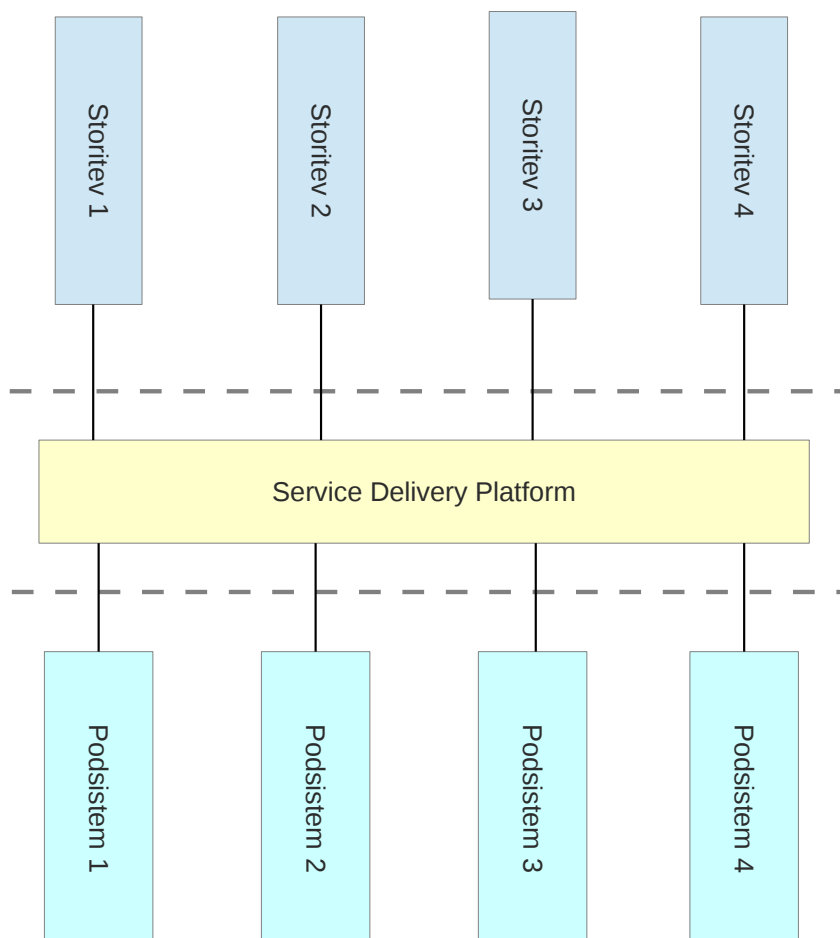
Potrebno je omeniti, da SDP sam po sebi ni standardiziran, in da se razlage, kaj naj bi SDP vseboval in kaj točno naj bi bila njegova vloga od razvijalca do razvijalca razlikujejo. Nekateri, kot na primer podjetje Comsys ([4]) pojmujejo podsisteme kot telekomunikacijska vozlišča, med tem ko so za druge SDP razvijalce, kot na primer Fujitsu ([2]) in Nil ([7]), podsistemi abstraktniješe komponente, na primer CRM, ERP, strežniki, aplikacije, itd. V grobem



Slika 2.1: Implementacija brez SDP.

se avtorji člankov [4], [2] in [7] strinjajo, da naj bi SDP omogočal naslednje:

1. **Okolje za tvorjenje nove storitve** - uporabniki lahko uvajajo nove lastne storitve brez sprememb podsistemov
2. **Okolje za izvajanje storitve** - omogočen je pretok informacij med storitvijo in končnimi uporabniki
3. **Nadzor nad izvajanjem** - upravljalci imajo na voljo aplikacije in orodja, preko katerih lahko nadzorujejo delovanje in upravljajo s SDP-jem
4. **Prostorsko neodvisnost** - SDP mora končnim uporabnikom omogočati storitve ne glede na napravo, ki jo končni uporabnik uporablja za dostop
5. **Enostavno integracijo** - potrebna integracija med posameznimi komponentami SDP-ja in zunanji sistemi je minimalna

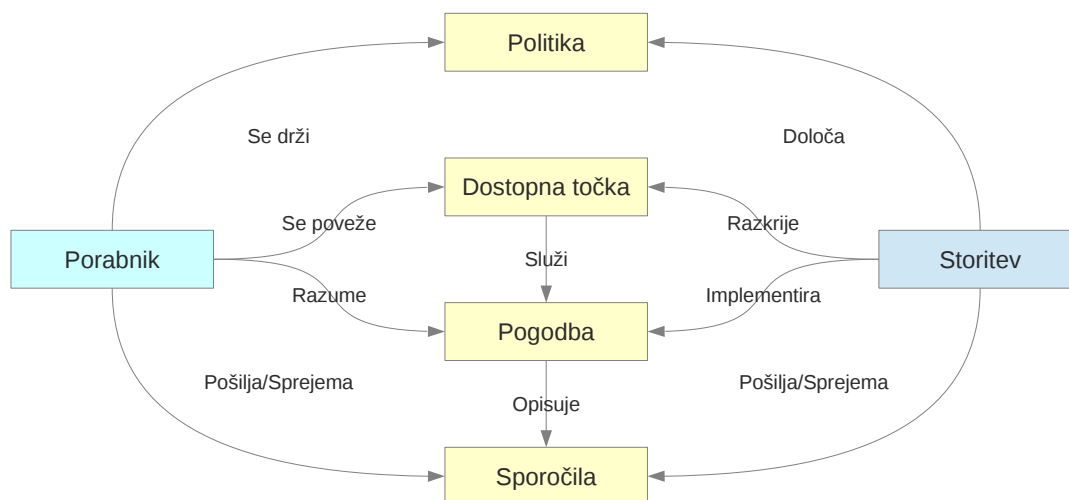


Slika 2.2: Implementacija s SDP.

S standardizacijo SDP-ja se ukvarja združenje podjetij s področja telekomunikacij, imenovano TM Forum, vendar le-ta še ni končana ([21]).

2.2 Storitveno usmerjena arhitektura

Storitveno usmerjena arhitektura (*ang.: Service-oriented Architecture*, v nadaljevanju: SOA) je nabor principov in metodologij, ki se uporabljajo za načrtovanje in razvoj programske opreme z uporabo povezljivih storitev. Storitve so samostojne programske komponente, ki nudijo določene funkcionalnosti. Za njih je značilno zakrivanje in šibka sklopljenost. Posamezne storitve ne



Slika 2.3: SOA komunikacija.

uporabljajo druga druge neposredno, temveč si izmenjujejo sporočila po standardih, kot sta na primer XML in JSON. To omogoča, da so posamezne storitve in porabniki storitev razviti v različnih programskih okoljih. SOA med drugim tudi nudi način za odkrivanje storitev in njihovih funkcionalnosti. Zgradbo sporočila, pričakovano obliko podatkov in komunikacijski protokol storitev tipično objavi v obliki WSDL dokumenta. Komunikacijski protokoli so večinoma SOAP, RPC ali REST.

Shema komunikacije med storitvijo in njenim porabnikom je prikazana na sliki 2.3. Politika, ki jo določa storitev je nabor specifikacij, ki opisuje zmožnosti in varnostne omejitve. Dostopna točka je URL naslov, na katerem je storitev dostopna. Storitvena pogodba definira, na kakšen način bo potekala komunikacija med porabnikom in storitvijo.

Thomas Erl ([6]) definira SOA skozi uporabo osmih principov:

1. **Standardizirana storitvena pogodba** - storitve se držijo komunikacijskih dogovorov, zapisanih v dokumentih, ki opisujejo storitev.
2. **Šibka sklopljenost** - odvisnost med storitvami je minimalna, potrebno je le, da vedo za obstoj druga druge.
3. **Abstrakcija** - programska logika storitve ni vidna navzven.

4. **Ponovna uporaba** - programska logika je razdeljena v storitve z namenom večkratne uporabe.
5. **Avtonomija** - samo posamezna storitev ima nadzor nad svojo programsko logiko.
6. **Odsotnost stanja** - storitev se izogiba hranjenju stanja, ko je to mogoče.
7. **Vidnost** - storitev nudi komunikacijske metapodatke, preko katerih je mogoč dostop do nje.
8. **Sestavljivost** - dodajanje novih storitev ne vpliva na že obstoječe storitve.

Cilj SOA je, da so čim večji deli funkcionalnosti, ki jih obsega arhitektura, implementirani v obliki storitev. Takšna vrsta arhitekture omogoča enostavno ad-hoc implementacijo aplikacij, ki uporabljajo že implementirane funkcionalnosti, ki jih ponujajo storitve. Manjše število storitev pomeni tudi manj integracijskih točk. Glavna prednost SOA pred ostalimi arhitekturami, na primer objektnimi, je v manjši kompleksnosti implementacije velikih sistemov. Manjše število komponent in lokacijska neodvisnost pomeni preprostejšo in bolj pregledno arhitekturo, kar je še posebej privlačno za snovalce sistemov.

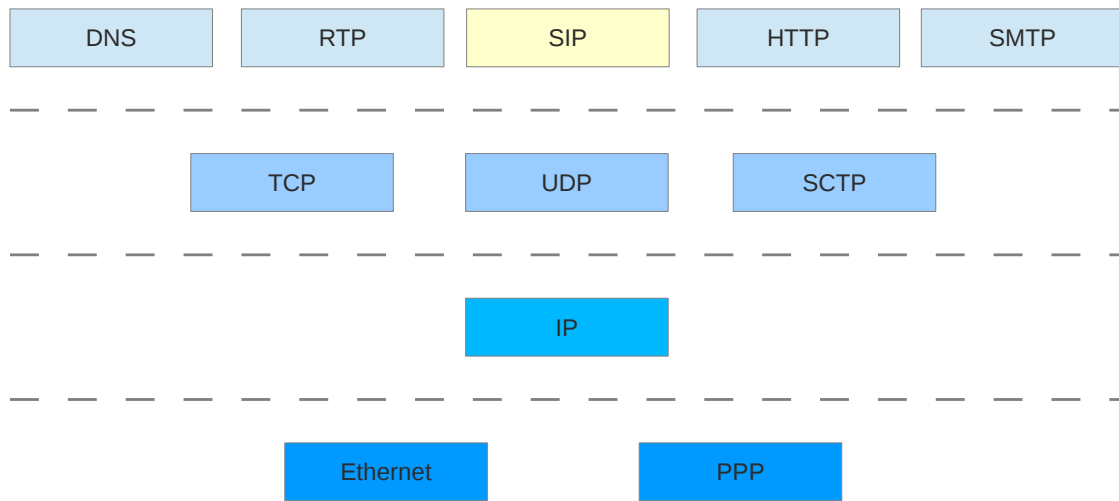
Pomanjkljivost takšne arhitekture je, da lahko storitve postanejo premalo specifične za uporabo. XML vmesnik povzroča tudi določen zmogljivostni strošek, saj potrebuje razpoznavanje vsebine besedila.

2.3 Session Initiation Protocol

2.3.1 Kaj je SIP?

SIP je signalizacijski protokol za upravljanje komunikacijskih sej preko IP omrežja. Razvit je bil v okviru IETF in leta 1999 priznan kot standard RFC 2543 ([11]) ter pozneje posodobljen v RFC 3261 ([20]). Uporablja se za ustvarjanje, spreminjanje in končevanje sej med agenti. SIP agenti so lahko VoIP telefoni, mobilni telefoni, računalniške aplikacije in drugi. Tipično se uporablja pri prenosu govora (VoIP), videa, takojšnjega sporočanja (IM) in iger. Deluje na aplikacijskem sloju, neodvisno od transportnega protokola. Za prenos se sicer običajno uporablja UDP. Umeščenost SIP-a v TCP/IP hierarhiji je prikazana na sliki 2.4.

SIP sporočila so besedila v UTF-8 formatu. Po zgradbi sporočil in delovanju je SIP podoben protokoloma HTTP in SMTP. SIP sam po sebi ne



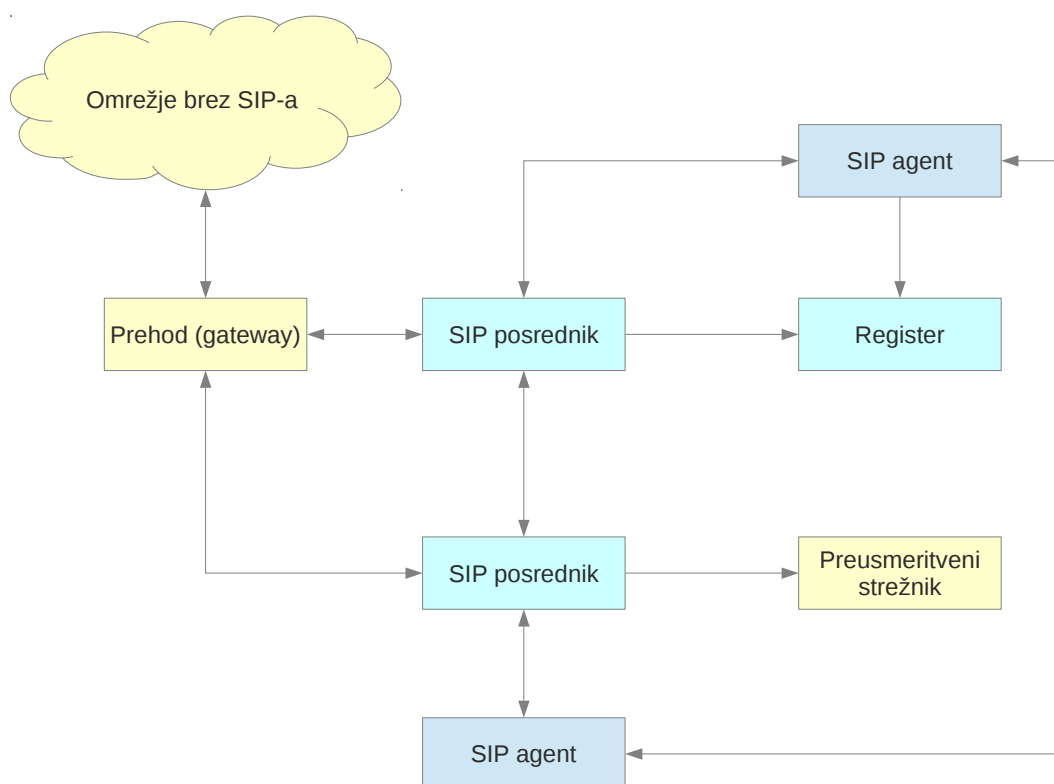
Slika 2.4: Položaj SIP-a v TCP/IP hierarhiji.

definira protokola za prenos podatkov med agenti, omogoča pa, da se za to dogovorijo. V primeru telefonskih oz. video klicev se za to ponavadi uporablja RTP.

Na sliki 2.5 so prikazani osnovne komponente, ki so lahko prisotne v SIP omrežju. Poleg SIP agentov so pogosto prisotni še SIP posredniki, registri, preusmeritveni strežniki in omrežni prehodi (*gateway*). SIP posredniki so posredniki sporočil med SIP agenti. Registri hranijo naslove naprav, na katerih so uporabniki dostopni. SIP posredniki lahko vpogledajo v register in s tem izvejo, kam morajo posredovati sporočilo. Preusmeritveni strežnik je agent, ki vrača odgovore s kodami 3xx (preusmeritve), kar zmanjša obremenitev SIP posrednikov in poveča robustnost. Prehodi se uporabljajo kot vmesniki med SIP omrežjem in omrežjem, ki ne podpira SIP-a, kot na primer PSTN.

2.3.2 Kako deluje?

Delovanje SIP-a najlažje razložimo na primeru. Slika 2.6 prikazuje enostaven primer vzpostavitve in zaključka seje z uporabo SIP protokola. Imamo dve osebi, Ana in Boruta. Komunikacija se začne, ko Ana pošlje Borutu povabilo v obliki INVITE zahtevka. Koda 2.1 je primer SIP sporočila, ki je poslano znotraj UDP datagrama. Sporočilo sestavljajo metapodatki, ločeni po vrsticah in telo



Slika 2.5: Omrežne SIP komponente

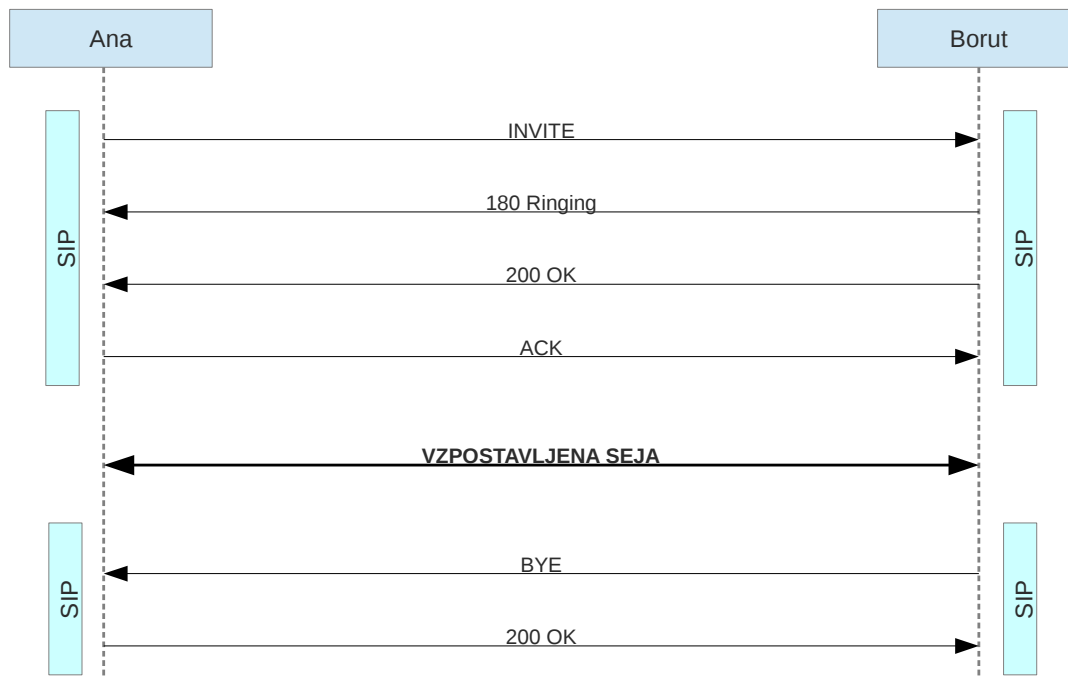
sporočila. Format zapisa metapodatkov je:

glava: podatki CRLF.

Prva vrstica vsebuje ime metode (**INVITE**), SIP URI zahtevka (**sip:borut@bovec.si**) in verzijo SIP-a (**SIP/2.0**). SIP URI je posebna oblika naslova, podobna naslovu za elektronsko pošto. Ponavadi je sestavljen iz uporabniškega imena in naslova strežnika, kjer je uporabnik registriran. Njegova zgradba in uporaba sta natančneje opisana v knjigi [14, str. 100-102]. Parameter **branch** je identifikator transakcije. Vsi odgovori na ta zahtevek imajo enak **branch** parameter.

Druga vrstica vsebuje naslove vseh naprav, ki so posredovale sporočilo. Naslovi so ponavadi v formatu URL oz. so IP naslovi. Tipična SIP vrata so 5060.

Tretja vrstica nam pove maksimalno število posredovanj sporočila, kar služi kot primitivna oblika za detekcijo cikla.



Slika 2.6: Primer vzpostavitve SIP klica.

Naslednji vrstici določata naslovnik in pošiljatelja sporočila. Potrebno je omeniti, da se sporočilo ne usmerja glede na SIP URI naslovnik, ampak SIP URI zahtevka. To pa zaradi tega, ker dejanski naslovnikov naslov ni nujno viden znotraj omrežja, kjer se pošiljatelj nahaja. SIP URI je naslov posrednika sporočila in se z vsakim posredovanjem lahko spremeni. Primer z uporabo posrednikov je predstavljen v naslednjem razdelku.

Call-ID služi kot identifikator seje med Ana in Borutom. Potreben je zaradi primera, ko imata Ana in Borut vzpostavljenih več sej. Označba (*tag*) pošiljateljevega naslova, ki jo Ana tvori ob pošiljanju sporočila, označba naslovnikovega naslova, ki jo tvori Borut ob odgovoru na sporočilo (primer: 2.2) in Call-ID skupaj predstavljajo univerzalni identifikator seje.

Sedma vrstica nam pove tip transakcije in vrstni red zahtevkov. Pri padajoča številka v tem primeru, ko opujemo INVITE metodo, ni pomembna, saj se uporablja samo pri Register metodi.

Via, Max-Forwards, To, From, Call-ID, CSeq so obvezni zapisi v glavi SIP sporočila. V primeru INVITE metode je obvezno tudi Contact polje, ki vsebuje naslov naprave, kjer Ana prejema sporočila.

Naslednji dve vrstici nam povesta, da je telo sporočila tipa SDP, dolžine 158 oktetov. Sledi telo sporočila, ki vsebuje Anin predlog komunikacijskega protokola - v tem primeru je to zvočni RTP, vzorčen s PCMU kodekom s frekvenco 8000Hz.

INVITE sporočilo je tip oz. metoda sporočila-zahtevka. V RFC 3261 ([20]) je definiranih dodatnih 5 osnovnih tipov zahtevkov, ki skupaj sestavljajo jedro standarda. Ti so: REGISTER, BYE, ACK, CANCEL in OPTIONS. V kasnejših dodatkih k RFC 3261 je bilo dodanih še več tipov zahtevkov.

```
INVITE sip:borut@bovec.si SIP/2.0
Via: SIP/2.0/UDP pc33.ankaran.si:5060;branch=z9hG4bK776asdhs
Max-Forwards: 70
To: Borut <sip:borut@bovec.si>
From: Ana <sip:ana@ankaran>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 1 INVITE
Contact: <sip:ana@pc33.ankaran.si>
Content-Type: application/sdp
Content-Length: 158

v=0
o=Ana 2890844526 2890844526 IN IP4 pc33.ankaran.si
s=Phone Call
c=IN IP4 100.101.102.103
t=0 0
m=audio 49170 RTP/AVP 0
a=rtptime:0 PCMU/8000
```

Izvorna koda 2.1: Primer SIP INVITE sporočila.

Naslednje sporočilo na sliki 2.6 je 180 Ringing, ki ga pošlje Borutova naprava Anini napravi, in je primer tipa sporočila-odgovor. Odgovor sporoča, da je naprava prejela INVITE zahtevek, in da Borutu zvoní telefon, mu utripa obvestilo na ekranu oz. katerakoli druga oblika opozorila. V izseku kode 2.2 vidimo, da je večina polj identičnih, kot pri INVITE metodi. V Via polju se doda omrežni naslov naprave, ki je sprejela INVITE sporočilo kot Received parameter. To je ponavadi tisti naslov, ki ga DNS razreši iz SIP URI naslova. Nasprotno kot bi morebiti pričakovali, se naslova To in From ne zamenjata, čeprav je Borut poslal sporočilo Ani. Odgovori na zahteveke namreč ohranjajo smer komunikacije zahtevka, na katerega je bil poslan odgovor. To polju

se doda še Borutov identifikator. `Contact` polje vsebuje Borutov naslov, s katerim lahko Ana direktno komunicira.

Sporočilo s kodo 180 je tip *obvestilnega odgovora*. To so odgovori s trimestnimi kodami, ki se začnejo s številko 1. Sporočilo, ki spremlja kodo odgovora, kot na primer *Ringling*, je poljubno. Poleg *informacijskih odgovorov* poznamo še odgovore o uspehu (2xx), preusmeritvi (3xx), napaki odjemalca (4xx), napaki strežnika (5xx) ter globalni napaki (6xx). Večina odgovorov je povzetih po HTTP. Natančnejši opisi odgovorov so v RFC 3261 ([20, str. 183-192]).

```
SIP/2.0 180 Ringing
Via: SIP/2.0/UDP pc33.ankaran.si:5060;branch=z9hG4bK776asdhds;
recieved=100.101.102.103
To: Borut <sip:borut@bovec.si>;tag=12315912
From: Ana <sip:ana@ankaran>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 1 INVITE
Contact: <sip:borut@srvr66.bovec.si>
Content-Length: 0
```

Izvorna koda 2.2: Primer odgovora s kodo 180.

Ko se Borut oglasi, se Ani pošlje odgovor s kodo 200. Odgovor sporoča, da Borutov agent sprejema parametre medijske seje in obenem pošlje SDP paket, ki nosi informacijo o lastnih medijskih sposobnostih. Samo sporočilo s kodo 2.3 je podobno odgovoru s kodo 180, z izjemo dodanega SDP paketa. Borut bi lahko klic tudi zavrnil in poslal na primer odgovor s kodo 600 (zaseden) ali 603 (zavrnitev).

V naslednjem koraku Ana pošlje Borutu ACK zahtevo 2.4, ki potrjuje medijske parametre, ki jih je Borut poslal v SDP paketu znotraj odgovora s kodo 200, in s tem začne neposredno medijsko komunikacijo preko RTP protokola. Sporočilo ima nov `branch` vrednost, ker je to nov zahtevek in s tem nova transakcija.

Iz primera lahko vidimo, da lahko dva SIP agenta enostavno vzpostavita povezavo, v kolikor vesta za naslov drug drugega. Vidimo lahko tudi, da SIP agenti v dialogu služijo kot odjemalci, ki pošiljajo zahteve, in obenem kot strežniki, ki na zahteve odgovarjajo. Na začetku Ana pošlje Borutu INVITE zahtevek, in Borut vrne Ani odgovor. Na koncu se Borut odloči, da bo pogovor končal, in pošlje Ani BYE zahtevek, na katerega Ana odgovori z odgovorom 200. BYE zahtevek 2.5 ima nov `Branch` zaznamek, ker gre za drugo transakcijo kot

```
SIP/2.0 200 Ok
Via: SIP/2.0/UDP pc33.ankaran.si:5060;branch=z9hG4bK776asdhs;
recieved=100.101.102.103
To: Borut <sip:borut@bovec.si>;tag=12315912
From: Ana <sip:ana@ankaran>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 1 INVITE
Contact: <sip:borut@srvr66.bovec.si>
Content-Type: application/sdp
Content-Length: 155

v=0
o=Borut 2759238342 2759238342 IN IP4 srvr66.bovec.si
s=Phone Call
c=IN IP4 200.201.202.203
t=0 0
m=audio 60000 RTP/AVP 0
a=rtpmap:0 PCMU/8000
```

Izvorna koda 2.3: Primer odgovora s kodo 200.

```
ACK sip:borut@bovec.si SIP/2.0
Via: SIP/2.0/UDP pc33.ankaran.si:5060;branch=z9hG4bK776ghj65fg
Max-Forwards: 70
To: Borut <sip:borut@bovec.si>;tag=12315912
From: Ana <sip:ana@ankaran>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 1 ACK
Contact: <sip:ana@pc33.ankaran.si>
Content-Length: 0
```

Izvorna koda 2.4: Primer SIP ACK sporočila.

INVITE ali ACK. Ker v tem primeru Borut pošilja zahtevek Ani, sta vsebini polj To in From zamenjani. Naslovom pripadajoče tag oznake in Call-ID ostanejo enake kot prej, da lahko Ana točno določi, kateri klic Borut končuje. Ana odgovori z OK sporočilom 2.6.

Slika 2.6 prikazuje vzpostavitev klica neposredno med dvema agentoma, kar pomeni, da Anin agent pozna IP naslov Borutovega agenta. V realnem sta Ana

```
BYE sip:ana@pc33.ankaran.si SIP/2.0
Via: SIP/2.0/UDP srvr66.bovec.si:5060;branch=z9hG4bK77hg39ajhj
Max-Forwards: 70
To: Ana <sip:ana@ankaran>;tag=1928301774
From: Borut <sip:borut@bovec.si>;tag=12315912
Call-ID: a84b4c76e66710
CSeq: 1392 BYE
Content-Length: 0
```

Izvorna koda 2.5: Primer SIP BYE sporočila.

```
SIP/2.0 OK
Via: SIP/2.0/UDP srvr66.bovec.si:5060;branch=z9hG4bK77hg39ajhj;
received=200.201.202.203
Max-Forwards: 70
To: Ana <sip:ana@ankaran>;tag=1928301774
From: Borut <sip:borut@bovec.si>;tag=12315912
Call-ID: a84b4c76e66710
```

Izvorna koda 2.6: Primer SIP OK sporočila.

in Borut povezana samo preko SIP naslova, točnega IP naslova drug drugega pa ne poznata. Na slika 2.7 je model SIP komunikacije, kjer vzpostavitev seje poteka SIP posrednikov. Ana ne ve, kje točno se Borut nahaja, ve pa, da je registriran na strežniku *bovec.si*. Zato pošlje zahtevo svojemu posredniku (*ankaran.si*), ki jo ta posreduje naprej. Odgovori na zahtevek se vračajo nazaj po isti poti. Ko Borut Ani pošlje 200 OK odgovor, Ana s tem zve Borutov dejanski naslov in od tega trenutka dalje lahko komunikacija poteka direktno.

V resnici velikokrat tudi SIP posrednik ne ve dejanskega naslova in mora poslati poizvedbo na registracijski strežnik. S tem je uporabnikom omogočeno, da so dostopni na istem SIP naslovu na različnih napravah. Borut lahko na primer uporablja IM program na računalniku, tablici, pametnem telefonu, itd. Ko se program zažene, prijavi IP naslov v register in tako lahko Ana zve, kateri izmed Borutovih naprav mora poslati sporočilo. Poleg registrov, se v SIP infrastrukturi pojavljajo še *preusmeritveni* strežniki, ki omogočajo poljubno relokacijo ostalih strežnikov, ter *strežnikov-prehodov*, ki služijo kot vstopna/izstopna točka v druga omrežja - na primer prehod iz IP omrežja v

telefonsko omrežje.

2.4 Okolja in orodja

2.4.1 Java™

Java je objektni programski jezik, načrtovan z namenom minimizirati odvisnost programa od ciljne platforme. Njegova posebnost je v tem, da se javanski programi prevajajo v bajtno kodo, ki se izvaja na navideznem stroju (JVM). Takšen koncept omogoča, da programerji pišejo in prevajajo program le enkrat, ta pa se lahko izvaja na poljubnih sistemih, v kolikor na njih teče JVM. JVM kodo izvaja na procesorju s pomočjo prevajalnika vrste JIT. Java ima tudi avtomatsko upravljanje s pomnilnikom.

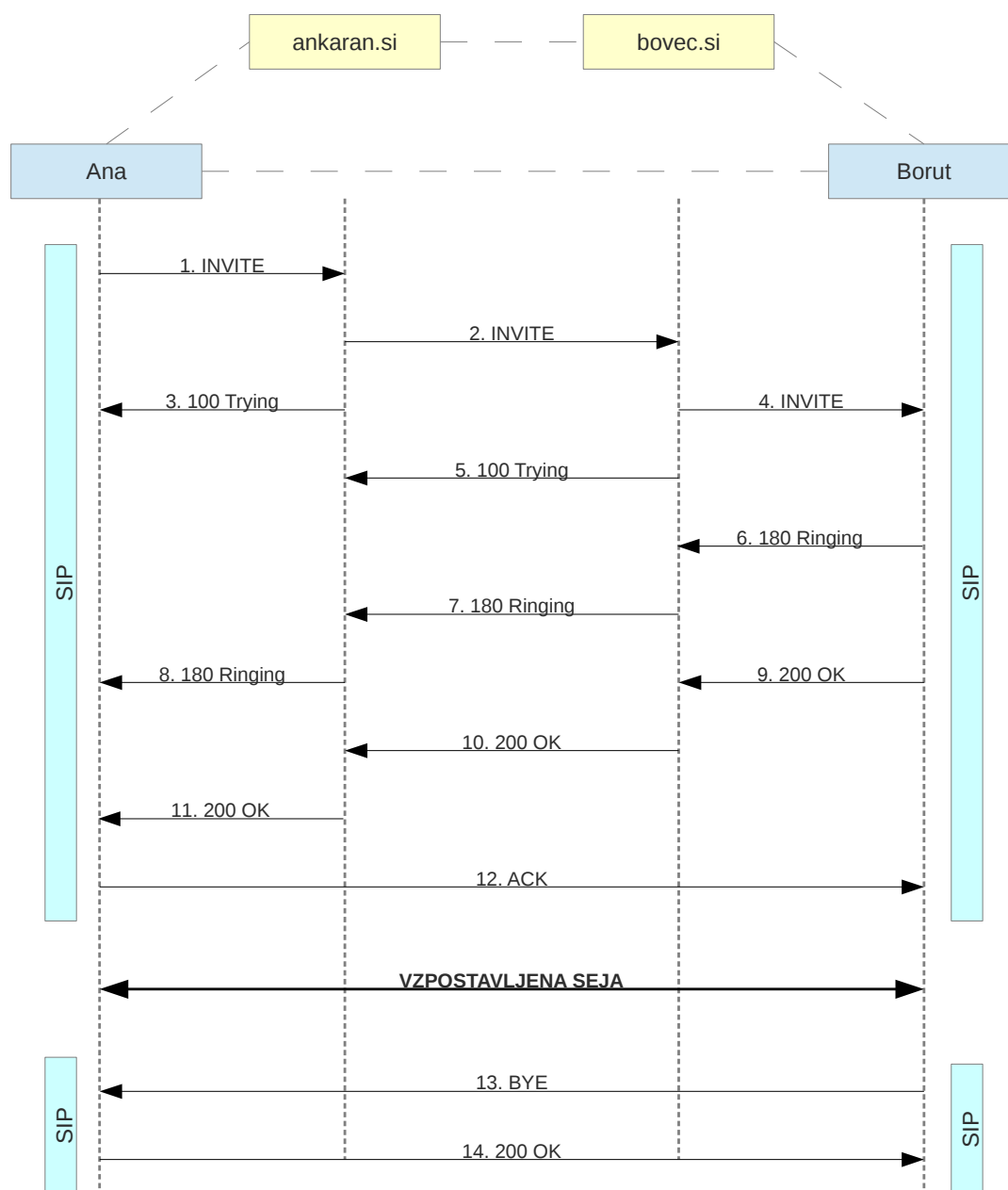
Java je verjetno najpopularnejši programski jezik na področju poslovnih spletnih aplikacij. Razlogi za to tičijo v dejstvu, da je odprto-kodni jezik, v prenosljivosti kode, v široki podpori internetnih standardov, v velikem številu programskih ogrodij in knjižnic ter v širokem naboru aplikacijskih strežnikov, kot so Oracle WebLogic, IBM WebSphere, Apache Tomcat, JBoss AS, Glassfish, itd. Specifikacija Jave verzije 7 je objavljena v [10].

2.4.2 JBoss AS

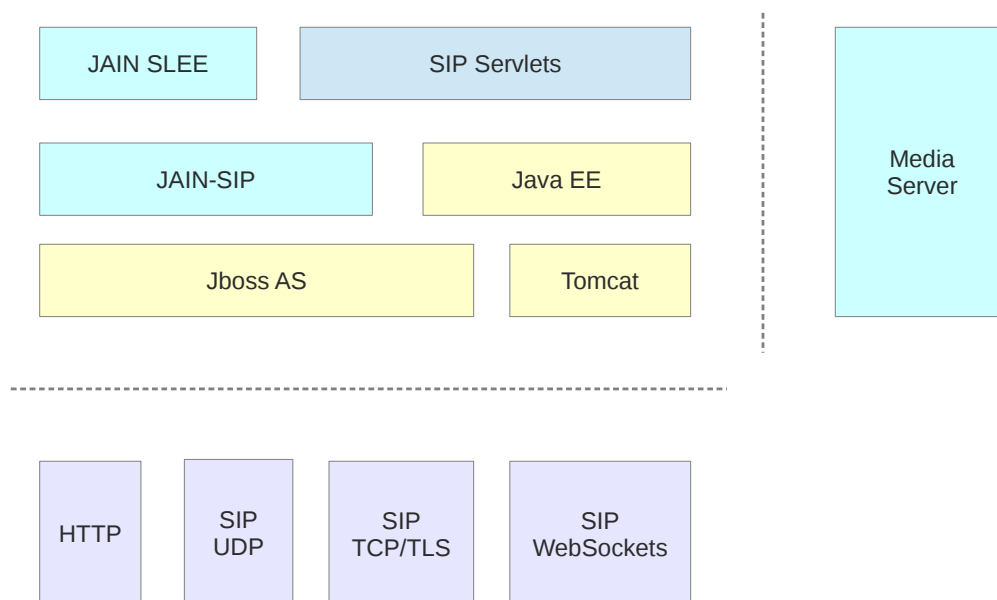
JBoss AS (ang.: JavaBeans Open Source Software Application Server) je aplikacijski strežnik, ki temelji na Javi EE (*Enterprise Edition*). Zaradi tega lahko deluje na mnogo različnih platformah, v kolikor le-te podpirajo Javo. Razvoj *JBoss-a* se je začel leta 1999 kot strežnik, ki implementira *EJB API*, definiran v Java EE. Je odprotkodni strežnik, prosto dostopen po licenci *GNU Lesser General Public License*. Nadaljnji razvoj in podporo strankam proti plačilu nudi JBoss oddelek znotraj podjetja Red Hat. JBoss strežnik, dokumentacija, in izvorna koda so dostopni na strani [12].

2.4.3 Mobicents SIP Servlets

Mobicents je odprto-kodna komunikacijska platforma. Deluje v okolju Java EE na aplikacijskih strežnikih JBoss ali Tomcat. Omogoča ustvarjanje, postavitve in upravljanje storitev ter aplikacij za zvočno, video in podatkovno komunikacijo preko različnih omrežij. Pomembna lastnost platforme je visoka prilagodljivost glede na število uporabnikov ter trenutne porabe virov. Zato je



Slika 2.7: Primer vzpostavitve SIP klica preko SIP posrednikov.



Slika 2.8: Mobicents SIP Servlets znotraj VoIP platforme

primerna tudi za največje telekomunikacijske operaterje. Na sliki 2.8 so komponente, ki skupaj tvorijo VoIP platformo.

JAIN-SIP je javanska knjižnica, ki nudi SIP sklad. Omogoča enostavno pošiljanje in sprejemanje SIP sporočil z uporabo javanskih metod. Uporabniku se tako ni treba ukvarjati z zgradbo SIP sporočil in prepoznavanjem njihove vsebine.

JAIN-SLEE je robustna dogodkovno-vođena aplikacijska platforma, ki omogoča povezovanje razliĝnih komunikacijskih protokolov, kot so na primer protokoli družine SS7, SIP, XMPP, USSD, MGCP, HTTP in še mnogo drugih.

SIP Servlets je aplikacijski vmesnik, ki temelji na Java EE. Omogoča razvoj SIP servletov, ki sluŝijo kot agenti-streŝniki v SIP dialogu. V delovanju se zgleđujejo po HTTP servletih.

Mobicents nudi tudi javanski medijski streŝnik, s katerim komunicira preko MGCP protokola. Njegov namen je zdruŝevanje medijskih vsebin, do katerih se dostopa iz razliĝnih omreŝij.

Knjiŝnice, dokumentacija, navodila in izvorna koda so dostopni na strani [5].

2.4.4 Seam

Seam je odprto-kodna razvojna platforma, namenjena izdelavi javanskih aplikacij. Je celovito okolje, ki zajema tehnologije kot so: AJAX, JSF, SPA, EJB 3.0, BPM. Razvita je bila z namenom zmanjšanja kompleksnosti integracije in konfiguracije programskih komponent. V ta namen omogoča razvijalcem med drugim uporabo številnih gradnikov uporabniškega vmesnika in *anotacij* programske kode. Najnovejšo verzijo *Seam-a* je mogoče dobiti na naslovu [13].

Anotacije so posebne oznake v programski kodi, ki neposredno ne vplivajo na delovanje programa. Služijo kot dodatne informacije za prevajalnik oz. druge programe, ki procesirajo kodo programa. *Anotacijo* določimo s simbolom "@", kateremu sledi ime *anotacije*. Poleg standardnih javanskih *anotacij*, ponuja *Seam* možnost dodatne konfiguracije programskih komponent s pomočjo lastnih *anotacij*, ki s tem nadomestijo konfiguracijske XML datoteke.

Posebnost *Seama* je zajem dogodkov in deklarativno upravljanje stanj aplikacije, zaradi česar je še posebej primeren za upravljanje z dialogi. Vsaka *Seam* komponenta lahko sproži določen dogodek, ki ga lahko zaznajo poljubne ostale komponente. Takšna arhitektura poleg drugega omogoča visoko stopnjo modularnosti in šibko sklopljenost razredov.

Koda 2.7 prikazuje dva javanska razreda, ki komunicirata s proženjem in poslušanjem dogodkov. Prvi je pošiljatelj, ki s pomočjo *anotacije*, označene kot `@RaiseEvent` sproži dogodek in zraven posreduje parameter, določen z *anotacijo* `@Out`. Prejemnik spremlja dogodke in s pomočjo *anotacije* `@Observer` zazna pošiljanje sporočila ter prejme parameter v spremenljivki, označeni z `@In`. *Anotacije* imajo lahko tudi dodatne parametre za konfiguracijo, kot je na primer `required` polje pri `@Out`.

2.5 Metrike

V tem poglavju so opisane metrike, ki so uporabljene za evalvacijo delovanja aplikacije v poglavju 4. Uporabili smo tri lastnosti, ki opredeljujejo tri vidike kakovosti rešitve, glede na skupine ljudi:

- Uporabnost – končni uporabniki (porabniki storitev)
- Modularnost – razvijalci in vzdrževalci
- Prenosljivost – uporabniki (ponudniki storitev)

```
@Name("sender")
public class Sender {

    @Logger
    Log log;

    private String text;

    @Out(required=true)
    private String message;

    @RaiseEvent("messageSent")
    public void send(){
        message=text;
        log.info("Sent message: "+message);
    }
}

@Name("recipient")
public class Recipient {

    @Logger
    Log log;

    private String text;

    @In
    private String message;

    @Observer("messageSent")
    public void receive(){
        text=message;
        log.info("Received message: "+message);
    }
}
```

Izvorna koda 2.7: Zajem in proženje dogodkov v Seam okolju.

2.5.1 Uporabnost

Na začetku smo si zadali cilj, da izdelamo aplikacijo, ki bi lahko bila primer rešitve za zagotavljanje storitev nekega telekomunikacijskega operaterja. Če želimo to doseči, mora aplikacija zagotavljati storitve za širok nabor končnih

uporabnikov.

Ciljne skupine uporabnikov so:

- spletni uporabniki,
- VoIP uporabniki,
- uporabniki mobilnih telefonov,
- uporabniki stacionarnih telefonov.

2.5.2 Modularnost

Modularni razvoj programske opreme narekuje delitev funkcionalnosti v ločene, med seboj izmenljive dele - module. Moduli vsebujejo vse, kar potrebujejo za opravljanje njim namenjene funkcionalnosti. Modularnost omogoča večjo preglednost, lažje vzdrževanje ter enostavnejše dograjevanje v prihodnosti. Posamezne module je mogoče tudi ponovno uporabiti v druge namene.

Lastnosti modularnih aplikacij, kot so priporočene v knjigi [22]:

- **Šibka sklopljenost** – moduli so med sabo ločeni v smislu, da posamezen modul ne vsebuje direktne povezave do drugega modula.
- **Zakrivanje** – modul omogoča dostop do lastnih atributov samo preko vmesnika.
- **Kohezija** – vsak modul opravlja samo določeno funkcionalnost. V kolikor določen modul potrebuje funkcionalnost, ki je v domeni drugega modula, lahko to naredi le z uporabo le-tega modula.

2.5.3 Prenosljivost

Prenosljivost aplikacije je zmožnost uporabe v različnih okoljih oziroma minimalno potrebno po prilagajanju v primeru spremembe okolja. Visoka stopnja prenosljivosti se doseže z abstrakcijo okolja, v katerem je nameščena aplikacija.

V splošnem se problem prenosljivosti deli na podprobleme:

- prenos v drugo okolje, ki je identično izvornemu,
- različno programsko okolje,
- različen operacijski sistem,
- različna strojna konfiguracija.

Poglavje 3

WebPhone

3.1 Ideja

WebPhone je spletna aplikacija, ki združuje dva različna komunikacijska sveta - svet telefonije in svetovni splet. Služi kot testni primer, kako lahko s pomočjo odprto-kodnih programskih orodij razvijemo konvergenčno rešitev za telefonske in spletne uporabnike. Sestavljajo jo spletni vmesnik, zaledna poslovna logika, ter SIP agent.

Spletnim uporabnikom omogoča vzpostavitev telefonskega klica med dvema SIP agentoma v omrežju oz. telefonske konference med poljubnim številom agentov. Telefonski uporabniki lahko s klicem na SIP naslov aplikacije na strežniku pustijo zvočno sporočilo, ki ga lahko pozneje predvajajo preko spletnega vmesnika.

Spletni vmesnik je *HTML* stran, na kateri imamo dva razdelka. V prvem lahko vzpostavimo klice. Vsebuje polja, kamor vpišemo naslove agentov, ki jih želimo poklicati in gumb, s katerim izvedemo klicanje. V spodnjem delu se nahaja seznam sporočil, ki so se posnela ob klicu SIP agenta aplikacije. Sporočila lahko predvajamo s klikom na ime.

3.1.1 B2BUA

WebPhone vzpostavlja klice na poseben način, s pomočjo vmesnega SIP agenta (*B2BUA*). Po delovanju je *B2BUA* podoben SIP posredniku, ki je opisan na koncu poglavja 2.3.2, razlikuje pa se predvsem v tem, da je navzven viden kot navaden SIP agent in se tudi obnaša kot tak. *B2BUA* sprejema SIP sporočila in kreira nova. Medtem ko SIP posrednik spreminja samo *Via* polje v sporočilu, lahko *B2BUA* spremeni celotno vsebino sporočila. S tem pridobi popoln nadzor

nad vzpostavitev seje. Uporaben je predvsem v primeru, ko želimo zagotoviti anonimnost uporabnikov storitve.

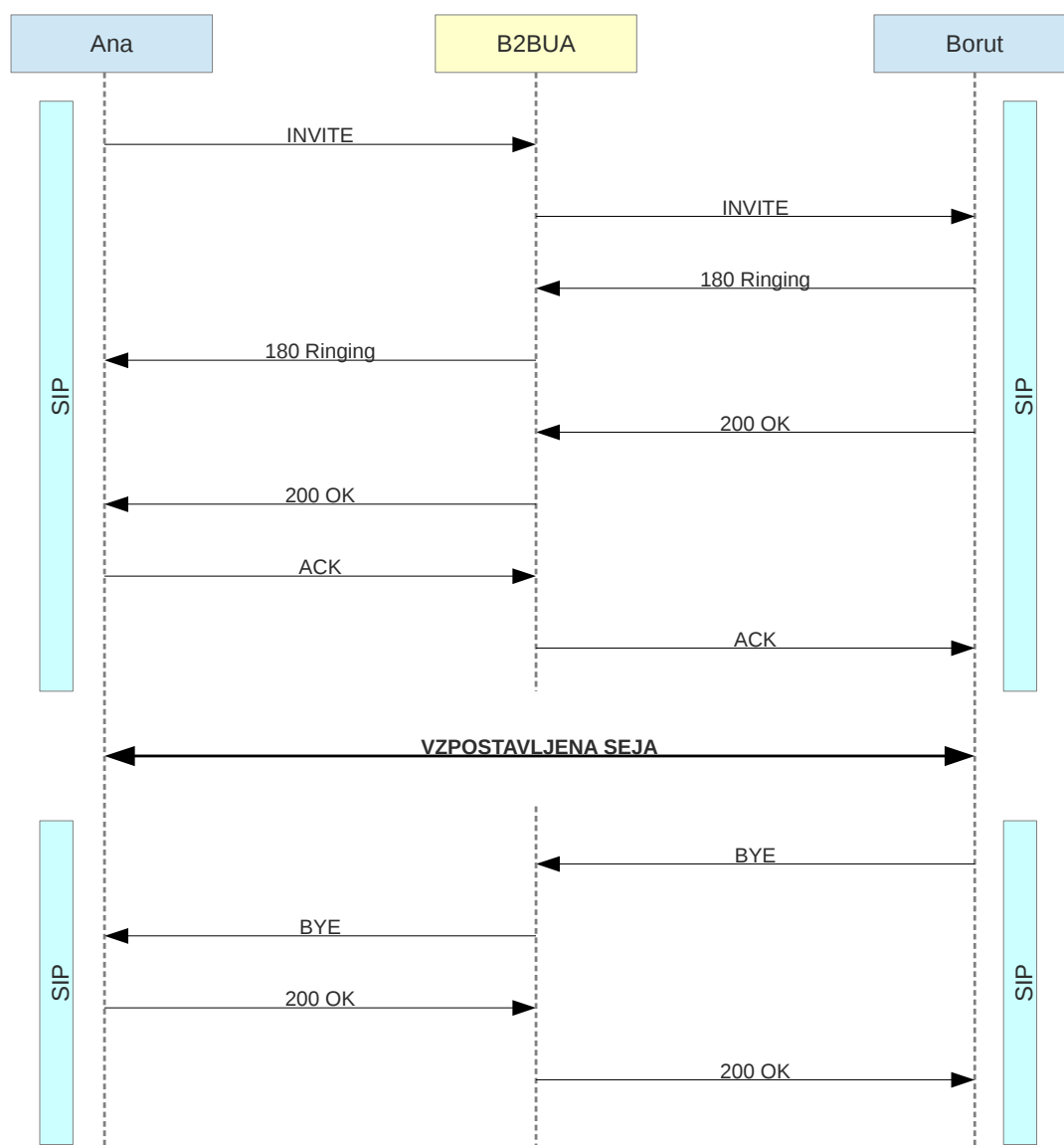
Vzpostavitev seje z uporabo *B2BUA* je prikazana na sliki 3.1. Za primer vzemimo ponovno Ano in Boruta. Ana v tem primeru ne pozna Borutovega naslova, ampak pošlje **INVITE** sporočilo vmesnemu agentu. Ta naredi novo **INVITE** sporočilo, ki izgleda, kot da bi pogovor začel *B2BUA*. Podobno, ko Borut odgovori na **INVITE** zahtevo, *B2BUA* posreduje spremenjeno sporočilo Ani. Na tak način med Borutom in Ano ni neposredne SIP komunikacije. *B2BUA* lahko vseeno omogoči, da se seja vzpostavi neposredno med Ano in Borutom, brez nepotrebnega vmesnika. To naredi tako, da v **200 OK** odgovoru in **ACK** zahtevku v polje **Contact** ne vnaša svojega naslova, ampak vnese Borutov oz. Anin dejanski naslov. S pomočjo dejanskih naslovov se seja tako vzpostavi neposredno. *B2BUA* lahko seveda zapiše v **Contact** polje svoj naslov, in ostane posrednik tudi v medijski seji med Ano in Borutom.

3.1.2 DTMF

Sporočila, ki jih uporabnik pusti na strežniku, se posnamejo s pomočjo *DTMF* signalizacije. Z *DTMF* tonom lahko uporabnik strežniku sporoči začetek in konec snemanja sporočila. *DTMF* predstavlja 16 tonov, ki označujejo številke (0-9), črke (A-D) in posebna znaka (#, *). Narejeni so s seštevkom signala s ti. *nizko* frekvenco (697 Hz, 770 Hz, 852 Hz, 941 Hz) in signala s ti. *visoko* frekvenco (1209 Hz, 1336 Hz, 1477 Hz, 1633 Hz). Po štiri frekvence vsake vrste nam dajo na voljo 16 različnih kombinacij novih tonov. Številko 0 tako predstavlja signal, v katerem sta sešteti sinusoidi s frekvencami 941 Hz in 1336 Hz. Frekvence so izbrane tako, da se v človeškem govoru pojavlja s čim manjšo verjetnostjo in je tako možnost napačne detekcije tem manjša.

Za detekcijo signala se uporablja *Goertzeljev algoritem*. *Goertzeljev algoritem* izračuna DFT signala za določeno frekvenco. Ker za računanje uporablja realne koeficiente, je v primeru računanje odziva celotnega spektra počasnejši kot FFT, hitrejši pa pri računanju odziva za manjše število frekvenc, kar ustreza *DTMF* signalom, ki so sestavljeni le iz dveh frekvenc. Poleg tega je tudi enostaven in lahek za implementacijo, kar posledično olajša njegovo uporabo v vezjih in enostavnih procesorjih v telefonskih napravah. Algoritem je podrobneje opisan v članku [9].

S pritiskom na tipko telefona, se pošlje impulz s frekvenco, ki pripada pritisnjenemu znaku. Če telefon nima aktivnega klica, z *DTMF* toni telefonski centrali sporočimo telefonsko številko, ki jo želimo poklicati, centrala pa nato vzpostavi povezavo. *DTMF* signali se pošljejo tudi ob vzpostavljeni povezavi



Slika 3.1: Vzpostavitev klica s pomočjo B2BUA.

in jih lahko uporabimo za proženje dogodkov na prejemnikovi strani - v našem primeru SIP aplikaciji.

3.2 Arhitektura

WebPhone je spletna aplikacija, napisana v programskem jeziku *Java*. *Java* je bila izbrana kot razvojni programski jezik predvsem zato, ker je velika večina poslovnih aplikacij, med drugim tudi v telekomunikacijskem svetu, razvitih v *Javi*. Ker želimo razviti aplikacijo, ki bi predstavljala neko telekomunikacijsko rešitev, je bila izbira programskega jezika enostavna odločitev.

Ogrodje aplikacije je programska platforma *Seam*. Glavni razlog za izbiro *Seama* je enostavna konfiguracija s pomočjo anotacij in zajemanje dogodkov, kar pride še posebej prav pri razvoju logike SIP agenta.

Za SIP komunikacijo se uporablja SIP sklad vmesnika *SIP Servlets*, ki je del *Mobicents* platforme. *Mobicents* je daleč najbolj razširjeno odprto-kodno ogrodje za razvoj telekomunikacijskih sistemov. *SIP Servlets* API se uporablja za sprejemanje in tvorjenje SIP sporočil.

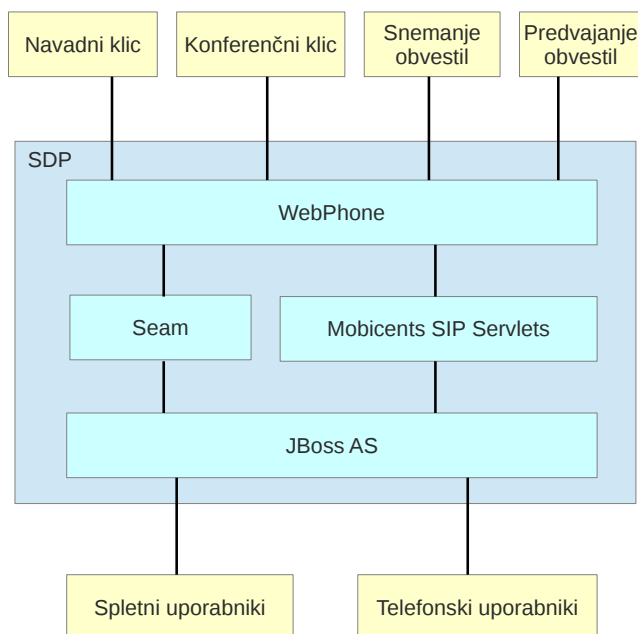
Za zvočno komunikacijo med aplikacijo in SIP uporabnikom, kot so na primer navodila, ki se predvajajo, ko uporabnik pokliče aplikacijo, se uporablja sintetizator slovenskega govora *Proteus* (TTS). *Proteus* je bil razvit v slovenskem podjetju *Alpineon*.

WebPhone deluje na aplikacijskem strežniku *JBoss AS*, katerega podpirata platformi *Seam* in *Mobicents*.

Na sliki 3.2 je prikazana storitvena arhitektura konvergenčne rešitve. Na vrhu imamo storitve, ki jih rešitev ponuja:

- navadni klic,
- konferenčni klic,
- snemanje obvestil,
- predvajanje obvestil

Programska logika aplikacije, okolje *Seam*, knjižnice *Mobicents* *SIP Servlets* in aplikacijski strežnik *JBoss AS* skupaj omogočajo zgoraj navedene storitve za dva podsistema (vmesnika za spletne in telefonske uporabnike). Če storitveno arhitekturo aplikacije primerjamo z arhitekturo na sliki 2.2, vidimo, da *WebPhone* aplikacija v kombinaciji s programskim okoljem (*Seam*, *Mobicents* *SIP Servlets*, *JBoss AS*) po zasnovi predstavlja platformo za zagotavljanje storitev (SDP).



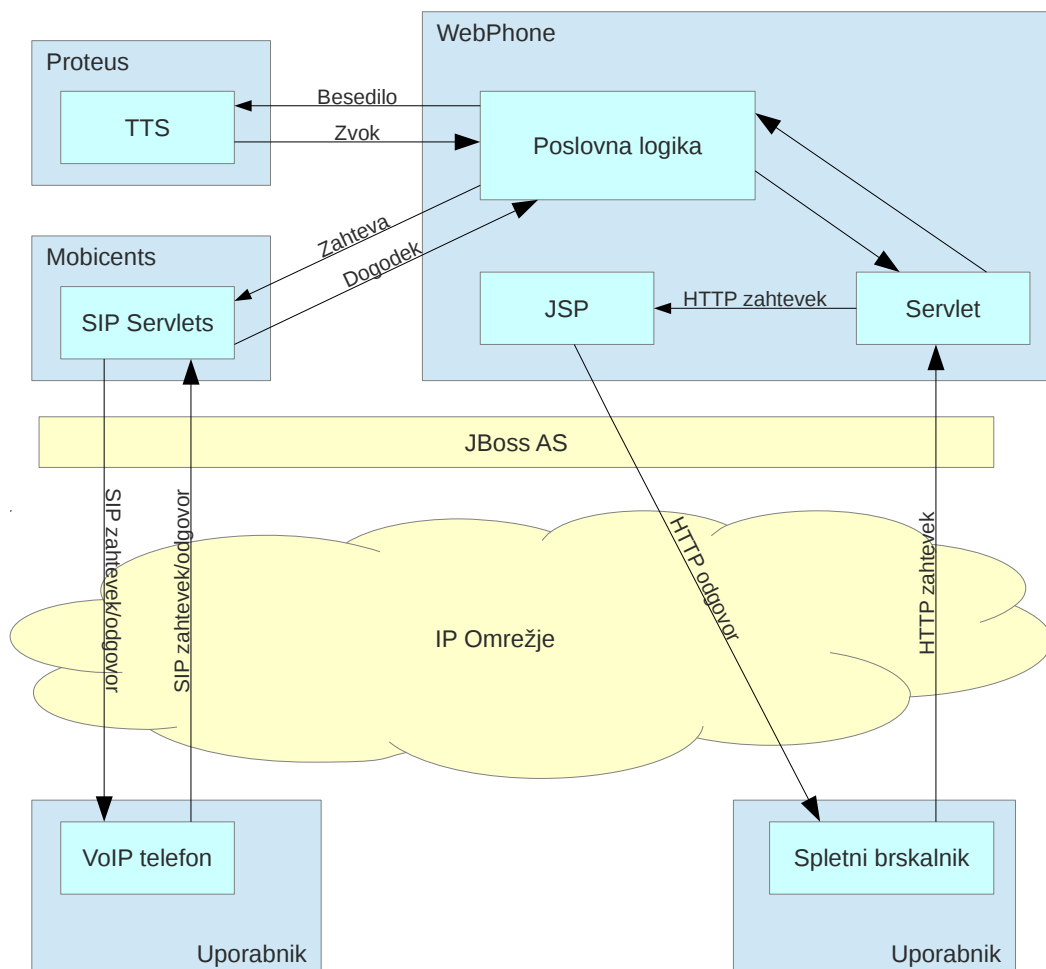
Slika 3.2: Storitvena arhitektura WebPhone aplikacije.

3.3 Delovanje

Slika 3.3 opisuje programsko arhitekturo sistema in kako poteka komunikacija med uporabniki in aplikacijo, ter med posameznimi elementi aplikacije. *WebPhone* ima dve uporabniški dostopni točki - HTML vmesnik in SIP agenta. Komunikacija poteka preko IP omrežja. Na strani aplikacije predstavlja komunikacijsko vstopno točko strežnik *JBoss*. Na njem teče spletna aplikacija, ki posluša na vratih 8080, ter SIP agent, ki posluša na vratih 5060. V nadaljevanju je opisano delovanje sistema v primeru dostopa preko spletnega vmesnika oz. VoIP klica.

3.3.1 Klicanje

Ko uporabnik preko brskalnika pošlje zahtevo (*GET*) na URL naslov aplikacije, jo prestreže *JBoss* strežnik, ki posluša na vratih 8080. Glede na URL naslov znotraj domene, posreduje zahtevo razredu tipa *servlet*. *Servleti* so posebni javanski razredi, ki strežejo HTTP zahteve. Konfiguracija, kam *JBoss* usmerja zahteve v domeni aplikacije, se nahaja v datoteki *web.xml*. V kodi *servleta* se izvedejo klici v zaledni del sistema in pripravijo podatki za prikaz, nato pa se zahteva s parametri posreduje JSP strani. Rezultat izvajanja JSP kode je



Slika 3.3: Shema komunikacije elementov WebPhone aplikacije.

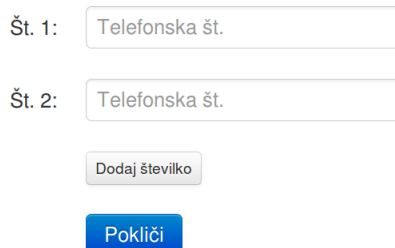
besedilo v formatu HTML, ki se pošlje kot odgovor na zahtevo uporabnika, ter se prikaže v njegovem brskalniku.

Na sliki 3.4 je spletni vmesnik aplikacije. V polje *št. 1* vnese SIP naslov, ki predstavlja kličočega, v polje *št. 2* pa vnese naslov klicanega. S klikom na gumb "Dodaj številko" se na maski prikaže dodatno polje za vnos. V kolikor se vnesejo trije SIP naslovi ali več se vzpostavi konferenčni klic, v primeru dveh naslovov pa navadni klic.

Vzemimo za primer ponovno Ano, ki želi tokrat s pomočjo *WebPhone* aplikacije poklicati Boruta. *WebPhone*, ki deluje po principu vmesnega agenta, poskuša posnemati klicanje tako, da najprej pokliče kličočega, in šele ko se ta oglasi pokliče klicanega. Na sliki 3.5 je prikazan postopek komunikacije. Ap-

Klicanje

Vzpostavi klic:



Obvestila

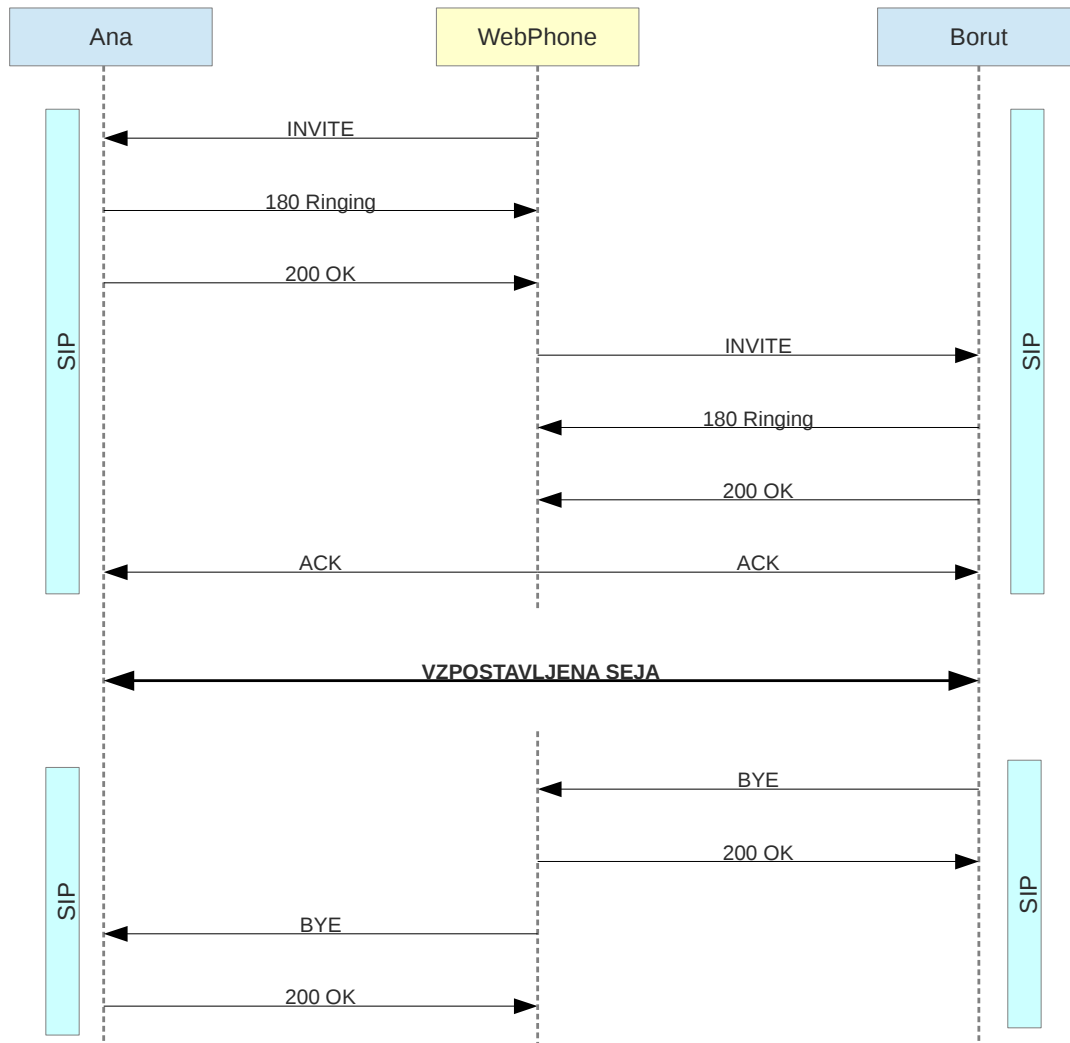
- [test.wav](#)
- [trololololo.wav](#)
- [040587954_15.2.2013\(2\).wav](#)
- [040587954_15.2.2013\(3\).wav](#)
- [040587954_15.2.2013\(1\).wav](#)
- [040587954_15.2.2013.wav](#)

Slika 3.4: Spletni uporabniški vmesnik.

likacija, pošlje Ani INVITE zahtevek, ki se po vsebini malenkostno razlikuje od običajnega INVITE zahtevka. Primer takšnega sporočila je prikazan v izseku kode 3.1. Vidimo, da imata polji **From** in **To** vsebini, kot da bi Boruta dejansko klicala Ana, medtem ko polje **Contact** vsebuje IP naslov strežnika *WebPhone* aplikacije. Kot že omenjeno v poglavju 2.3.2, SIP agenti pridobijo naslov, kamor pošljejo odgovor, iz polja **Contact** in zato tudi Anin agent odgovori strežniku in ne Borutovemu agentu.

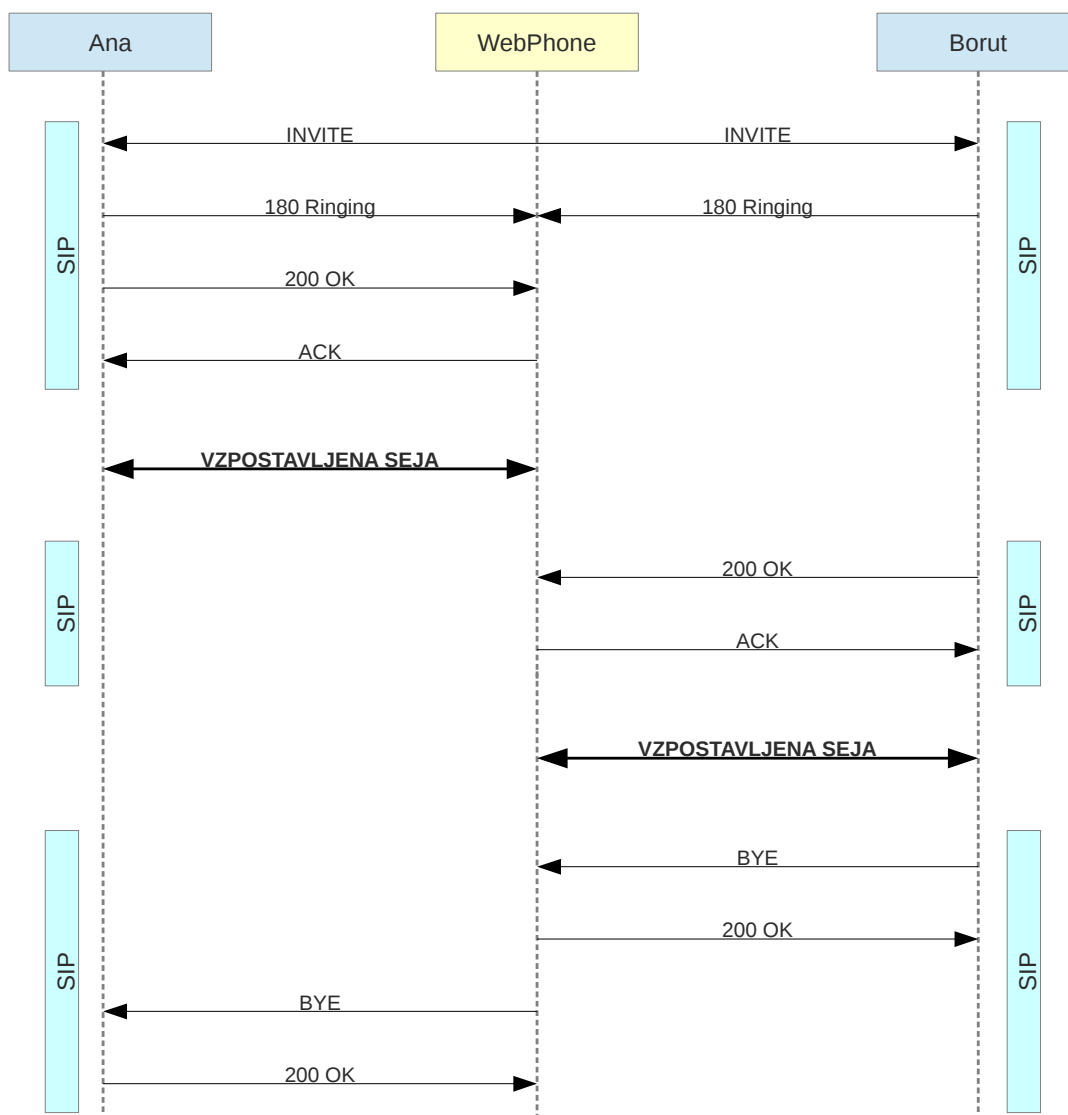
Ko Ana sprejme klic, se INVITE zahtevek pošlje še Borutu. Ko Borut pritrdilno odgovori na povabilo, dobi *WebPhone* potrebne podatke za vzpostavitev povezave med Ano in Borutom. To naredi tako, da obema pošlje ACK zahtevek, v katerem vsebuje **Contact** polje Anin oz. Borutov dejanski naslov. Borutov agent prikaže *WebPhone* aplikacijo kot kličočega agenta, povezavo pa v resnici vzpostavi z Aninim agentom. Ko se eden izmed agentov odloči prenehati klic, pošlje BYE zahtevek, ki ga aplikacija posreduje drugemu agentu.

Vzpostavitev konferenčnega klica je prikazana na sliki 3.6. *WebPhone* vsem



Slika 3.5: SIP komunikacija pri navadnem klicu.

prisotnim v konferenci pošlje **INVITE** zahtevek. V tem primeru sta to Ana in Borut, sicer pa je število naslovnikov poljubno. Ko Ana odgovori na zahtevek s kodo 200, se med njenim SIP agentom in aplikacijo vzpostavi povezava. Za razliko od navadnega klica, kjer *WebPhone* deluje samo kot organizator pri vzpostavitvi SIP seje, je v konferenčnem klicu aktivno vključen kot gostitelj konference. Za Ana na povabilo odgovori še Borut in se s tem priključi konferenci. V primeru, da bi v konferenci sodelovali še drugi agenti, bi se le-ti priključili ob poljubnih časih. Zaporedje vzpostavljenih povezav ni pomembno. Za razliko od navadnega klica, kjer se medijska seja začne, ko so povezani vsi agenti, je pri konferenčnem klicu medijska seja postavljena že od vsega začetka, še preden so poslana prva povabila.



Slika 3.6: SIP komunikacija pri konferenčnem klicu.

```

INVITE sip:ana@ankaran.si SIP/2.0
Call-ID: d384be4fbe79c1df33cab0e3a7d6baf4@0.0.0.0
CSeq: 1 INVITE
From: <sip:ana@ankaran.si>;tag=48638199_936771e3
To: <sip:borut@bovec.si>
Max-Forwards: 70
Via: SIP/2.0/UDP 10.1.10.79:5060;branch=z9hG4bKa7acba35f298d82
db4c34ef524f1c9a393137;appname=webphone
Contact: ana@ankaran.si <sip:10.1.10.79:5060;transport=udp>
Content-Length: 0

```

Izvorna koda 3.1: Primer INVITE zahtevka WebPhone aplikacije

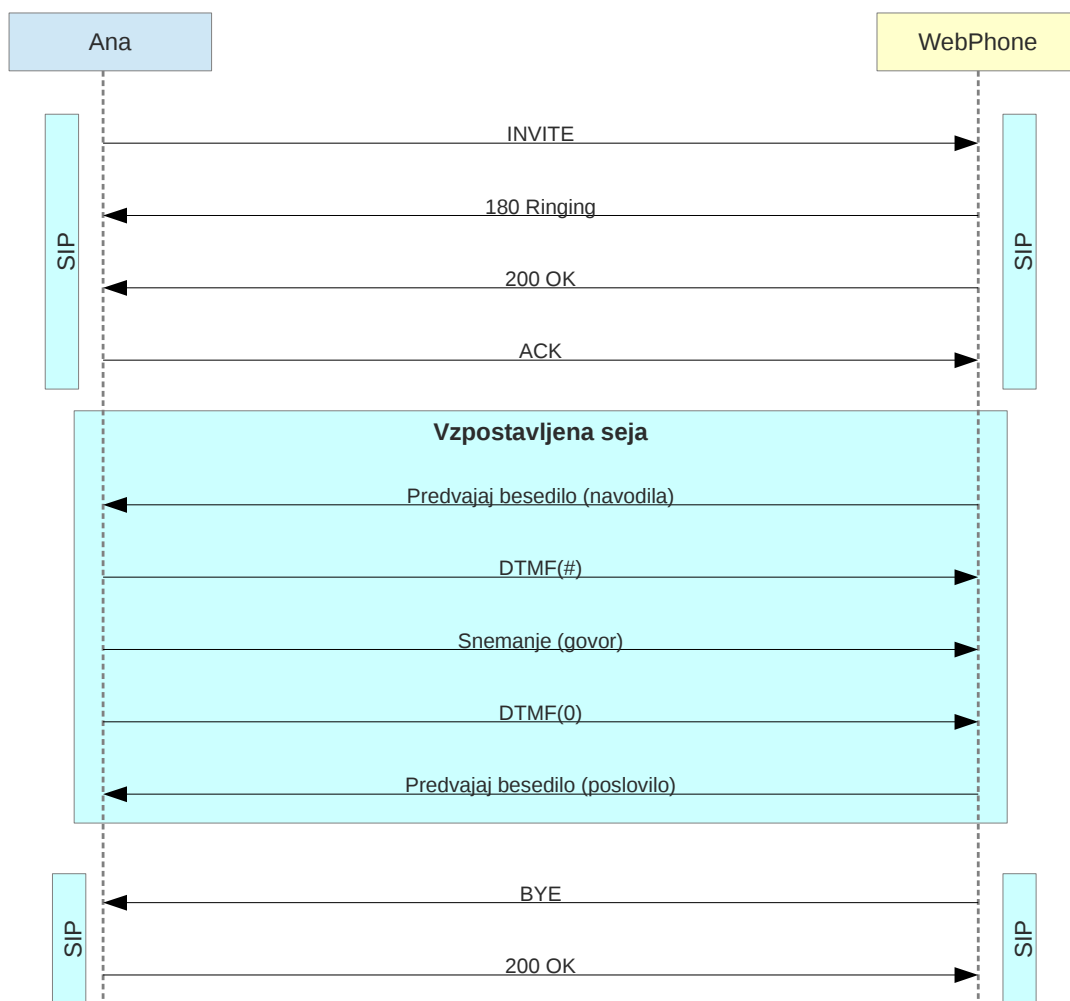
3.3.2 Obvestila

Pri snemanju obvestil, se dialog začne iz obratne smeri. Uporabnik storitve z VoIP telefonom pokliče SIP naslov *WebPhone* aplikacije. Zaporedje dogodkov je prikazano na sliki 3.7. Vzpostavitev povezave poteka po postopku, že opisanem v poglavju 2.3.2. Ko uporabnik, v tem primeru Ana, vzpostavi govorno povezavo s SIP agentom aplikacije, mu aplikacija predvaja zvočna navodila za uporabo storitve. Navodila, ki se tvorijo iz besedila s pomočjo sintetizatorja *Proteus*, Ani povejo, da mora za začetek snemanja pritisniti tipko '#' in za zaključek tipko '0'. Ko Ana na tipkovnici telefona pritisne tipko, se pošlje DTMF signal, ki da aplikaciji znak, da začne snemati pogovor. Po pritisku na tipko '0' se snemanje zaključi. Po končanem obvestilu pošlje *WebPhone* Ani BYE zahtevek in s tem zaključi dialog.

Aplikacija zvočne posnetke shranjuje v imenik na strežniku. Poimenuje jih z naslovom uporabnika in datumom. Pri pripravi podatkov za prikaz na maski spletnega vmesnika aplikacija prebere vsebino imenika s posnetki ter izpiše povezave do posnetkov.

3.3.3 Razred *EventHandler*

Jedro logike, ki se ukvarja s SIP dialogi, se nahaja v javanskem razredu, imenovanem *EventHandler* (upravljaivec z dogodki). Okolje *Seam* nam omogoča, da s pomočjo *@Observer* anotacij zajamemo SIP dogodke in izvedemo ustrezne ukaze. SIP dogodke proži SIP sklad znotraj *Mobicents SIP Servlets* paketa. *EventHandler* vsebuje samo programsko logiko, ki upravlja s SIP dialogom. Ostale operacije se izvajajo preko klicev metod v drugih razredih. Takšna



Slika 3.7: Komunikacija pri snemanju obvestil.

zasnova omogoča visoko stopnjo modularnosti, in s tem lažje vzdrževanje ter enostavnejšo implementacijo morebitnih razširitve aplikacije. Dogodki, ki jih *EventHandler* zazna so prejeti SIP zahteve (*INVITE*, *BYE*), SIP odgovori na poslano zahteve, ter ostali dogodki, ki se tičejo medijske seje, kot so vzpostavitev seje, končanje seje, prejet *DTMF* signal, itd.

Izsek kode 3.2 iz razreda *EventHandler* prikazuje konfiguracijo razreda in deklaracije vhodnih parametrov. Objekt vrste *Stateless* ne hrani lastnega stanja, kar pomeni, da se morajo parametri, ki jih potrebuje za izvedbo, poslati ob vsakem klicu znova. Rezultat tega je med seboj neodvisno procesiranje dogodkov. Atribut *@Transactional* poskrbi, da je vsak klic metode razreda transakcija. To pomeni, da se v primeru, ko se metoda ne izvrši uspešno do

konca, spremembe entitetnih zrn (EJB) ne uveljavijo.

```
...
@Name("mediusSDPEventHandler")
@Scope(ScopeType.STATELESS)
@Transactional
public class EventHandler {
    @In
    MediaController mediaController;
    @In
    SipSession sipSession;
    ...
    @In
    SipFactory sipFactory;
    @In(required = false)
    MsEventFactory eventFactory;
    ...
}
```

Izvorna koda 3.2: Deklaracije in vhodni parametri razreda *EventHandler*

Metoda `doInvite()`, deli katere so prikazani v kodi 3.3, sprejema povabila v pogovor (INVITE zahteve), ki prispejo s strani uporabnika. Na začetku se iz zahtevka izloči uporabnikov SDP paket in SIP naslov ter nanj pošlje odgovor s kodo 180 (Ringing). V SIP sejo se shranijo informacije, ki se bodo potrebovale v kasnejših delih dialoga. Primer takšne informacije je atribut **ACTION**, ki pove, da se izvaja akcija snemanja. SIP seja je podobna HTTP seji, s tem da se za identifikacijo uporabnika ne uporabljajo IP naslovi oz. sejni piškotki, temveč SIP naslovi. Enako kot pri HTTP seji, lahko aplikacijski strežnik v SIP sejo shrani serializirane javanske objekte kot sejne attribute.

Na koncu se vzpostavi povezavo preko razreda **MediaController**, ki je del *Mobicents* knjižnice, tako da se mu posreduje uporabnikov SDP paket, v katerem so predlagani parametri medijske komunikacije. **MediaController** poskrbi za pošiljanje odgovora s kodo 200 in vzpostavitev povezave.

Metoda `doResponse` sprejema odgovore SIP agentov. Zanimajo nas tisti odgovori, ki so bili poslani na povabilo aplikacije (INVITE zahtevek). Začetek metode je prikazan v izvorni kodi 3.4.

```
@Observer("INVITE")
public void doInvite(SipServletRequest request) throws
Exception {
    SipURI suri = (SipURI) request.getTo().getURI();
    String account = suri.getUser();
    ...
    String sdp = new String((byte[])request.getContent());
    request.createResponse(SipServletResponse.SC_RINGING)
        .send();
    ...
    sipSession.setAttribute(Constants.FIRST_SIP_MESSAGE,
        request);
    Address from = sipFactory.createAddress(request.getFrom()
        .getURI().toString());
    log.info("User from: " + from.toString());
    sipSession.setAttribute(Constants.REQUEST_URI, from
        .getURI());
    sipSession.setAttribute(Constants.ACTION, Constants
        .ACTION_RECORD);
    try {
        mediaController.createConnection(PR_JNDI_NAME)
            .modify("$", sdp);
        ...
    }
```

Izvorna koda 3.3: Sprejem povabila

```
@Observer("RESPONSE")
public void doResponse(SipServletResponse response)
    throws ServletException, IOException {
    int status = response.getStatus();
    if (status == SipServletResponse.SC_RINGING) {
        ...
        log.info("Ringing...");
    } else if (status == SipServletResponse.SC_OK) {
        if ("INVITE".equals(response.getMethod())) {
            ...
        }
    }
}
```

Izvorna koda 3.4: Procesiranje odgovora

Deli kode, kjer sestavimo in pošljemo povabilo posameznemu uporabniku, so prikazani v kodi 3.5. Na začetku metoda ustvari nov zahtevek. Vanj vstavi pošiljatelj in naslovnikov SIP naslov, kar daje videz, kot da prvi uporabnik pošilja povabilo drugemu uporabniku. Nato ustvari novo SIP sejo in ji nastavi dodatne attribute. Nastavi tudi ID klica, preko katerega lahko pozneje identificiramo transakcijo. Na koncu zahtevek odpošlje. Razred `SipFactory` je del vgrajene *JAIN-SIP* knjižnice.

Povabilo v konferenco pošlje koda 3.6. Za vsak naslov, ki je podan v seznamu `tos` se naredi nov zahtevek, v katerega se vstavi SIP naslov aplikacije. Preden se zahtevek odpošlje, se nastavijo še atributi seje.

```
public CallResponse call(String from, String to) {
    try {
        URI touri = sipFactory.createAddress(URIUtil.toSipUri(to))
            .getURI();
        URI fromuri = sipFactory.createAddress(URIUtil
            .toSipUri(from)).getURI();
        SipApplicationSession appSession = sipFactory
            .createApplicationSession();
        SipServletRequest req = sipFactory.createRequest(
            appSession, "INVITE", fromuri, touri);
        req.setRequestURI(fromuri);
        SipSession session = req.getSession();
        session.setAttribute(Constants.SECOND_PARTY_ADDRESS,
            sipFactory.createAddress(touri));
        ...
        session.setAttribute(Constants.ACTION, Constants
            .ACTION_CLICK_2_DIAL);
        long id = new Random().nextLong();
        session.setAttribute(Constants.CALL_ID, new Long(id));
        CallResponse response = new CallResponse();
        response.setId(id);
        log.info("Sending request" + req);
        req.send();
        ...
    }
}
```

Izvorna koda 3.5: Pošiljanje povabila v klic

```
...
for (int i = 0; i < tos.length; i++) {
    try {
        touri = sipFactory.createAddress(URIUtil.toSipUri(tos[i]))
            .getURI();
        URI from = fromAddr == null ? null : sipFactory
            .createAddress(fromAddr).getURI();
        SipServletRequest req = sipFactory.createRequest(
            appSession, "INVITE", from, touri);
        req.setRequestURI(touri);
        req.getSession().setAttribute(Constants.ACTION, Constants
            .ACTION_CONFERENCE);
        ...
        req.getSession().setAttribute(Constants.ACCOUNT_NUMBER,
            tos[i]);
        req.send();
        ...
    }
}
```

Izvorna koda 3.6: Pošiljanje povabila v konferenco

```
...
String action = (String) response.getSession().getAttribute(
    Constants.ACTION);
if (Constants.ACTION_CLICK_2_DIAL.equalsIgnoreCase(action)) {
    //navadni klic
    Address secondPartyAddress =(Address) response.getSession()
        .getAttribute(Constants.SECOND_PARTY_ADDRESS);
    if (secondPartyAddress != null) {
        //prejeli smo OK od prvega agenta
        ...
    } else {
        //prejelo smo OK od drugega agenta - poslji ACK
        ...
    }
} else if (Constants.ACTION_CONFERENCE.equalsIgnoreCase(action))
{
    //konferencni klic
    ...
}
```

Izvorna koda 3.7: Določanje izvora odgovora

Prejeto sporočilo je odgovor na povabilo ene izmed dveh vrst – povabilo v klic, ali povabilo v konferenco. V izseku kode 3.7 je prikazano, kako določimo, katera akcija je bila povod za prejeti odgovor. Akcijo smo predhodno shranili v sejni atribut, ko smo uporabniku pošiljali povabilo. V kodi 3.5 je bila shranjena akcija `CLICK_TO_CALL`, ki označuje navadni klic. V kodi 3.6 je bila shranjena akcija `CONFERENCE`, ki označuje konferenčni klic. V primeru navadnega klica je potrebno še preveriti, ali je bilo prejeto sporočilo odgovor na prvo povabilo, ali drugo. Pri navadnem klicu namreč pošljemo povabilo drugemu uporabniku šele, ko dobimo odgovor prvega uporabnika. To določimo z atributom `SECOND_PARTY_ADDRESS`.

Če je v seji še vedno shranjen naslov drugega uporabnika (`SECOND_PARTY_ADDRESS`), to pomeni, da je bil prejet odgovor prvega uporabnika. V kodi 3.8 je prikazan postopek prejema odgovora prvega uporabnika v navadnem klicu. Iz odgovora se izloči naslov prvega uporabnika in zgradi novo `INVITE` sporočilo, v katerem je pošiljatelj prvi uporabnik in prejemnik drugi uporabnik. Sporočilo se pošlje na naslov drugega uporabnika. Obenem se ustvari tudi `ACK` zahtevek, ki se shrani za prihodnje pošiljanje prvemu uporabniku.

```
...
Address fromAddress = (Address) response.getSession()
    .getAttribute(Constants.FROM_ADDRESS);
SipApplicationSession appSession = sipFactory
    .createApplicationSession();
SipServletRequest inviteRequest = sipFactory.createRequest(
    appSession, "INVITE", fromAddress, secondPartyAddress);
inviteRequest.send();
SipServletRequest ack = response.createAck();
ack.setRequestURI(response.getTo().getURI());
SipSession session = inviteRequest.getSession();
session.setAttribute(Constants.FIRST_PARTY_ACK, ack);
session.setAttribute(Constants.FIRST_PARTY_CONTENT, response
    .getContent());
session.setAttribute(Constants.ACTION, Constants
    .ACTION_CLICK_2_DIAL);
session.setAttribute(Constants.FIRST_SIP_MESSAGE, inviteRequest);
...
```

Izvorna koda 3.8: Sprejem odgovora prvega uporabnika v klicu

Sprejem odgovora drugega uporabnika prikazuje koda 3.9. Najprej se ustvari potrditveni (ACK) zahtevek za drugega uporabnika. Iz seje se pridobi še potrditveni zahtevek za prvega uporabnika, ki je bil shranjen v kodi 3.8. Zahtevku, ki se pošlje prvemu uporabniku se nastavijo parametri, kot da bi ga pošiljal drugi uporabnik, in zahtevku, ki se pošlje drugemu uporabniku, obratno. Na koncu se oba zahtevka odpošljeta.

```
...
SipServletRequest secondPartyAck = response.createAck();
secondPartyAck.setRequestURI(response.getTo().getURI());
SipServletRequest firstPartyAck = (SipServletRequest) response
    .getSession().getAttribute(Constants.FIRST_PARTY_ACK);
if (response.getContentType() != null && response
    .getContentType().equals(Constants.APPLICATION_SDP)) {
    firstPartyAck.setContent(response.getContent(), Constants
        .APPLICATION_SDP);
    secondPartyAck.setContent(response.getSession().getAttribute(
        Constants.FIRST_PARTY_CONTENT), Constants.APPLICATION_SDP);
}
firstPartyAck.send();
secondPartyAck.send();
...
```

Izvorna koda 3.9: Sprejem odgovora drugega uporabnika v klicu

V primeru, da je bil prejet odgovor na povabilo v konferenco se iz odgovora izlušči SDP paket, s pomočjo katerega se vzpostavi povezava. Trivialen postopek je prikazan v kodi 3.10.

V prejšnjem primeru je bila vzpostavljena povezava med aplikacijo in uporabnikom, potrebno pa je uporabnika še povezati v dejansko sejo. To se stori v delu kode 3.11. Metoda `doLinkConnected` čaka na trenutek, ko se vzpostavi povezava. Iz seje se določi akcija. V primeru, da gre za konferenčni klic, se potekajoči konferenci priključi novega uporabnika. Konferenca se pridobi s pomočjo imena, shranjenega v seji. V primeru, da gre za akcijo snemanja obvestila, se izvede predvajanje navodil preko metode `playAnnouncement()`.

Ko uporabnik pritisne na gumb na tipkovnici, se pošlje DTMF signal, ki ga ulovi metoda `doDtmf`. Metoda prejme parameter, ki pove, kateri gumb je bil pritisnjen. Če je bil pritisnjen gumb '0', je to znak za začetek snemanja pogovora. Iz seje se pridobi prvo sporočilo, ki je bilo poslano s strani uporabnika. Iz tega sporočila se razbere uporabnikov naslov, ki se uporabi za ime

```

...
Object sdpObj = response.getContent();
byte[] sdpBytes = (byte[]) sdpObj;
String sdp = new String(sdpBytes);
sipSession.setAttribute(Constants.ACTION, Constants
    .ACTION_CONFERENCE);
sipSession.setAttribute(Constants.FIRST_SIP_MESSAGE, response);
mediaController.createConnection(PR_JNDI_NAME).modify("$", sdp);
...

```

Izvorna koda 3.10: Sprejem odgovora na povabilo v konferenco

```

@Observer("linkConnected")
public void doLinkConnected(MsLinkEvent event)
    throws IOException {
    String action = (String) sipSession.getAttribute(
        Constants.ACTION);
    mediaSessionStore.setMsLink(event.getSource());
    if (Constants.ACTION_CONFERENCE.equalsIgnoreCase(action)){
        MsLink link = mediaSessionStore.getMsLink();
        String confName = (String) sipSession.getAttribute(
            Constants.CONFERENCE_NAME);
        Conference conference = ConferenceCenter.getInstance().
            getConference(confName);
        conference.setConferenceEndpoint(link.getEndpoints()[0]);
    } else if (Constants.RECORD.equalsIgnoreCase(action)) {
        playAnnouncement(Constants.IVR.WELCOME_MSG);
        sipSession.setAttribute("firstTimeZero", true);
    }
    ...
}

```

Izvorna koda 3.11: Povezava v konferenco in predvajanje obvestil

zvočne datoteke, kamor se shrani posneti zvok. Dejansko snemanje se izvaja v metodi `record()`. Ime datoteke se shrani v sejo, da se bo datoteka lahko pozneje našla v začasem imeniku. Nadaljnji prejeti DTMF signali simbola '0' se ignorirajo.

Prejeti znak '#' pomeni konec snemanja. Posneta datoteka se prenese iz začasnega imenika v imenik, kjer se hranijo zvočni posnetki. To opravi metoda `copyFileToDir()`. Za zaključek se uporabniku predvaja poslovljno besedilo.

```
@Observer("DTMF")
public void doDtmf(String button) throws IOException, {
    SipServletRequest ivrRequest = (SipServletRequest)
    sipSession.getAttribute(Constants.FIRST_SIP_MESSAGE);
    SipURI suri = (SipURI) ivrRequest.getFrom().getURI();
    String accountNumber = suri.getUser()
    if ("0".equals(button) && action.equalsIgnoreCase(Constants
        .ACTION_RECORD)) {
        if (firstTimeZero != null && firstTimeZero) {
            sipSession.setAttribute("firstTimeZero", false);
            String filename = accountNumber + "_" + date + ".wav";
            record(filename);
            sipSession.setAttribute(Constants.IVR.RECORDED_FILE,
                filename);
        }
    }
    else if ("#" .equals(button) && action.equalsIgnoreCase(
        Constants.ACTION_RECORD)) {
        URL urlRecordedFile = copyFileToDir();
        sipSession.setAttribute("playRecorded", true);
        playAnnouncement(Constants.IVR.BYE_MSG);
        ...
    }
}
```

Izvorna koda 3.12: Detekcija DTMF signalov

Poglavje 4

Evalvacija

4.1 Razširitev – povezava v telefonsko omrežje

V poglavju 2.5 je v razdelku *Uporabnost* podana zahteva, da je storitev omogočena za štiri skupine uporabnikov. *WebPhone* sam po sebi, kot je opisan v poglavju 3, omogoča dostop za spletne ter VoIP uporabnike.

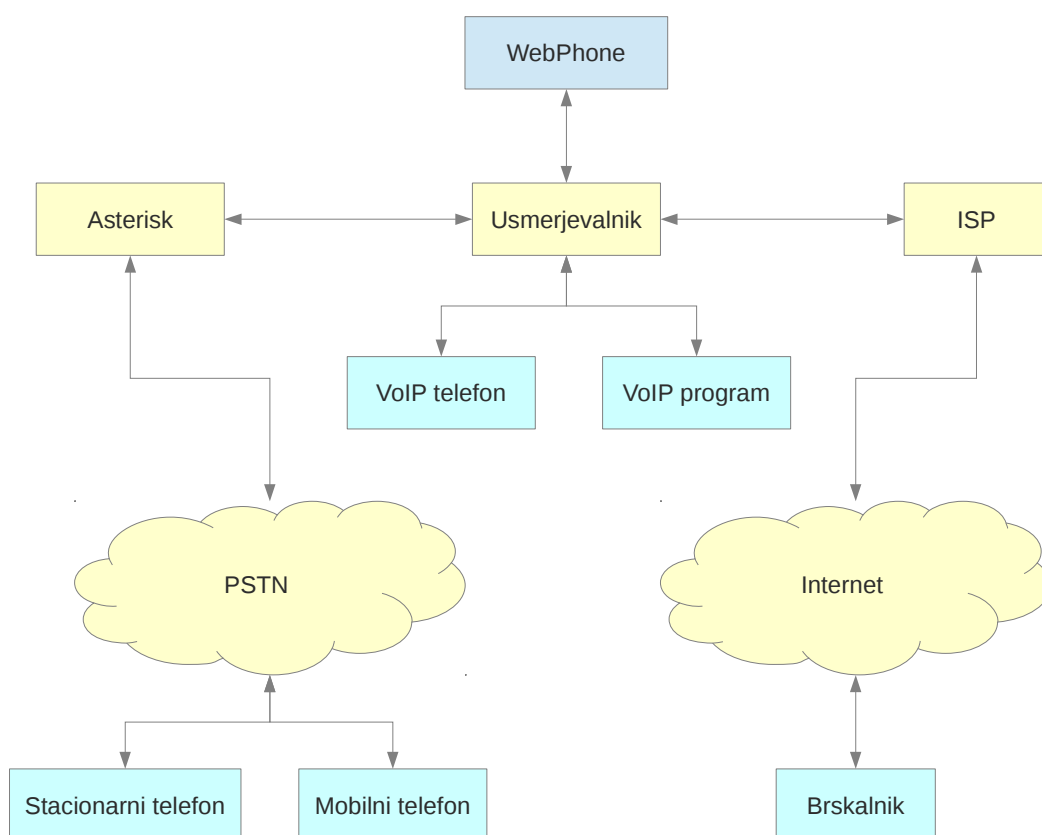
Dostop za spletne uporabnike je omogočen tako, da se omrežni usmerjevalnik priključi na omrežje ponudnika internetnih storitev. Na usmerjevalniku mora biti nastavljeno posredovanje zahtevkov na HTTP vratih 8080 in 80 na naslov strežnika znotraj omrežja, na katerem teče aplikacija *WebPhone*. To naredi aplikacijo dostopno iz svetovnega spleta.

Podobno velja za VoIP uporabnike. Če želimo storitev ponuditi uporabnikom izven lokalnega omrežja, moramo na usmerjevalniku nastaviti posredovanje še na standardnih SIP vratih 5060. V primeru, da uporabljamo RTP za medijsko komunikacijo, moramo nastaviti posredovanje še dodatno na vratih 5004.

Ostaneta še dve skupini nepovezanih uporabnikov: uporabniki stacionarnih in mobilnih telefonov. Če želimo tudi njim omogočiti dostop do storitve je potrebna dograditev sistema.

4.1.1 Asterisk

Asterisk je odprto-kodna programska implementacija telefonske centrale, napisana v programskem jeziku *C*. Omogoča klice med priključenimi telefoni ter druge storitve, kot so VoIP storitve in povezovanje v telefonsko omrežje. *Asterisk* je bil razvit v podjetju *Digium* in je dostopen v okviru licence *GNU General Public License*. V osnovi je bil razvit za operacijski sistem Linux,



Slika 4.1: Arhitektura SIP sistema z uporabo *Asterisk* centrale

danesh pa je dostopen na vseh večjih operacijskih sistemih.

Asterisk sam po sebi ne omogoča telefonskih storitev, ponuja pa okolje, v katerem lahko te storitve enostavno vzpostavimo. V ta namen vsebuje poseben skriptni programski jezik, s pomočjo katerega lahko predpišemo logiko usmerjanja telefonskih klicev v obliki pravil. Ta so zapisana v datoteki *extensions.conf*. Pravila imajo obliko `exten => 'številka', 'prioriteta', 'akcija';`. Akcije so lahko klici vgrajenih programov, kot je na primer ukaz za klicanje (`Dial`), makro ukazi ali klici uporabniških programov. Podrobnejši opis delovanja in navodila za konfiguracijo *Asterisk* strežnika so v knjigi [16].

4.1.2 Izvedba

Na sliki 4.1 je prikazan sistem, kjer se za povezavo telefonskih uporabnikov uporablja telekomunikacijska centrala *Asterisk*. Zaradi robustnosti SIP protokola, uvedba centrale ne zahteva nobenih sprememb na sami aplikaciji, z izjemo popravkov v konfiguracijski datoteki, kjer se spremeni SIP domena naslovov. Potrebno je le spremeniti nastavitve usmerjevalnika, da se SIP paketi, ki izvirajo s strežnika *WebPhone* aplikacije posredujejo računalniku, na katerem teče *Asterisk*. V tem primeru deluje *Asterisk* kot SIP SIP posrednik, z dodatno funkcijo prehoda (*gateway*), usmerjevalnik pa kot SIP preusmeritveni strežnik.

Na *Asterisku* je potrebno nastaviti, kako se poveže v telefonsko omrežje, in kam mora posredovati SIP pakete. V kodi 4.1 so nastavitve v datoteki *sip.conf*, ki to omogočajo. Najprej deklariramo SIP uporabnika, z imenom 5006, ki se nahaja na IP naslovu 10.1.10.79. Uporabnik se avtomatsko registrira ob zagonu *Asteriska*. V naslednjih vrsticah se nahajajo nastavitve, ki se navezujejo na telefonsko povezavo. V tem primeru je *Asterisk* povezan s Telekomovim omrežjem, v katerem je zakupljena številka 82001832 znotraj SIP domene *ngn.telekom.si*. S tem smo poskrbeli, da lahko *WebPhone* pokliče poljubne telefonske številke.

```
...
[5006](server-client)
host = 10.1.10.79

[ngn.telekom.si]
type = peer
host = ngn.telekom.si
fromdomain = ngn.telekom.si
fromuser = 82001832
outboundproxy = ngn.telekom.si
canreinvite = no
```

Izvorna koda 4.1: Izsek datoteke *sip.conf*

Poskrbeti pa moramo tudi za povezavo v obratni smeri. S pomočjo skriptnega jezika, ki ga nudi *Asterisk*, naredimo logiko, ki obravnava vhodne klice. Pravila, ki določajo logiko, so shranjena v datoteki *extensions.conf*. Del datoteke, ki določa usmerjanje klicev, namenjenih *WebPhone* aplikaciji je v

izvorni kodi 4.2. V prvi vrstici nastavimo spremenljivko `callerid`, ki identifi-
cira klicatelja. V naslednji vrstici omejimo možne klicatelje le na določene SIP
uporabnike oz. telefonske številke. V primeru, da klicatelj ustreza omejitvam,
se ga poveže s SIP uporabnikom 5006, ki je bil definiran v datoteki *sip.conf*
(koda 4.1). V nasprotnem primeru klic zavrnemo.

```
...
exten => _500,n,Set(callerid=${CALLERID(num)})
exten => _500,n,GotoIf($[${callerid} = 6000 | ${callerid} =
    6001 | ${callerid} = 6002 | ${callerid} = 040587954]?stef:
    prekinitev
exten => _500,n(stef),Dial(SIP/5006)
exten => _500,n(prekinitev),Background(/var/lib/asterisk/sounds
    /thanks)
exten => _500,n,Hangup
...
```

Izvorna koda 4.2: Izsek datoteke *extensions.conf*

Z zgoraj opisanim postopkom je bila demonstrirana uporabnost *WebPhone*
aplikacije v smislu, da jo je brez sprememb mogoče vgraditi v telekomunikaci-
jski sistem kot storitev.

4.2 Razširitev – nova funkcionalnost

Kot ena izmed metrik, opisanih v poglavju 2.5, je bila podana tudi modularnost
aplikacije. V nadaljevanju bomo modularnost *WebPhone* aplikacije preverili
tako, da bomo poskusili v aplikacijo vgraditi novo funkcionalnost. Na primer,
da želimo spletnim uporabnikom omogočiti, da lahko posneta obvestila, ki so
shranjena na strežniku, pošljejo na poljubne SIP naslove.

Spremembe na spletnem vmesniku so malenkostne. Posameznim obvestilom
je potrebno je dodati vnosna polja za naslov prejemnika. S klikom na gumb
Pošlji se informacija o posnetku in SIP naslovu posreduje servletu. Servletu
je potrebno dodati logiko, ki izvede ustrezne klice v zaledni del aplikacije.
Popravljen maska je prikazana na sliki 4.2.

Obvestilo se pošlje tako, da se najprej z naslovnikom vzpostavi povezava,
nato pa se predvaja zvočna datoteka. Potreba dodatna logika v razredu *Even-
tHandler* je prikazana v kodi 4.3.

Klicanje

Vzpostavi klic:

Št. 1:

Št. 2:

Obvestila

1	test.wav	Pošlji
2	trololololo.wav	Pošlji
3	040587954_15.2.2013(2).wav	Pošlji
4	040587954_15.2.2013(3).wav	Pošlji
5	040587954_15.2.2013(1).wav	Pošlji
6	040587954_15.2.2013.wav	Pošlji

Slika 4.2: Nov uporabniški vmesnik.

Naslovníku se pošlje povabilo (INVITE zahtevek), kot v primeru navadnega klica ali konference. Razlikuje se v parametrih, ki se shranijo v sejo. Kot akcija se v sejo shrani `ACTION_PLAY_ANNOUNCEMENT` in nato še pot do zvočne datoteke na strežniku.

V primeru da je naslovnik sprejel klic, in da akcija, shranjena v seji, ustreza akciji `ACTION_PLAY_ANNOUNCEMENT`, ki je bila nastavljena pri pošiljanju povabila, se vzpostavi povezava. Po vzpostavljeni povezavi se v klic predvaja zvočni posnetek, čigar naslov se prebere iz seje, kamor je bil shranjen ob pošiljanju povabila. Za predvajanje se uporabi metoda `playAnnouncement()`, ki je bila že uporabljena za predvajanje navodil pri snemanju obvestila.

Iz zgoraj opisanega postopka lahko razberemo, da je dodajanje nove funkcionalnosti v aplikacijo razmeroma enostavno, saj so potrebni gradniki dovolj

modularni. Pri pošiljanju povabila in ustvarjanju uporabljajo moduli knjižnice *Mobicents SIP Servlets*, za samo predvajanje zvočnih datotek pa že prej implementirana metoda. Modularnost bi lahko še izboljšali z združenjem nekaterih ukazov za pošiljanje povabila, ki se izvedejo v vseh primerih akcij, kot so na primer ukazi za shranjevanje podatkov v sejo.

4.3 Prenosljivost

4.3.1 EAR

EAR je datotečni format, ki opisuje arhiv, v katerem je *Java EE* aplikacija. Običajno se uporablja za prenos in namestitvev aplikacije na aplikacijski strežnik. Vsebuje različne module aplikacije, kot so: *web* modul, *EJB* modul, knjižnice v obliki *JAR* arhivov, itd. Omogoča, da se moduli namestijo istočasno in celovito. Vsebuje tudi namestitvene opisnike v obliki XML dokumentov. *EAR* je identičen navadni Zip datoteki, z izjemo direktorija *meta-inf*, ki vsebuje namestitvene opisnike. *EAR* arhivi se ponavadi naredijo z orodji *Ant* ali *Maven*, s pomočjo katerih lahko opišemo postopek grajenja arhiva.

4.3.2 Namestitev na drug strežnik

WebPhone je nameščen na aplikacijski strežnik *JBoss* s pomočjo *EAR* arhiva. Arhiv je sestavljen in iz *WAR* arhiva, ki vsebuje *JSP* datoteke, servlete in druge datoteke, ki so del spletnega vmesnika, *JAR* arhiva, ki vsebuje entitetna zrna, imenika *lib*, ki vsebuje potrebne knjižnice, ter *JAR* arhiv *jboss-seam.jar*.

Namestitev *EAR* datoteke na aplikacijski strežnik je enostavna. Potrebno jo je le prenesti v namestitveni imenik znotraj strežnika. Strežnik poskrbi za postavitve aplikacije ob zagonu oziroma spremembi v namestitvenem imeniku. Migracija aplikacije iz enega *JBoss* strežnika na drugega je zato trivialna.

Potrebno je omeniti, da sta *Proteus*, ki ga *WebPhone* uporablja za sintezo zvoka, in *Asterisk* samostojna aplikaciji, ki se lahko nahajata na poljubnem strežniku. V primeru, da se je *WebPhone* preselil na strežnik v drugo omrežje, oziroma se je spremenila tudi lokacija *Asterisk* in *Proteus* strežnikov, je potrebno popraviti nastavitve v konfiguracijskih datotekah.

4.3.3 Sprememba programskega okolja

Sprememba programskega okolja bi imela na *WebPhone* velike vplive. Logika aplikacije, ki krmili SIP agenta, sloni na okoljih *Seam* in *Mobicents*. Menjava

ali odstranitev katerega izmed okolij bi pomenila povsem drugačno zasnovano aplikacije. Enak rezultat bi povzročila menjava aplikacijskega strežnika *JBoss*, od katerega je odvisno okolje *Seam* in prav tako *Mobicents*.

Sprememba samega programskega jezika ne bi zahtevala samo ponovne implementacije aplikacije, ampak bi pod vprašaj postavila tudi dejansko možnost izvedbe. Le redki jeziki poleg *Java* imajo na voljo knjižnice in orodja za delovanje v SIP okolju, kar bi lahko pomenilo, da bi bila potrebna implementacija na zelo nizkem programskem nivoju.

Iz vidika programskega okolja je zato *WebPhone* aplikacija izrazito nefleksibilna.

4.3.4 Sprememba operacijskega sistema ali strojne konfiguracije

Ker je *WebPhone* razvita v *Javi*, in ker javanski programi tečejo znotraj navideznega stroja, se odvisnost aplikacije od operacijskega sistema in strojne konfiguracije prenese na odvisnost JVM.

Uradna verzija JVM, ki jo izdaja podjetje *Oracle*, z najnovejšo verzijo JRE in JDK podpira operacijski sistem *Windows* na procesorskih arhitekturah x86 in x64, *Linux* distribucije *Oracle*, *Suse*, *Red Hat* in *Ubuntu* na arhitekturah x86 in x64, *Solaris* na arhitekturah x86, x64 in Sparc 64, ter *Mac OS X* na arhitekturi x64. Celotna specifikacije je objavljena na strani [19].

Razvijalci operacijskih sistemov drugi razvijalci pogosto razvijejo tudi lastne implementacije JVM, ki so lahko posebej prilagojene operacijskemu sistemu, oziroma strojni opremi. Iz tega sledi, da bi dandanes verjetno težko našli mikroprocesorski sistem, na katerem ne bi bilo mogoče poganjati javanskih programov.

```

protected CallResponse playAnnouncement(
...
    SipApplicationSession appSession = sipFactory.
        createApplicationSession();
    SipServletRequest req = sipFactory.createRequest(
        appSession, "INVITE", from, touri);
    req.getSession().setAttribute(Constants.TO_ADDRESS,
        sipFactory.createAddress(touri));
    req.getSession().setAttribute(Constants.REQUEST_URI, touri);
    req.getSession().setAttribute(Constants.FROM_ADDRESS,
        sipFactory.createAddress(fromAddr));
    req.getSession().setAttribute(Constants.ACTION,
        Constants.ACTION_PLAY_ANNOUNCEMENT);
    req.getSession().setAttribute(Constants.AUDIO_FILE,
        audiofile);
    req.send();
...
@Observer("RESPONSE")
public void doResponse(
...
    else if (status == SipServletResponse.SC_OK) {
        if ("INVITE".equals(response.getMethod())) {
            ...
            if (Constants.ACTION_PLAY_ANNOUNCEMENT.equalsIgnoreCase(
                action)) {
                Object sdpObj = response.getContent();
                byte[] sdpBytes = (byte[]) sdpObj;
                String sdp = new String(sdpBytes);
                mediaController.createConnection(PR_JNDI_NAME)
                    .modify("$", sdp);
                ...
            }
        }
    }
@Observer("linkConnected")
public void doLinkConnected(
...
    } else if (Constants.ACTION_PLAY_ANNOUNCEMENT
        .equalsIgnoreCase(action)){
        URL url = (URL) sipSession.getAttribute(
            Constants.AUDIO_FILE);
        if (url != null) {
            playAnnouncement(url);
            ...
        }
    }
}

```

Izvorna koda 4.3: Spremembe v *EventHandler.java*

Poglavje 5

Zaključek

S pomočjo *WebPhone* aplikacije smo pokazali, da je z uporabo pravih orodij mogoče izdelati konvergenčno aplikacijo, ki združuje spletne in telefonske storitve. Pri evalvaciji uporabnosti smo ugotovili, da se da aplikacijo umestiti v telefonski sistem praktično brez sprememb na sami aplikaciji. Aplikacija se je pokazala tudi zadosti modularna, da dodajanje nove funkcionalnosti ni predstavljalo večjih težav. Prenosljivost aplikacije se je v kontekstu prenosljivosti iz enega strežnika na drugega, ter odvisnosti od operacijskega sistema in strojne konfiguracije izkazala za odlično, v spremembi programskega okolja pa prenosljivost ovira odvisnost od uporabljenih orodij.

Pri sami implementaciji ni bilo večjih težav, pokazal pa se je problem pri razširitvi z uporabo *Asterisk* strežnika, opisani na začetku poglavja 4. Ni nam namreč uspelo zagotoviti, da bi *Asterisk* posredoval DTMF signale aplikaciji, namesto da jih prestreže.

Ena izmed željenih uporabljenih metrik je bila tudi zmogljivost sistema, vendar se je izkazalo, da je težko zagotoviti ustrezno testno okolje. Da bi zadostno obremenili sistem, bi potrebovali veliko število SIP agentov, ki pa jih težko simuliramo na enem računalniku. Vsak SIP agent namreč zahteva lastna vrata s številko 5060. Zaradi tega je bila zmogljivost kot metrika opuščena.

Tudi sama aplikacija je izdelana zelo minimalistično, njen namen je bolj demonstracija, kot dejanska uporabnost. Če bi želeli izdelati aplikacijo, ki bi bila po volji uporabnikov, bi jo lahko dopolnili z dodatnimi elementi. Primer je podatkovna baza, v kateri bi hranili podatke uporabnikov: njihova uporabniška imena in gesla, obvestila, zgodovino klicev, kontakte, itd. Uporabniški vmesnik bi lahko razširili, dodali nove vsebine. Implementirali bi lahko tudi nove funkcionalnosti, kot so na primer koledarski opomniki.

Poseben SIP problem, ki se ga diplomska naloga ni dotaknila, predstavlja

NAT. Za prebijanje NAT-a v SIP-u obstajajo posebne metode, ki so opisane v knjigi [14, str. 235-261], in bila lahko bile tema nadaljnjega dela.

Literatura

- [1] C. Blackman, “Convergence between telecommunications and other media: how should regulation adapt?” *Telecommunication Policy*, zv. 22, št. 3, str. 163–170, 1998.
- [2] T. Chujo, M. Katoh, A. Chugo, in H. Sekine, “Service Delivery Platform Technologies,” *FUJITSU Sci. Tech. J.*, zv. 45, št. 3, str. 415–421, 2009.
- [3] R. Collins, “Back to the future: Digital Television and Convergence in the United Kingdom,” *Telecommunication Policy*, zv. 22, št. 4-5, str. 383–96, 1998.
- [4] Comsys, “Comsys Service Delivery Platform,” Dostopno na: <http://www.comsys.net/products-solutions/products/service-delivery-platform.html>, pogledano 11.4.2013.
- [5] J. Deruelle, “Mobicents SIP Servlets Leading Open Source Implementation and HTML5 WebRTC Compliant SIP AS,” Dostopno na: <http://code.google.com/p/sipservlets/>, pogledano 11.4.2013.
- [6] T. Erl, *SOA Principles of Service Design*. Prentice Hall, 2007.
- [7] D. Eror, “Mobile Service Delivery Platform,” Dostopno na: <http://stack.nil.com/ipcorner/MobileServiceDeliveryPlatform/>, pogledano 11.4.2013.
- [8] A. Gate, “Convergence and competition: Technological change, industry concentration and competition policy in the telecommunications sector,” *University of Toronto Faculty of Law Review*, zv. 58, št. 2, str. 83–117, 2000.
- [9] G. Goertzel, “An Algorithm for the Evaluation of Finite Trigonometric Series,” *The American Mathematical Monthly*, zv. 65, str. 34–35, Jan. 1958.

- [10] J. Gosling, B. Joy, G. Steele, G. Bracha, in A. Buckley, *The Java™ Language Specification*, Oracle America, 2012.
- [11] M. Handley, H. Schulzrinne, E. Schooler, in J. Rosenberg, “SIP: Session Initiation Protocol,” RFC 2543 (Proposed Standard), Internet Engineering Task Force, Mar. 1999, zastarano z RFC-ji 3261, 3262, 3263, 3264, 3265. Na voljo na spletu: <http://www.ietf.org/rfc/rfc2543.txt>
- [12] JBoss, “JBoss AS,” Dostopno na: <http://www.jboss.org/jbossas>, pogledano 11.4.2013.
- [13] —, “Seam 2 Distribution Downloads,” Dostopno na: <http://www.seamframework.org/Seam2/Downloads>, pogledano 11.4.2013.
- [14] A. Johnston, *SIP Understanding the Session Initiation Protocol*, 3. izdaja. Artech House, 2009.
- [15] T. Keating, “Internet Phone Release 4,” *Computer Telephony Interaction Magazine*, zv. 1, št. 1, str. 28–31, 1996.
- [16] L. Madsen, J. V. Meggelen, in R. Bryant, *Asterisk, The Definitive Guide*, 3. izdaja. O’Reilly, 2011.
- [17] misch@elara.fsag.de, “MTALK repozitorij,” Dostopno na: <ftp://sunsite.unc.edu/pub/Linux/apps/sound/talk/>, pogledano 11.4.2013.
- [18] M. Mueller, “Digital Convergence and its consequences,” *The Public*, zv. 6, št. 3, str. 11–27, 1999.
- [19] Oracle, “Oracle JDK 7 and JRE 7 certified system configurations,” Dostopno na: <http://www.oracle.com/technetwork/java/javase/config-417990.html>, pogledano 11.4.2013.
- [20] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, in E. Schooler, “SIP: Session Initiation Protocol,” RFC 3261 (Proposed Standard), Internet Engineering Task Force, Jun. 2002, posodobljeno z RFC-ji 3265, 3853, 4320, 4916, 5393, 5621, 5626, 5630, 5922, 5954, 6026, 6141, 6665. Na voljo na spletu: <http://www.ietf.org/rfc/rfc3261.txt>
- [21] K. Sasaki, “Standardization Trends in the TeleManagement Forum (TM Forum),” *Global Standardization Activities*, zv. 10, št. 7, 2009.

- [22] K. Sierra in B. Bates, *SCJP, Sun® Certified Programmer for Java™ 6 Study Guide*. McGraw-Hill, 2008.