KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

PROGRAMŲ INŽINERIJOS KATEDRA

Vilius Panevėžys

Intelektualių tinklų protokolų integracijos MOBICENTS platformoje tyrimas

Magistro darbas

Vadovas
prof. Kęstutis Motiejūnas

KAUNAS, 2011

KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

PROGRAMŲ INŽINERIJOS KATEDRA

Vilius Panevėžys

Intelektualių tinklų protokolų integracijos MOBICENTS platformoje tyrimas

Magistro darbas

Recenzentas **dr. Sigitas Drąsutis** **2011-05-30**	**Vadovas** **prof. Kęstutis Motiejūnas** **2011-05-30** **Atliko** **Vilius Panevėžys, IFM-9/2** **2011-05-30**

KAUNAS, 2011

Turinys

IN protocol integration to Mobicents platform....5
Summary....5
1 Įvadas....6
1.1 Santrauka....6
1.2 Darbo tikslas ir adresatas....6
2 Analitinė dalis....7
2.1 Literatūros analizė....7
2.2 Problemos sprendimas pasaulyje....8
2.2.1 jNetX....8
2.2.2 OpenCloud....9
2.2.3 Situacijos Lietuvoje įvertinimas....9
2.3 Projekto pagrindimas....10
3 Projektinė dalis....12
3.1 Produkto apibūdinimas....12
3.1.1 Programų sistemos funkcijos....12
3.1.2 Sistemos kontekstas....12
3.1.3 Vartotojo charakteristikos....12
3.1.4 Vartotojo problemos....12
3.1.5 Vartotojo tikslai....12
3.1.6 Bendri apribojimai....13
3.2 Projekto įgyvendinimo planai ir kokybės vertinimas....13
3.3 Projektavimo metodologijos ir technologijų analizė....13
3.4 Reikalavimų specifikavimas....14
3.4.1 Sistemos tikslai (paskirtis)....14
3.4.2 Panaudojimo atvejų vaizdas....15
3.4.2.1 Aktoriai ir panaudos atvejai....15
3.4.2.2 PAM diagrama....16
3.4.2.3 Aktorių funkcijos....17
3.4.2.3.1 Administratorius....17
3.4.2.3.2 SBB....17
3.4.2.4 PA specifikacijos....17
3.4.2.4.1 Paleisti RA Entity....17
3.4.2.4.2 Stabdyti RA Entity....18
3.4.2.4.3 Keisti konfigūraciją....18
3.4.2.4.4 Peržiūrėti veiklos statistiką....19
3.4.2.4.5 Trečios šalies skambutis....19
3.4.2.4.6 Specifinio pranešimo siutimas į tinklą....20
3.4.2.4.7 Įvykių filtro taikymas....20
3.5 Funkciniai reikalavimai....21
3.5.1 Veiklos sfera (The scope of the work)....21
3.5.1.1 Veiklos kontekstas....21
3.5.1.2 Veiklos padalinimas....21
3.5.2 Produkto veiklos sfera (The scope of the product)....22
3.5.2.1 Sistemos ribos....22
3.6 Architektūros specifikacija....22
3.6.1 Architektūros tikslai ir apribojimai....22
3.6.2 Sistemos statinis vaizdas....23
3.6.2.1 Komponentų vaizdas....23
....23
3.6.2.2 Klasių diagrama....23
3.6.3 Sistemos dinaminis vaizdas....25
3.6.3.1 Skambučių modelis....25
3.6.4 Būsenų mašinos....26
3.6.4.1 Provider būsenos....26
3.6.4.2 Call būsenos....26
3.6.4.3 Connection būsenos....27
3.6.4.4 TerminalConnection būsenos....28

3.6.5 Tipinių scenarijų sekų diagramos....28
3.6.5.1 Išeinančio skambučio sujungimas....29
3.6.5.2 Įeinančio skambučio sujungimas....34
3.6.5.3 Įeinančio skambučio nukreipimas....35
3.6.5.4 Garsinio pranešimo grojimas (announcement) visiems pokalbio dalyviams....36
3.6.5.5 Garsinio pranešimo grojimas vienam pokalbio dalyviui....36
3.6.5.6 Duomenų surinkimas iš vartotojo....37
3.6.5.7 Apmokestinimas....38
3.6.5.8 Išdėstymo vaizdas....38
4 Tiriamoji ir eksperimentinė dalis....39
4.1 Virtualių mašinų panaudojimo specifika....39
4.1.1 Dinaminis transliavimas (Just-in-time compilation)....39
4.1.1.1 Kas yra JIT?....39
4.1.1.2 JIT privalumai ir trūkumai....40
4.1.1.3 Pradinis uždelsimas....41
4.1.2 Šiukšlių surinkimo (Garbage collection) strategijos....42
4.2 Tyrimo aplinka....43
4.3 Tyrimo rezultatai....45
4.3.1 Nuolatinės apkrovos eksperimentas....45
4.4 Didėjančios apkrovos eksperimentas....46
5 Išvados....48
6 Literatūra....49
7 Terminų ir santrumpų žodynas....50
8 PRIEDAI....51
8.1 Priedas Nr. 1....51
8.2 Priedas Nr. 2....51

IN PROTOCOL INTEGRATION TO MOBICENTS PLATFORM

Summary

Current developments in telecommunication technologies show that price of the basic services have reached minimal values. Costs for service providers to maintain infrastructure, staff have not declined. Therefore, the main method to remain competitive in this dynamic market is by introducing value-added services.

To support easy introduction of new services a specialized platform is required. This paper focuses on integration of IN protocols to an open-source MOBICENTS platform. A solution is proposed to ease service creation employing the MOBICENTS platform. Telecommunication service developers will be presented with an abstract, protocol-agnostic API for call control to enable easy and fast development of universal business logic. Moreover, reliance on open-source components brings many advantages like code reviewed by many developers, tests conducted in various environments, cost-effectiveness.

1 ĮVADAS

1.1 Santrauka

Vystantis ir tobulėjant telekomunikacijų technologijoms ir paslaugoms, paremtoms jomis, bazinių paslaugų kainos krenta. Tačiau paslaugų tiekėjų infrastruktūros, personalo išlaikymo ir kiti kaštai nemažėja. Šiame kontekste konkurencinė kova iš konkuravimo žemiausiomis kainomis persiorientuoja į konkuravimą siūlant įdomias, naudingas papildomas paslaugas, kurios taip pat gali padėti taupyti kaštus telekomunikacinių paslaugų tiekėjų klientams, t.y. praktiškai bet kuriai verslo įmonei.

Tokioms paslaugoms vystyti reikalinga specializuota platforma. Šiame darbe analizuojama galimybė panaudoti atvirojo kodo MOBICENTS platformą integruojant IN protokolus. Kuriama sistema siekia palengvinti skambučių valdymo paslaugų kūrimą MOBICENTS platformos pagrindu. Telekomunikacinių paslaugų kūrėjams bus pateikta programinė sąsaja su papildomu abstrakcijos lygiu, leisiančiu realizuoti nuo konkretaus protokolo nepriklausomą biznio logiką. Kadangi projektas remiasi atviro kodo sprendimais, galutinis produktas turės konkurencinį pranašumą prieš visiškai komercinius analogiškus produktus.

1.2 Darbo tikslas ir adresatas

Darbo tikslas – išanalizuoti IN protokolų integracijos problemas, suprojektuoti ir realizuoti programinį komponentą IN [3] protokolų integracijai (JSLEE Resource Adaptor) į Mobicents [9] platformą. Projekto produktas įgalintų realizuoti įvairias skambučių valdymo paslaugas Mobicents platformos pagrindu. Galutinis produkto vartotojas – fiksuoto ir mobiliojo ryšio operatorių abonentai besinaudojantys tokiomis paslaugomis kaip VPN (*Virtual Private Numbering* – „trumpieji numeriai“). Naudos gavėjas yra operatorius pasirinkęs inovatyvų paslaugų pateikimo būdą: taupomi kaštai dėl IN architektūros savybių, gaunamas papildomas pelnas iš teikiamų papildomų paslaugų arba įgyjamas konkurencinis pranašumas.

2 ANALITINĖ DALIS

2.1 Literatūros analizė

Ilgą laiką telefonijos paslaugos apsiribojo tik dviejų vartotojų sujungimu pokalbiui. To pilnai pakako iki XX amžiaus vidurio. Tačiau operatoriai susidūrė su problema, kad pajamų augimas iš tradicinių paslaugų žymiai sumažėjo. Taip pat verslo klientams nepakako paprastos dviejų abonentų sujungimo paslaugos. Nuo VI dešimtmečio vidurio ryšio tinklų operatoriai pradėjo teikti papildomas paslaugas. Tai, kas šiuo metu suprantama kaip intelektualus tinklas (IN [3]), prasidėjo nuo BELL Labs inžinierių VII dešimtmečio idėjos apie papildomų paslaugų teikimą bendro naudojimo tinkle.

Kai įvairios skambučių valdymo paslaugos (paskutinio skambinusiojo abonento numerio rinkimas, skambučio nukreipimas, blokavimas ir t.t.) buvo įgyvendinamos komutavimo įrangoje, kildavo suderinamumo problemų tarp skirtingų gamintojų įrangos, naujų paslaugų pristatymas buvo neįmanomas be komutacinės įrangos gamintojo įsikišimo. Sekantis žingsnis buvo paslaugų atskyrimas nuo šios komutavimo įrangos ir jos patalpinimas atskiruose tinklo mazguose.

Naujos kartos intelektualusis tinklas [3] suprantamas, kaip paslaugoms atviras telekomunikacijų tinklas. Čia intelektualumas yra perkeliamas iš komutavimo stoties (SSP) į kompiuterinius serviso valdymo mazgus (SCP), kurie yra paskirstyti visame tinkle (šiame darbe daroma prielaida, kad tinklo architektūra paremta SS7 [6][7]). Tokiu būdu tinklo operatoriui sudaroma galimybė daug efektyviau kontroliuoti teikiamas paslaugas bei kurti naujas. Papildomi pajėgumai gali būti sparčiai įvedami į tinklą, o juos įvedus paslaugos gali būti nesunkiai derinamos tarpusavyje pritaikant jas prie kliento poreikių.

Šiuo metu kuriamos IN paslaugos kuriamos naudojant servisų platformą (SDP [8]). SDP koncepcija išsivystė per keletą pastarųjų metų. Terminas SDP reiškia sistemos architektūrą, kuri įgalina efektyvų vienos ar daugiau paslaugų klasių kūrimą, vystymą, organizavimą bei valdymą. SDP atsirado kaip telekomunikacinio tinklo vystymosi pasekmė.

Siekiant sumažinti galutinę paslaugų kainą labai patrauklų SDP pagrindu naudoti atviro kodo paslaugų logikos vykdymo aplinką Mobicents [9]. Mobicents - didelio pralaidumo, įvykiais valdomas taikomųjų programų serveris palaikantis komponentinį modelį ir užtikrinantis stabilią taikomųjų programų vykdymo aplinką. Mobicents yra pirmoji ir vienintelė atviro kodo platforma sertifikuota pagal JSLEE [1][10] standartą.

Kiekvienos SDP platformos svarbi charakteristika – palaikomi telekomunikacinių tinklų protokolai. Tai apsprendžia platformos suderinamumą su egzistuojančia tinklo infrastruktūra ir galimų paslaugų aibę bei funkcionalumą. Projekto tikslas – sukurti programinį komponentą (RA – Resource Adaptor) įvairių skambučių valdymo protokolų integracijai į Mobicents platformą. Siekiama sukurti homogenišką programinę sąsają (API – Application Programming Interface) telekomunikacinių paslaugų kūrėjams. Numatoma API apibrėžiama remiantis egzistuojančiais ir vis dar tobulinamais JAIN (Java APIs for Integrated Networks) grupės standartais: JCC (Java Call Control - JSR 21 [11]) ir JCAT (Java Coordination and Transaction - JSR 122 [12]).

2.2 Problemos sprendimas pasaulyje

Kadangi projekto objektas – programinis komponentas – SDP platformos modulis, tiesioginiais konkurentais tikslinga laikyti alternatyvių SDP platformų gamintojus.

2.2.1 jNetX

Nuo 2001 metų jNetX [2] kompanija pirmoji pradėjo naudoti „carrier“ klasės Java technologijas telekomunikacijų tinkle. jNetX dabar yra pasaulinės rinkos lyderė, užtikrinanti standartus, pagrįstus Java technologijomis, ir taikomąsias programas susiejančias („convergent“) fiksuoto, mobilaus, IMS (ryšio) ir interneto / intraneto aplinkose. jNetX kompanijos lyderystė rinkoje pirmiausia siejama su šiais trimis principais:

Pirma, jNetX technologija užtikrina aukščiausią atviro kodo programavimo, stabilumo ir funkcionalumo rinkoje kombinaciją. jNetX technologija išgauna konkrečios gamintojo tinklo techninės įrangos pajėgumus paslaugai, paverčiant šiuos pajėgumus į bendrą Java sąsajų rinkinį. Tai užtikrina paslaugos mobilumą ir paslaugos pakartotinį naudojimą skirtinguose tinkluose. Toks būdas leidžia grupei veikiančių kompanijų išvystyti tinkamą paslaugos planą („roadmap“) sparčiau ir žymiai mažesnėmis sąnaudomis.

Antra, kaip pirmojo principo pasekmė, jNetX sulaužo tradicinį gamintojo "lock-in“ modelį ir leidžia operatoriui valdyti savo rinkodaros planą („roadmap“), savo paslaugų logiką ir greitai reaguoti į rinkos grėsmes ir galimybes. Su jNetX operatoriai gali laisvai rinktis savo pageidaujamą partnerį integracijai ir turėti naudos iš paslaugų vystytojų kūrybinio fondo.

Trečia, jNetX pramonės trauka skatino turtingą partnerių ekosistemą, jNetX Developer Network Area (DNA), kuri siūlo operatorius visame pasaulyje su plačiu tinklo paslaugų portfeliu. Daugiau nei šimtas tinklo paslaugų yra aprašyta jNetX DNA 100+ tinklo paslaugos

kataloge, siekiant padėti operatoriams kurti naujas idėjas, naujus verslo modelius ir rinkos strategijas.

jNetX pasižymi mobilumu ir nuolat viršija savo verslo tikslus. jNetX yra JAV kompanija, kurioje dabartiniu metu dirba daugiau nei 160 patyrusių komunikacijų ir informacinių technologijų profesionalų tokiose šalyse, kaip JAV (Dalasas), Jungtinė Karalystė (Londonas), Prancūzija (Sofija-Antipolis), Tailandas (Bankokas), Meksika (Meksika) ir Rusija (Maskva)).

2.2.2 OpenCloud

OpenCloud [5] aprūpina telekomunikacijų sferą Rhino serveriu. Tai realaus laiko taikymų serveris siekiant sparčiai vystyti, naudoti ir efektyviai valdyti asmuo-į-asmenį komunikacijų paslaugas visoje dabartinės ir būsimos kartos technologijų srityje. Rhino yra didelio našumo, paslaugų vykdymo aplinka realizuoti paslaugų pateikimą (*Next Generation Service Delivery* platformą (NG-SDP)). OpenCloud būstinė yra Kembridžas, Jungtinė Karalystė su R&D, inžinerija ir palaikymu Naujojoje Zelandijoje ir Ispanijoje bei filialais JAV, Singapūre ir Japonijoje. Kompanijoje dirba daugiau nei 50 darbuotojų.

Rhino galima įsigyti tiesiogiai arba per pasirinktus tinklo techninės įrangos gamintojus, vietinius partnerius bei per globalius paslaugų / taikymų vystymo partnerius. Sukurti taikymo pavyzdžiai yra virtualus privatus tinklas (VPN), realaus laiko personalizuotas garsinis skambėjimo tonas, reklamos remiamas skambutis, personalizuotos skambučio kontrolės taisyklės, iš anksto apmokėtos paslaugos skambučio kontrolė, keli numeriai vienam įrenginiui, skambučio kontrolė ir skambučio tęstinumas.

2.2.3 Situacijos Lietuvoje įvertinimas

Šiuo metu įkainiai už mobilius pokalbius sumažėję, yra arti minimumo ribos ir panašūs pas visus Operatorius. Pagrindinė konkurencijos tarp Operatorių priemonė - tai papildomos paslaugos. Svarbiausia papildomų paslaugų dalis - tai intelektualių tinklų paslaugos (IN Services/IN Paslaugos). Tokių paslaugų realizavimas, stebėsena bei valdymas reikalauja paslaugų pateikimo platformos (SDP - Service Delivery Platform). Diegiant pavienes paslaugas įvairių technologijų pagrindu ir nenaudojant vieningos SDP kiekvienos paslaugos kaina bei eksploatacijos kaštai ženkliai išauga, taip pat nukenčia ir paslaugų kokybė. Todėl SDP ir jos pagrindu sukurtos IN paslaugos yra paklausios rinkoje.

2.3 Projekto pagrindimas

Šiuo metu telekomunikacinių tinklų operatoriai dėl konkurencijos priversti mažinti pokalbių kainas, todėl yra ieškomi kiti pajamų šaltiniai. Vienas iš būdų - teikti intelektualių tinklų paslaugas. Todėl išaugo poreikis IN paslaugų kūrimui. Kuriant minėtas paslaugas susiduriama su keliomis problemomis. Pirmoji - aukšta IN paslaugų teikimo platformų kaina. Nors rinkoje yra visa eilė platformų tiekėjų HP, jNetX, Alcatel, tačiau šių platformų kaina labai iškelia galutinę paslaugos kainą operatoriui ir tai ženkliai mažina IN paslaugų pardavimo/diegimo galimybes. Antra problema - IN paslaugoms labai svarbus „Time to Market“ parametras, t. y. jei paslauga realizuojama per ilgai, ji pasensta moraliai ir tampa nebereikalinga. Šio projekto tikslas sumažinti bent kai kurias iš šių problemų: aktualių protokolų palaikymas didina Mobicents platformos vertę ir įgalina kurti inovatyvias paslaugas už konkurencingą kainą.

IN evoliucionuojant, paslaugų tiekėjai susiduria su nevienareikšmišku galimybių pasirinkimu vystant paslaugų tinklą. Kadangi intelektualaus tinklo paskirtis yra tenkinti vis besikeičiančius klientų poreikius, tinklo intelektualumas tampa vis daugiau paskirstytas ir sudėtingas. Pavyzdžiui, kai trečiosios šalies paslaugų tiekėjai jungiami bendros paskirties tinklais vietinio numerio perkeliamumas (LNP – local number portability) sukelia daug problemų. Tai gali būti išspręsta IN aplinkoje. Kiti IN paslaugų pavyzdžiai:

- Trečios šalies skambutis;
- Skambučio pranešimas;
- Skambučio valdymas;
- Konferencija;
- Netrukdyti (DND);
- Skambučio atranka;
- Telefono numerio mobilumas;
- Nemokamas skambutis;
- Skambučių paskirstymas remiantis įvairiais skambučio kriterijais (vietovės, laiko, proporcinis paskirstymas);
- Išankstinis apmokėjimas ir t. t.

Didžioji dauguma SDP išvystymų pastūmėti problemų siejamų su naujų paslaugų kūrimo laiku ir kaštais. Rinkos apžvalgos rodo, kad du trečdaliai SDP investicijų buvo įpareigotos sumažinti veiklos ir kapitalo išlaidas, ir ženkliai pagreitinti naujų paslaugų

įsiskverbimą į rinką. Dauguma operatorių tikėjosi, kad bendra atvirais standartais pagrįsta architektūra, išspręs prisirišimą prie konkretaus paslaugos išdirbėjo.

Tyrimas patvirtino, kad dauguma sėkmingų SDP diegimų, kai operatorius turėjo aiškiai apibrėžto verslo modelį. Priešingai, kai SDP diegimas buvo paskatintas grynai „technologijos spaudimo“ ir stokojo komercinės prasmės, privesdavo prie nesėkmių. Šiandien, nei vienas operatorius neinvestuoja į SDP be verslo plano, kuris garantuotų greitą investicijų grąžą ir komercinę sėkmę.

Šiandien, SDP yra integruotos su dideliu pasisekimu mobilių paslaugų ir turinio srityse. Bet SDP integracija balso ir kitoms pagrindinėms telekomunikacinėms paslaugoms nepasiekė to paties rinkos sėkmės lygio. Dabartinis rinkos susijungimas SDP erdvėje yra vienoje eilėje kartu su normalia rinkos dinamika ir verslo gyvavimo ciklo konkurenciniu spaudimu. Šis susijungimas taip pat įrodo, kad paslaugų tiekėjai žvalgosi visapusių, galutinių sprendimų SDP produktus atitinkančius jų techninius ir verslo tikslus.

Moriana tyrimo [4] rezultatai rodo, kad daugelis operatorių galiausiai investuos į SDP, bet pabrėžia, kad tai bus ilgalaikis procesas ir 2/3 operatorių perėjimas prie SDP paslaugų teikimo, priklausys nuo technologinių sprendimų su gera investicijų grąža atsiradimo, t.y. SDP versijų atsiradimo, kurios turės didelės integracijos paslaugų ir jų taikymų.

3 PROJEKTINĖ DALIS

3.1 Produkto apibūdinimas

3.1.1 Programų sistemos funkcijos

Kuriama programinė sistema – tarsi tarpininkas tarp telekomunikacinio tinklo ir Mobicents platformos, integruojantis Mobicents su egzistuojančiu tinklu. Iš sistemos apibrėžimo išplaukia du pagrindiniai darbo scenarijai:

- iš tinklo gaunamų telekomunikacinių protokolų, pavyzdžiui, CAP, INAP pranešimų dekodavimas ir atitinkamų įvykių pateikimas į JSLEE platformą – Mobicents;
- JSLEE platformoje veikiančių telekomunikacinių paslaugų kreipinių per pateiktą programinę sąsają konvertavimas į atitinkamus telekomunikaciniame tinkle veikiančių protokolų pranešimus.

3.1.2 Sistemos kontekstas

Programinė sistema – JSLEE platformos komponentas, neveikiantis nepriklausomai. Iš kylantys reikalavimai – integracija su platformos konfigūracijos valdymo, statistinių ir gyvybinių parametrų stebėsenos ir kitomis posistemėmis.

3.1.3 Vartotojo charakteristikos

Vartotojas – telekomunikacinių paslaugų abonentas – niekada tiesiogiai su sistema nesąveikauja. Su sistema dirba administratorius atsakingas už platformos eksploataciją.

3.1.4 Vartotojo problemos

Kaip jau minėta, abonentams sistema tiesioginės įtakos nedaro. Sistemos administratorius atliks įprastas platformai eksploatacines užduotis.

3.1.5 Vartotojo tikslai

- Palengvinti telekomunikacinių paslaugų kūrimą;
- sutrumpinti paslaugų kūrimui reikalingą laiką;
- sumažinti paslaugų kūrimo kaštus;
- užtikrinti stabilų ir nenutrūkstamą paslaugų teikimą.

3.1.6 Bendri apribojimai

Kadangi kuriama ne nepriklausoma sistema, o programinis komponentas Mobicents platformos galimybių išplėtimui, didžiausi apribojimai kyla dėl integracijos su platforma. Kita vertus, sistema nepriklausoma nei nuo operacinės sistemos, nei nuo aparatinės įrangos.

3.2 Projekto įgyvendinimo planai ir kokybės vertinimas

Produkto kokybės patikrai pilna testavimo aplinka – platforma, kadangi kuriamas komponentas nėra savarankiška sistema. Kokybės kriterijai:

- funkciniai
  - protokolo realizacijos atitikimas standartams
  - pateikiamos sąsajos patogumas, lankstumas, pilnumas;
- našumo;
- patikimumo.

Funkcinis kokybės kriterijus tenkinamas jeigu realizuotas komponentas geba dirbti realiame tinkle, t.y. tinklo protokolų realizacija teisinga. Kitas svarbus funkcinis kriterijus – pateikiamos sąsajos patogumas, kadangi įterpiamas papildomas abstrakcijos lygis, turintis padėti atsiriboti nuo konkretaus protokolo specifikos, o lankstumo ir pilnumo reikalavimai užtikrina, kad pateikta sąsaja nebūtų ribojanti.

Dėl taikomosios srities specifikos itin svarbūs kriterijai – našumas ir patikimumas. Našumas testuojamas dirbtinai sukeliant pikines apkrovas. Tai taip pat atlieka ir patikimumo patikrą. Papildomai patikimumas testuojamas provokuojant nenumatytas situacijas, su klaidingais duomenimis ir pan.

3.3 Projektavimo metodologijos ir technologijų analizė

Projektavimas remiasi UML modeliavimu. Naudotas įrankis BoUML, kuriuo ne tik sukurtas sistemos modelis, bet ir sugeneruotas pradinis kodas, įrankis taip pat palaiko *round-trip* inžineriją.

Dėl projektuojamos sistemos specifikos (skambučių apdorojimas), modelyje svarbiausios – sekos. Šioje dalykinėje srityje dažnai vadinamos „*call flows*“. Šiomis sekomis detaliai specifikuota sistemos dinaminė elgsena.

Be UML modelio, visi sprendimai fiksuojami architektūrinėje dokumentacijoje. Vėlesniuose projekto vystymo etapuose ji naudojama, kaip šaltinis vartotojo vadovams kurti.

Naudoti įrankiai:

- Eclipse IDE su papildomais įskiepiais;
- BoUML UML modeliavimui;
- SVN versijų kontrolei;
- Ant automatizuotam transliavimui ir paketų ruošimui;
- Wireshark protokolų analizei.

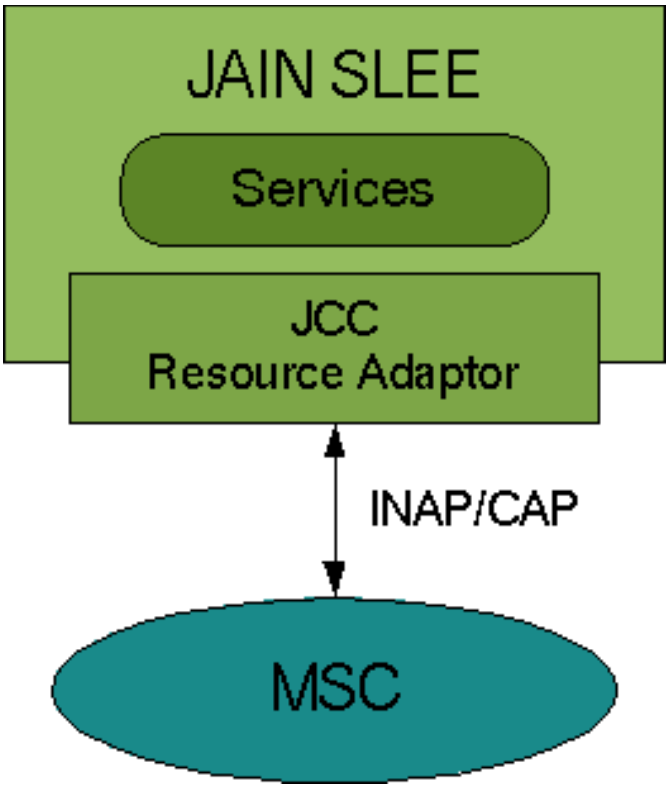
Iš reikalavimų sistemai (tikslinė aplinka - JSLEE konteineris) išplaukia, kad pagrindinė programavimo technologija – Java.

3.4 Reikalavimų specifikavimas

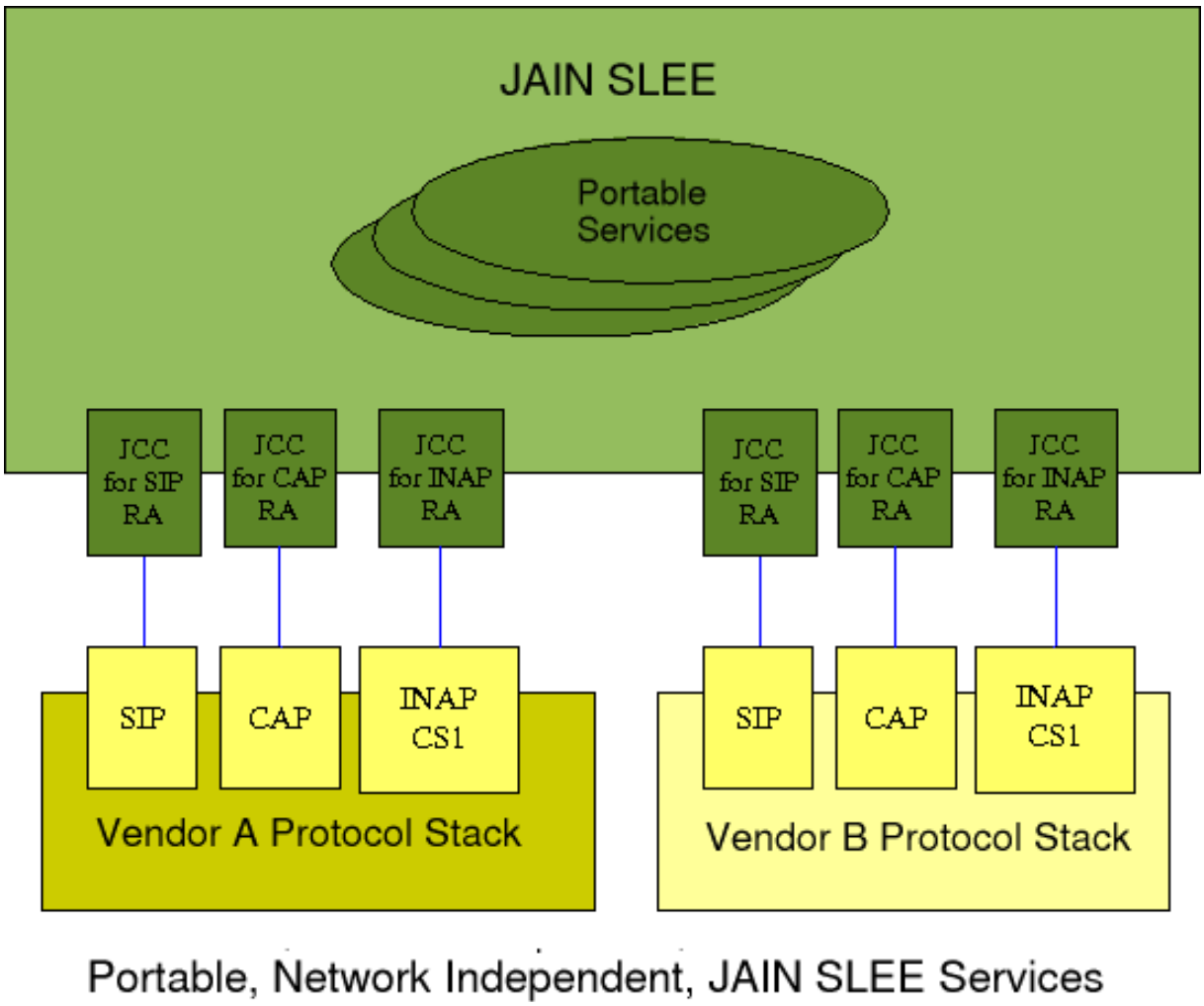
3.4.1 Sistemos tikslai (paskirtis)

Projektuojamas JSLEE modulis (Resource Adaptor) Mobicents platformos integracijai su įvairiais telekomunikaciniuose tinkluose naudojamais skambučių valdymo protokolais, kaip parodyta 3.1. Pav. SLEE ir RA sąryšis. .

3.2. Pav. Paslaugos, sukurtos remiantis projektuojamu RA nepriklauso nei nuo protokolų steko, nei nuo konkretaus protokolo. iliustruoja kitą sistemos paskirties aspektą: sukūrimą abstrakčios programinės sąsajos leidžiančios kurti paslaugas nepriklausančias nuo konkrečių protokolų.



3.1. Pav. SLEE ir RA sąryšis.



3.2. Pav. Paslaugos, sukurtos remiantis projektuojamu RA nepriklauso nei nuo protokolų steko, nei nuo konkretaus protokolo.

Projektuojamo modulio ir jo pateikiamo API tikslas – pateikti telekomunikacinių paslaugų kūrėjams patogią, efektyvią ir homogenišką programinę sąsają skambučių valdymui, leidžiančią ja paremtoms paslaugoms maksimaliai atsiriboti nuo konkretaus skambučių valdymo protokolo. Tokiu būdu supaprastės telekomunikacinių paslaugų kūrimas, jas bus galima pristatyti greičiau ir pigiau, taip operatoriai galės efektyviau konkuruoti pristatydami aktualias paslaugas.

3.4.2 Panaudojimo atvejų vaizdas

3.4.2.1 Aktoriai ir panaudos atvejai

Administratorius

- paleisti *RA entity*
- stabdyti *RA entity*
- keisti konfigūraciją

- peržiūrėti veiklos statistiką

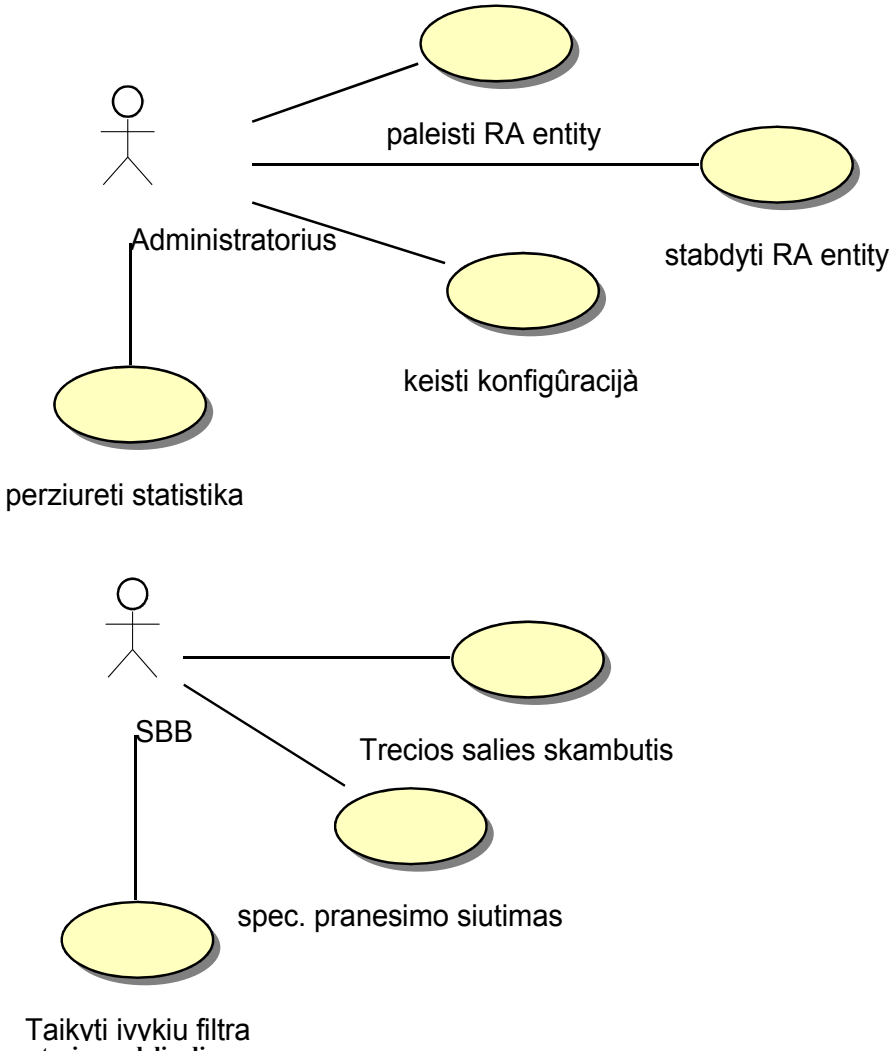
SBB (*Service Building Block*)

- Trečios šalies skambutis
- specifinio pranešimo siuntimas į tinklą
- taikyti įvykių filtrą

SLEE

SLEE nuolat sąveikauja su šiuo komponentu, kaip konteineris su valdomu moduliu: būsenos valdymas (per specifikacijos apibrėžtus *life-cycle* metodus), informavimas apie (ne)sėkmingą įvykio apdorojimą, egzistuojančių sesijų būsenos pasikeitimus ir t.t. Šios sąveikos yra esminės integracijai su platforma ir modulio veikimui, tačiau projektavimo požiūriu trivialios, todėl nebus nagrinėjamos.

3.4.2.2 PAM diagrama



3.3. Pav. Panaudos atvejų modelio diagrama.

3.4.2.3 Aktorių funkcijos

3.4.2.3.1 Administratorius

Administratorius – asmuo atsakingas už sklandžią platformos, įskaitant ir projektuojamą modulį, eksploataciją. Vienu metu platformoje, kurioje įdiegtas modulis, gali egzistuoti daugiau nei vienas aktyvus RA (*RA entity*), pavyzdžiui, su skirtinga konfigūracija. Administratorius gali stabdyti bet kurį iš jų, keisti jų konfigūraciją, taip pat peržiūrėti statistinius duomenis tokius, kaip aktyvios sesijos, nepavykę skambučiai, maksimalus skambučių per sekundę skaičius ir pan.

3.4.2.3.2 SBB

Service Building Block – telekomunikacinės paslaugos realizacija, veikianti platformoje. Kiekvienas SBB – komponentas paruoštas pagal JSLEE reikalavimus ir realizuojantis biznio logiką tam tikrame scenarijuje. Bendru atveju, tai, kas vartotojo suvokiama, kaip viena paslauga, gali sudaryti ir daugiau nei vienas SBB. Priklausomai nuo konkrečios paslaugos logikos SBB dalyvauja vienoje ar daugiau skambučio apdorojimo fazių gaudamas apie skambučio būseną informuojančius įvykius ir kaip atsakymą pateikdamas instrukcijas tolesniam skambučio apdorojimui. Šiame darbe nagrinėjami panaudos atvejai:

- skambučio inicijavimas;
- specifinio pranešimo į tinklą siutimas (reikalingas, kai prieinamas API yra nepakankamas ir reikalinga suformuoti specifinį konkretaus protokolo pranešimą);
- įvykių filtro taikymas (naudojamas norint gauti tik tam tikrus,o ne visus įvykius, pavyzdžiui, kai SBB dalyvauja tik pradinėje skambučio apdorojimo fazėje).

3.4.2.4 PA specifikacijos

3.4.2.4.1 Paleisti *RA Entity*

- Tikslas: Aktyvuoti *RA entity*
- Aktoriai: administratorius
- Ryšiai su kitais PA: nėra
- Nefunkciniai reikalavimai: operacijos vykdymo laikas – iki 10 sekundžių
- Prieš-sąlygos: startuojamas *RA entity* turi būti neaktyvus
- Sužadinimo sąlyga: administratorius įvykdo *start* komandą su argumentu – startuojamo *RA entity* vardu

- Po-sąlyga: nurodytas *RA entity* pereina į būseną *STARTED*
- Pagrindinis scenarijus: vykdoma inicializacija, darbui reikalingų resursų alokavimas (pavyzdžiui, prisijungimas prie protokolų steko)
- Alternatyvūs scenarijai: nepavykus inicializacijai, jau alokuoti resursai dealokuojami, problema raportuojama administratoriui

3.4.2.4.2 Stabdyti *RA Entity*

- Tikslas: sustabdyti *RA entity*
- Aktoriai: administratorius
- Ryšiai su kitais PA: nėra
- Nefunkciniai reikalavimai: operacijos vykdymo laikas – iki 10 sekundžių
- Prieš-sąlygos: stabdomas *RA entity* turi būti būsenoje *STARTED*
- Sužadinimo sąlyga: administratorius įvykdo *stop* komandą su argumentu – stabdomo *RA entity* vardu
- Po-sąlyga: nurodytas *RA entity* pereina į būseną *STOPPED*
- Pagrindinis scenarijus: RA vykdo deinizializaciją ir visų resursų dealokavimą
- Alternatyvūs scenarijai: nepavykus operacijai administratorius informuojamas apie priežastis

3.4.2.4.3 Keisti konfigūraciją

- Tikslas: pakeisti platformoje veikiančio *RA entity* konfigūracinius parametrus
- Aktoriai: administratorius
- Ryšiai su kitais PA: nėra
- Nefunkciniai reikalavimai: manipuliuojamo *RA entity* būsena nekinta. Jeigu nurodyto parametro naujos reikšmės pritaikymui reikalingas stabdymas ir startavimas – apie tai turi būti informuotas administratorius.
- Prieš-sąlygos: *RA entity* turi bent vieną konfigūracinį parametrą
- Sužadinimo sąlyga: administratorius iškviečia atitinkamą komandą ir pateikia naują parametro reikšmę
- Po-sąlyga: nurodytam konfigūraciniam parametrui suteikta nauja reikšmė
- Pagrindinis scenarijus: pritaikomi ir išsaugomi konfigūracijos pakeitimai

- Alternatyvūs scenarijai: jeigu nauja parametro reikšmė sutampa su senąja, jokie pakeitimai nevykdomi, nereikalaujamas perstartavimas, net jei tai būtų reikalinga pasikeitus šio parametro reikšmei

3.4.2.4.4 Peržiūrėti veiklos statistiką

- Tikslas: statistikos peržiūra. Administratorius remdamasis gautais duomenimis gali daryti išvadas apie modulio veikimo efektyvumą, diagnozuoti kilusias problemas, daryti išvadas apie apkrautumą tam tikru metu ir t.t.
- Aktoriai: administratorius
- Ryšiai su kitais PA: nėra
- Nefunkciniai reikalavimai: visi raportuojami parametrai turi būti teisingi, atnaujinami ne rečiau kaip kas 10 sekundžių.
- Prieš-sąlygos: *RA entity* turi būti būsenoje *STARTED*
- Sužadinimo sąlyga: administratorius iškvietė atitinkamą komandą
- Po-sąlyga: rodomos visų statistinių parametrų reikšmės
- Pagrindinis scenarijus: parodomos momentinės statistinių parametrų reikšmės ir tęsiamas jų skaičiavimas
- Alternatyvūs scenarijai: jei operacija iškviečiama ne *STARTED* būsenoje, rodomos pradinės parametrų reikšmės

3.4.2.4.5 Trečios šalies skambutis

- Tikslas: skambučio inicijavimas iš aplikacijos
- Aktoriai: SBB
- Ryšiai su kitais PA: nėra
- Nefunkciniai reikalavimai: skambutis inicijuojamas panaudojant maksimalią aibę parametrų iš įvykio sužadinusio SBB
- Prieš-sąlygos: *RA entity* turi būti būsenoje *STARTED*, SBB turi būti aktyvuotas kurio nors iš užsakytų įvykių
- Sužadinimo sąlyga: SBB aktyvavo operaciją *createCall()*
- Po-sąlyga: sukuriamas skambutis *IDLE* būsenoje
- Pagrindinis scenarijus: skambutis sėkmingai sukurtas

- Alternatyvūs scenarijai: nesėkmės atveju problema raportuojama standartinėmis priemonėmis (*Java Exception*)

3.4.2.4.6 Specifinio pranešimo siutimas į tinklą

- Tikslas: išsiųsti pranešimą, nenumatytą API operacijose
- Aktoriai: SBB
- Ryšiai su kitais PA: nėra
- Nefunkciniai reikalavimai: pranešimo ilgis bus apribotas naudojamo protokolų steko
- Prieš-sąlygos: *RA entity* turi būti būsenoje *STARTED*, SBB turi būti aktyvuotas kurio nors iš užsakytų įvykių
- Sužadinimo sąlyga: SBB aktyvavo atitinkamą operaciją
- Po-sąlyga: suformuotas pranešimas išsiųstas
- Pagrindinis scenarijus: pranešimas sėkmingai išsiunčiamas
- Alternatyvūs scenarijai: jei pranešimas neteisingai suformuotas, problema raportuojama standartinėmis priemonėmis (*Java Exception*)

3.4.2.4.7 Įvykių filtro taikymas

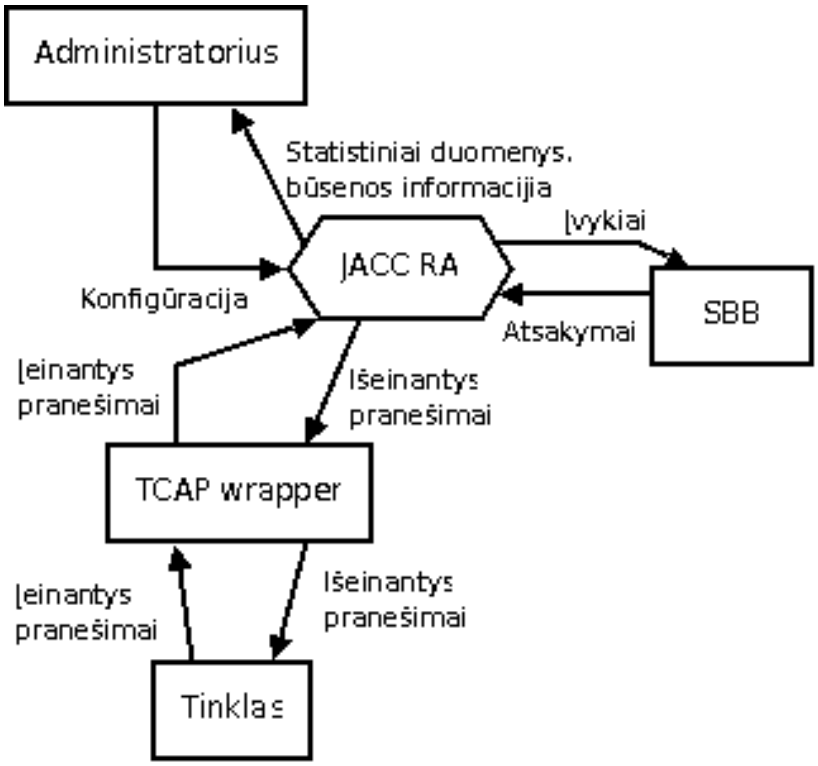
- Tikslas: pritaikyti įvykių filtrą siekiant sumažinti įvykių, nebūtinų paslaugos logikos vykdymui, srautą ir taip padidinti efektyvumą, bendrą pralaidumą ir greitaveiką
- Aktoriai: SBB
- Ryšiai su kitais PA: nėra
- Nefunkciniai reikalavimai: filtro pritaikymas turi būti vykdomas iškart, ir nepriklausomai nuo to kiek ir kokių įvykių jau yra gauta
- Prieš-sąlygos: *RA entity* turi būti būsenoje *STARTED*
- Sužadinimo sąlyga: SBB iškvietė atitinkamą operaciją su nurodytu standartiniu ir specifiniu įvykių filtru.
- Po-sąlyga: SBB bus notifikuojamas tik apie įvykius leistinus pagal nurodytą filtrą

- Pagrindinis scenarijus: korektiškas filtras pritaikomas ir susiejamas su šiuo SBB
- Alternatyvūs scenarijai: jei filtras nekorektiškas, įvykiai toliau pristatomi pagal ankstesnį korektišką filtrą. Jei joks filtras nebuvo pritaikytas – pristatomi visi įvykiai.

3.5 Funkciniai reikalavimai

3.5.1 Veiklos sfera (The scope of the work)

3.5.1.1 Veiklos kontekstas



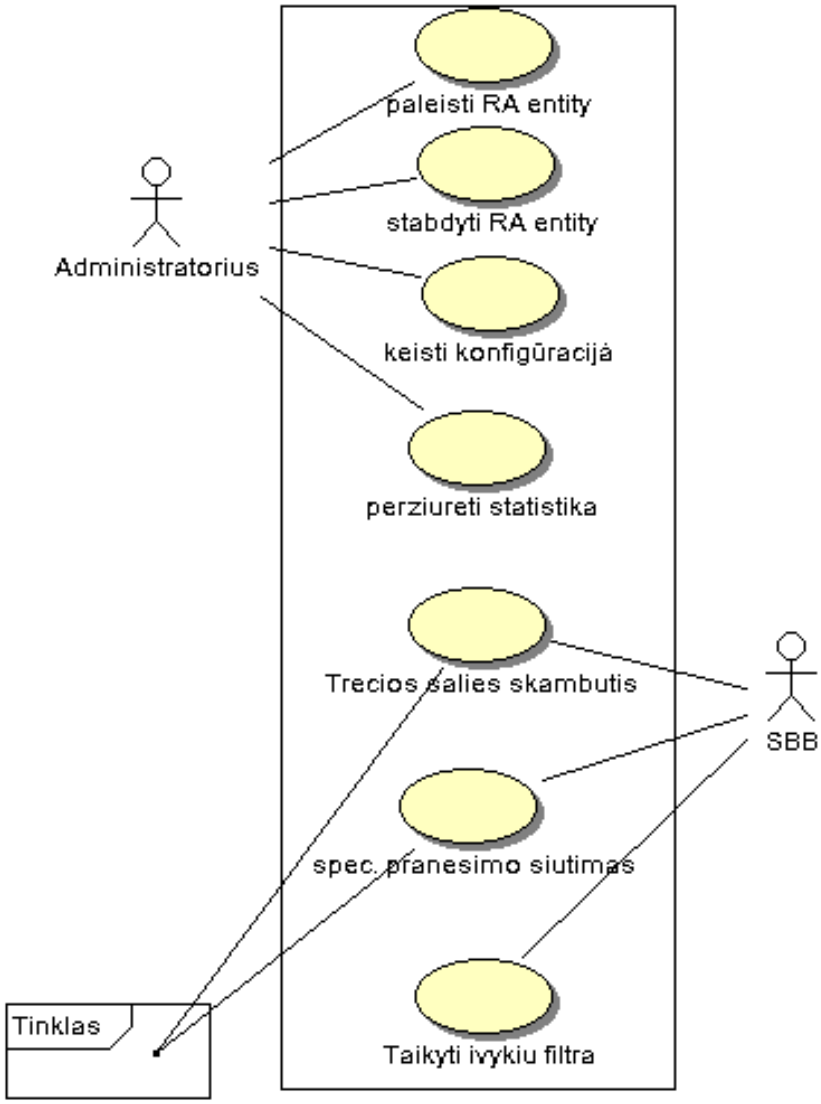
3.4. Pav. Sistemos kontekstas.

3.5.1.2 Veiklos padalinimas

Nr.	Įvykio pavadinimas	Įeinantys/išeinantys informacijos srautai
1	Įeinantis pranešimas iš tinklo	Įvykio detalės (in)
2	Pranešimo išsiuntimas į tinklą	Pranešimo argumentai (out)
3	Konfigūracijos keitimas	Konfigūraciniai parametrai (in)
4	Statistikos užklausa	Statistiniai ir būsenos duomenys (out)
5	Įvykio raportavimas paslaugos logikai	Įvykio detalės (out)
6	Atsakymas iš paslaugos logikos	Tolesnio skambučio apdorojimo instrukcijos

3.5.2 Produkto veiklos sfera (*The scope of the product*)

3.5.2.1 Sistemos ribos



3.5. Pav. Sistemos ribos.

3.6 Architektūros specifikacija

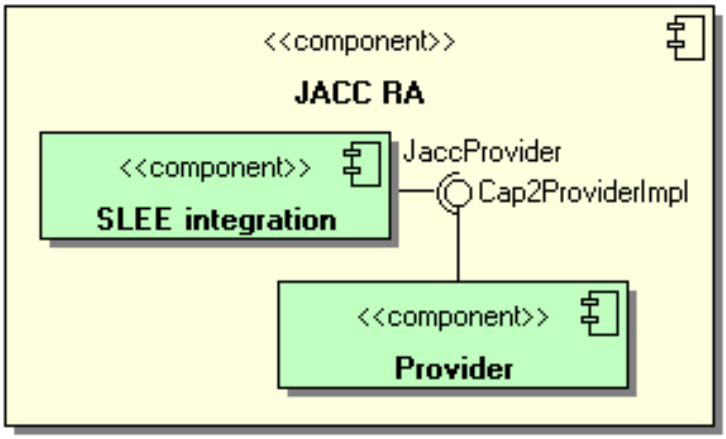
3.6.1 Architektūros tikslai ir apribojimai

Kuriama ne atskira sistema, o komponentas, todėl būtina sąlyga – integracija su tiksline platforma – Mobicents, paremta JSLEE specifikacija.

Komponento paskirtis – pranešimų apdorojimas realiu laiku. Iš to išplaukiantys nefunkciniai reikalavimai – aukštas pralaidumas ir greitaveika.

3.6.2 Sistemos statinis vaizdas

3.6.2.1 Komponentų vaizdas

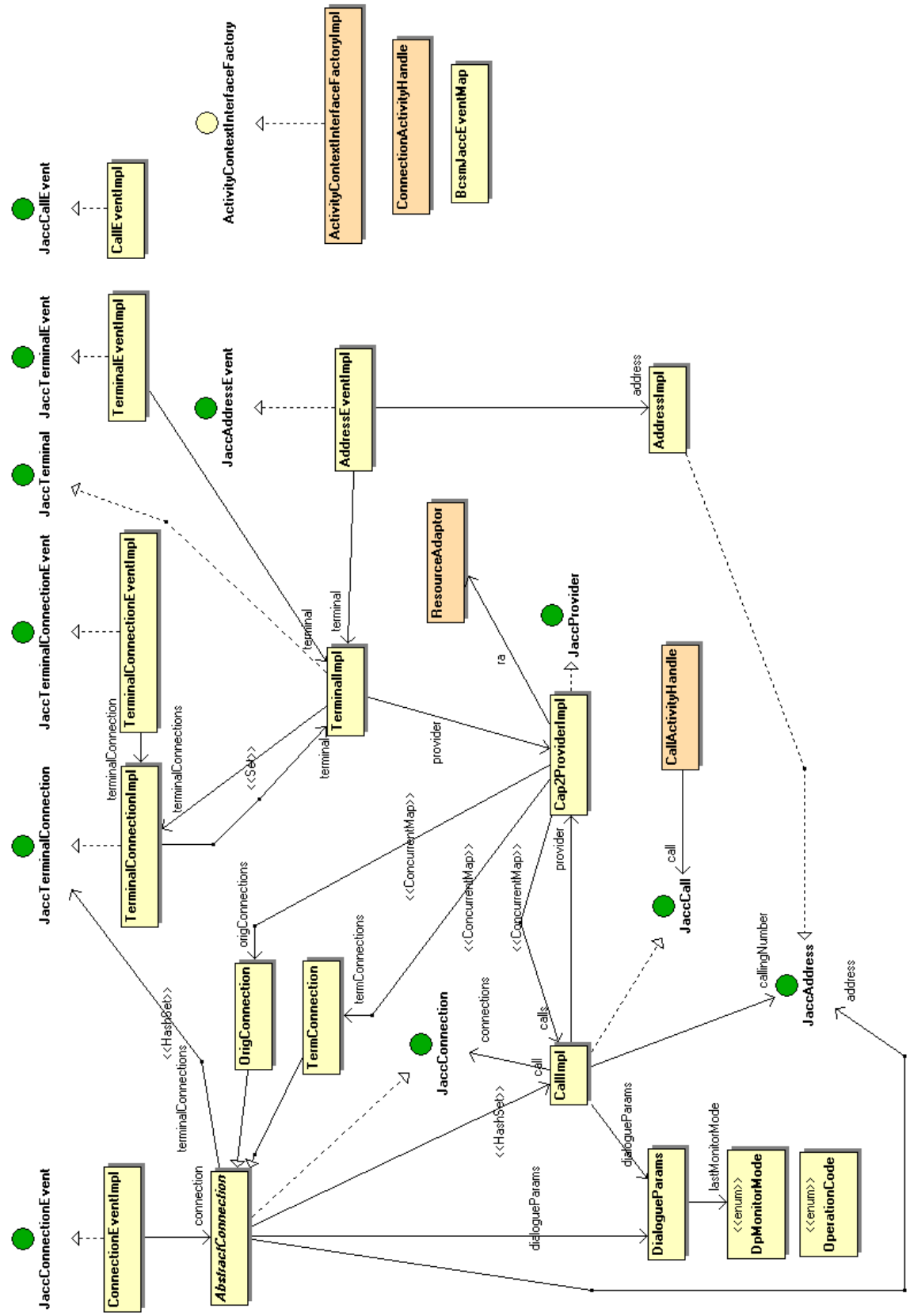


3.6. Pav. Pagrindiniai sistemos komponentai.

Sistemą galima suskirstyti į dvi pagrindines funkciškai skirtingas dalis, kaip parodyta 3.2. Pav. Paslaugos, sukurtos remiantis projektuojamu RA nepriklauso nei nuo protokolų steko, nei nuo konkretaus protokolo. „SLEE integration“ komponentas – aibė klasių atsakingų už RA integraciją su konteineriu. Daugelis jų – įgyvendina JSLEE standarte numatytas sąsajas. „Provider“ komponentas įgyvendina konkretaus protokolo logiką. Architektūroje iš anksto numatoma patogi galimybė realizuoti papildomų protokolų palaikymą – tam apibrėžtos sąsajos.

3.6.2.2 Klasių diagrama

3.7. Pav. Sistemos klasių diagrama pavaizduota bendra sistemos klasių diagrama.



3.7. Pav. Sistemos klasių diagrama

3.6.3 Sistemos dinaminis vaizdas

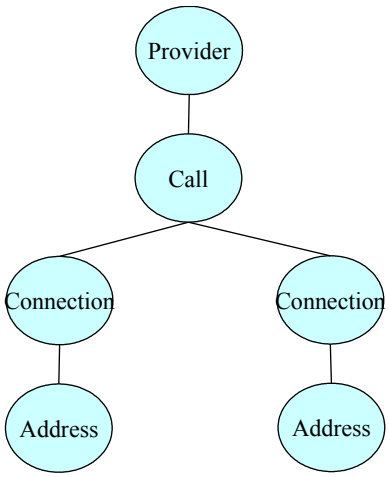
3.6.3.1 Skambučių modelis

Kuriama programinė sąsaja pateikia bendrinį skambučių modelį atvaizduojantį esminius aspektus remiantis egzistuojančiais modeliais. Tokiu būdu siekiama vartotojui pateikti patogią ir funkcionalią programinę sąsają skambučių valdymui.

Programinę sąsają sudaro penki pagrindiniai objektai modeliuojantys esmines esybes, reikalingas daugelio paslaugų realizavimui:

- Provider – lyg langas, per kurį taikomoji programa stebi skambučių apdorojimą;
- Call – atitinka skambutį ir yra fizinių ir loginių esybių rinkinys, sujungiantis du ar daugiau galinių įrenginių;
- Address – adresas – atitinka loginį galinį įrenginį;
- Connection – atspindi dinaminį sąryšį tarp Call ir Address.
- Terminal – vaizduoja fizinį galinį įrenginį, pavyzdžiui, telefoną, delninį kompiuterį ir pan.
- TerminalConnection – modeliuoja būseną nusakančią sąryšį tarrp Terminal ir Connection.

Šių objektų tarpusavio sąryšiai pavaizduoti 3.5. Pav. Sistemos ribos. Papildomi pokalbio dalyviai gali būti modeliuojami be jokių apribojimų – pridedant papildomus Connection ir Addess objektus susietus su tuo pačiu skambučiu. Šis modelis yra simetrinis, nes nediferencijuojamas skambučio iniciatorius ir gavėjas. Modelis taip pat yra pilnas, nes gali būt modeliuojami visos skambučio šalys.

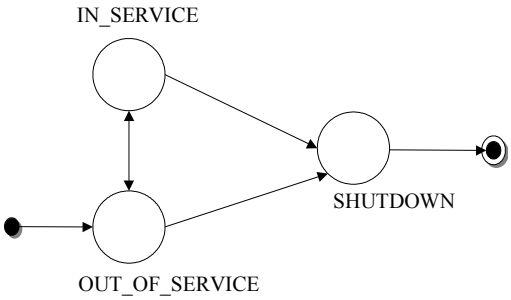


3.8. Pav. Objektinis skambučio modelis.

3.6.4 Būsenų mašinos

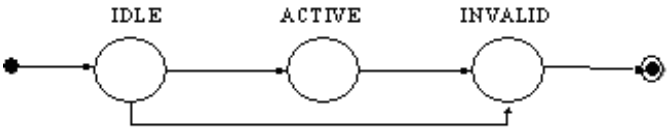
Provider, Call, Connection ir TerminalConnection objektų elgsena apibrėžiama baigtiniais automatais, nusakančiais būsenų kitimą.

3.6.4.1 Provider būsenos



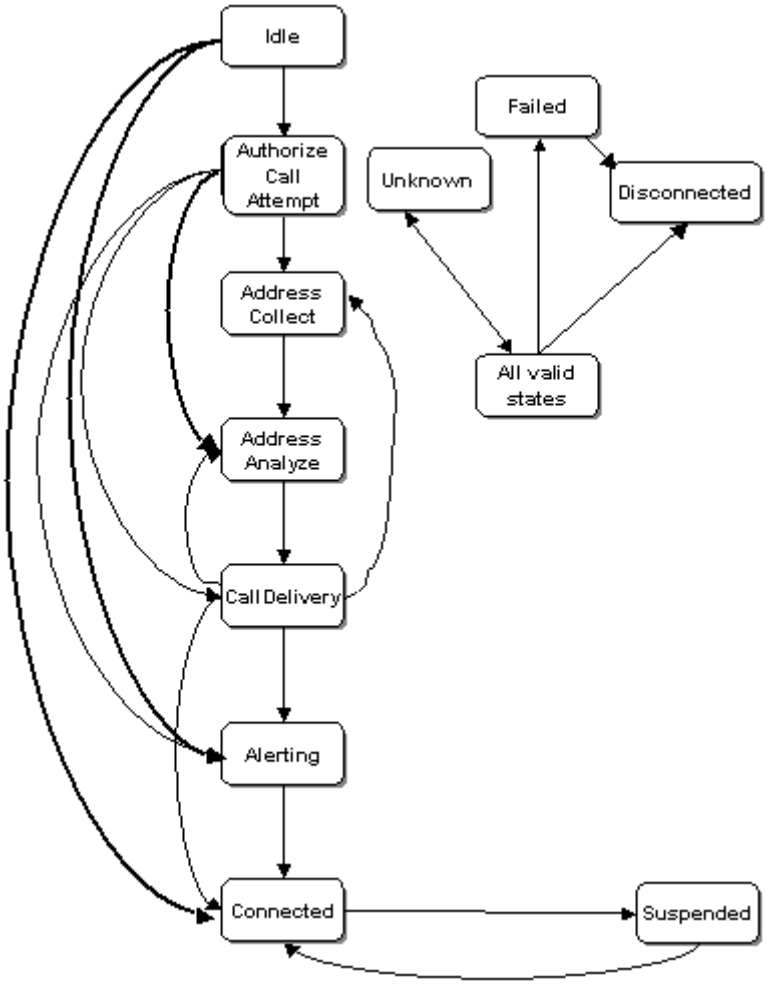
3.9. Pav.Provider būsenų automatas.

3.6.4.2 Call būsenos



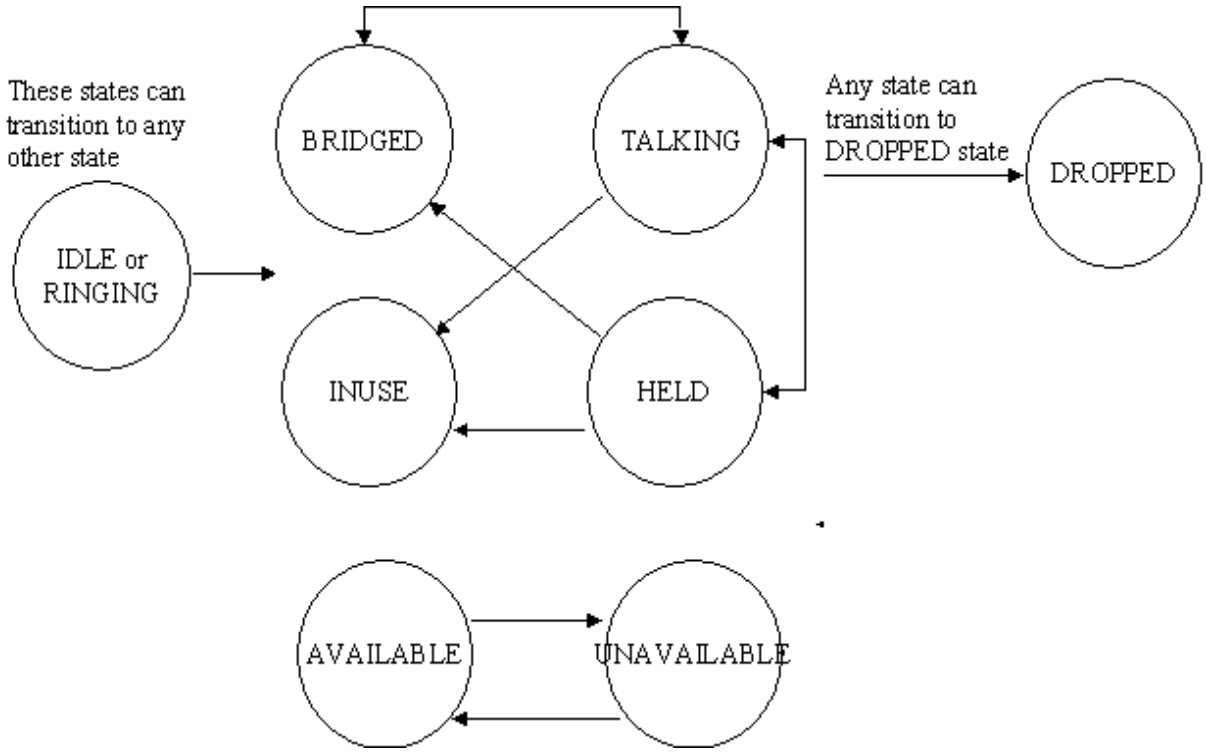
3.10. Pav. Call būsenų automatas.

3.6.4.3 Connection būsenos



3.11. Pav. Connection būsenų automatas.

3.6.4.4 TerminalConnection būsenos

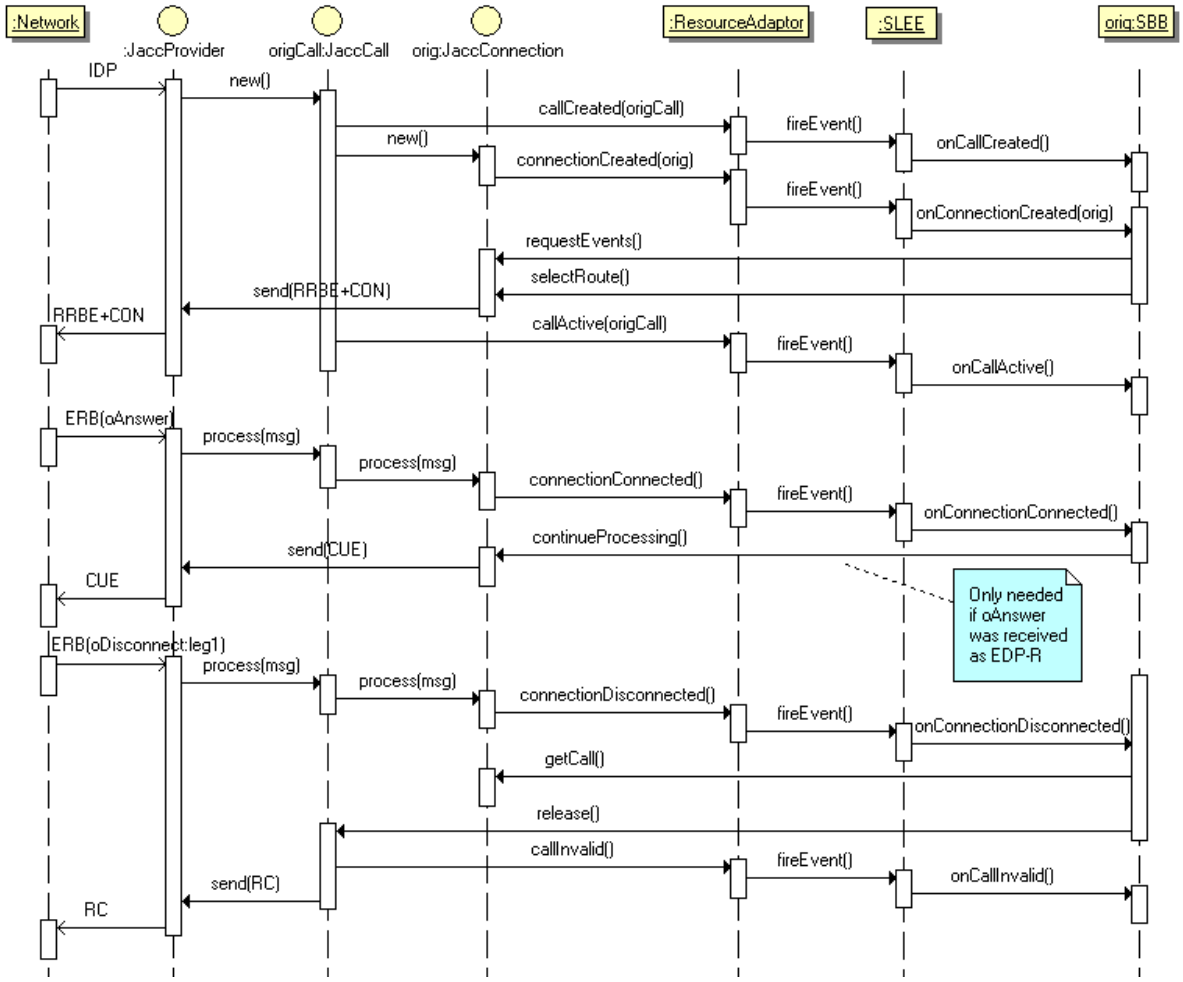


3.12. Pav. TerminalConnection būsenų automatas.

3.6.5 Tipinių scenarijų sekų diagramos

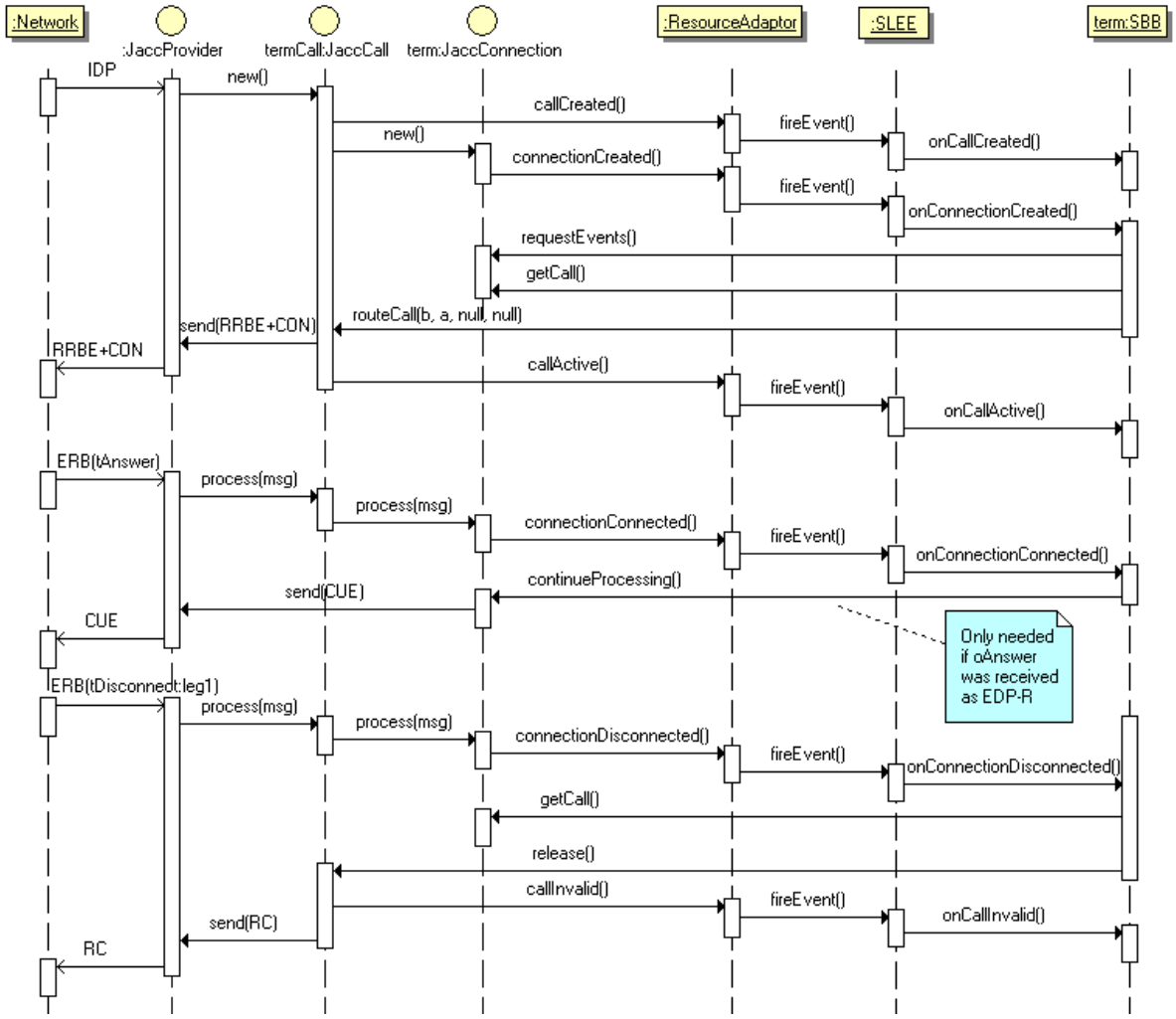
Diagramose vaizduojamos klasės „Network“, „SLEE“ ir „SBB“ nėra sistemos klasės. Tai abstraktūs telekomunikacinio tinklo, Mobicents SLEE konteinerio ir konteineryje veikiančios paslaugos atvaizdai siekiant atvaizduoti pilną pranešimo apdorojimo kelią nuo tinklo iki konkrečios telekomunikacinės paslaugos.

3.6.5.1 Išeinančio skambučio sujungimas



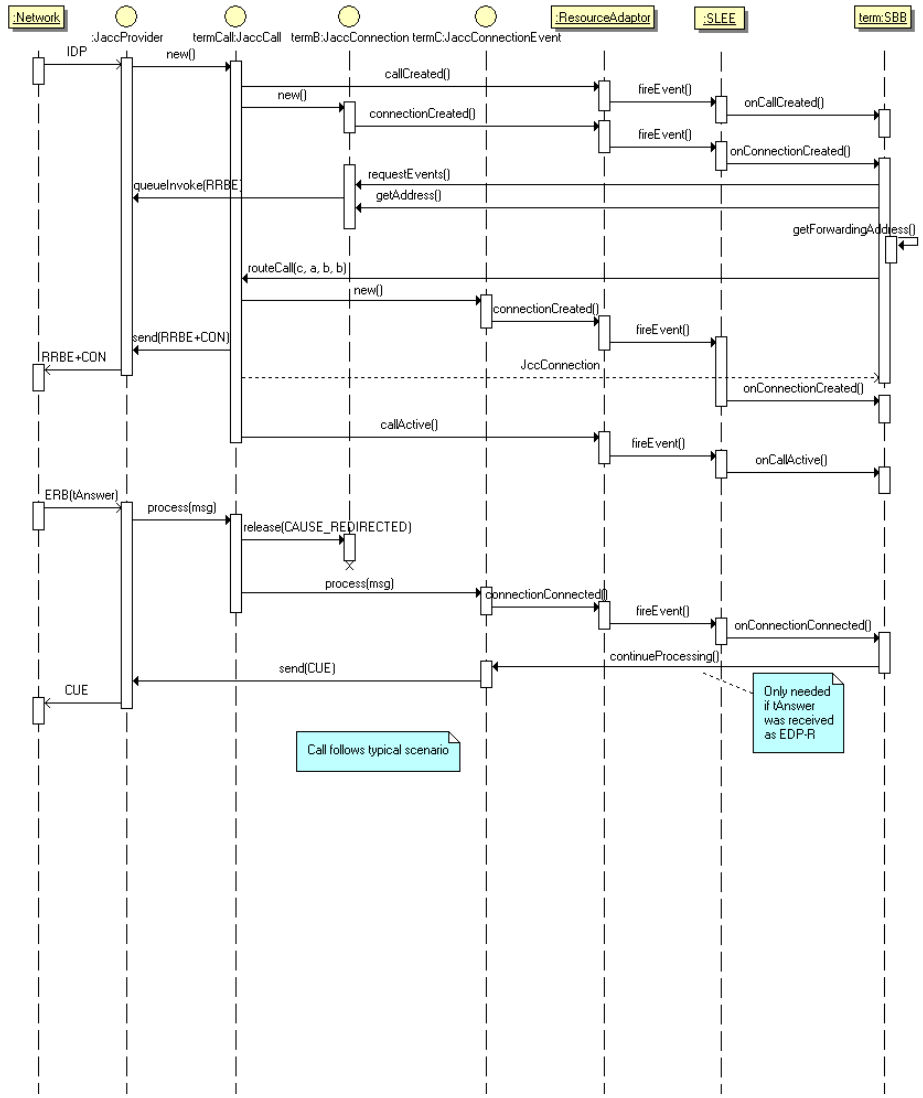
3.13. Pav.Išeinančio skambučio apdorojimas.

3.6.5.2 Įeinančio skambučio sujungimas



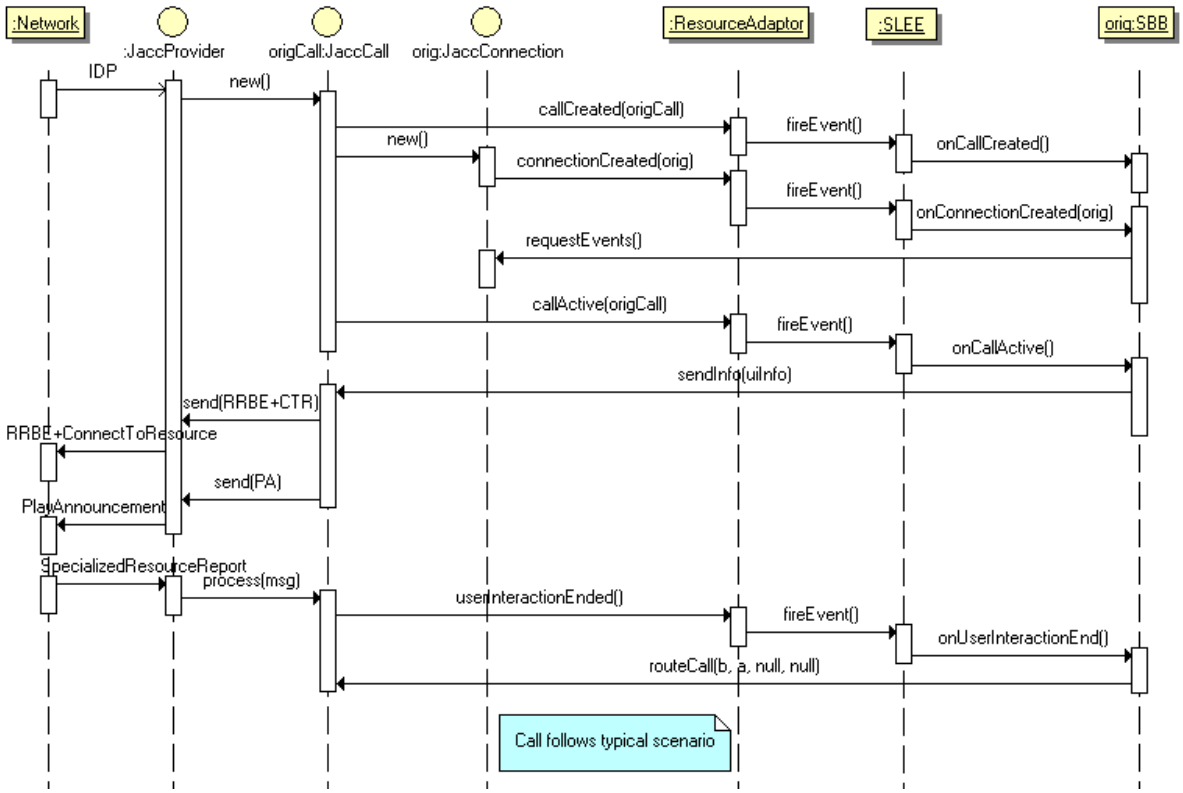
3.14. Pav.Įeinančio skambučio apdorojimas.

3.6.5.3 Įeinančio skambučio nukreipimas



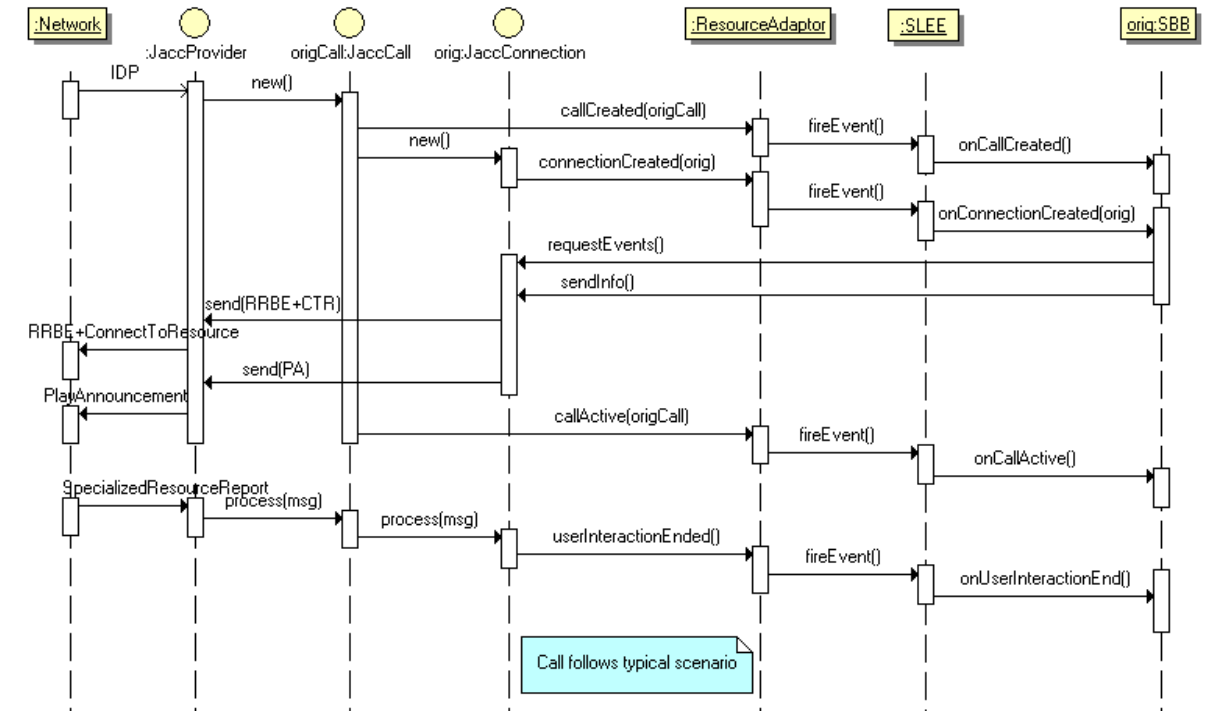
3.15. Pav. Įeinančio skambučio nukreipimo kreipinių seka.

3.6.5.4 Garsinio pranešimo grojimas (announcement) visiems pokalbio dalyviams



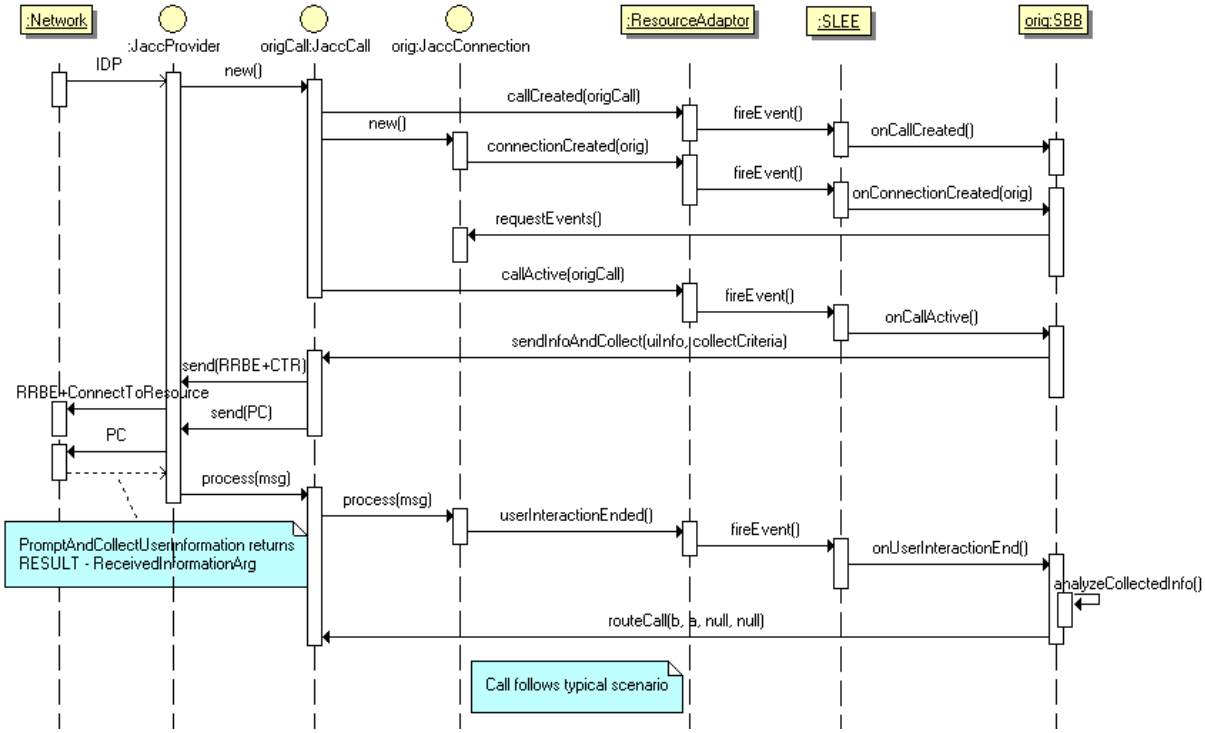
3.16. Pav. Pranešimas pokalbio dalyviams.

3.6.5.5 Garsinio pranešimo grojimas vienam pokalbio dalyviui



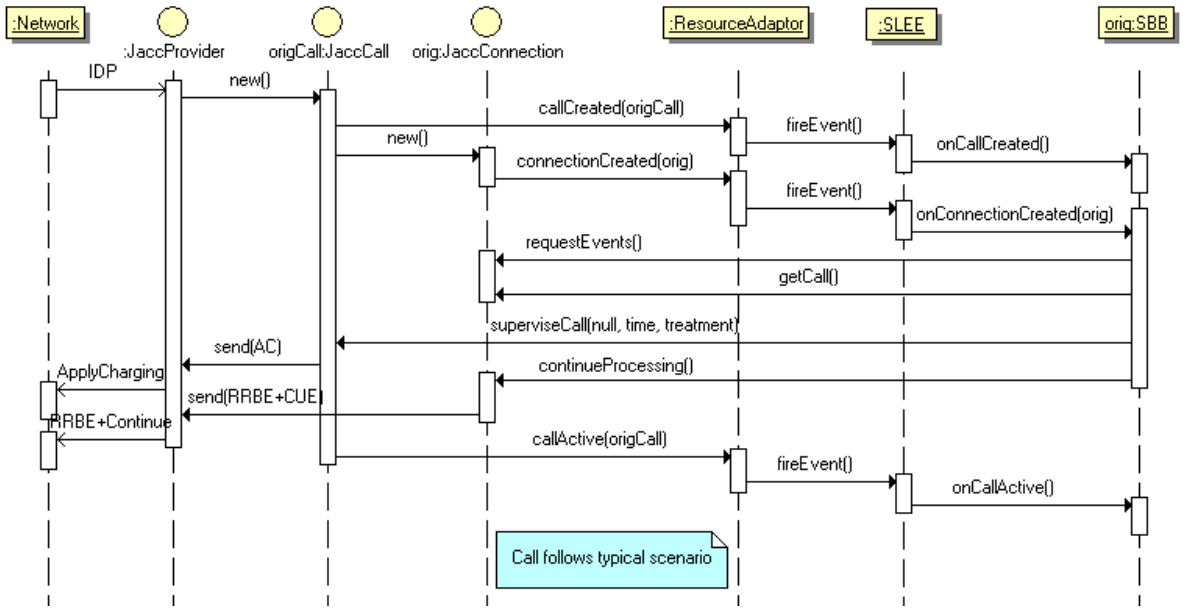
3.17. Pav. Pranešimas vienam pokalbio dalyviui.

3.6.5.6 Duomenų surinkimas iš vartotojo



3.18. Pav.Vartotojo duomenų surinkimas.

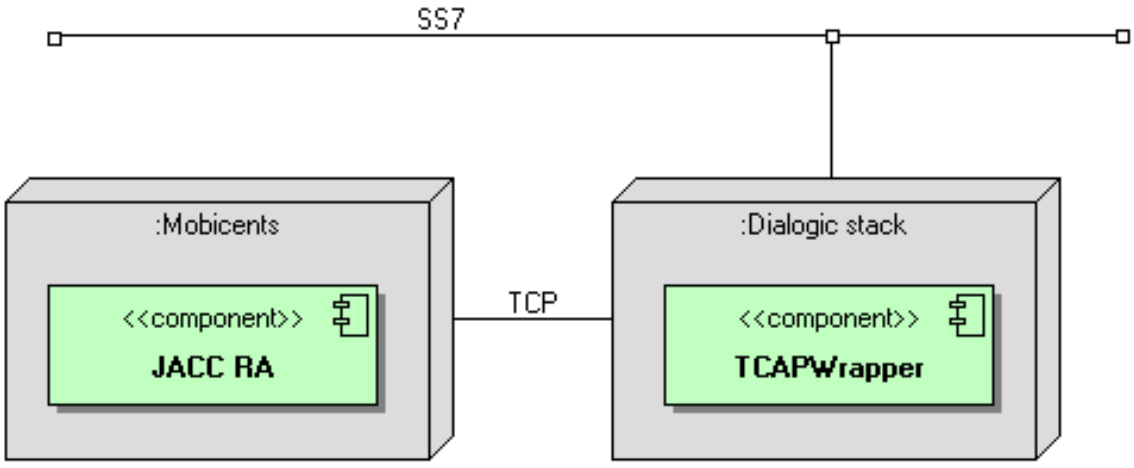
3.6.5.7 Apmokestinimas



3.19. Pav. Abonento apmokestinimas.

3.6.5.8 Išdėstymo vaizdas

Šiame darbe projektuojamas komponentas 3.20. Pav. Sprendimą sudarantys komponentai. pavaizduotas „JACC RA“. TCAPWrapper – komponentas diegiamas atskirame fiziniame mazge. Šis komponentas Dialogic protokolų steko pagalba jungiamas prie SS7 telekomunikacinio tinklo. JACC RA ir TCAPWrapper bendrauja TCP pagalba naudojant specialiai sukurtą protokolą apibrėžtą ASN.1, suprojektuotą efektyviam, didelės spartos pranešimų apsikeitimui.



3.20. Pav. Sprendimą sudarantys komponentai.

4 TIRIAMOJI IR EKSPERIMENTINĖ DALIS

Sistemos taikymo sritis nusako, kad vienas esminių nefunkcinių reikalavimų – sistemos našumas, t.y. sistemos gebėjimas apdoroti didelį skambučių srautą.

Sistemos realizavimai naudojamos technologijos taip pat turi esminę įtaką eksploatacinėms sistemos savybėms – našumui, veikimo stabilumui, prognozuojamumui.

Šio darbo tiriamojoje dalyje analizuojamas sistemos našumas, Java technologijos ir virtualių mašinų panaudojimo specifika.

4.1 Virtualių mašinų panaudojimo specifika

Virtualių mašinų vystymo aušroje jos laikytos tinkamomis tik tam tikrais atvejais, pavyzdžiui, ilgai veikiančioms serverių sistemoms su dideliais atminties resursais. Manyta, kad virtualios mašinos – neperspektyvi platforma klientinių aplikacijų kūrimui ar juo labiau realaus laiko sistemoms.

Šiame skyriuje apžvelgiamos virtualių mašinų ypatybės, dinaminio optimizavimo, šiukšlių surinkimo strategijų įtaka virtualioje mašinoje veikiančioms taikomosioms programoms.

Analizė remiasi Sun Java realizacija, tačiau ir kitos modernios virtualios mašinos turi analogiškas technologijas.

Šis skyrius remiasi [14], [15], [16] šaltiniais.

4.1.1 Dinaminis transliavimas (*Just-in-time compilation*)

4.1.1.1 Kas yra JIT?

Įprastai programos veikdavo vienu iš dviejų būdų: programos kodas interpretuojamas arba transliuojamas iš anksto. Pirmu atveju programos kodas nuolat transliuojamas iš aukšto lygio programavimo kalbos į mašininį kodą kiekvieno vykdymo metu. Antruoju atveju (statinis transliavimas) kodas transliuojamas tik vieną kartą – prieš vykdymą.

JIT kompiliatoriai leidžia alternatyvų, hibridinį spendimą. Transliavimas atliekamas nuolat, kaip interpretuojamo kodo atveju, tačiau sutransliuotas kodas kešuojamas siekiant didesnio našumo.

JIT remiasi dviem pagrindinėmis idėjomis: byte kodo kompiliavimas ir dinaminis kompiliavimas, t.y. kodas verčiamas į mašininį kodą vykdymo metu (pavyzdžiui, byte kodas į mašininį kodą).

JIT naudojamas ne vienoje modernioje VM, pavyzdžiui, daugelyje Java realizacijų, Microsoft .NET Framework.

4.1.1.2 JIT privalumai ir trūkumai

Tradicinės virtualios mašinos tiesiog interpretuoja kodą, bendru atveju daug lėčiau, nei sistemos naudojančios JIT. Kai kurie interpretatoriai netgi interpretuoja patį išeities tekstą, dėl to vykdymo laikas dar prastesnis.

Statiškai transliuojamas kodas verčiamas į mašininį prieš vykdymą. Dinaminio transliavimo esmė – kompiliatorius naudojamas vykdymo metu. Pavyzdžiui, daugelis Common Lisp sistemų turi galimybę kompiliuoti funkcijas sukurtas vykdymo metu. Tai suteikia tam tikrus JIT privalumus, tačiau kodo kompiliavimą valdo programuotojas, ne vykdymo aplinka. Tai gali būti panaudota ir dinamiškai sugeneruotam kodui kompiliuoti, kuris gali duoti rezultatus geresnius ne tik lyginant su statiškai kompiliuotu kodu, bet ir JIT.

Byte kodu paremtoje sistemoje programos išeities tekstas verčiamas į tarpinę formą, vadinamą byte kodu. Byte kodas nėra konkretaus kompiuterio mašininis kodas ir yra lengvai perkeliamas (*portable*) tarp skirtingų architektūrų. Byte kodas interpretuojamas virtualios mašinos. JIT kompiliatorius naudojamas byte kodo vykdymo paspartinimui: byte kodo vykdymo metu JIT kompiliatorius visą ar jo dalį verčia į mašininį kodą. JIT apdorojama kodo dalis gali būti failas, funkcija ar bet koks kitas kodo fragmentas. JIT pavadinimas kilo iš to, kad kodas kompiliuojamas prieš pat jo vykdymą ir kešuojamas pakartotiniam panaudojimui – taip išvengiama pakartotinio to paties kodo transliavimo.

JIT technologijų tikslas – pasiekti ar pralenkti statinės kompiliacijos našumą, išlaikant interpretuojamo byte kodo pranašumus. Išeities teksto analizė ir bazinės optimizacijos atliekamos kompiliavimo į byte kodą metu, o byte kodo vertimas į mašininį kodą daug spartesnis, tokius kompiliatorius paprasčiau parašyti. Be to, byte kodas yra nepriklausomas nuo architektūros, t.y. lengvai perkeliamas. Kadangi šį kompiliavimą kontroliuoja vykdymo aplinka, nepažeidžiamas sistemos saugumas.

JIT taikančios sistemos turi daug geresnę greitaveiką lyginant su tradiciniais interpretatoriais. Tam tikrais atvejais JIT pasiekia geresnį našumą, nei statinė kompiliacija, nes tam tikros optimizacijos galimos tik vykdymo metu:

1. Pritaikymas konkrečiam CPU ir OS. Pavyzdžiui, JIT gali pasirinkti SSE2 instrukcijas, jei CPU jas palaiko. Kompiliuojant statiškai reikėtų kompiliuoti iš naujo kiekvienai platformai/architektūrai arba viename vykdomajame faile įtraukti kelias tų pačių procedūrų versijas.

2. Virtuali mašina renka statistiką apie tai kaip iš tiesų vykdoma programa konkrečioje aplinkoje ir tuo remiantis pertvarkyti ir perkompiliuoti kodą optimaliam veikimui. Kai kurie statiniai kompiliatoriai taip pat gali remtis profiler'io duomenimis.

3. Sistema taip pat gali taikyti globalų optimizavimą, pavyzdžiui, funkcijų iš bibliotekų įtraukimas (*inlining*).

4. Nors tai įmanoma ir statiškai kompiliuojamoms kalboms su šiukšlių surinkimu, tačiau byte kodu paremta sistema gali lengviau perorganizuoti kodą veikimo metu ir geriau pasinaudoti kešavimu.

4.1.1.3 Pradinis uždelsimas

JIT paprastai sukelia tam tikrą uždelsimą pirmą kartą vykdant aplikaciją dėl laiko reikalingo pirminiam byte kodo užkrovimui ir sukompiliavimui. Šis reiškinys vadinamas „starto delsa“ (*startup delay*). Kuo daugiau optimizacijų pritaiko JIT, tuo geresnis kodas gaunamas, tačiau nuo to taip pat didėja pradinis uždelsimas. Dėl šių priežasčių JIT taikymas visada yra kompromisas tarp uždelsimo trukmės ir gaunamo kodo efektyvumo. Be to, pradinė delsa susijusi ir su įvesties operacijomis, pavyzdžiui, Java standartinės bibliotekos JAR failas ~ 40 MB – VM turi atlikti daug paieškų šiame faile.

Viena iš Sun HotSpot JVM taikomų optimizacijų – interpretavimo ir JIT kompiliacijos kombinavimas. Aplikacijos kodas pirmiausiai interpretuojamas, bet VM stebi kurie byte kodo fragmentai yra dažnai vykdomi ir verčia juos į mašininį kodą tiesioginiam vykdymui. Jei kodas vykdomas tik keletą kartų – išvengiama kompiliavimo (sutaupomas tam reikalingas laikas, mažėja starto delsa), dažnai vykdomas kodas sukompiliuojamas ir pasiekiamas kur kas geresnis efektyvumas. Taigi JIT leidžia greitą kodo vykdymą po pradinės lėtos interpretavimo fazės. Kadangi bet kuri programa didžiąją laiko dalį vykdomą tą patį kodą, kompiliavimui sugaištas laikas nėra reikšmingas, o per pradinę interpretavimo fazę surenkama naudinga statistinė informacija vėliau panaudojama taikant optimizacijas.

Teisingas JIT taikymas priklauso nuo taikymo atvejo. Pavyzdžiui, Sun JVM turi du pagrindinius veikimo režimus: *client* ir *server*. Režime *client* kompiliavimas ir optimizavimas taikomi minimaliai siekiant minimizuoti starto delsą. *server* režime kompiliavimas ir optimizavimas taikomi plačiai siekiant geriausio našumo ir aukojant starto laiką. Kiti JIT kompiliatoriai taiko kitokias euristikas:

- metodo vykdymų skaičius ir jo byte kodo dydis [14];
- metodo vykdymų skaičius ir aptikti ciklai [15].

Trumpai veikiančiose programose daug sunkiau teisingai identifikuoti kurias kodo dalis verta optimizuoti [16].

Microsoft *Native Image Generator (Ngen)* taiko kitokį metodą pradiniam uždelsimui sumažinti. *Ngen* iš anksto kompiliuoja byte kodą į mašininį. Taip išvengiama kompiliavimo programos veikimo metu. Pavyzdžiui, *.NET framework 2.0* pateikiamas su *Visual Studio 2005* atlieka išankstinį kompiliavimą instaliacijos metu. Tai pagerina startavimo laiką, tačiau gaunamo kodo kokybė nėra tokia, kaip naudojant JIT dėl tų pačių priežasčių dėl kurių statiškai sukompiliuotas kodas be profiler duomenų nusileidžia JIT.

Kai kurios Java realizacijos taip pat taiko išankstinį kompiliavimą kartu su JIT (*Excelsior JET*) ar interpretatoriumi (*GNU Compiler for Java*).

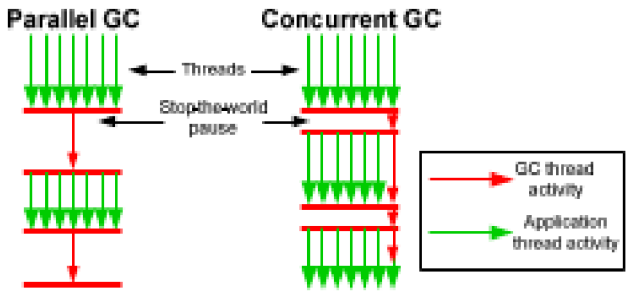
4.1.2 Šiukšlių surinkimo (*Garbage collection*) strategijos

Daugeliu atvejų automatinis nebenaudojamos atminties atlaisvinimas ir defragmentavimas yra naudingas. Tačiau tai taip pat gali sukelti problemų, kai nuo to priklausančiai sistemai keliami realaus laiko apribojimai. Tiesiogiai valdant atmintį lengva padaryti klaidų, kurių išvengti leidžia automatinis šiukšlių surinkimas (*Garbage Collection* – GC).

Paprastai programuotojas negali visiškai valdyti GC elgsenos – vienas iš šalutinių efektų yra tas, kad GC gali neprognozuojamai sustabdyti vykdomą programą. Siekiant išvengti šios problemos Java 6 VM turi kelis alternatyvius GC algoritmus, skirtus skirtingoms taikomųjų programų klasėms. Trumpai aptarsime numatytojo *Parallel* GC ir alternatyvaus *Concurrent* GC savybes.

Parallel GC yra numatytasis GC režimas ir jis yra tinkamiausias daugeliu atvejų, tačiau jį naudojant galimos neprognozuojamos programos vykdymo pauzės net iki kelių sekundžių esant didelei apkrovai. Tai itin nepageidautina realaus laiko sistemoms: tokiu atveju sustoja įeinančių pranešimų apdorojimas ir pan. GC baigus darbą sistemą turi apdorotų visus eilėje susikaupusius pranešimus, o tai pakenks sistemos laikiniams parametrams bei stabilumui.

Sistemoms, kurioms keliami laikiniai apribojimai rekomenduojama naudoti *Concurrent* GC. Tai vėlgi yra kompromisas: ši strategija siekia minimizuoti laiką reikalingą nebenaudojamos atminties atlaisvinimui tai atliekant tuo pat metu, kai veikia ir taikomosios programos.

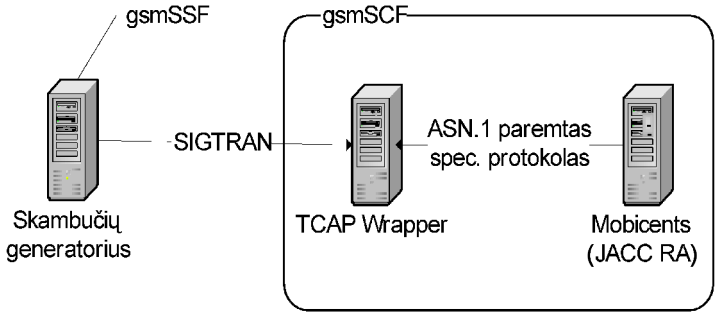


4.1. Pav. Parallel ir Concurrent GC palyginimas.

Concurrent GC atveju tik dvi trumpos pauzės paveikia taikomųjų programų gijas: surinkimo pradžioje ir pabaigoje (4.1. Pav. Parallel ir Concurrent GC palyginimas.). Šia strategija išvengiama neprognozuojamų užlaikymų, tačiau paaukojami procesoriaus resursai, nes jo ciklais nuolat naudojami GC. Dėl aptartų priežasčių tyrimo metu naudota *Concurrent GC* strategija.

4.2 Tyrimo aplinka

Sistemos našumo ir kitų eksploatacinių savybių tyrimui buvo paruošta aplinka pagal schemą 4.2. Pav. Tyrimo aplinkos schema.



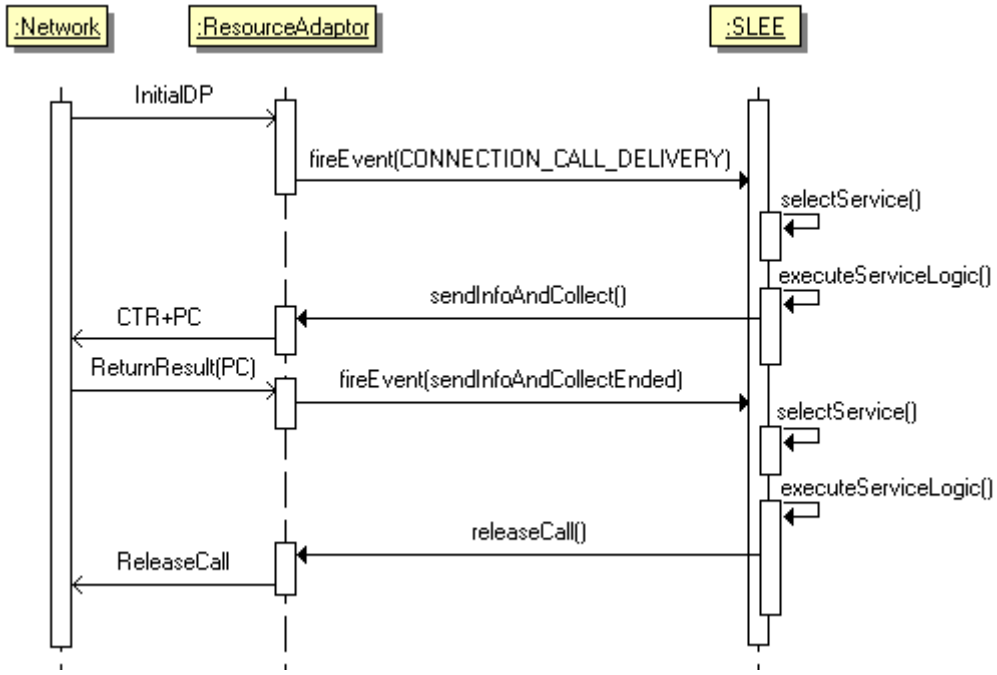
4.2. Pav. Tyrimo aplinkos schema

TCAP Wrapper ir Mobicents komponentai įdiegti vienoje fizinėje mašinoje siekiant eliminuoti galimus neapibrėžtumus dėl tinklo įtakos. Skambučių generatorius – atskira tarnybinė stotis, kurioje diegiama programinė įranga skambučių generavimui.

Remiantis IN architektūros principais šioje schemoje skambučių generatorius laikytinas gsmSSF mazgu, o TCAP Wrapper ir Mobicents komponentai kartu sudaro gsmSCF mazgą. Šie mazgai sujungti IP tinklu ir bendrauja naudodami SIGTRAN.

Techninė įranga panaudota šiam diegimui – HP ProLiant DL 360 G5 (4 branduoliai @2,4 GHz, 8 GB RAM).

Tyrimo aplinkoje taip pat būtina testuojamoji paslauga, t.y. SLEE servisas. Našumui ir skirtingoms veiksenos savybėms tirti realizuotas servisas, kurio sąveikos su tinklu (apsikeitimo pranešimais) seka vieno skambučio metu (angl. *call flow*) pavaizduota 4.3. Pav. Abonento paslaugų valdymo paslaugos pranešimų seka.



4.3. Pav. Abonento paslaugų valdymo paslaugos pranešimų seka

Pavaizduotos logikos taikymo pavyzdys galėtų būti abonento papildomų paslaugų, pavyzdžiui, balso pašto, valdymas interaktyviu balsiniu meniu (IVR).

1. Abonentas surenka trumpąjį paslaugos valdymo numerį. JACC RA gauna iš tinklo apie tai informuojantį pranešimą *InitialDP*.
2. JACC RA sukuria reikalingas skambučio modelio esybes ir sugeneruoja atitinkamus įvykius į SLEE, kurie perduodami šį skambutį aptarnausiančiam servisui.
3. Servisas inicijuoja pranešimo grojimą skambinančiajam (tinklo pranešimai *ConnectToResource (CTR)* ir *PromptAndCollectUserInformation (PC)*), pavyzdžiui, „Paspauskite 1, jei norite įjungti balso pašto paslaugą, kai esate užimtas, paspauskite 2, jei norite įjungti balso pašto paslaugą, kai Jūsų telefonas išjungtas arba už ryšio zonos ribų, paspauskite 3, jei norite įjungti balso pašto paslaugą abiem atvejais, paspauskite 4, jei norite išjungti balso pašto paslaugą.“
4. Abonentui pasirinkus vieną iš alternatyvų ir paspaudus atitinkamą telefono mygtuką JACC RA gauna iš tinklo *ReturnResult* pranešimą su šia informacija. Šis pranešimas

JACC RA konvertuojamas į SLEE įvykį ir perduodamas atitinkamam servisui. Servisas išsaugo nustatymus ir baigia skambutį pranešimu *ReleaseCall*.

Šios paslaugos aptarnaujamuose skambučiuose galima išskirti dvi fazes:

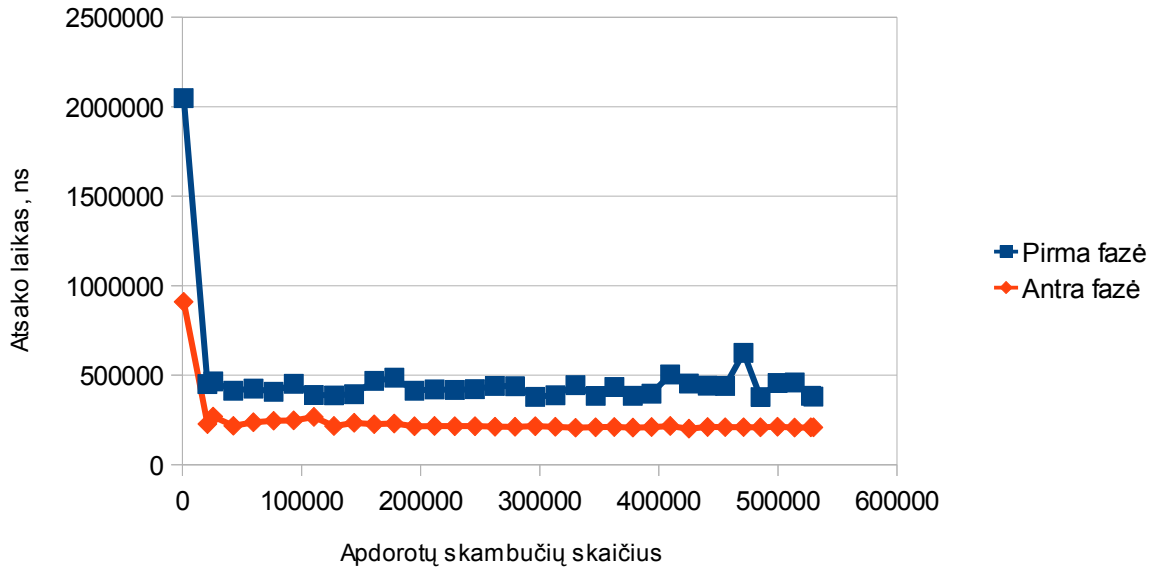
1. nuo pirmo pranešimo iš tinklo gavimo iki atsakymo;
2. nuo antro pranešimo iš tinklo gavimo iki skambučio baigimo.

4.3 Tyrimo rezultatai

4.3.1 Nuolatinės apkrovos eksperimentas

Šio eksperimento metu tiriama sistemos elgsena pradedant generuoti skambučius stabiliu dažniu nuo sistemos starto, t.y. prieš pradedant eksperimentą sistema dar nėra apdorojusi nė vieno skambučio.

Tyrimas parodė aiškią pradinės delsos mažėjimo tendenciją dėl anksčiau aptartos dinaminės optimizacijos įtakos.



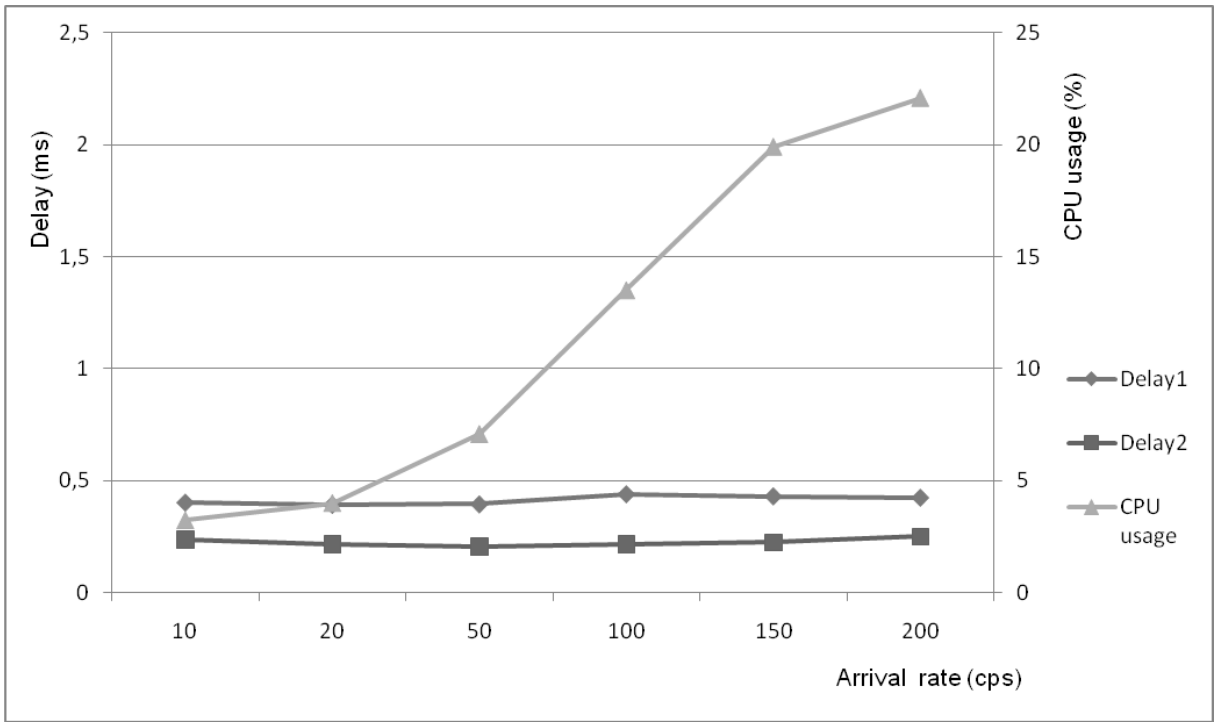
4.4. Pav. Atsako laikai generuojant 30 skambučių per sekundę

Iš grafiko pavaizduoto 4.4. Pav. Atsako laikai generuojant 30 skambučių per sekundę matoma, kad pirmieji skambučiai po platformos starto apdorojami kelis kartus lėčiau, nei pasiekus optimalią būseną. Atsako laikai stabilizuojasi maždaug po 20000 apdorotų skambučių. Įvertinant skambučių generavimo dažnį paaiškėja, kad dinaminis optimizavimas pasiekia stabilią būseną maždaug po 10 min. nenutrūkstamo darbo.

Be to, viso eksperimento metu išlieka ženklus skirtumas tarp atsako laikų į pirmą ir į antrą skambučio pranešimą. Pirmo pranešimo apdorojimo metu atliekama visų reikalingų esybių inicializacija, dėl skambučių modelio savybių generuojami net trys įvykiai. Antrojo įvykio apdorojimo metu logika kur kas paprastesnė, dėl to atsako laikas šiuo atveju trumpesnis.

4.4 Didėjančios apkrovos eksperimentas

Šio eksperimento išankstinė sąlyga – sistema turi būti apdorojusi pakankamai skambučių, kad būtų pasiekta optimali veiksena, t.y. eliminuota dinaminio optimizavimo įtaka atsako laikui. Įvykdžius pradinę generavimo sesija, pradedamas eksperimentas, kur nuolat didinamas generuojamų skambučių per sekundę skaičius ir stebimi atsako laikai bei procesoriaus apkrautumas.



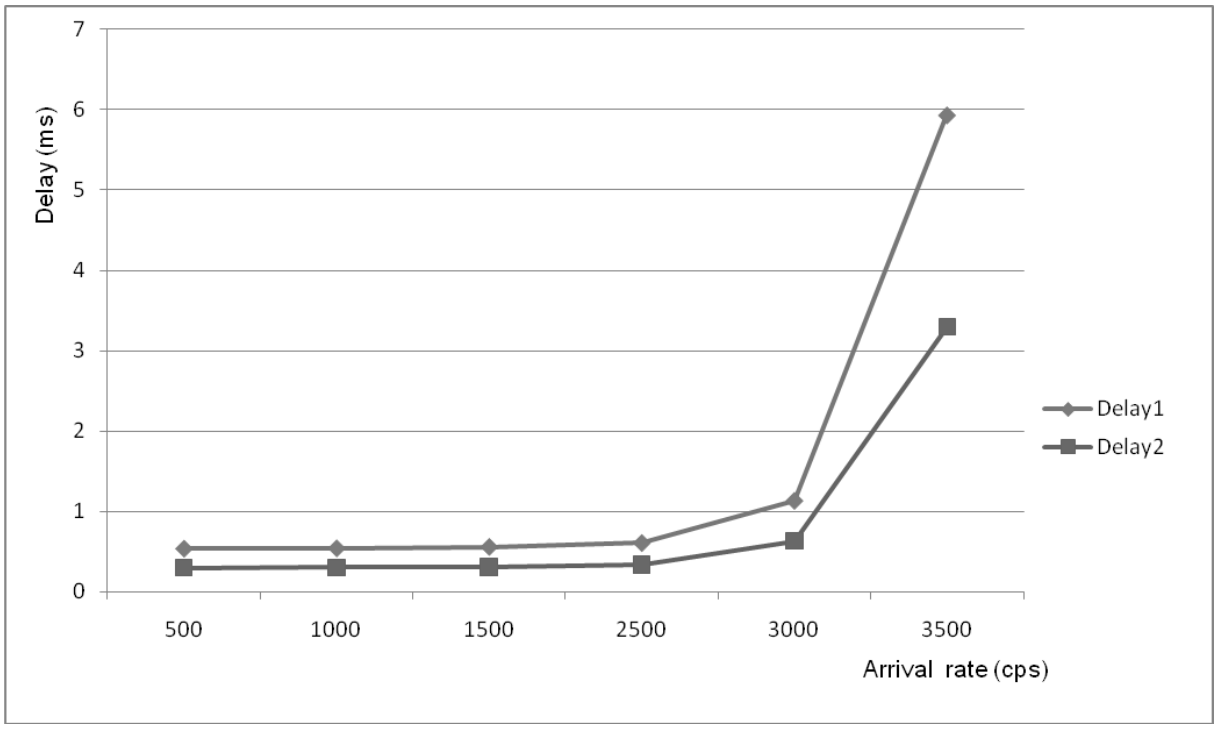
4.5. Pav. Atsako laikai didinant apkrovą

Grafike pavaizduotame 4.5. Pav. Atsako laikai didinant apkrovą matoma, kad sistemai pasiekus stabilią būseną atsako laikai abiejose skambučio fazėse (grafike šie atsako laikai pažymėti „Delay 1“ ir „Delay 2“) nekinta apkrovą didinant iki 200 skambučių per sekundę. Procesoriaus apkrautumas pastebimai augti pradeda maždaug nuo 30 skambučių per sekundę apkrovos ir ties maksimalia 200 skambučių per sekundę apkrova pasiekia 22% apkrautumą.

Rezultatai rodo, kad sistemos pajėgumų visiškai pakanka tokiai apkrovai apdoroti. Dėl testinės aplinkos apribojimų praktiniais eksperimentais ieškoti sistemos galimybių ribos neįmanoma: testavimui skirta SS7 protokolų steko licencija turi ribotą pralaidumą, kuris

reiškia tam tikrą skambučių per sekundę ribą, priklausomai nuo generuojamų skambučių scenarijaus. Naudojant šiam tyrimui parinktą skambučio scenarijų sistema geba generuoti daugiausiai 200 skambučių per sekundę.

Dėl aptartų ribojimų teorinėms sistemos galimybių riboms tirti sudarytas imitacinis modelis [17], paremtas atkarpomis tiesinių agregatų formalizmu [18] diskrečių įvykių simuliavimui [19].



4.6. Pav. Itin didelės apkrovos simuliavimas

4.6. Pav. Itin didelės apkrovos simuliavimas pavaizduoti simuliacijos metu gauti duomenys. Rezultatai rodo, kad sistema stabiliai apdoroja apkrovą be atsako laiko pasikeitimų iki 2500 skambučių per sekundę. Toliau didinant apkrovą atsako laikai itin sparčiai auga. Tai rodo, kad pasiekta sistemos pralaidumo riba ir sistema nebepajėgia apdoroti generuojamo srauto, užsipildo vidinės įvykių eilės.

5 IŠVADOS

Šiame darbe išanalizuota IN protokolų integracijos su Mobicents SLEE platforma problema. Suprojektuotas ir realizuotas sprendimas.

Kadangi Mobicents – JSLEE realizacija, sprendime taip pat pagrindinė technologija – Java. Darbe ištirtos Java ir kitų virtualių mašinų panaudojimo taikymo atvejuose su laikiniais apribojimais ypatybės. Realizuotam sprendimui parinkta optimali atminties (*garbage collection*) valdymo strategija.

Sistemos našumas ir kitos veiksenos savybės tirtos eksperimentiškai naudojant specialiai realizuotą tipinę IN paslaugą ir naudojant imitacinį sistemos modelį.

Šis tyrimas parodė, kad sistema veikia efektyviai ir našiai netgi netaikant jokios plečiamumui pritaikytos diegimo schemos. Eksperimentais praktiškai išbandytas skambučių apdorojimas iki 200 skambučių per sekundę, tokiu režimu su naudota įranga procesoriaus apkrautumas neviršyja 25%. Imitacinio modelio pagalba parodyta, kad teorinė sistemos pralaidumo riba su pasirinkta tyrimui aparatūrine įranga yra apie 3500 skambučių per sekundę.

Eksperimentai ir simuliacija parodė, kad šis Mobicents paremtas sprendimas yra visiškai tinkamas panaudojimui kaip skambučių apdorojimo platforma didelio intensyvumo telekomunikaciniuose tinkluose.

Našumo tyrimas parodė aiškią pradinės delsos įtaką ir dinaminio optimizavimo įtaką sistemos darbo pradžioje. Šios įgytos žinos apie sistemos dinaminę elgseną gali būti panaudotos formuluojant eksploatacines rekomendacijas:

- prieš paleidžiant sistemą (prijungiant prie skambučių srauto) būtina atlikti bandomųjų skambučių sesiją (*warm-up round*) arba realių riboto srauto skambučių;
- jei įmanoma, rekomenduojama apkrovą didinti palaipsniui.

Įgyvendinta programinė sąsaja paremta standartais ir patobulinta papildymais, kurie būtina realių paslaugų įgyvendinimui. Pasinaudojus standartais projektuotais taikymams su Java technologija rezultate gauta programinė sąsaja, kuri yra patogi naudoti, nesunkiai suprantama ir įsisavinama. Visos operacijos realizuojamos kaip atitinkamų skambučių modelio esybių metodai – toks programavimo modelis modernesnis ir patogesnis prie jo pripratusiems žmonėms lyginant su alternatyvomis, pavyzdžiui, Parlay.

Pasiūlytos sprendimo architektūros lankstumas leidžia lengvą plečiamumą horizontaliai – sistemos našumas gali būti didinamas tiesiog pridedant papildomus mazgus.

Suprojektuotas ir realizuotas sprendimas tenkina jam suformuluotus reikalavimus tiek funkciniu, tiek nefunkciniu (našumas, patogumas) aspektais. Naudojant šį sprendimą Mobicents platformą taptų tinkama panaudojimui kaip IN SDP.

6 LITERATŪRA

[1] JAINSLEE.ORG, JAIN SLEE ir susijusios technologijos, http://www.jainslee.org/othertechnologies/other.html [2011-05-24]

[2] jNetX Konverguojančių paslaugų platforma, http://www.jnetx.com/ [2011-05-24]

[3] Tarptautinis inžinerijos konsorciumas: Sumanus tinklas (Intelligent Network), http://www.iec.org/online/tutorials/in/index.asp [2011-01-23]

[4] The Moriana Group "Service Delivery Platforms in the Web 2.0 era" http://www.morianagroup.com/index.php?option=com_docman&task=cat_view&gid=155&Itemid=86 [2011-05-24]

[5] OpenCloud Next Generation IN Solutions http://www.opencloud.com/ [2011-05-24]

[6] SS7 Network Architecture Tutorial https://staff.ti.bfh.ch/mdm1/.../SS7/Architecture%20SS7.pdf [2010-01-24]

[7] Tarptautinis inžinerijos konsorciumas: SS7 (Signalizacijos sistema nr. 7) http://www.iec.org/online/tutorials/ss7/index.asp [2011-01-24]

[8] Hiroshi Sunaga, Yoji Yamato, Hiroyuki Ohnishi, Masashi Kaneko, Masami Iio, and Miki Hirano, Service Delivery Platform Architecture for the Next-Generation Network, https://www.icin.biz/files/2008papers/Session9A-2.pdf [2011-05-24]

[9] Mobicents: The Open Source SLEE and SIP Server, http://www.mobicents.org/ [2011-05-24]

[10] JSR 240: JAIN™ SLEE (JSLEE) v1.1, http://jcp.org/en/jsr/summary?id=240 [2011-05-24]

[11] JSR 21: JAIN™ JCC Specification, http://jcp.org/en/jsr/detail?id=21 [2011-05-24]

[12] JSR 122: JAIN™ JCAT, http://jcp.org/en/jsr/detail?id=122 [2011-05-24]

[13] JSLEE and the JAIN Initiative, http://java.sun.com/products/jain/ [2011-05-24]

[14] Schilling, Jonathan L. (February 2003). "The simplest heuristics may be the best in Java JIT compilers"

[15] Toshio Suganuma, Toshiaki Yasue, Motohiro Kawahito, Hideaki Komatsu, Toshio Nakatani, "A dynamic optimization framework for a Java just-in-time compiler", Proceedings of the 16th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications (OOPSLA '01), pp. 180–195, October 14–18, 2001.

[16] Matthew Arnold, Michael Hind, Barbara G. Ryder, "An Empirical Study of Selective Optimization", Proceedings of the 13th International Workshop on Languages and Compilers for Parallel Computing-Revised Papers, pp. 49–67, August 10–12, 2000.

[17] Vytautas Pilkauskas, Vilius Panevėžys, „Analysis and Simulation of a JAIN SLEE-based Platform for Intelligent Network Services". Proceedings of the 17th International Conference on Information and Software Technologies, IT 2011, Kaunas, 2011.

[18] Pranevicius H. Aggregate approach for specification, validation, simulation and implementation of computer network protocols. Lecture Notes in Computer Science No 502, Springer. 1992.

[19] Giambiasi N., Escude B., Ghosh S. GDEVS: a generalized discrete event specification for accurate modeling of dynamic systems. Proceedings. 5th International Symposium on Autonomous Decentralized Systems. 2001.

7 TERMINŲ IR SANTRUMPŲ ŽODYNAS

IN (*Intelligent Network*) – „Sumanusis tinklas“ - telekomunikacinio tinklo architektūra leidžianti atskirti komutavimo funkcijas nuo paslaugų logikos.

SSP (Service Switching Point) – IN architektūros mazgas, atliekantis komutavimo funkciją.

SCP (Service Control Point) – IN architektūros mazgas, kuriame veikia paslaugų logika.

SDP (Service Delivery Platform) – paslaugų teikimo platforma

JSLEE (Java Service Logic Execution Environment) – specifikacija, apibrėžianti Java SLEE realizacijos programines sąsajas ir vidinę veikseną.

RA (Resouce Adaptor) - JSLEE komponentas skirtas išorinių resursų integravimui (duomenų bazės, protokolų stekai ir pan.)

SS7 (Signalling System #7) - telefonijos signalizacinių protokolų grupė. Taip pat žinoma, kaip C7.

MSC (Mobile Switching Center) – telefonijos komutacinė stotis mobiliojo ryšio tinkle.

SIGTRAN (*signalling transport*) – IETF apibrėžta grupė protokolų leidžianti patikimą paketų perdavimą, skirtą SS7 signalizacijai per IP tinklus.

IVR (Interactive Voice Response) – interaktyvus meniu, kur balsu pranešami galimi pasirinkimai ir pasirinkimas pateikiamas paspaudžiant atitinkamą mygtuką.

CPS (Calls Per Second) – skambučiai per sekundę. Vartojama kaip vienetas skambučių srautui įvertinti.

ASN.1 – Abstract Syntax Notation One. Standartizuota ir lanksti notacija, skirta duomenų struktūroms apibrėžti kodavimui ir perdavimui kompiuterių bei telekomunikaciniais tinklais.

8 PRIEDAI

Prieduose išlaikoma vidinė numeracija.

8.1 Priedas Nr. 1

Straipsnis paskelbtas tarptautinėje konferencijoje IT 2011 (anglų kalba).

8.2 Priedas Nr. 2

Įdiegimo aktas.

ANALYSIS AND SIMULATION OF A JAIN SLEE-BASED PLATFORM FOR INTELLIGENT NETWORK SERVICES

Vytautas Pilkauskas[1], Vilius Panevezys[2]

[1]Kaunas University of Technology, Department of Business Informatics, Studentu 50, Kaunas, Lithuania, vytpilk@ktu.lt

[2]UAB ELITNET, Pašilės 102, Kaunas, Lithuania, vilius@elitnet.lt

Abstract. Intelligent Network is a telephone network architecture in which the service logic for a call is located separately from the switching facilities, allowing services to be added or changed without having to redesign switching equipment. The Java Community, within the Java APIs for Integrated Networks activities, offers a set of standard frameworks and open protocol APIs for the creation of telecommunications services. Currently the only open source implementation of JSLEE specifications is Mobicents. This paper proposes a solution for protocol-agnostic service creation in Mobicents platform. In our proposed solution, services depend on an abstract API defined by specifications from the JAIN group and implemented by a custom JSLEE Resource Adaptor. Discrete-event simulation model was used, in order to evaluate the performance of the proposed solution.

Keywords: Intelligent network, JSLEE, Mobicents, simulation model.

Introduction

Basic call setup functions are not enough for an operator to survive in modern competitive markets. Telecoms strive to get revenue by providing value-added services to their customers. One of value-added service enablers is Intelligent Network (IN) [1]. Intelligent Network is a telephone network architecture originated by Bell Communications Research in which the service logic for a call is located separately from the switching facilities, allowing services to be added or changed without having to redesign switching equipment. IN eases deployment, management and significantly reduces "time to market" of innovative services. One of key IN services is prepaid, requiring online charging. Other popular IN services include Friends and Family Promo, Corporate User Group, Single IMSI multiple MSISDN and others.

The Java Community, within the Java APIs for Integrated Networks (JAIN) activities, offers a set of standard frameworks and open protocol APIs for the creation of telecommunications services. In particular, the JAIN SLEE (JSLEE) provides a set of specifications [2] for the implementation of a Service Logic Execution Environment (SLEE) container. A SLEE is a hosting environment for telecom applications, also known as application server (AS), specifically designed to support asynchronous services with real-time constraints. Being composed of several layers of abstraction, a SLEE AS simplifies service design and implementation by providing the non-functional features needed for service execution. The resulting benefit is that developers can focus on aspects related to the implementation of the service logic, since the underlying complexity is masked by high-level APIs [3]. This kind of AS is a major candidate for deploying telecom application services [4].

Currently the only open source implementation of JSLEE specifications is Mobicents [5], which relies on the JBoss AS hosting environment. JBoss offers capabilities for service and SLEE management through Java Management Beans (MBean), service deployment, and thread pooling [6].

Telecom services are intrinsically asynchronous and typically require short latency, high throughput and high availability. Their integration is not simple as they may rely on several network protocols and interoperate with different software and hardware platforms. [7] presents the results of an experimental campaign in which several SIP-based VoIP services were tested, in order to evaluate the performance and scalability of the Mobicents SLEE platform. This paper presents a solution for IN protocol integration with Mobicents JSLEE and a simulation model of the system.

The paper is organized as follows. In section 2, we give some background information. In section 3, we illustrate in detail our proposal. In section 4, we describe an example of IN service implementation of the proposed solution, and section 5, outlines the simulation model used to evaluate the platform performance, and the relevant results. Finally, section 6 reports our conclusions.

Background

JAIN SLEE specifications

The JSLEE is the Java standard specification for the creation of a SLEE application environment, as reported in JSR 22 (v.1.0) and JSR 240 (v.1.1) [1]. The specification includes a component model for structuring

the application logic of communications applications. It views such applications as a collection of reusable object-orientated components, which are composed into higher level and more sophisticated services. Well behaved applications may be deployed on any application environment that implements the JSLEE specification. Figure 1 presents a simplified view of the JSLEE Architecture.

JSLEE is an event driven component-based container technology, supporting high performance, asynchronous application services. The service logic is implemented in system components called SBB, managed as a pool. SBBs operate asynchronously by receiving, processing, and firing events. They can be attached to event streams called Activity Contexts (AC), by which they receive notifications from other entities. When SBBs are no longer used, they are passivated, that is they are put within an instance pool waiting to be re-used.

External network events are translated into internal Java events by JSLEE modules called Resource Adaptors (RAs), which constitute an abstract interface layer that allows JSLEE AS to access external resources. RAs, as its name suggests adapts particular properties and behavior of an external Resource into normalized, portable Java interfaces and events visible to application code inside the JSLEE container. Events are internally routed by a system component called Event Router.

In addition to the application component model, JSLEE also defines management interfaces and a set of built-in Facilities.

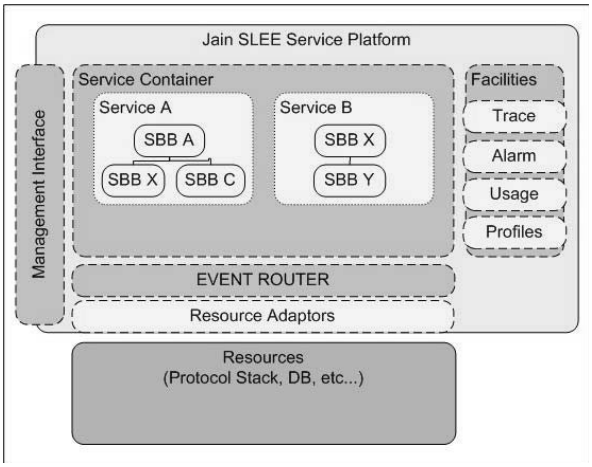


Figure 1. Simplified JSLEE Architecture

Open source implementation of JSLEE

Mobicents is an open-source communication platform that includes a SLEE, a Media Server, a Presence Server and a SIP Servlet AS. Below the Mobicents SLEE will be referred to as MSLEE. At the time of this writing the most recent version of Mobicents was 2.2.1.FINAL based on JBoss 5.1.0.GA. JBoss offers capabilities for service and SLEE management through Java Management Beans (MBean), service deployment, and thread pooling. It represents a container for higher level service containers. MSLEE does not rely on the CMPs offered by the JBoss container.

Comparison with commercial JSLEE solutions

There are other tools and architectures used to simplify JSLEE service development, but they are bundled within commercial products. JNetX JSLEE [7] implementation Open Convergent Feature Server (OCFS) comes with an IDE tool called Telecom Service Studio that claims to ease the whole development process. OCFS includes implementations of the IN protocols. Service designer may use the implementations directly to build the service using the protocol API directly or alternatively an intermediate protocol abstraction layer may be used. The abstraction layer is a protocol independent API called Multi-Party Call Control (MPCC). It should be possible to reuse the same service built using MPCC with different supported protocols. MPCC is based on Parlay call control API [8].

Instead, the OpenCloud Rhino [7] platform is very well documented. It consists of an XDocklet module and the FSM tool. The first one is used to write the deployment descriptor code directly inside Java class files, utilizing special tags without switching between different XML files (but the developer must yet write this type of data). The FSM tool is used for the creation of a finite state machine inside the service. It uses the VFMS description language to describe the state machine behavior and allows three different type of actions: entry, exit and input, executed when entering a state, leaving a state and corresponding to an event occurrence, respectively; obviously transitions can be defined as well. The tool parses this description and generates an SBB Java class

that defines the abstract behavior of the component. The developer must extend this class, implementing the actions with Java code and mapping actions with RAs callback methods. This type of approach, which delegates event handling to other methods and classes, is very similar to our SBB starting point/event router solution, but we think that our solution is more powerful. Opencloud's VFSM language is relatively simple and with few pattern options, unable to create complex state machine diagrams. Instead, we integrate the JSLEE AS with a whole WFMS capable of advanced control-flow patterns (e.g. split, merge, synchronization) and features. In the OpenCloud solution, the communication channels between different components still relies on JSLEE features (SbbLocalObject interface or ACI), whereas we can overtake this issue through the usage of transient variables which are more easy and flexible to use. Finally, using only one SBB, we do not have to care of multiple deployment descriptor XML files. Rhino has support for IN protocols as well, though, does not provide any abstraction to ease service development and make the service logic portable to different protocols.

Proposed solution

Despite the ever growing interest in VoIP services telecom operators still heavily rely on existing IN infrastructure, which has required considerable investments in the past so it is very unlikely to be withdrawn heedlessly. This historical background suggests that MSLEE would only be attractive to telecom operators with reliable and convenient IN integration.

This problem was approached with these key qualities in mind: reliability, high-availability, scalability, deployment flexibility.

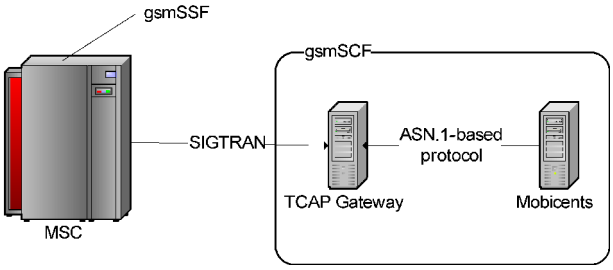


Figure 2. Mobicents IN integration solution scheme

Figure 2 depicts the basic scheme of the solution and how it aligns with IN architecture and SS7. IN Service Control Function is constituted of two components TCAP Gateway and MSLEE.

TCAP Gateway resides in a node equipped with a Dialogic® SIGTRAN-based SS7 stack (M3UA, SCCP-CL, TCAP). The component directly integrates with the protocol stack, takes care of TCAP dialogue management and acts as a gateway for higher-level protocol payloads. Flexible architecture of the solution allows this component to either be located on a separate node or co-located with MSLEE. This allows various deployment schemes and easy horizontal scalability for either of the components.

The second component is the MSLEE enhanced with a custom Resource Adaptor for Java Advanced Call Control that communicates with TCAP Gateway and implements logic of IN call control protocols such as CAP, INAP. Java Advanced Call Control is an API that builds upon existing specifications for Java call control [9,10] from the JAIN group and introduces user interaction and other advanced call control functions.

TCAP Gateway directly communicates with Mobile Switching Center, a Service Switching Function in IN parlance, which is responsible for routing voice calls, SMS and other services. The other link maintained by TCAP Gateway is used to exchange messages of IN call control protocols with the JACC RA deployed in MSLEE. The link uses a specially designed efficient ASN.1-based protocol, TCP is used for transport.

The JACC RA integrates IN protocols to MSLEE and presents an easy to use, protocol-independent, standardized API for service logic composition.

Analysis of performance and scalability of the solution

In order to assess the call processing performance of the solution and calibrate the simulation model, timing probes have been injected into JACC RA to measure time taken for particular segments of a call processing scenario.

For this test a real world use case has been chosen based on the flow depicted in Figure 3. This is an example of a trivial subscriber service management service, for example an IVR-like service to configure voice mail service:

1. Subscriber dials a short code dedicated for voice mail management. JACC RA gets notified about the initiated call by *InitialDP* message from network.

2. JACC RA fires appropriate event to SLEE, which selects a service to handle this event and the event gets delivered to the service for voice mail configuration.
3. Service starts playing and announcement for the subscriber, e.g. “Press 1 to enable voice mail on busy, press 2 to enable voice mail when your phone is turned off or out of the radio coverage, press 3 to enable voice mail in both cases or press 4 to disable voice mail service.” The appropriate network node (gsmSRF) is instructed to do this by sending out messages *ConnectToResource* and *PromptAndCollectUserInformation*.
4. After the user has chosen one of the alternatives and pressed the corresponding digit, JACC RA receives *ReturnResult* message from network with the collected information. SLEE selects the service that is handling this call and delivers the second event. Service would then persist the new configuration defined by the subscriber to a data store and release the call (possibly after playing back another announcement to the subscriber to assure the new setting has been saved). Interactions with the data store are omitted for simplicity.

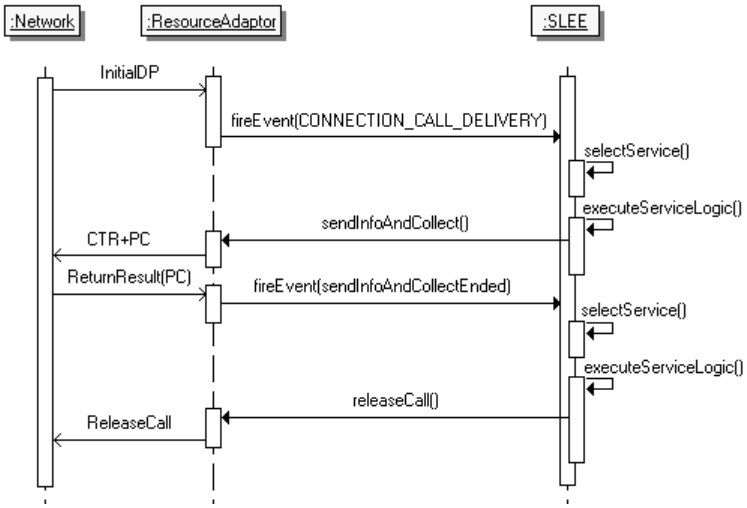


Figure 3. Call flow of an interactive service

Segments of this scenario have been timed to get experimental data for calibration of the simulation model.

	Segment		Time taken (ms)
	Start	End	
1	Initial event received from network	Initial event fired to SLEE	0.278
2	Initial event fired to SLEE	Service logic execution started	0.309
3	Start of service logic execution	Start of user interaction	0.002
4	Start of user interaction	End of user interaction	0.247
5	End of user interaction	End of service logic execution	0.001
6	Result event received from network	Event fired to SLEE	0.034
7	Event fired to SLEE	Service logic execution started	0.069
8	Start of service logic execution	Start of release call operation	0.001
9	Start of release call operation	End of release call operation	0.184
10	End of release call operation	End of service logic execution	0.001

Table 1. Timed segments of call processing

Testing methodology used was the following:

1. Start SLEE with the deployed service and run a “warm-up” round of 50000 calls to avoid cold start effects of the JVM like JIT and other dynamic optimization techniques interfering with the test results.
2. Run 5 calls at 1 CPS rate.
3. Average results of the last 5 calls.

The tests have been performed using HP ProLiant DL360 G5 server.

Performance evaluation using simulation model

Discrete-event simulation modeling has become the most commonly used tool for performance evaluation of stochastic dynamic systems in science and engineering, including such complex systems as computer and communication systems [11]. Simulation modeling can be used both as an analysis tool for predicting the effect of changes to existing systems and as a design tool to predict the performance metrics such as average queue length and utilization of resources of new systems under varying sets of circumstances.

Simulation Model Description

In this paper we use a piece-linear aggregate (PLA) [12] formalism for creation of discrete event models [13]. In the application of the PLA approach for system specification, the system is represented as a set of interacting piece-linear aggregates. The PLA is taken as an object defined by a set of states Z, input signals X, and output signals Y. A behavior of an aggregate is considered in a set of time moments $t \in T$. State $z \in Z$, input signals $x \in X$, and output signals $y \in Y$ are considered to be time functions. An execution of PLA corresponds to a sequence of internal and external events. Transition and output operator H must be known as well. PLA formalism has been used for creating object-oriented library for development of simulation models [14].

During research the formal (mathematical) model of MSLEE system realizing IN services was created. In the PLA system model five types of aggregates were developed (Figure 4):

- ***Network*** **–** for message stream generation of IN services in the model;
- ***ResourceAdaptor*** – for simulation of resource's adapters which connects MSLEE container with the external devices;
- ***SLEE*** **–** for simulation IN service processing;
- ***ProcUnit*** **–** for simulation of functioning maintenance resources adapters and MSLEE container's program components in server process unit.

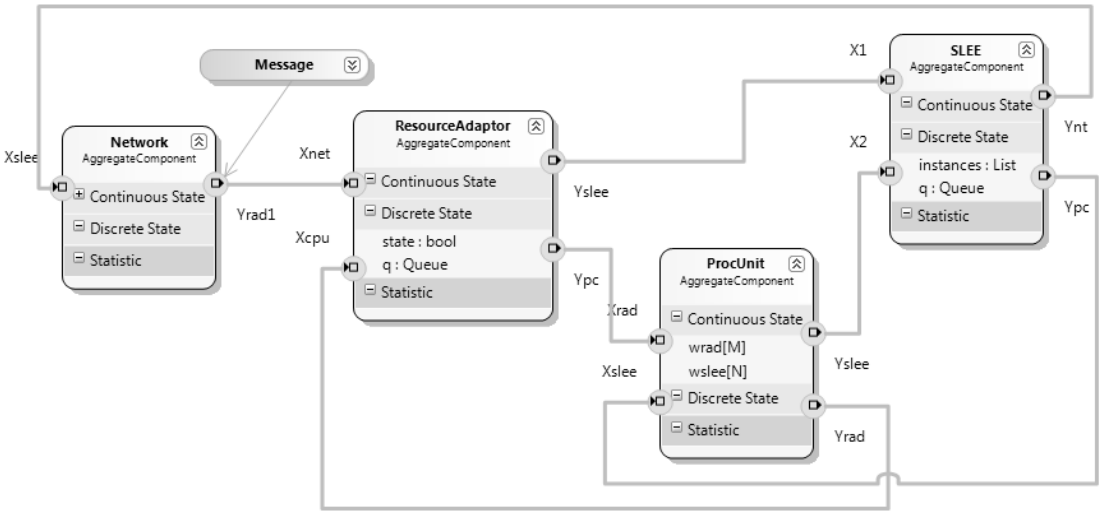


Figure 4. Structural scheme of PLA simulation model.

Due to the formal model simulation modelling program was created. For program code creation of simulation models specificated using PLA formalism, object -oriented imitation modeling library PLASim, created in KTU business department, was used as well as graphical redactor specialized for PLA simulation models creation. In programming model IN services specificated using sequences diagram (Figure 3) are simulated. This sequences diagram is considered as the model's input parameter. Created model allows imitation of several types of IN services processing with different sequence diagrams of such type.

Simulation Results

Simulation experiments with the program model were performed. The call processing scenario depicted in Figure 3 can be split into two phases:

1. Starting with call initiation indication from network (*InitialDP* message), ending with response from service logic (*ConnectToResource* and *PromptAndCollectUserInformation* messages)
2. Starting with *ReturnResult* message from network and ending with service logic release the call indicated to network by *ReleaseCall* message.

Graphs presented in this chapter (Delay1, Delay2) correspond to the call processing phases 1 and 2 respectively as described above.

IN service's processing time dependencies for the execution of software components of MSLEE were gained (Figure 5, Figure 6).

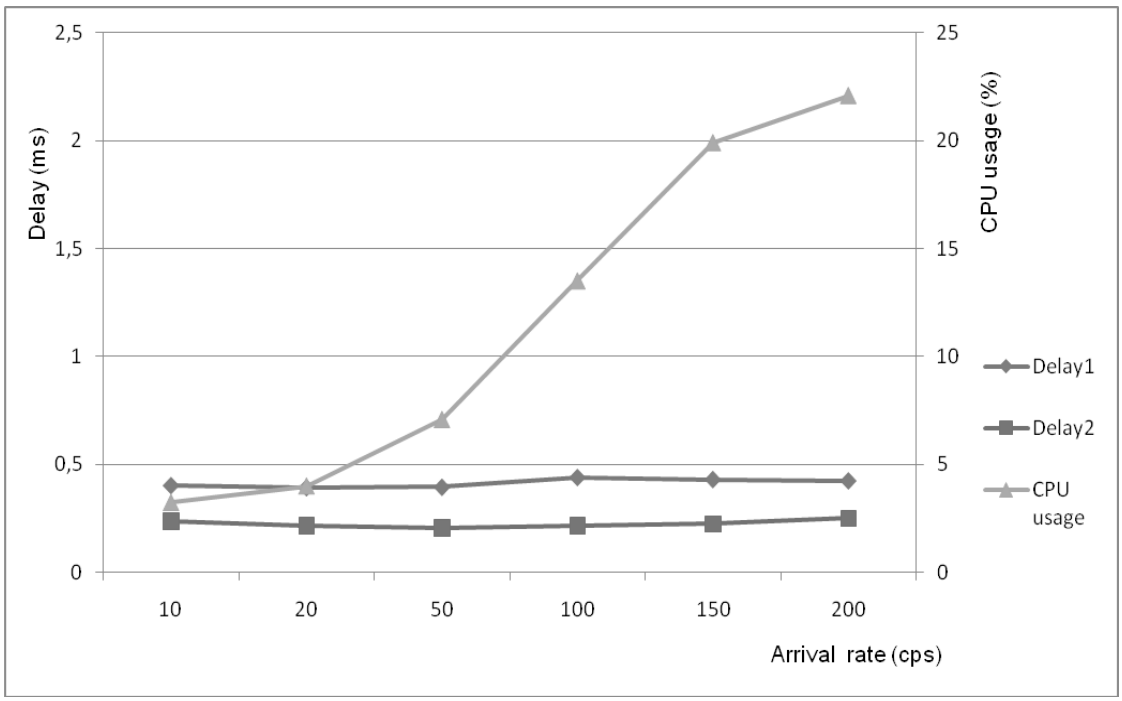


Figure 5. Testing results: *Delay1*, *Delay2* and *CPU usage* dependencies from arrival rate

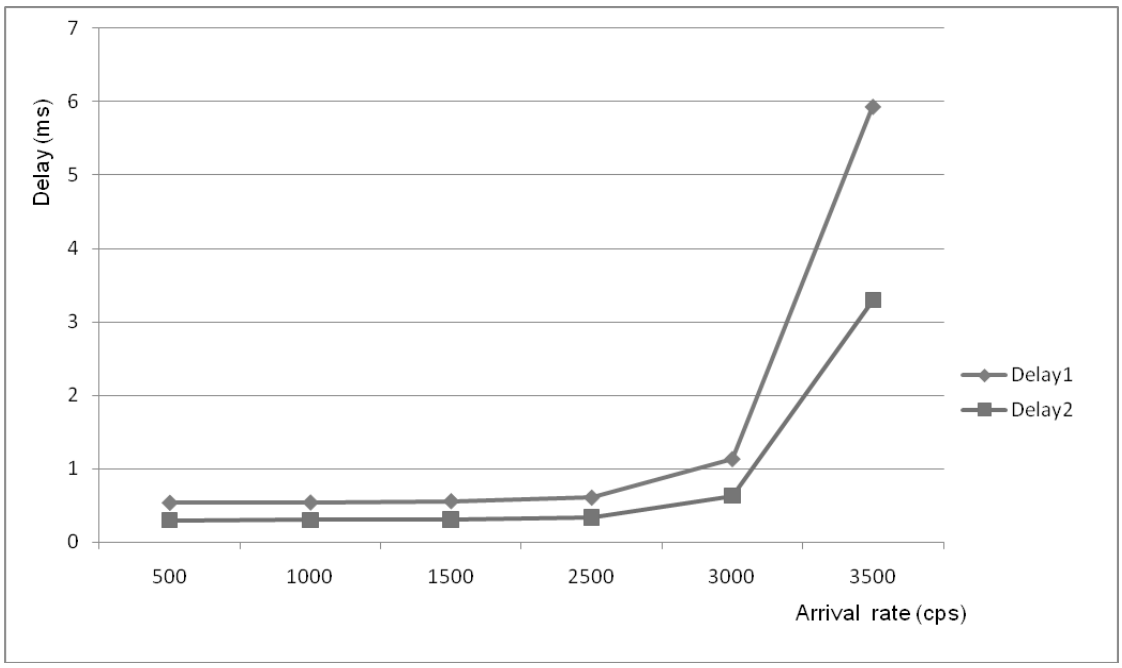


Figure 6.Simulation results: *Delay1* and *Delay2* dependencies from arrival rate

Conclusions

In this paper we have analyzed the open source Mobicents JSLEE server scalability performance through a typical IN service. The proposed solution proved to be highly effective even without employing any scalability-oriented deployment scheme. Testing and simulation have shown that Mobicents is fully capable to

be used for call processing in high traffic networks. Moreover, flexibility of the solution allows easy horizontal scalability, which can be used to achieve both higher throughput and high availability. Having integrated IN protocols Mobicents becomes a suitable and cost-effective replacement for commercial IN platforms.

Acknowledgements

This work was supported in part by measure "IntelektasLT" of EU structural programme for Economic Growth in Lithuania. Project "Development of the third generation service delivery platform based on open source service logic execution environment for telecommunication intelligent networks" VP2-1.3-ŪM-02-K-01-058.

References

[1] **Faynberg I., Gabuzda L.R., Kaplan M.R., Shah N.J.** Intelligent Network Standards: Their Application to Services. *McGraw-Hill.* 1997.

[2] **Sun Microsystems**. JSLEE Specification JSR 240. 2008 [interactive 2010.12.12] access via the Internet: http://jcp.org/en/jsr/detail?id=240.

[3] **Femminella, M.; Maccherani, E.; Reali, G.** A software architecture for simplifying the JSLEE service design and creation. *Proceedings of 8th International Conference on Software, Telecommunications and Computer Networks (SoftCOM).* 2010.

[4] **Khlifi H., Gregoire J.-C.** IMS application servers: roles, requirements, and implementation technologies. *IEEE Internet Computing.* 2008, vol. 12(3).

[5] **Mobicents.** The Open Source SLEE and SIP Server. [interactive 2010.12.12] access via the Internet: http://www.mobicents.org.

[6] **Bhayani, A.** Developing Converged Application using Open Source software. *IEEE Symposium on Industrial Electronics & Applications ISIEA.* 2009. vol.1.

[7] **Femminella M., Francescangeli R., Giacinti F., Maccherani E., Parisi A., Reali G.** Scalability and performance evaluation of a JAIN SLEE-based platform for VoIP services. *21st International Teletraffic Congress ITC 21.* 2009.

[8] **OSA/Parlay.** Open Application Programming Interface (API) for application access to telecoms network resources. [interactive 2011.01.14] access via the Internet: http://etsi.org/WebSite/Technologies/OSA.aspx.

[9] **The Java Community Process Program**. JSR 21: JAIN™ JCC Specification [interactive 2011.01.14] access via the Internet: http://jcp.org/en/jsr/detail?id=21.

[10] **The Java Community Process Program**. JSR 122: JAIN™ JCAT [interactive 2011.01.14] access via the Internet: http://jcp.org/en/jsr/detail?id=122.

[11] **Tian G., Fidge C., Tian Y.** Hybrid system simulation of computer control applications over communication networks. *IEEE International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems, MASCOTS '09*. 2009.

[12] **Pranevicius H.** Aggregate approach for specification, validation, simulation and implementation of computer network protocols. *Lecture Notes in Computer Science No 502, Springer*. 1992.

[13] **Giambiasi N., Escude B., Ghosh S**. GDEVS: a generalized discrete event specification for accurate modeling of dynamic systems. *Proceedings. 5th International Symposium on Autonomous Decentralized Systems*. 2001.

[14] **Guginis G., Pilkauskas V.** The system for generating simulation model code from the PLA specifications. *Proceedings of the 14th International Conference on Information and Software Technologies, IT 2008,* Kaunas, 2008.

DIEGIMO AKTAS

2011-05-25

Kaunas

Viliaus Panevėžio magistro darbo „Intelektualių tinklų protokolų integracijos MOBICENTS platformoje tyrimas“ rezultatai įdiegti UAB „Elitnet“ kuriant Intelektualių telekomunikacinių tinklų paslaugų teikimo platformą atvirojo kodo Mobicents pagrindu.

UAB „Elitnet“ Projektų vadovas

(Parašas) (V. Pavardė)

A.V.