KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PRAKTINĖS INFORMATIKOS KATEDRA

Evaldas Miliauskas

Mobilaus įrenginio ir serverio duomenų sinchronizacijos galimybių tyrimas esant nepastoviam interneto ryšiui

Magistro darbas

Vadovas
doc. V. Pilkauskas

Kaunas, 2009

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PRAKTINĖS INFORMATIKOS KATEDRA

Evaldas Miliauskas

Mobilaus įrenginio ir serverio duomenų sinchronizacijos galimybių tyrimas esant nepastoviam interneto ryšiui

Magistro darbas

Recenzentas
doc. dr. E. Valakevičius
2009-05-22

Vadovas
doc. V. Pilkauskas
2009-05-25

Atliko
IFM-3/2 gr. stud.
Evaldas Miliauskas
2009-05-25

Kaunas, 2009

Turinys

Paveikslų sąrašas

Lentelių sąrašas

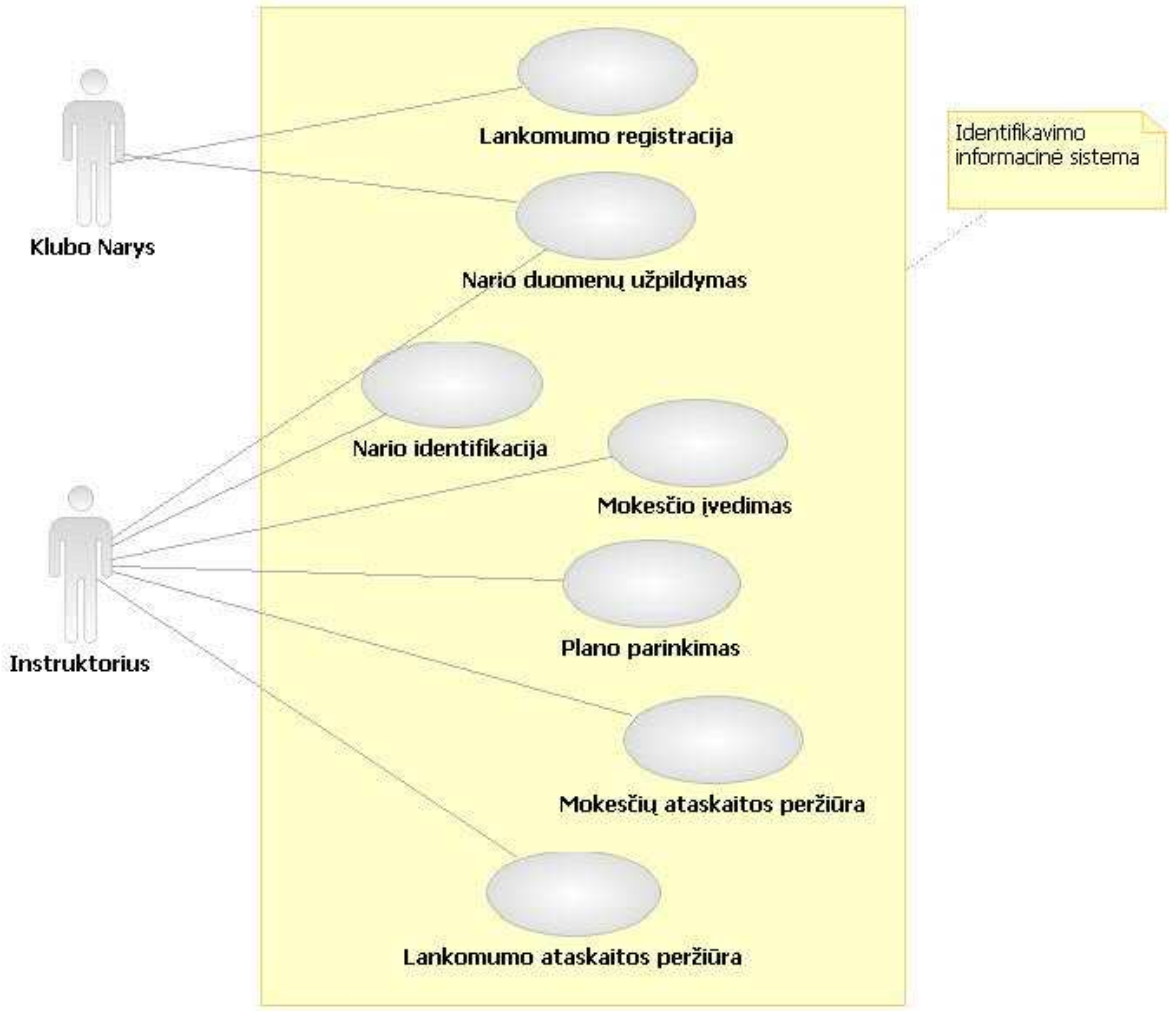
1 ĮVADAS

1.1 Darbo tikslas

Šis darbas skirtas išnagrinėti mūsų projekto (klubo valdymo informacinė sistema) siaurą sritį, t.y. duomenų sinchronizaciją tarp serverio ir mobilaus įrenginio esant nepastoviam interneto ryšiui. Šiame darbe apžvelgsime galimus sinchronizacijos metodus, duomenų sinchronizacijos techninius ypatumus, projektavimo aspektus norint įtraukti sinchronizaciją į jau egzistuojančią sistemą bei sudarysime vieno sinchronizacijos algoritmo modelį ir jį ištirsime. Norint geriau suprasti šios srities naudą turime trumpai aptarti magistrinio projekto kontekstą (tikslus), kuriamos sistemos funkcionalumą ir technines ypatybes.

1.2 Magistrinio projekto (klubo valdymo sistema) funkciniai ypatumai

Pagrindinis sistemos tikslas – narių identifikacija pagal korteles. Norima, jog naudojantis sistema laikas reikalingas narių apskaitai sumažėtu dvigubai. Taip pat, sistema turi dalinai arba pilnai automatizuoti treniruočių planų parinkimą, mokesčių skaičiavimą nariams, užsakymų atlikimą. Taip pat, turi būti galimybė gauti mokesčių ataskaitas pagal pasirinktus duomenis ir laikotarpį. Paveiksle 1.1 pateikiame panaudos atvejų diagramą, kuri vaizduoja pagrindinį sistemos funkcionalumą. [1]



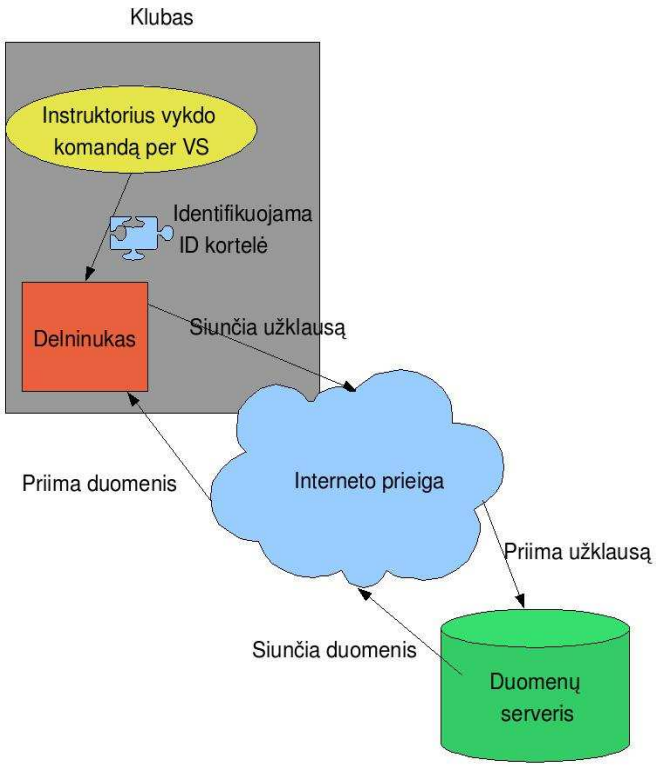
Paveikslas 1.1 Klubo valdymo sistemos panaudos atvejų diagrama

Kiekvieną funkciją trumpai aprašysime:

- Vartotojo autentifikacija – Vartotojo atpažinimas sistemoje, pagal jo prisijungimo duomenys.
- Nario identifikacija – Identifikuojamas klubo narys, pagal jo ID kortelę.
- Nario duomenų užpildymas – Duomenų bei kontaktų, reikalingų klubo nario identifikavimui, įvedimas.
- Lankomumo registracija – Užregistruojamas klubo narys, pagal jo ID kortelę.
- Mokesčių įvedimas – Įvedamas naujas klubo nario mokėjimas (pagal pasirinktą tipą).
- Plano Parinkimas – Parenkamas ir priskiriamas planas klubo nariui.
- Mokesčių ataskaitos – Peržiūrima mokesčių ataskaita iš DB pagal pasirinktą laiko periodą, narius, klubus ar kitus kriterijus.

- Lankomumo ataskaitos – Peržiūrima lankomumo ataskaita iš DB pagal pasirinktą laiko periodą, narius, klubus ar kitus kriterijus. [1]

Šios sistemos techninės ypatybes gerai atvaizduoja paveiksle 1.2 pateikta bendro pobūdžio diegimo aplinkos diagrama. Joje vaizduoja sistemos išskirstymas į dvi pagrindines dalis, t.y. serverio ir kliento puses. Serverio dalyje saugomi visi duomenys ir funkcijos skirtos manipuliuoti šiais duomenimis. Taip pat, jis suteikia prieigą prie minėtų funkcijų, tam, kad būtų galima atlikti pakeitimus naudojantis nuotoliniu klientu. Kliento pusėje naudojama mobilus įrenginys, su kuriuo dirba vartotojas norėdamas atlikti duomenų pakeitimus. Mobiliajam kliente taip pat yra specialus įėjimas skirtas nuskaityti identifikavimo kortelėms, kuriomis atpažįstami klubo nariai.



Paveikslas 1.2 Klubo valdymo sistemos diegimo aplinkos diagrama.

Taigi sistemos visi duomenų apsikeitimai vyksta internetu ir kliento pusėje jokie duomenys nelaikomi. Toks veikimo principas yra saugus, tačiau nėra efektyvus, jeigu vartotojui nėra prieinamas spartus internetas. Todėl šiame darbe panagrinėsime kaip

galima patobulinti šią sistemą naudojant duomenų sinchronizacijos metodus bei sukursime modelį, kurį naudosime savo tyrimams pagrįsti.

1.3 Santrauka

Duomenų sinchronizacija yra neatsiejama šiomis dienomis, kuomet eina kalba apie paskirstytas aplikacijas, kurios veikia mobiliuose įrenginiuose. Kuomet duomenys yra sinchronizuojami mobiliajame įrenginyje, jų vartotojai yra nepriklausomi nuo duomenų serverio ir gali laisvai naudotis lokaliais duomenis atsijungę nuo tinklo. [14]

Šiame darbe išnagrinėjome galimas duomenų prieigos architektūras:

1. Visuomet atsijungęs
2. Visuomet prisijungęs
3. Mišrus

Taip pat aptarėme techninės realizacijos ypatumus, problemas su kuriomis susiduriama, kuriant sinchronizacijos algoritmus bei būtiną informaciją reikalingą šių algoritmų funkcionavimui (ID valdymas, pakeitimų sekimas, greita/lėta sinchronizacija ir kt.).

Projektinėje dalyje buvo išnagrinėtos jau egzistuojančios sistemos architektūra, bei aprašyti reikalingi pakeitimai, tam kad būtų galima įtraukti sinchronizacijos procesą į sistemos funkcionalumą.

Galiausiai detaliai išanalizavome mišrios duomenų prieigos architektūrą, jos paskirtį, privalumus ir trūkumus. Taip pat sudarėme imitacinį modelį pagal šios architektūros principus panaudodami sudėtingų sistemų formalizavimo agregatinį metodą bei atlikome jo parametrų analizę.

2 ANALITINĖ DALIS

Šioje dalyje plačiai apžvelgsime mūsų projekto (klubo valdymo informacinė sistema) siaurą sritį, t.y. duomenų sinchronizaciją tarp serverio mobilaus įrenginio. Norėdami tai atlikti pirmiausia aprašysime kelis galimus sprendimo būdus. Išnagrinėsime jų privalumus ir trūkumus bei vieną būdą, kurį mes modeliuosime projektinėje dalyje. Taip pat šiame skyriuje apžvelgsime duomenų sinchronizacijos techninius ypatumus bei siūlomą realizacijos algoritmą.

2.1 Sinchronizacijos metodai [2]

Šiuolaikinės verslo aplikacijos visuomet naudoja įvairaus pobūdžio duomenis. Pavyzdžiui klientų, užsakymų, knygų, darbuotojų, inventoriaus duomenys ir kt. Šių duomenų pagrindinė versija (angl. vert. „master“) yra saugoma centrinio serverio reliacinėje duomenų bazėje. Verslo aplikacijos, kurios gali veikti tiek asmeniniuose kompiuteriuose, tiek mobiliuose įrenginiuose, gali skaityti ir keisti šiuos duomenis įvairiais būdais, todėl pakeisti duomenys turi būti sinchronizuoti su duomenimis esančiais centrinėje saugykloje. Todėl šioje skiltyje aprašysime skirtingas duomenų prieigos architektūras, kurios gali būti panaudotos kliento/serverio aplikacijose.

Kliento/serverio duomenų prieigos architektūros gali būti suskirstytos į kelis tipus:

1. Visuomet atsijungęs
2. Visuomet prisijungęs
3. Mišrus

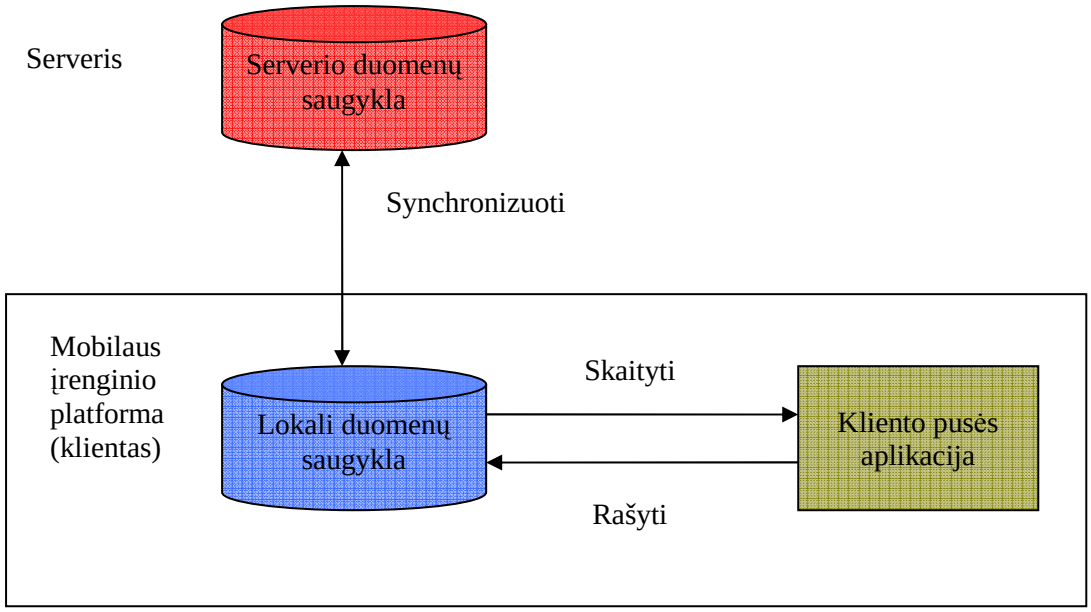
Kiekvienas iš šių duomenų prieigos šablonų turi skirtingus reikalavimus kliento pusės vartotojo sąsajai, kuri naudojama kreipiantis į suprojektuotą duomenų saugyklą.

Šiuose šablonuose visuomet yra serverio duomenų saugyklą ir lokali duomenų saugykla. Serverio duomenų saugykla yra kuri nors reliacinė duomenų bazė, kuri naudojama verslo sistemose, kaip pavyzdys galėtų būti IBM DB2, Oracle ar Microsoft SQL Server. Lokali duomenų saugykla yra lokali reliacinė duomenų bazė, kuri

naudojama mobilaus įrenginio platformoje. Šiuo atveju tai gali būti IBM DB2 Everyplace, Derby, Cloudscape ar kt.

2.1.1 Visuomet atsijungęs

Paveiksle 2.1 pateiktame paveiksle atsispindi visuomet atsijungusio kliento architektūros šablonas. Šiame šablone, kliento aplikacija visuomet kreipiasi į lokalią duomenų bazę, nepriklausomai nuo to ar kliento aplikcija gali fiziškai prisijungti prie centrinės duomenų saugyklos. Kuomet yra galimybė, lokali duomenų bazė yra sinchronizuojama su serverio duomenų baze tam tikrais nustatytais laikais arba pagal vartotojo užklausą.



Paveikslas 2.1 Visuomet atsijungusio kliento/serverio duomenų prieigos architektūra.

Sinchronizacija ne visuomet turi būti vykdoma nustatytu grafiku. Kuomet klientas yra prisijungęs, t.y. kada jis turi prieigą prie serverio – duomenų sinchronizacija gali būti vykdoma iškart po pakeitimų lokalioje duomenų bazėje. Tuomet duomenys tuo pačiu metu yra atnaujinami serverio pusės duomenų bazėje kuomet kliento pusėje įvyksta pakeitimai. Mobilaus įrenginio platformos suteikia įrankius kuriais naudojantis galima nustatyti kada klientas yra prisijungęs prie tinklo.

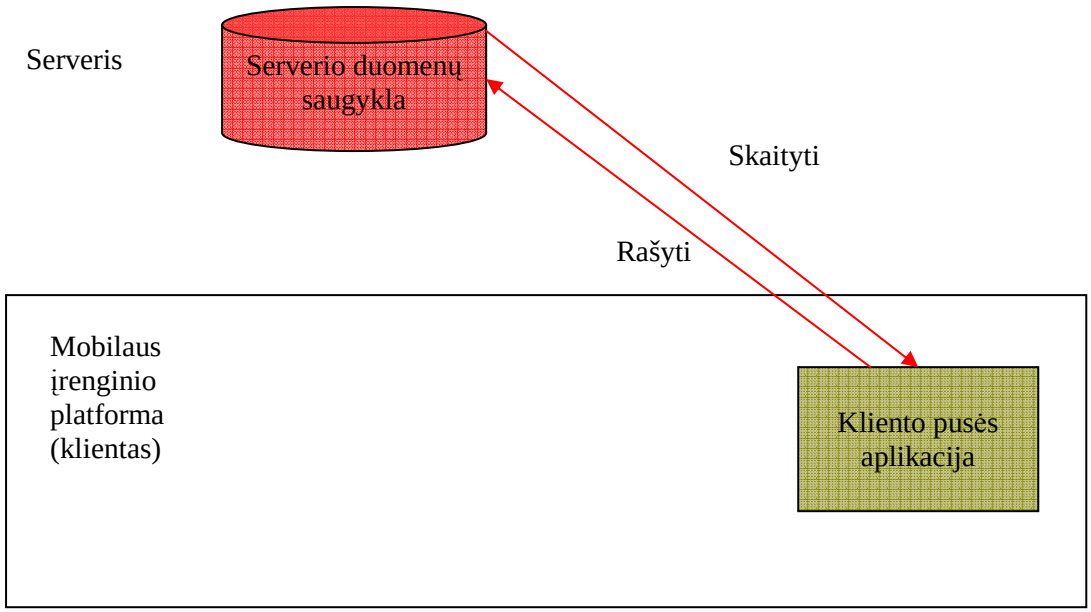
Visuomet prisijungusio šablono privalumai:

- Prieinamumas. Kliento aplikacija galima naudotis nepriklausomai ar jis yra prisijungęs prie tinklo.
- Sparta. Kreiptis į lokalią duomenų bazę yra visuomet greičiau nei nuotolinę.
- Sumažinta serverio apkrova. Klientas ir serveris sąveikauja tik sinchronizacijos metu.

2.1.2 Visuomet prisijungęs

Toliau nagrinėsime visuomet prisijungusios duomenų prieigos šabloną, kuri pavaizduota sekančiame paveiksle. Šiame šablone klientas skaito ir rašo tiesiai iš serverio duomenų saugyklos. Tokio pobūdžio duomenų apsikeitimam paprastai naudojama internetinės paslaugos (angl. ver. web Services) arba REST-pagrindu veikiantis interfeisas.

Jeigu naudojama internetinės paslaugos, tuomet klientas atlieka duomenų keitimo operacijas kviesdamas resursus nuotoliniu būdu. Pavyzdžiui, gautiAsmeni() ar atnaujintiAsmeni(). Abidvi užklausos ir atsakymai yra apjungiami specialiu SOAP formatu. Modernios PĮ kūrimo priemonės suteikia galimybę sugeneruoti SOAP formatus automatiškai, todėl PĮ kūrėjas neapkraunamas papildomu rankiniu darbu.



Paveikslas 2.2 Visuomet prisijungusio kliento/serverio duomenų prieigos architektūra.

Taigi internetinių paslaugų metodai kreipiasi tiesiogiai į serverio pusės aplikaciją, kad galėtų atlikti duomenų pakeitimus. Tiesioginis ryšys su duomenų baze iš kliento aplikacijos į serverį yra nepatartinas, kadangi tai sukuria stiprų priklausomybės ryšį, kuris apsunkina pačios aplikacijos palaikymą.

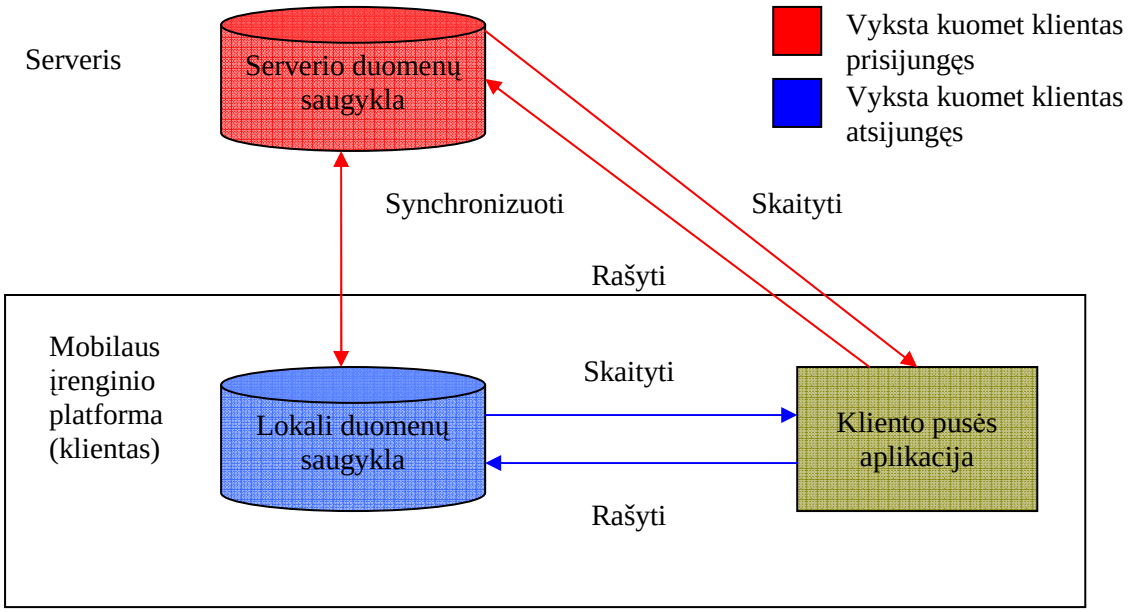
Tokia architektūra primena internetinės naršyklės veikimą, kuomet naudojama Ajax technologija. Šioje technologijoje, vartotojo sąsaja ir aplikacijos logika yra parsiunčiama ir vykdoma pačioje naršyklėje. Tuomet naršyklė naudoja JavaScript kalbą, kad galėtų rašyti/skaityti duomenis neperkraunant tinklapio lango. Taip pat ir mobilioje platformoje, kuri turi vartotojo sąsają ir aplikacijos logiką, reikalinga tik priėjimas prie duomenų serveryje.

Pagrindiniai trys privalumai naudojant šią architektūrą:

- Klientas visuomet dirba su naujausiais duomenimis.
- Serverio duomenys yra pastoviai atnaujinami.
- Nereikalinga jokia sinchronizacijos logika ar duomenų konfliktai kuriuos reikalinga išspręsti.

2.1.3 Mišrus

Paskutinis metodas duomenų sinchronizacijai, kurį aprašysime, yra mišrios architektūros šablonas. Mišrioje architektūroje, kliento aplikacija gali nustatyti ar ji turi prieigą prie tinklo ar ne. Kuomet ji turi prieigą kliento pusės aplikacija dirba su serverio pusės duomenų prieiga. Kuomet prieigos nėra, kliento pusės aplikacija dirba su lokalia duomenų baze. Kuomet klientas yra prisijungęs prie tinklo, jis periodiškai sinchronizuoja duomenų pasikeitimus, padarytus lokalioje duomenų bazėje, su serveriu. Vartotojo sąsajos lygyje turi būti mechanizmas nustatantis, ar aplikacija yra prisijungus prie tinklo ir pagal tai rodyti atitinkamus komponentus.



Paveikslas 2.3 Mišri kliento/serverio duomenų prieigos architektūra.

Norint, kad aplikacija veiktu skirtingai, kai ji yra prisijungusi prie tinklo, mobili platforma gali atpažinti kliento būseną ar jis turi prieigą prie tinklo ar ne. Prisijungusi/atsijungusi būsena gali būti nustatyta naudojanti vartotojo sąsaja arba programiniu būdu. Kliento pusės aplikacija turi galimybę siųsti užklausą mobilioje platformoje, kuri grąžintu prisijungimo būseną arba iššauktų įvykį, kuris pakeistu pačios aplikacijos veikimo būdą.

Šis šablonas yra sudėtingiausias iš visų trijų, tačiau suteikia privalumus iš abiejų architektūrų. Todėl jis gerai tinka vartotojams kurie:

- Didžiąją laiko dalį yra prisijungę prie didelio pralaidumo tinklo.
- Kartais dirba atsijungia nuo tinklo.
- Turi poreikį keisti duomenis, kuomet yra atsijungę nuo tinklo.

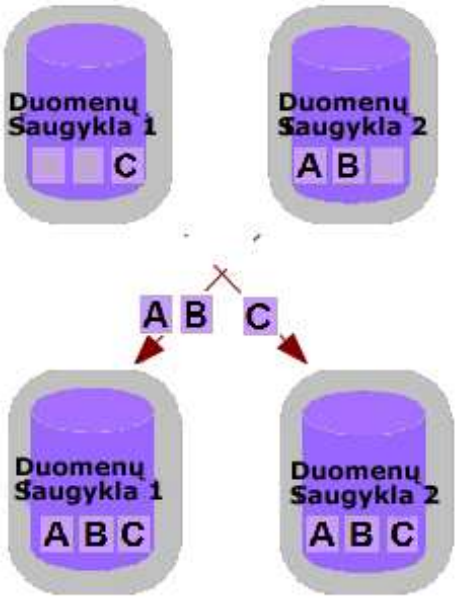
Alternatyva šiam šablonui būtų visuomet atsijungęs šablonas, tačiau sukonfiguruojant aplikaciją taip, jog ji sinchronizuotų duomenys esant kiekvienam pakeitimui lokalioje duomenų bazėje. Tokio pobūdžio veikimas yra labai panašus į mišrios architektūros veikimą, tačiau yra paprasčiau sukurti, kadangi aplikacijos logika turi tik vieną veikimo būda – atsijungusį.

Ši architektūra suteikia tokius privalumus:

- Naudojasi tinklo prieigos privalumais, kas suteikia galimybę dirbti su duomenimis realiu laiku.
- Turi galimybę veikti atsijungusi nuo tinklo.

2.2 Techninės realizacijos galimybės [4]

Visi mobiliųjų įrenginiai – delniniai kompiuteriai, mobilūs telefonai, peigeriai, nešiojamieji kompiuteriai – turi sinchronizuoti duomenis su serveriu, kuriame saugoma informacija. Ši savybė leidžianti atnaujinti duomenis bet kuriuo momentu yra neatskiriama nuo mobilios kompiuterizacijos. Tačiau šiandien beveik kiekvienas įrenginys naudoja skirtingas technologijas atlikti duomenų sinchronizacijai. Toliau aprašysime pagal SyncML standartą pagrindines duomenų sinchronizacijos sprendžiamas problemas. [5]



Paveikslas 2.4 Duomenų Sinchronizacijos procesas

Duomenų sinchronizacija yra naudinga daugelyje sričių:

- Siųsti atnaujinimus į vis augantį skaičių naujų aplikacijų;
- Išvengti apribojimų tarp mobilių įrenginių ir bevielio ryšio;
- Pagerinti vartotojo galimybes ir sumažinti duomenų prieigos atsako laiką;
- Sudaryti galimybes IT infrastruktūros išplečiamumui, tokioje aplinkoje, kur įrenginių (klientų) ir prisijungimų skaičius visuomet auga;

- Padeda suprasti mobilių aplikacijų reikalavimus, suteikia vartotojui galimybes, kurios nesudaro nepatogumų atliekant mobilius darbus.

Duomenų sinchronizacija tai procesas, kuris suvienodina dvi duomenų aibes (pateiktas paveikslas 2.4). Tai įtraukia ne vieną koncepciją, svarbiausios yra:

- ID valdymas
- Pakeitimų aptikimas
- Modifikacijų apsikeitimas
- Konfliktų aptikimas
- Konfliktų sprendimas
- Lėta ir greita sinchronizacija

2.2.1 ID valdymas

Iš pirmo žvilgsnio ID valdymas atrodo trivialus procesas, kuris neturėtų kelti didelio susidomėjimo. Iš tiesų, ID valdymas yra svarbi sinchronizacijos proceso dalis, kuri nėra tokia triviali. Kiekviena duomenų dalis yra identifikuojama jos tam tikru duomenų poaibiu. Pavyzdžiui, jeigu turi kontaktų įrašą, naudodami vardą ir pavardę gauname visus likusius kontakto duomenis iš savo duomenų katalogo. Kitais atvejais naudojamas ID, kuris saugomas specialiame duomenų lauke, dėl galimybės išgauti įrašą iš duomenų bazės. Tai gali būti naudojama pardavimų automatizavimo mobilioje aplikacijoje, kur užsakymas yra atpažįstamas pagal užsakymo numerį arba nuorodą. Tokiu būdu prekės ID yra sugeneruojamas bei negali būti nustatytas iš anksto. Tokio pobūdžio duomenys dažniausiai būna specifiniai aplikacijoms ir įrenginiams.

Korporacijų sistemose duomenys yra saugomi centralizuotoje duomenų bazėje, kuria dalinasi visi vartotojai. Kiekviena prekė yra atpažįstama sistemoje pagal globalų ir unikalų ID. Kai kuriais atvejais egzistuoja dvi aibės duomenų (pavyzdžiui užsakymas kliento aplikacijoje ir užsakymas serveryje), kurios atitinka tą pačią informaciją (pirkėjo atliktas užsakymas), tačiau yra skirtingos. Kokių priemonių tokiu atveju reiktu imtis, norint suvienodinti kliento ir serverio ID, tam kad informacija būtu vientisa? Galimas ne vienas sprendimo būdas:

- Klientuose ir serveryje suderinama ID schema (turi būti sudaryta taisyklė nusakanti kaip generuojami ir naudojami ID raktai);

- Kiekvienas klientas generuoja globalius unikalius ID (GUID) ir serveryje saugomi kliento sugeneruoti ID;
- Serveris generuoja globalius globalius unikalius ID (GUID) ir kliente saugomi serverio sugeneruoti ID;
- Klientas ir serveris generuoja savo ID atskirai, tuomet saugoama ryšiai tarp šių skirtingų ID. Kliento pusės ID vadinami lokalūs unikalūs identifikatoriai (LUID) ir serverio pusės ID yra vadinami globalūs unikalūs identifikatoriai (GUID). Šių dviejų tipų ID ryšys vadinamas LUID-GUID ryšių.

2.2.2 Pakeitimų aptikimas

Pakeitimų aptikimas – tai procesas, kurio metu aptinkama pasikeitę duomenys nuo tam tikro laiko taško (pavyzdžiui, nuo paskutinės sinchronizacijos). Šis funkcionalumas realizuojamas pridedant papildomos informacijos, t.y. laiko žymes bei būsenos informaciją. Tarkime turime duomenų bazę, kuriai turi būti atlikti pakeitimai, pavaizduota lentelėje 2.1.

Lentelė 2.1 Duomenų bazė turinti galimybę sekti pakeitimus

ID	vardas, pavardė	telefonas	būsena	paskutinis atnaujinimas
12	John Doe	+1 650 5050403	N	2003-04-22 13:22
13	Mike Smith	+1 469 4322045	D	2003-05-21 17:32
14	Vincent Brown	+1 329 2662203	U	2003-05-21 17:29

Nors kartais senesnės duomenų bazės nesuteikia reikiamos informacijos norint efektyviai aptikti pakeitimus. Tokiu atveju reikalingi papildomi įrankiai norint aptikti pakeitimus (kaip pavyzdys gali būti turinio palyginimas).

2.2.3 Modifikacijų apsikeitimas

Kaip modifikacijos apsikeičiamos tarp serverio ir kliento yra svarbiausias duomenų sinchronizacijos infrastruktūros komponentas. Tam tikslui aprašomas sinchronizacijos protokolas, kurį klientas ir serveris turi naudoti inicijuojant ir vykdant

sinchronizacijos sesiją. Be modifikacijų apsikeitimo, sinchronizacijos protokolas taip pat turi aprašyti kokias modifikavimo operacijas jis gali vykdyti. Minimalios komandos naudojamos modifikacijai yra:

- Įterpti
- Pakeisti
- Ištrinti

2.2.4 Konfliktų aptikimas

Tarkime du vartotojai iš ryto, prieš eidami į ofisą, sinchronizuoja savo lokalių kontaktų duomenų bazę su centriniu serveriu. Prieš sinchronizacija jie turi visiškai vienodus kontaktus savo PDA (mobiliajame įrenginyje). Toliau sakykime jie pakeičia to paties asmens „John Doe“ įrašą, bet dėl kažkokios priežasties jie įveda skirtingus numerius (vienas iš jų padarė klaidą). Kas atsitiks sekantį rytą, kai jie vėl sinchronizuos duomenis? Kurią iš dviejų naujų John Doe įrašo versijų reikia išsaugoti ir laikyti serveryje? Ši būsena vadinama konfliktu ir serverio užduotis yra ją identifikuoti ir išspręsti.

Paprasčiausias būdas identifikuoti konfliktą naudojant „sinchronizacijos matricą“.

Lentelė 2.2 Sinchronizacijos matrica

Duombazė A→ ↓ Duombazė B	Naujas	Ištrintas	Atnaujintas	Sinchronizuotas/ Nepakeistas	Neegzistuojantis
Naujas	C	C	C	C	B
Ištrintas	C	X	C	D	X
Atnaujintas	C	C	C	B	B
Sinchronizuotas/ Nepakeistas	C	D	A	=	B
Neegzistuojantis	A	X	A	A	X

Dėl to, kad vartotojai sinchronizuojasi su centrine duomenų baze, mes galime nuspręsti, kas vyksta tarp serverio duomenų bazės ir vieno iš klientų duomenų bazės konkrečiu laiko momentu: pavadinkime duombazę A kaip kliento duombaze, o duombazę B – serverio. Sinchronizacijos matricoje esantys simboliai turi šiais reikšmes:

- X : nieko nedaryti

- A : įrašas A pakeičia įrašą B
- B : įrašas B pakeičia įrašą A
- C : konfliktas
- D : ištrinti įrašą saugykloje, kurioje jis saugomas

2.2.5 Konfliktų sprendimas

Kuomet įvyksta konfliktas ir jis yra aptinkamas, reikalinga imtis tam tikro veiksmo. Gali būti pritaikyta keletas skirtingų taisyklių:

- Nusprendžia vartotojas: vartotojas yra informuojamas apie įvykusį konfliktą ir nusprendžia ką su juo daryti; ši strategija, kaip ir „klientas laimi“ yra šiek tiek problematiška, kuomet serveris yra centrinis sinchronizacijos taškas: kiekvienas vartotojas gali turėti įrašo modifikavimo teisę ir kartais gali nenustatyti ar jo modifikuotas įrašas turi pakeisti kito įrašą.
- Klientas laimi: serveris, pagal nutylėjimą, pakeičia konflikto būsenoje esančius įrašus su kliento siunčiamais.
- Serveris laimi: klientas turi pakeisti savo įrašus su tais, kurie yra serveryje.
- Pagal laiko žymę: paskutinis modifikuotas (laike) įrašas laimi.
- Paskutinis/pirmas laimi: paskutinis/pirmas atsiųstas įrašas laimi.
- Nesprendžiama.

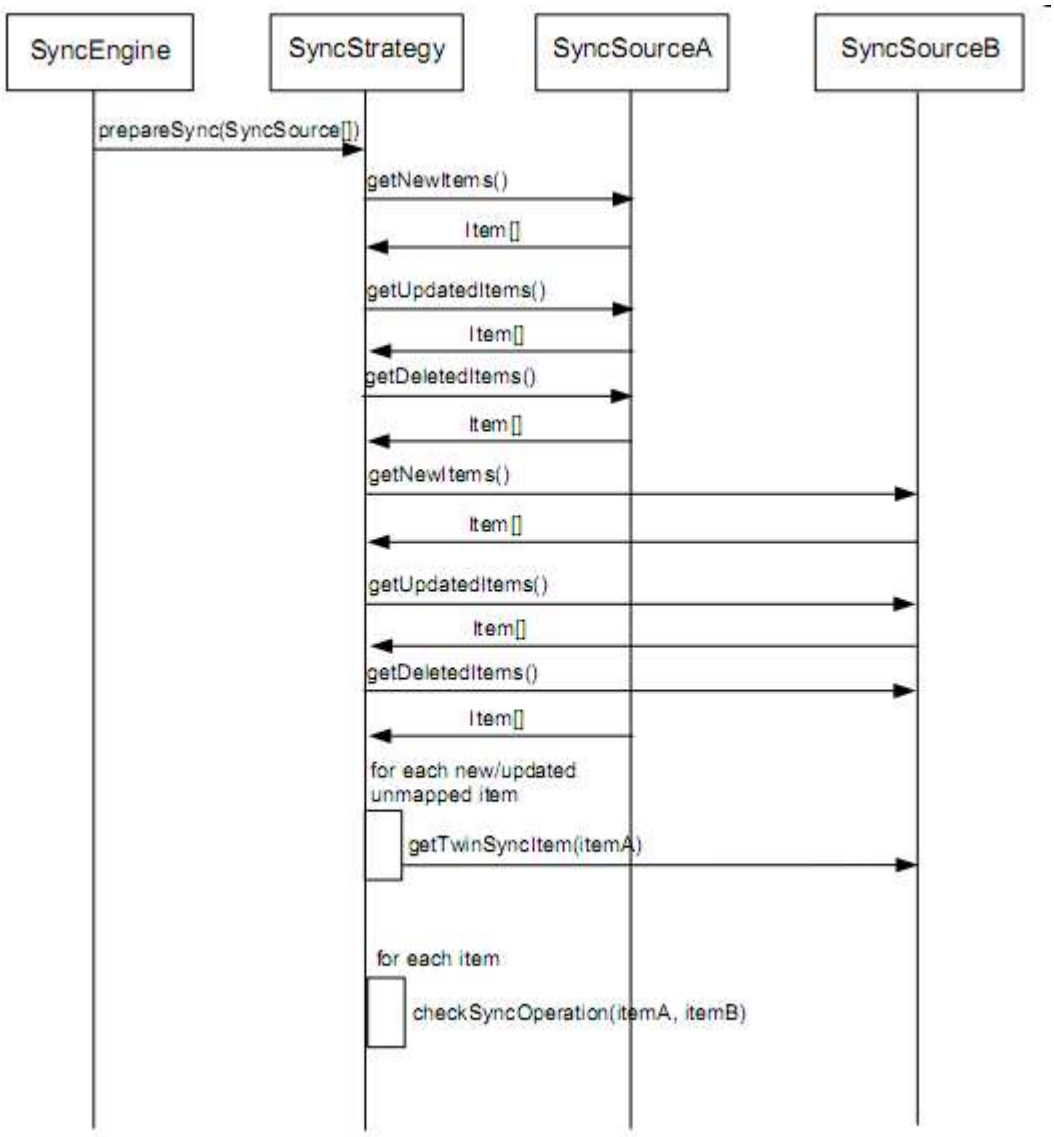
2.2.6 Lėta ir greita sinchronizacija

Sinchronizacijos procese yra ne vienas rėžimas, kaip jis gali būti vykdomas. Pagrindinis skirtumas yra tarp greitos ir lėtos sinchronizacijos. Greita sinchronizacija įtraukia tik tuos įrašus, kurie buvo pakeisti nuo paskutinės sinchronizacijos tarp dviejų įrenginių. Žinoma šis optimizuotas procesas naudojasi faktu, jog abu įrenginiai ankščiau buvo pilnai sinchronizuoti; todėl, abiejų pusių būsena sinchronizacijos pradžioje yra žinoma. Jeigu šis reikalavimas nėra išpildytas (dėl to, kad mobilusis įrenginys buvo išvalytas ir prarado visas laiko žymes nuo praėjusios sinchronizacijos), tuomet reikalinga vykdyti lėtą sinchronizacija. Šiuo atveju, klientas siunčia visą savo duomenų bazė

serveriui, kuris palygina su lokalia duomenų baze ir grąžina klientui modifikacijas, kurios turi būti atliktos, kad duombazė būtų atnaujinta.

Tiek greitu, tiek lėtu rėžimu sinchronizacija gali būti atlikta keleta būdu:

- Iš kliento į serverį: serveris atnaujiną duombazę su kliento modifikacijomis, bet nesiunčia serverio pusėje padarytų modifikacijų.
- Iš serverio į klientą: klientas atnaujiną duomenų bazė su serverio modifikacijomis, bet nesiunčia kliento pusės modifikacijų.
- Abipusė: klientas ir serveris apsikeičia modifikacijomis ir pagal tai atnaujina abidvi duombazes.



Paveikslas 2.5 Sinchronizacijos proceso sekų diagrama

Aukščiau pateikiame aukšto lygio sekų diagramą kokia seka atliekami veiksmai sinchronizuojant įrašus tarp duombazių. Kaip matome sinchronizacija atliekama keliais žingsniais. Pirmiausia gaunami visų pakeistų įrašų sąrašas iš prieigos A (SyncSourceA), po to iš prieigos B (SyncSourceB) ir tuomet kiekvienam įrašui atliekama atitinkama operacija (įterpimo, keitimo ar šalinimo).

2.3 Technologijų pasirinkimas

Paskutinė svarbi dalis, kurią aptarsime analitinėje dalyje, yra technologijos, kurios bus naudojamos mobiliuose įrenginiuose. Šios technologijos, skirtingai nei įprastuose asmeniniuose ar nešiojamuose kompiuteriuose, yra daug įvairesnės ir mažiau standartizuotos. Dėl to kurti aplikacijas, kurios veiktu visose platformose yra didelis iššūkis. Mūsų projekte buvo pasirinkta Pocket PC (Windows Mobile) platforma, o aplikacijos kūrimui naudojome J2ME programos kūrimo įrankius. Taigi toliau trumpai aptarsime kokios platformos yra šiuo metu rinkoje populiarios tarp vartotojų ir trumpai aprašysime jas.

2.3.1 Mobiliųjų įrenginių aplikacijų kūrimo platformos [18]

Pocket PC

Tai platforma, kuri naudojama Microsoft korporacijos. Ji sukurta Windows pagrindu ir dalinai palaiko programas veikiančias Windows platformoje. [19] Šios platformos turi eilę realaus gyvenimo pritaikymų tiek vyriausybinėse tiek verslo organizacijose. [20]

Symbian

Ši operacinė sistema yra plačiai naudojama Nokia, Ericsson, Siemens korporacijų gaminamuose mobiliuose telefonuose ir kituose įrenginiuose. Taip pat visai neseniai ši operacinė buvo paskelbta kaip atviro kodo programinė įranga, kas leidžia kiekvienam prisidėti prie jos kūrimo ir tobulinimo. [21] Symbian jau gyvuoja daugiau nei 10 metų ir yra jungia nemažą programinė įrangos kūrėjų bendruomenę. [22]

Java 2 ME

Sun korporacijos populiari platforma naudojama daugelyje mobiliųjų telefonų. Jos privalumai, jog pakanka aplikaciją sukurti vienai šiai platformai ir ji gali veikti ant skirtingų mobilių įrenginių. [23] Šioje platformoje svarbiausi API (aplikacijos programuojamas interfeisas) yra MIDP ir CLDC.

Pirmoji konfigūracija suteikia galimybę kurti parsisiunčiamas aplikacijas aplikacijas ar paslaugas, kurios gali naudoti mobilųjį ryšį. [24] CLDC konfigūracija skirta labiau resursų apribotiems įrenginiams (atminties, procesoriaus greičio ir kt.). [25]

Mobile Linux

Atviro kodo bendruomenės, kuriama ir tobulinama operacinė sistema. Pagrindinis skirtumas nuo kitų platformų yra jog, ji buvo kuriama ir tobulinama Linux bendruomenės narių, o ne vienos korporacijos. Šios operacinės sistemos vartotojų skaičius nėra gausus ir kolkas nėra daug informacijos apie aplikacijų kūrimą šiai platformai. [26]

Palm OS

Šio tipo operacinės sistemos buvo naudojami pirmuose delniniuose kompiuteriuose. Jie priklauso Palm kompanijai, kuri pradėjo juos gaminti kaip prototipus jau 1996-tais. Nors tai pirmoji mobili operacinė sistema, tačiau jos populiarumas išblėso iki šių dienų. [27]

iPhone OS

Šiuo metu vienas populiariausių mobiliųjų įrenginių pasaulyje yra Apple kompanijos „iPhone“. Ši kompanija visuomet pasižymėjo originaliais įrenginiais, kurie sulaukia didelio vartotojų dėmesio. Šis įrenginys taipogi turi savo atskirą OS, vadinamą iPhone OS, kurį sukūrė pati Apple kompanija. Norint kurti aplikacijas šiai sistemai, kolkas yra tik vienas pasirinkimas naudoti specialią „CoCo“kalbą, kuri yra sukurta objektinės C kalbos pagrindu. [28]

Android

Taipogi naujo tipo operacinė sistema, kuri sukurta naudojant Java kalbą. Ji buvo sukurta Google kompanijos ir panaudota vadinamam „gPhone“ (Google Phone). Nors buvo naudojama Java kalba, tačiau pati platforma realizuota ne pagal J2ME standartus ir nėra suderinama su šios platformos aplikacijomis. Nors ši platforma

pranoksta J2ME pagal savo galimybes, tačiau ji yra skirtinga savo panaudojimu ir sukelią naujų sunku suderinamumui kuriant aplikacijas skirtingoms platformoms. [29]

BlackBerry OS

Visai neseniai Kanados kompanijos RIM sukurti mobilieji įrenginiai vadinami BlackBerry sparčiai paplito tarp vartotojų (daugiau nei 21 milijonas prisiregistravusių). Ji taip pat turi savo operacinę sistemą, tačiau ji palaiko J2ME standartus ir suderinama su MIDP kuriamomis aplikacijomis Java kalba. [30]

2.3.2 Duomenų bazės parinkimo kriterijai [16]

Ne mažiau svarbi detalė nei platforma yra duomenų bazė, kurią naudosime mobiliajame įrenginyje. Mūsų darbe buvo pasirinkta „Embedded Derby“ duombazė, kadangi ji yra atviro kodo bendruomenės produktas ir viena iš populiariausių duombazių saugoti nedidelių kiekių duomenis. Taip pat žemiau aptarsime kriterijus, kuriuos naudinga įvertinti renkantis duombazę.

- Reikalingas funkcionalumas. Renkantis duombazę reikia atkreipti dėmesį į funkcijas, kurios prieinamos ir reikalingos kuriamai aplikacijai. Keletas pagrindinių yra sparta, galimas duomenų kiekis, transakcijų palaikymas, OS palaikymas ir kūrimo įrankių integracija.
- Aplikacijų sudėtingumas. Nustatoma kokio sudėtingumo aplikacijai kuriama duomenų struktūrą, kokie ryšiai ir duomenų keitimo operacijos reikalingos.
- Turimi resursai. Čia būtina apsvarstyti finansines ir kūrimo galimybes. Dažnai ieškomas kompromisas tarp kainos ir funkcionalumo. Taip pat reikia palyginti vidinį kūrimą ir palaikymo kaštus su komerciniu sprendimu.
- Diegimo dydis. Skaičius, kuris rodo kiek aplikacijų bus diegiama gali turėti didelę įtaka pasirenkamai technologijai. Jeigu nedidelis kiekis nėra didelis, dažniausiai kurti savo sprendimą būna per brangu, iš kitos pusės jeigu vartotojų skaičius didelis, tuomet palaikymo kaštai taipogi gali labai išaugti. Dėl to būtina surasti tą tašką, kuris atitinką kainos ir efektyvumo pusiausvyrą.

- Standartų palaikymas. Nustatoma ar reikalinga, jog sistema palaikytų standartus, kaip pavyzdžiui SQL, ODBC ir JDBC.
- Korporacijos integracija. Sprendžiama, kokio pobūdžio integracija gali būti reikalinga korporacijoje. Daugelis komercinių sistemų turi įdiegtas galimybes duombazės sinchronizacijai ir darbui tarp skirtingų protokolų. Todėl būtina nuspręsti ar kuriamai aplikacijai reikalinga toks funkcionalumas.

Tolesniame skyrelyje pateiksime projektuojamo modelio algoritmo diagramą, bei panagrinėsime kaip šis modelis gali būti integruojamas į magistrinio projekto sistemą.

3 PROJEKTINĖ DALIS

Šioje dalyje pateiksime esminius projektuojamos sistemos architektūros aspektus bei atvaizduosime jų veikimą naudodami UML diagramas. Naudodami šias diagramas atliksime sistemos pakeitimus įtraukiant duomenų sinchronizaciją į duomenų apsikeitimo procesą. Taip pat pateiksime projektuojamo duomenų sinchronizacijos metodo algoritmo bendradarbiavimo diagramą.

Architektūros pagrindinis tikslas – sudaryti bendrą sistemos vaizdą, kuris palengvina, programų kūrėjams, jos kūrimą ir keliamų reikalavimų įgyvendinimą. Architektūra turi sujungti techninę pusę su veiklos sritimi, kurioje sistema bus taikoma.

Projektuojant, buvo kreipiamas dėmesys į sistemos išskaidymą, t.y. panaudota 3-jų lygių architektūra: esybės-paslaugos-vaizdavimas. Tai yra naudinga, jeigu sistemai reiktų komunikuoti su kita sistema, paslaugų mechanizmas tam puikiai pritaikytas. Lygių atskyrimų pagrindinis privalumas, jog jie vienas nuo kito nėra priklausomi, dėl to juos nesunkiai galima pakeisti (realizacija naudojant sąsajas).

3.1 Duomenų sinchronizacijos projektavimas

Toliau panagrinėsime kiekvieną diagramą, aprašysime kokius patobulinimus reikia įgyvendinti norint įtraukti duomenų sinchronizacijos procesą į sistemą.

Išskirsime kelių pobūdžių diagramas:

- Paketų
- Klasių
- Veiklos
- Sekų
- Išdėstymo
- Esybių-ryšių

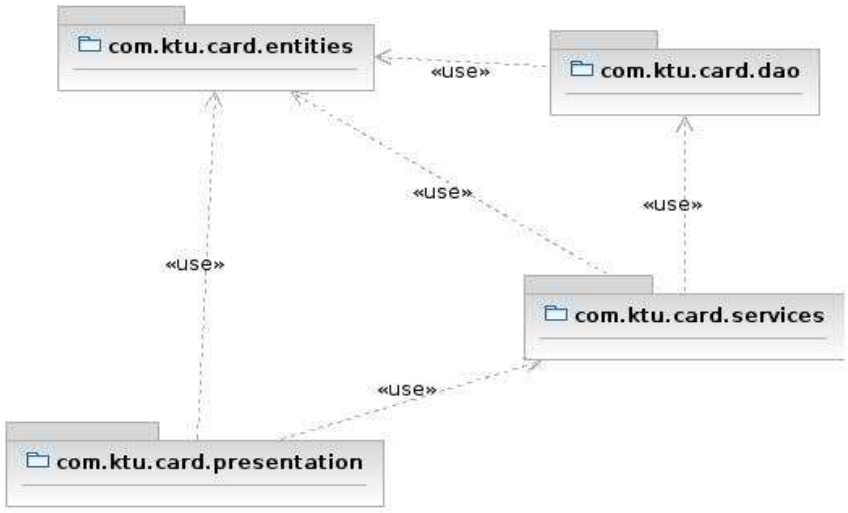
Paketų ir klasių diagramos nurodo sistemos statinį vaizdą, t.y. kokie objektai naudojami realizuojant tam tikras funkcijas.

Veiklos, Sekų vaizduoja kaip klasių objektai bendrauja tarpusavyje, t.y. dinaminis sistemos vaizdas.

Išdėstymo – išdėstymo diagrama aprašo kaip sistemos komponentai išdėstomi fiziniuose įrenginiuose, bei kokiais ryšiais jie susiejami.

Esybių-ryšių – vaizduojamas sistemos duomenų modelis, t.y. nusako kokio duomenys, kurie saugomi duombazėje bei aprašo jų tipus ir tarpusavio ryšius.

3.2 Paketų ir klasių diagramų pakeitimai



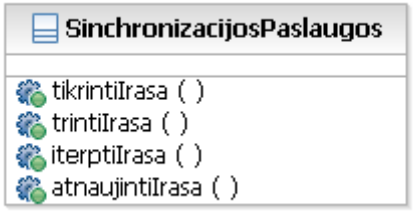
Paveikslas 3.1 Sistemos išskaidymas į paketus.

Paketų diagramoje pakeitimai nereikalingi, kadangi duomenų sinchronizacija bus atliekama paslaugų lygyje. Klientas naudodamas paslaugas iš serverio tikrins, kuriuos duomenis reikia sinchronizuoti. Toliau panagrinėsime paslaugų sluoksnį ir pridėsime funkcijas, kurios reikalingos duomenų sinchronizacijai. [3]

Paveikslas 3.2 Paslaugų sluoksnio sistemos klasių diagrama.

Pridedame naują klasę „SinchronizacijosPaslaugos“, su operacijomis:

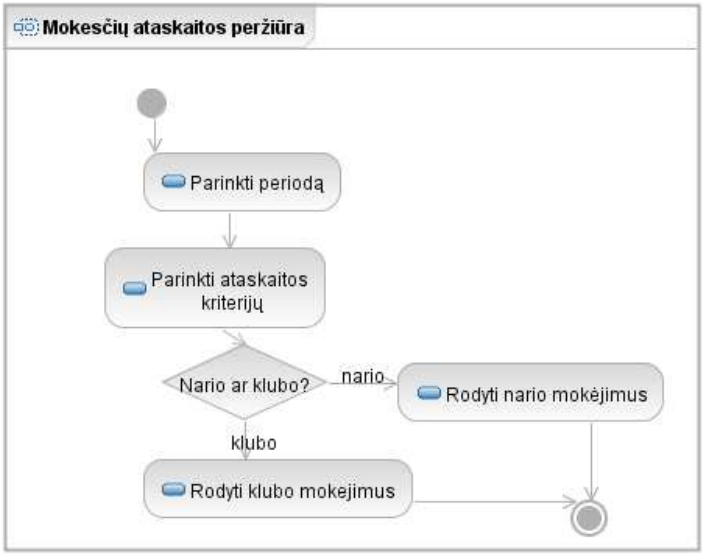
- tikrintiIrasa – lygina serverio ir kliento duomenų įrašus ir pagal datą bei statusą nustatoma koks sinchronizacijos veiksmas turi būti atliekamas.
- trintiIrasa – pagal tipą ir identifikatorių trinamas įrašas serverio duomenų bazėje arba grąžinamas statusas, jog turi būti trinama kliento duomenų bazėje.
- iterptiIrasa – pagal tipą ir identifikatorių įterpiamas įrašas serverio duomenų bazėje arba grąžinamas statusas, jog turi būti įterpiama kliento duomenų bazėje.
- atnaujintiIrasa – pagal tipą ir identifikatorių atnaujinamas įrašas serverio duomenų bazėje arba grąžinamas statusas, jog turi būti atnaujinama kliento duomenų bazėje.



Paveikslas 3.3 Sinchronizacijos paslaugų klasių diagrama.

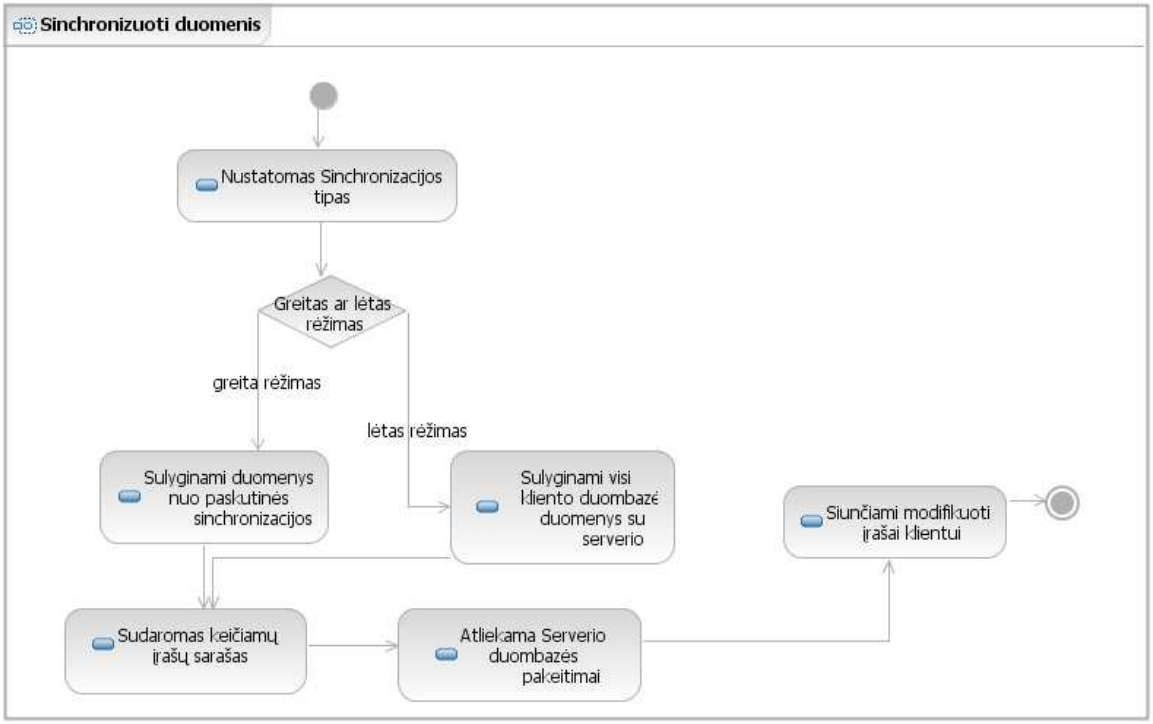
3.3 Veiklos diagramų pakeitimai

Toliau panagrinėsime sistemos veiklos diagramas bei suprojektuosime duomenų sinchronizacijos veiklos diagramą. Paveiksle 3.4 matome kokia veiksmų seka vykdoma užklausa norint peržiūrėti mokesčių ataskaitas. Reikia nepamiršti, jog duomenų užklausos šiuo atveju vyksta į sistemą naudojantis internetu. Kuomet bus įtraukta lokali duomenų bazė tuomet užklausos bus vykdomos į ją, tačiau šiuo atveju pagrindinių veiksmų seka nekinta, keičiasi tik realizacijos ypatumai.



Paveikslas 3.4 Mokesčių ataskaitos peržiūros veiklos diagrama.

Sekančiu žingsniu sudarysime veiklos diagramą, skirtą sinchronizuoti duomenis tarp mobilaus kliento aplikacijos ir serverio. Kaip matome pirmiausia nustatomas sinchronizacijos tipas, t.y. ar duomenys sinchronizuojami pirmą kartą (lėtai) ar naudojama informacija iš praėjusių sinchronizacijų ir galima atlikti tik naujų duomenų sinchronizavimą. Toliau atliekami atitinkamų duomenų sulyginimai bei vykdomos sinchronizacijos operacijos serveryje ir siunčiami pakeitimai klientui.

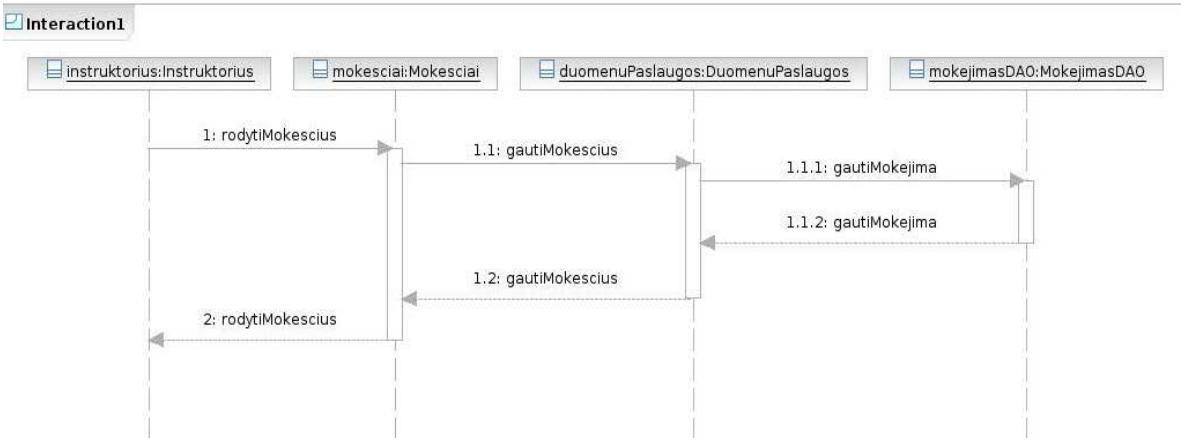


Paveikslas 3.5 Duomenų sinchronizacijos veiklos diagrama.

3.4 Sekų diagramų pakeitimai

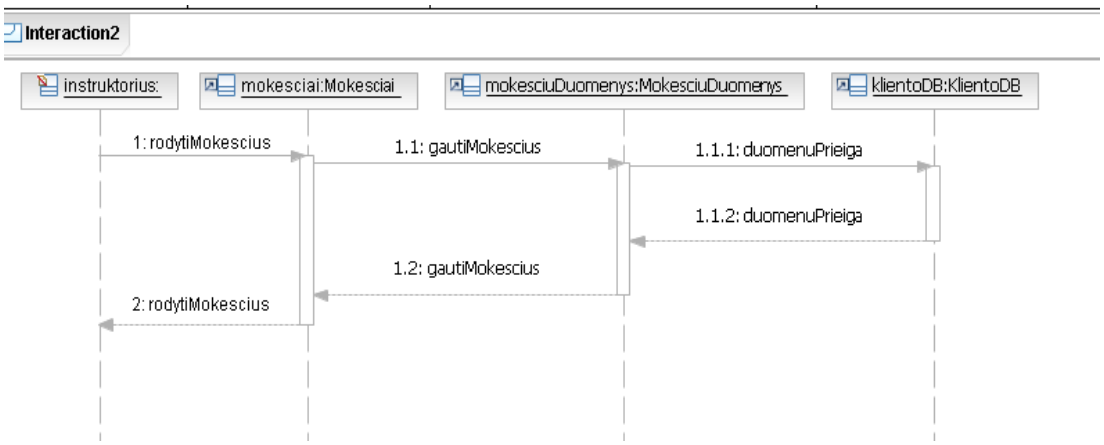
Kliento aplikaciją mūsų projekte buvo realizuota naudojant MIDP (angl. vert. Mobile Information Device Profile), kuriame buvo panaudota RMS (angl. vert. Record Management Structure) tam, kad būtų supaprastintas duomenų valdymas naudojant Java kalbos objektus. [15]

Norėdami suprasti, kaip objektai sąveikauja tarpusavyje, būtina atsižvelgti į sekų diagramas, kurios apibūdina tokio pobūdžio ryšius. Paveiksle 3.6 matome sekos diagramą, kuri nurodo mokesčių peržiūrėjimo funkcijos veiksmų seką bei objektus, kurie įtraukiami į šį procesą. Pirmojoje diagramoje vaizduojama dabartinė architektūra, kuomet kreipiamasi tiesiogiai į serverį.



Paveikslas 3.6 Pradinė mokesčių peržiūros sekų diagrama

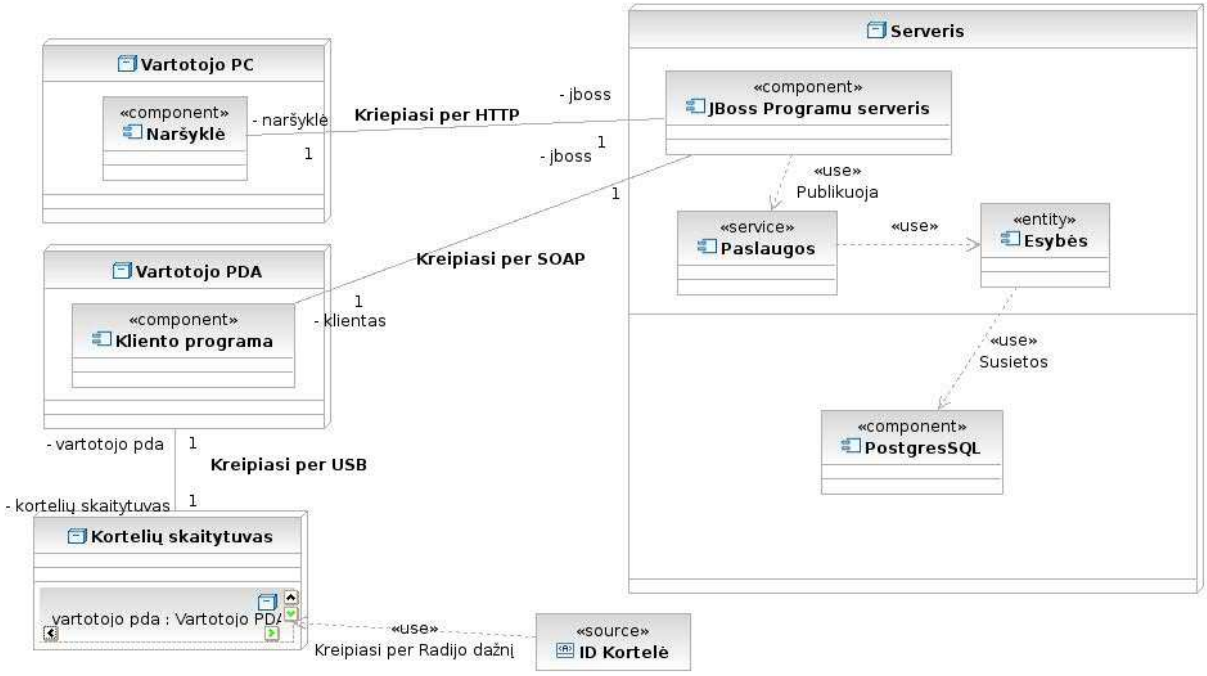
Kadangi integruojat sinchronizaciją į kliento aplikaciją tiesioginis kreipimasis į serverį nebūtinas, todėl pakeičiame sekų diagramą į pateiktą paveiksle 3.7. Pagrindinis skirtumas, kad klientas kreipiasi visuomet į lokalią duomenų bazę. O duomenų sinchronizacija atliekama kuomet klientas yra prisijungęs prie interneto. Taip pat galimas variantas, jog klientas gali dirbti ir su tiesioginiu serverio ryšiu kai yra prisijungęs prie interneto ir su lokaliais duomenimis, kuomet yra atsijungęs.



Paveikslas 3.7 Mokesčių peržiūros sekų diagrama po sinchronizacijos pakeitimų

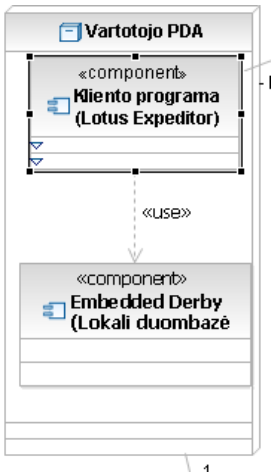
3.5 Išdėstymo diagramos pakeitimai

Vienas iš svarbiausių sistemos projektavimo dalių yra jos fizinis išdėstymas. Todėl išdėstymo diagrama nurodo aukšto lygio architektūrą iš kurios galima matyti ryšius tarp pagrindinių komponentų. Iš paveikslo 3.8 matome, kad norėdami įtraukti sinchronizacijos funkcionalumą į sistemą, būtina atlikti pakeitimus kliento pusėje.



Paveikslas 3.8 Sistemos išdėstymo diagrama

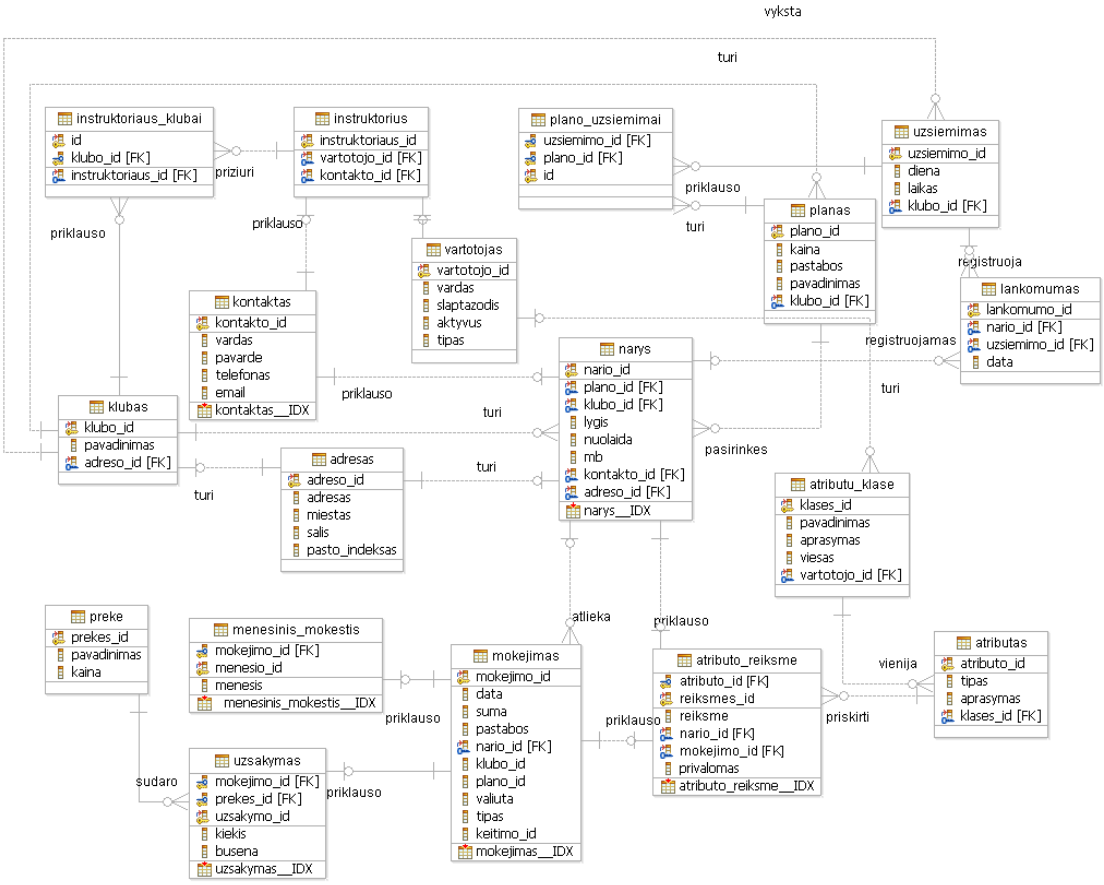
Kliento mobiliajame įrenginyje turime įdiegti lokalią duombazę, tam kad galėtume keisti ir saugoti duomenis kuomet nėra prieinamas interneto ryšys. Taigi aplikacija visuomet kreipiasi į lokalią duombazę, tačiau turi galimybę keisti duomenis ir tiesioginiu ryšiu su serveriu.



Paveikslas 3.9 Kliento pusės pakeitimai sinchronizacijai integruoti

3.6 Duomenų struktūrų pakeitimai

Paskutinė sistemos projektavimo dalis, kurios pakeitimus turime apibrėžti yra duomenų modelis esybių-ryšių diagramos pavidalu. Jame apibrėžiame visus duomenis, kurie nusako sistemos veiklos sritį ir yra naudojami jos funkcionalumui realizuoti. Paveiksle 3.10 pateikta pilnas esybių-ryšių diagrama.

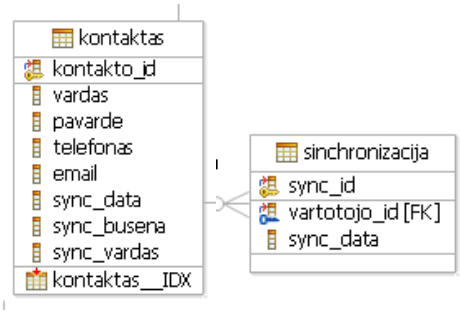


Paveikslas 3.10 Sistemos esybių-ryšių diagrama

Tam, kad duomenis galėtume sinchronizuoti būtina kiekvienai lentelei pridėti papildomus laukus:

- sync_data – laiko žymė kuomet duomenų lentelės įrašas buvo sinchronizuotas su serveriu. Pagal šį lauką nustatoma ar įrašą reikia sinchronizuoti.
- sync_busena – duomenų lentelės įrašo būsena gali būti: N – naujas, M – modifikuotas, D – pašalintas. Pagal tai galima atitinkamai spręsti, ką daryti su įrašu. Kaip pavyzdys jei būsena D, tuomet galima visus įrašus pašalinti iš kliento, tačiau serveryje gali būti laikoma, dėl duomenų istorijos kaupimo.
- sync_vardas – sinchronizuoto įrašo pakeitimo autoriaus vardas, t.y. vartotojo vardas, kuris yra atsakingas už paskutinį įrašo pakeitimą. Ši informacija reikalinga norint sekti įrašo pakeitimo autorius. Taip pat, iškilus duomenų konfliktų lengva išsiaiškinti, kurių vartotojų duomenys nesutampa.

Be šių laukų pridėjimo taip pat turime pridėti naują lentelę sekti duomenų sinchronizacijai tarp serverio ir kliento. Tokia lentelė reikalinga, norint žinoti, kurie vartotojai paskutinį kartą sinchronizavo savo duomenis, bei klientam suteikti galimybę atlikti greitą sinchronizaciją (tik duomenų, kurie naujesni už paskutinę sinchronizaciją). Pateiktas paveikslas 3.11 iliustruoja dvi lenteles su šiais pakeitimais.



Paveikslas 3.11 Sistemos esybių-ryšių sinchronizacijos duomenys

Sekančiame skyrelyje detaliau panagrinėsime mišraus duomenų sinchronizacijos modelį. Aprašysime pagrindinius parametrus, kurie turi įtaką sinchronizacijos kokybei ir veikimui. Taip pat trumpai aptarsime kokią paskirtį turi tokio modelio funkcionalumas realiame darbe.

3.7 Mišraus duomenų sinchronizacijos modelio analizė

Mūsų atveju nagrinėsime mišrios duomenų prieigos architektūrinį modelį, kadangi jis suteikia daugiausiai lankstumo ir privalumų realizuojant realioje sistemoje. Norėdami šį modelį suprasti pirmiausia turime aprašyti jo naudojimo pavyzdį bei problemas, su kuriomis susidursime.

Taigi sakykime įmonėje veikia centrinė duomenų saugykla, kurioje saugomi visi pagrindiniai duomenys reikalingi mobiliem darbuotojam. Kiekvienas darbuotojas turi mobilius įrenginius su kuriais jis gali prieiti prie centrinės duomenų bazės duomenų. Darbuotojai didžiąją laiko dalį praleidžia ne savo darbo vietoje. Kiekvienas darbuotojas norėdamas atsinaujinti duomenis turi dvi galimybes:

- Naudoti mobilųjį tinklą (internetą).
- Prisijungti prie bevielio tinklo perduodančio plačiajuostį internetą.

Abu variantai turi savų privalumų ir trūkumų. Mobilus tinklas yra gerai, kuomet darbuotojui esant bet kurioje vietoje reikia patikrinti naujų duomenų kiekį bei parsiųsti tam tikrą dalį naujų duomenų. Tačiau mobilaus tinklo kaštai yra kur kas didesni nei bevielio tinklo ir duomenų srauto sparta taipogi yra gerokai mažesnė lyginant su bevieliu tinklu. Bevielis tinklas yra gerai, kuomet darbuotojas turi atnaujinti didelį duomenų kiekį. Tačiau norint juo naudotis, būtina sugrįžti į darbovietę, kas kainuoja papildomo laiko ir gali būti nepriimtina jeigu darbuotojas turi dirbti nutolęs.

Šiom problemom spręsti naudosime mišrios duomenų prieigos modelį, kadangi jis aprašo tiek prisijungusio tiek atsijungusio kliento veikimą. Prieš nagrinėdami veikimą įsivesime pagrindinius parametrus nuo kurių priklausys duomenų sinchronizacijos priimtinumas pagal darbuotojo poreikius.

1. Atsako laikas (nuo 1 min) – laiko intervalas per kurį darbuotojas gauna kliente naujus duomenis.
2. Duomenų siuntimo/gavimo sparta (nuo 1kb/sek) – duomenų siuntimo ir gavimo sparta klientui prisijungus prie mobilaus ar bevielio tinklo.
3. Duomenų kiekis (nuo 1 mb) – duomenų kiekis centrinėje ir kliento duomenų bazėje.

4. Vidutinė duomenų kaita serveryje (nuo 1mb/h) – naujų duomenų kaita serveryje megabaitais per vieną valandą.
5. Naujų duomenų tinkamumo tikimybė kliento duomenų bazėje (nuo 1% per visa kaitą) – kiek procentų atnaujintų duomenų gali būti tinkama kliento duomenų bazės poaibiui.
6. Duomenų srauto kaina (Kaina 1 mb) – kokia vieno megabaito kaina esant mobiliam ar bevieliam tinklo srautui.
7. Kliento aplikacijos užklausos intervalas (nuo 1 min.) – laiko intervalas, tarp kurio kliento aplikacija automatiškai tikrina ar centrinėje duomenų bazėje atsirado naujų duomenų.
8. Klientų kiekis, atnaujinantis centrinę duomenų saugyklą – šis kiekis gali įtakoti, duomenų kaitos parametrą bei duomenų srauto gavimo spartą (sparta dalinama iš klientų kiekio).

Taigi sakykime mobilus darbuotojas kiekvieną dieną, darbo pradžioje, visada atsinaujina pilnai kliento duomenų bazę. Tuomet, kada jis yra nutolęs nuo darbovietės naudodamasis mobilų internetą gali patikrinti ar centrinėje duomenų bazėje yra atnaujintų duomenų reikalingų jam. Čia turime dvi galimybes:

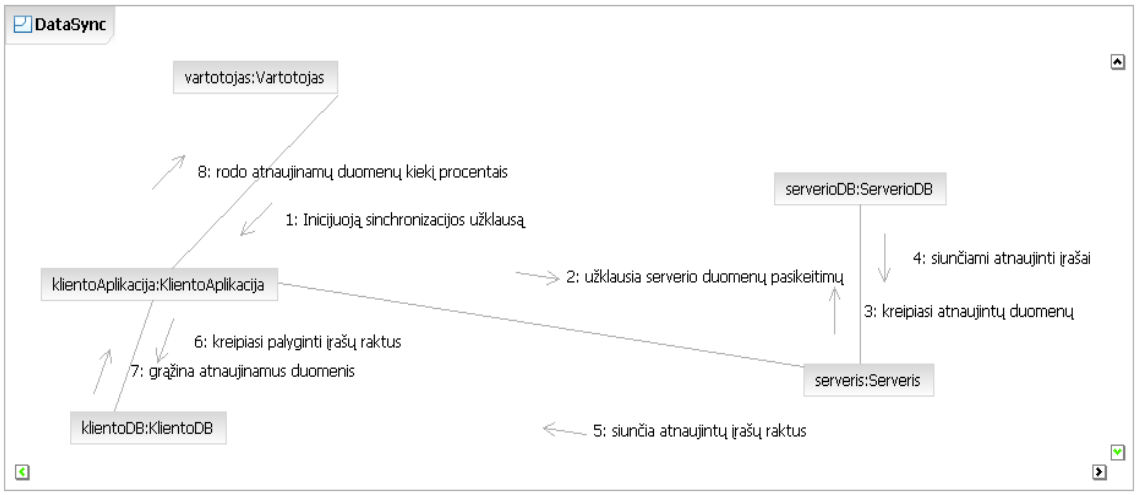
- Kliento aplikacija tam tikru intervalu užklausia serverio koks procentas naujų duomenų atsirado serveryje.
- Darbuotojas pats inicijuoja užklausą pagal, kurią gali daryti sprendimus ar reikalinga sinchronizacija.

Pirmuoju atveju kliento aplikacija gali pati inicijuoti duomenų atnaujinimą, kuomet yra atitinkamas naujų duomenų kiekis. Tai pagerintų aplikacijos atsako laiką, kadangi sumažėja darbuotojui tikimybė, kad reikės laukti kol bus parsiųsti duomenys. Tačiau tam aplikacija turi visada naudoti mobilų srautą ir gali padidinti mobilaus tinklo naudojimo kaštus.

Antruoju atveju darbuotojas gali pats daryti sprendimą ar verta atnaujinti duomenis ar verta palaukti kol jis vėl bus savo darbovietėje, kad galės atnaujinti visą kliento duomenų bazę. Toks veikimas mažiną atsako laiką (reikia laukti kol bus padaryta užklausa ir atsiųsti nauji duomenys).

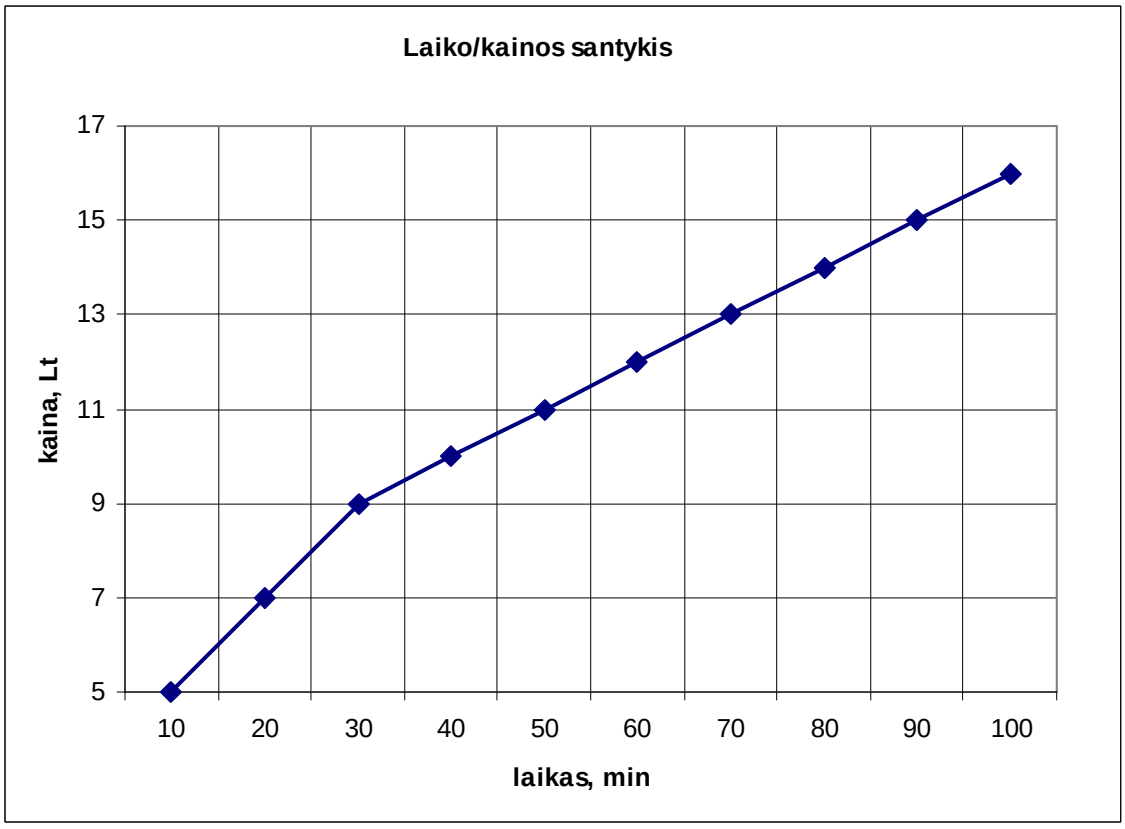
Pateiktame paveiksle 3.12 vaizduojami pagrindiniai elementai, kurie dalyvauja duomenų sinchronizacijos procese, bei jų tarpusavio komunikaciją. Šiuo atveju

procesas prasideda ir baigiasi vartotojo pusėje, kuomet jis inicijuoją duomenų sinchronizacija mobilaus įrenginio kliente.

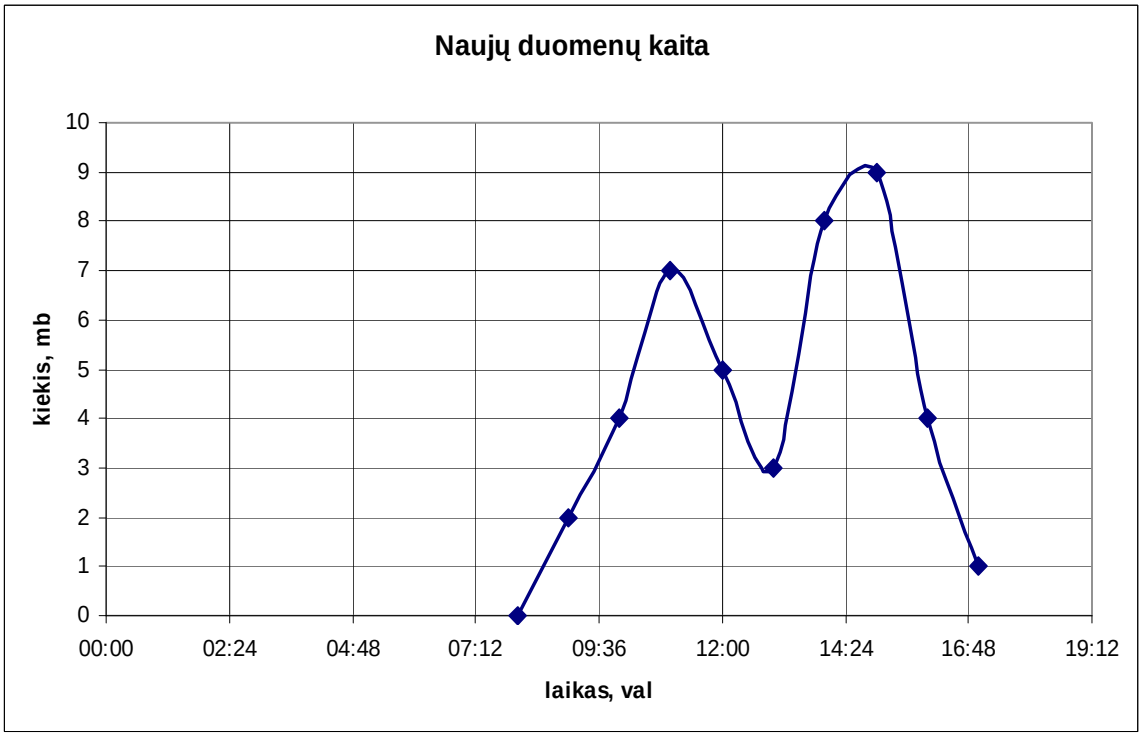


Paveikslas 3.12 Aukšto lygio komunikacijų diagrama.

Norint ištirti kaip šiame procese atsispindi pasirinkti parametrai galime naudoti paveiksle 3.13 pateiktus pavyzdinius grafikus.



Paveikslas 3.13 Kainos kitimas sinchronizuojant duomenis mobiliu tinklu.



Paveikslas 3.14 Duomenų kaita laiko atžvilgiu.

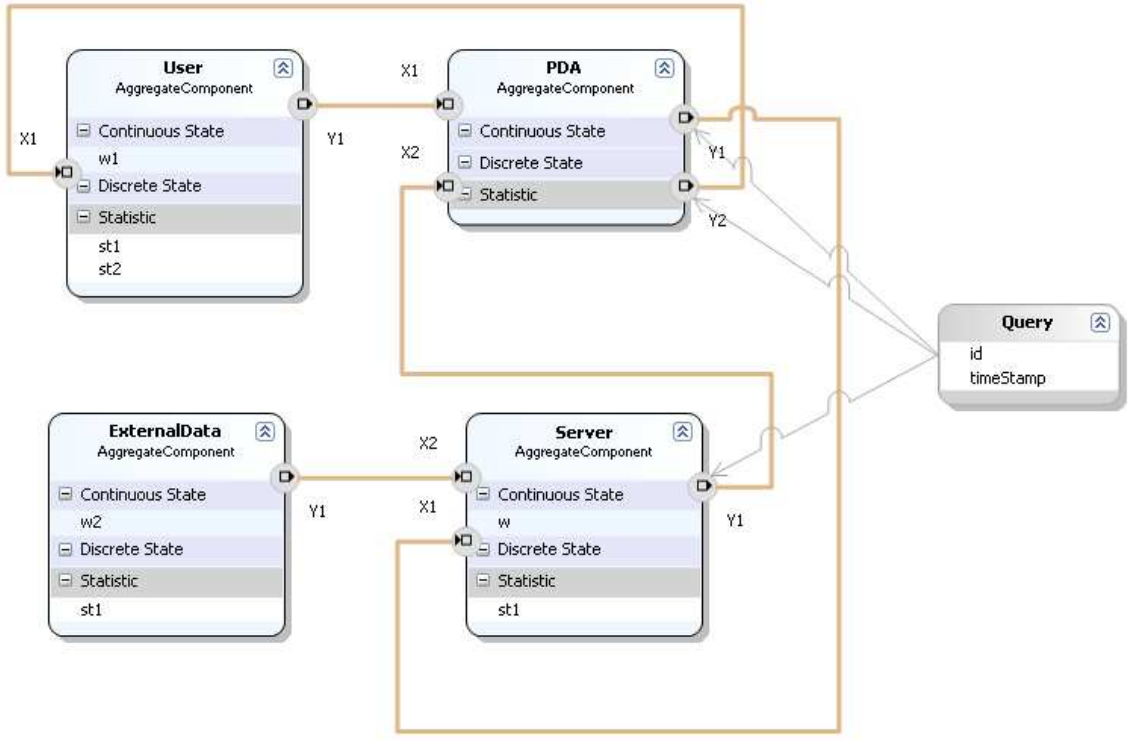
Taigi naudojantis šiuo modeliu galima tyrinėti kaip vienas parametras įtakoją kitą ar net keletą kitų parametrų. Kaip pavyzdys galėtų būti paros laikas, nuo kurio priklauso ne tik kiek duomenų pasikeičia centrinėje duomenų saugykloje, tačiau tai įtakoja ir duomenų srauto gavimą, kas tuo pačiu keičia ir kainą jei parsiunčiant duomenis naudojamas mobilusis tinklas.

4 TYRIMO DALIS

Tam kad išanalizuoti pasiūlytą duomenų sinchronizacijos algoritmų, jam buvo sukurtas sistemos imitacinis modelis. Modelio sudarymui panaudotas sudėtingų sistemų formalizavimas agregatiniu metodu. Agregatų funkcionavimo aprašymui buvo pritaikytas valdymo sekų metodas. [31]

Šio modelio agregatų sujungimo schema pateikta pav. 4.1.

Taigi šiame skyriuje aptarsime sudarytą duomenų sinchronizacijos algoritmo modelį, jo veikimo principą, parametrų įtakojančių atsako laiką (kuris yra svarbiausias mūsų tyrimo rezultatas), bei imitacinio modelio sudedamąsias dalis (agregatus) aprašysime matematinėmis išraiškomis.



Paveikslas 4.1 Sinchronizacijos sistemos imitacinio modelio agregatų sujungimo schema.

Parametrai, kuriuos nagrinėsime:

- ΔQ – siunčiamų naujų duomenų atsako laikas vartotojui.
- λ – kliento duomenų sinchronizacijos inicijavimo intervalas.
- τ – serverio duomenų atnaujinimo inicijavimo intervalas.

- μ – serverio vėlinimo laikas, kuomet siunčiami nauji duomenys klientui.

Iš paveikslo 4.1 galima pastebėti, kad modelį sudaro pagrindinės 4 dalys:

- Vartotojas (User) – tai duomenų sinchronizacijos kliente iniciatorius.
- Kliento aplikacija (PDA) – saugo kliento duombazės duomenis ir perduodą sinchronizacijos užklausą serveriui.
- Serveris (Server) – priima kliento užklausas ir išorinius duomenų atnaujinimus. Taip pat grąžina duomenis, kuriuos reikia sinchronizuoti kliente.
- Išoriniai duomenys (External Data) – siunčia pastovius atnaujinimus į serverį tam tikru intervalu (generuojama tikimybinis laikas, pagal apibrėžtas ribas).

Taip pat reiktų paminėti, jog duomenys siunčiami modelyje, kaip duomenų identifikatorius ir laiko žymė, tam kad būtų galima atpažinti, kurį duomenų lauką norime atnaujinti ir sulyginti jų naujumą.

Pagal šią diagramą naudojant Visual Studio 2008 paketą panaudojant modelių transformacijas [32] sugeneravome programos kodo skeletą su pagrindinėmis funkcijomis, kurias turime modifikuoti. Norėdami geriau suprasti kaip veikia modelis panagrinėsime kiekvienos dalies modifikacijas ir aprašysime jas pseudo kodu.

4.1 Vartotojo (User) agregato aprašmas

1. Įėjimo signalų aibė X = {x1_Query(id, timeStamp)}
   čia x1_Query – tai sinchronizacijos signalas su parametrais: id – siunčiamų duomenų masyvo identifikatorius, timeStamp – laiko žymė nurodanti duomenų sinchronizacijos datą ir laiką.
2. Išėjimo signalų aibė Y = {y1},
   čia y1 užklausa, kuri inicijuoja sinchronizacijos procesą.
3. Išorinių įvykių aibė E‘ = {e1’},
   čia e1‘ atėjo Query signalas.

4. Vidinių įvykių aibė E'' = {e1''},
   čia e1'' vartotojas baigė formuoti užklausą y1.
5. Valdymo sekos e1'' $\{\lambda_j\}$, j=1...D,
   čia j-tosios vartotojo paraiškos formavimo trukmė.
6. Diskrečiosios būsenos dedamoji ∅.
7. Tolydžioji būsenos dedamoji $z(t_m) = \{w(e1'', t_m)\}$,
   čia $w(e1'', t_m)$ laiko momentas kada baigsis vartotojo paraiškos formavimas.
8. Pradinė būsena $z(0) = \{\lambda_1\}$.
9. Perėjimo operatoriai H(e1'):
   $w(e1'', t_{m+1}) = t_m$ + lamda.NextExp(p_lamda);
   H(e1''):
   Y = y1;

4.2 Kliento aplikacijos (PDA) agregatas

1. Įėjimo signalų aibė X = {x1, x2_Query(id, timeStamp)},
   čia x1 užklausa, kuri inicijuoja sinchronizacijos procesą;
   x2_Query – tai sinchronizacijos signalas su parametrais: id – siunčiamų duomenų masyvo identifikatorius, timeStamp – laiko žymė nurodanti duomenų sinchronizacijos datą ir laiką.
2. Išėjimo signalų aibė Y = {y1_Query(id, timeStamp), y2_Query(id, timeStamp)},
   čia y1_Query – tai sinchronizacijos signalas su parametrais: id – siunčiamų duomenų masyvo identifikatorius, timeStamp – laiko žymė nurodanti duomenų sinchronizacijos datą ir laiką perduodamas kuomet vartotojas inicijuoja sinchronizacijos procesą;
   y2_Query – tai sinchronizacijos signalas su parametrais: id – siunčiamų duomenų masyvo identifikatorius, timeStamp – laiko žymė nurodanti duomenų sinchronizacijos datą ir laiką perduodamas kuomet gaunamas atsakymas iš serverio.
3. Išorinių įvykių aibė E' = {e1', e2'},
   čia e1' atėjo x1 užklausa;
   e2' atėjo x2_Query signalas.
4. Vidinių įvykių aibė ∅.
5. Valdymo sekos ∅.

6. Diskrečiosios būsenos dedamoji $\upsilon(t_m)$ ={{localArray$_i$}, i=1,N}
   čia localArray$_i$ masyvas su duomenų sinchronizacijos laiko žymėmis.
7. Tolydžioji būsenos dedamoji ∅.
8. Pradinė būsena localArray$_i$=0, i=1,N.
9. Perėjimo operatoriai H(e1’):
   Y = y1_Query(id, localArray$_{id}$).
   čia id atsitiktinis skaičius iš intervalo [1,N].
   H(e2’):

```
if (x2_Query.timeStamp > 0)
{
        localArray[x2_Query.id] = x2_Query.timeStamp;
}
Y = x2_Query;
```

4.3 Serverio (Server) agregatas

1. Įėjimo signalų aibė X = {x1_Query(id, timeStamp), x2 },
   čia x1_Query – tai sinchronizacijos signalas su parametrais: id – siunčiamų duomenų masyvo identifikatorius, timeStamp – laiko žymė nurodanti duomenų sinchronizacijos datą ir laiką, kuris gaunamas iš kliento aplikacijos norint palyginti duomenų įrašą su serveryje esančiu duomenų įrašu.
   x2 – tai sinchronizacijos inicijavimo signalas gaunamas iš išorės.
2. Išėjimo signalų aibė Y = {y1_Query(id, timeStamp)},
   čia y1_Query – tai sinchronizacijos signalas su parametrais: id – siunčiamų duomenų masyvo identifikatorius, timeStamp – laiko žymė nurodanti duomenų sinchronizacijos datą ir laiką perduodamas po duomenų sulyginimo.
3. Išorinių įvykių aibė E‘ = {e1’, e2’},
   čia e1‘ atėjo x1_Query signalas;
   e2’ atėjo x2 užklausa.
4. Vidinių įvykių aibė {e1’’},
   čia e1’’ serveris baigė formuoti duomenų sinchronizacijos atsakymo signalą y1_Query.
5. Valdymo sekos e1’’ {μ_j}, j=1...D,
   čia j-tosios serverio duomenų sinchronizacijos atsakymo signalo formavimo trukmė.

6. Diskrečiosios būsenos dedamoji $\upsilon(t_m)$ ={{centralArray$_i$}, i=1,N}

   čia centralArray$_i$ masyvas su duomenų sinchronizacijos laiko žymėmis.

7. Tolydžioji būsenos dedamoji $z(t_m) = \{w(e1'', t_m)\}$,

   čia $w(e1'', t_m)$ laiko momentas kada baigsis serverio duomenų sinchronizacijos atsakymo signalo formavimas.

8. Pradinė būsena $z(0) = \{\mu_1\}$ ir localArray$_i$=0, i=1,N.

9. Perėjimo operatoriai H(e1'):

```
if (x1_Query.timeStamp < centralArray[x1_Query.id])
{
        x1_Query.timeStamp = centralArray[x1_Query.id];
        w(e1'', tm+1) = tm + miu.NextExp(p_miu);
}
else
{
        x1_Query.timeStamp = 0;
        Y = x1_Query;
}
```

   H(e2'):

   centralArray$_{id}$ = tm;

   čia id atsitiktinis skaičius iš intervalo [1,N].

   H(e1''):

   Y = y1_Query(id, centralArray[id]);

4.4 Išoriniai duomenys (External Data)

1. Įėjimo signalų aibė ∅.
2. Išėjimo signalų aibė Y = {y1},

   čia y1 užklausa, kuri inicijuoja sinchronizacijos procesą serveryje.

3. Išorinių įvykių aibė ∅.
4. Vidinių įvykių aibė E'' = {e1''},

   čia e1'' išoriniai duomenys baigė siųsti duomenų sinchronizacijos inicijavimo signalą y1.

5. Valdymo sekos e1'' $\{\tau_j\}$,

   čia j-tosios išorinių duomenų sinchronizacijos paraiškos formavimo trukmė.

6. Diskrečiosios būsenos dedamoji ∅.
7. Tolydžioji būsenos dedamoji $z(t_m) = \{w(e1'', t_m)\}$,

čia w(e1'', t_m) laiko momentas kada baigsis išorinių duomenų sinchronizacijos paraiškos formavimas.

8. Pradinė būsena z(0) = {τ_1}.
9. Perėjimo operatoriai H(e1''):

   Y = y1;

   w(e1'', t_{m+1}) = t_m + tau.NextExp(p_tau);

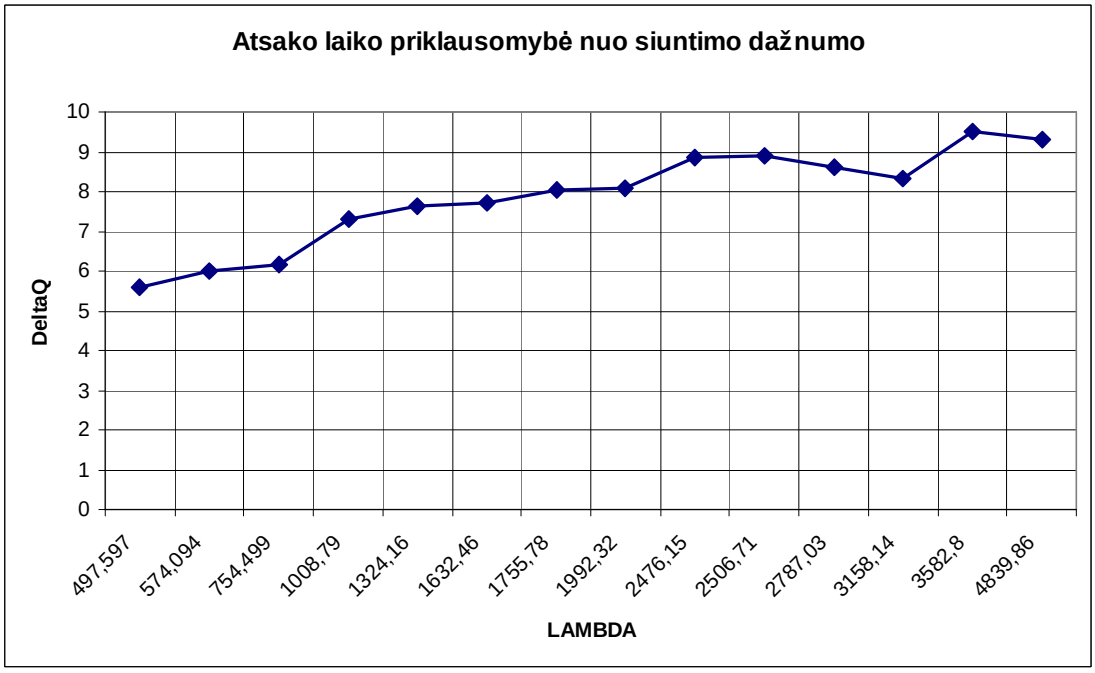
Pagal agregatų aprašymus buvo sugeneruotas programinis sistemos modelis su kuriuo buvo atliekami eksperimentai. Sekančiame skyrelyje pateikti imitacinių eksperimentų rezultatai grafikų pavidalu.

5 EKSPERIMENTINĖ DALIS

Šiame skyriuje trumpai aprašysime, kokius eksperimentus atliksime su sugeneruotu modeliu. Kiekvieną iš jų įvertinsime atitinkamais grafikais. Pagrindinis eksperimento tikslas įvertinti atsako laiką (žymėsim ΔQ arba DeltaQ), ir jo pokyčio priklausomybę nuo modelio parametrų. Taigi tyrimai susidės iš:

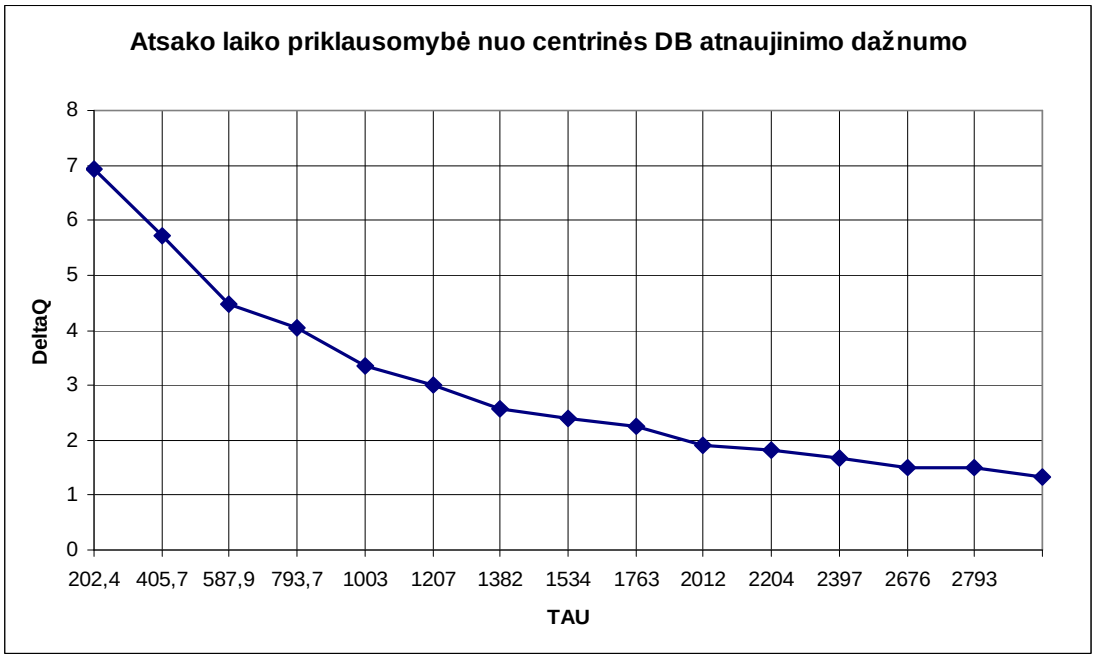
- ΔQ priklausomybės nuo λ (kliento duomenų sinchronizacijos inicijavimo intervalo).
- ΔQ priklausomybės nuo τ (serverio duomenų sinchronizacijos inicijavimo intervalo).
- ΔQ priklausomybės nuo μ (serverio vėlinimo laiko, kuomet siunčiami nauji duomenys klientui).

Pirmajame grafike matome, ΔQ priklausomybę nuo λ (kliento duomenų sinchronizacijos inicijavimo intervalo). Matome, atsako laikas linkęs didėti, kuomet intervalas didėja. Tai reiškia, kuo dažniau vartotojas sinchronizuoja duomenis tuo ilgiau jam reikia laukti, kol jį pasieks nauji duomenys iš serverio.



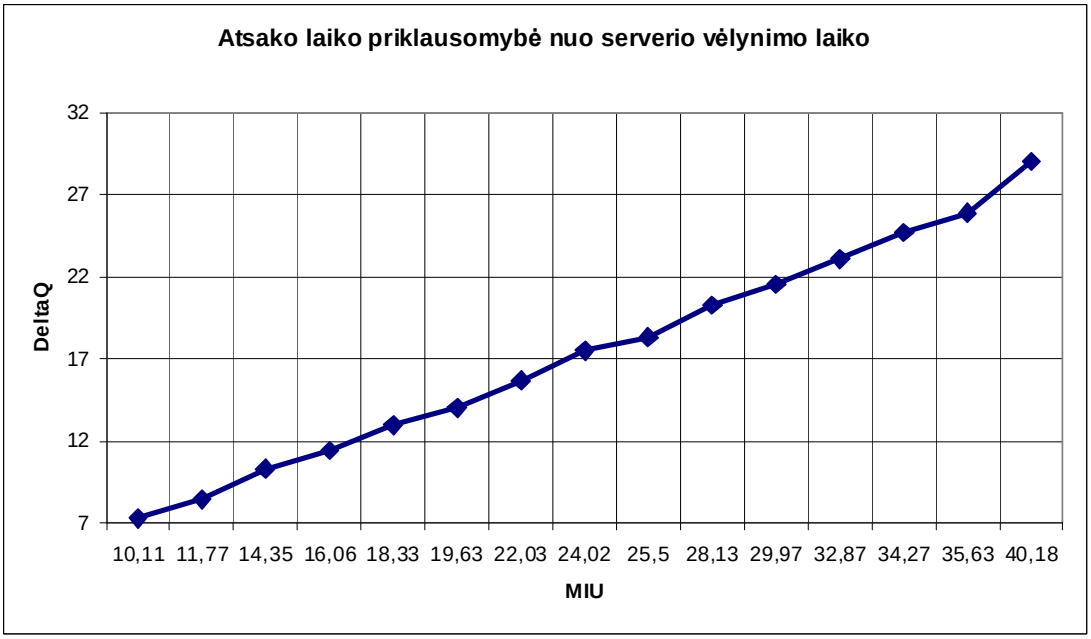
Paveikslas 5.1 ΔQ priklausomybės nuo λ (kliento duomenų sinchronizacijos inicijavimo intervalo).

Antrame grafike matome, jog ΔQ linkęs mažėti kai mažiname τ. Tai parodo, kuo rečiau sinchronizuojami serverio duomenis, tuo trumpesnis atsako laikas. Iš grafiko matome, kad ΔQ artėja į 0, kuomet τ intervalas didinamas. Vadinasi jeigu serverio duomenys atnaujinami retai, tada kliento aplikacijoje didžioji dalis duomenų yra visuomet atnaujinta.



Paveikslas 5.2 ΔQ priklausomybės nuo τ (serverio duomenų atnaujinimo inicijavimo intervalo).

Paskutiniame grafike matome, ΔQ linkęs tiesiogiai didėti kai didėja μ. Tai rodo, jog atsako laikas yra tiesiogiai proporcingas serverio duomenų siuntimo vėlinimui. Taigi norint gauti priimtiną atsako laiką, serverio vėlinimas turėtų būti kuo mažesnis.



Paveikslas 5.3 ΔQ priklausomybės nuo μ (serverio vėlinimo laiko, kuomet siunčiami nauji duomenys klientui).

6 IŠVADOS

- Duomenų sinchronizacija mobiliųjų įrenginių aplikacijose suteikia nemažai privalumų, lyginant su aplikacijomis, kurios neturi šio funkcionalumo. Pagrindiniai yra: vartotojo nepriklausomybė nuo tinklo jungties, geresnis atsako laikas ir aplikacijos sparta bei duomenys yra dažniau atnaujinami.
- Sukurtas mišrios duomenų prieigos architektūros imitacinis modelis, kuris simuliuoja mūsų apibrėžtos sistemos darbą. Naudodami šį modelį galėjome apskaičiuoti atsako laiko (ΔQ) priklausomybę nuo kliento duomenų sinchronizacijos inicijavimo intervalo (λ), serverio duomenų atnaujinimo intervalo (τ) ir sinchronizacijos vėlinimo laiko (μ).
- Pagal gautus modelio rezultatų grafikus matome, jog atsako laikas didėja kuo rečiau sinchronizuojame duomenis (reikalinga daugiau laiko parsiųsti didesniam duomenų kiekiui). Jeigu serveryje duomenys atnaujinami retai, tuomet atsako laikas kliente artėja prie nulio, tai reiškia esant retam atnaujinimui į serverį nereikia dažnai kreiptis. Paskutinis grafikas parodo tiesioginę atsako laiko priklausomybę nuo serverio vėlinimo, t.y. esant dideliam serverio vėlinimui, kliento atsako laikas bus taipogi didelis.

7 LITERATŪRA

[1] MILIAUSKAS, E. *Klubo valdymo sistema:* magistrinis projektas (Vartotojo dokumentacija). KTU Informatikos fakultetas. [Kaunas], 2009. 122 p.;

[2] MEDNEY, H. Building an offline application in IBM Lotus Expeditor. 2007, [Žiūrėta 2009-05-06]. Prieiga per internetą: <http://www.ibm.com/developerworks/lotus/library/expeditor-offline/>;

[3] MILIAUSKAS, E. *Klubo valdymo sistema:* magistrinis projektas (Architektūros specifikacija). KTU Informatikos fakultetas. [Kaunas], 2008. 54 p.;

[4] Funambol, *Sync4J Server ds Developer guide*. 2005, [Žiūrėta 2009-04-06]. Prieiga per internetą: <http://download.forge.objectweb.org/sync4j/sync4j_server_ds_developer_guide.pdf>;

[5] OMA, *SyncML Sync Protocol, version 1.1*, [Žiūrėta 2009-05-06]. Prieiga per internetą: <http://www.openmobilealliance.org/tech/affiliates/syncml/syncml_represent_v11_20020215.pdf>;

[6] OMA, SyncML Device Management Security 1.1, [Žiūrėta 2009-05-06]. Prieiga per internetą: <http://www.openmobilealliance.org/tech/affiliates/syncml/syncml_dm_sec_v11_20020215.pdf>;

[7] Valentina Dagienė, Gintautas Grigas, Tatjana Jevsikova, *Enciklopedinis kompiuterijos žodynas* [interaktyvus]. [Vilnius]: VU Matematikos ir informatikos institutas, 2008 [Žiūrėta 2009-05-16]. Prieiga per internetą: <http://www.likit.lt/term/enc.html>;

[8] Rodriguez, J. R., et al. *Building Composite Applications in Lotus Expeditor V6.1*. IBM Redbooks, 2007. 118 p. [Žiūrėta 2009-05-16]. Prieiga per internetą: <http://www.redbooks.ibm.com/redpapers/pdfs/redp4241.pdf>;

[9] FusionOne, *Corporate Overview*, [Žiūrėta 2009-05-16]. Prieiga per internetą: <http://www.fusionone.com/docs/FusionOne_Corporate_Overview.pdf>;

[10] FusionOne, *Handset Transfer Solution*, [Žiūrėta 2009-05-16]. Prieiga per internetą: <http://www.fusionone.com/docs/Handset_Transfer_Solution.pdf>;

[11] HP Mobile Management Center, *Device Management Functional Whitepaper*, [Žiūrėta 2009-05-16]. Prieiga per internetą: <https://h30406.www3.hp.com/campaigns/2008/promo/1-4T4KV/images/HP-MMC-4.5-Functional-Whitepaper-final-2-12-09.pdf>;

[12] Sang-Wook, K. *Synchronization in an Embedded DBMS Environment*. Division of Information and Communications, Hanyang University, Korėja 2006 July. 6 p. [Žiūrėta 2009-05-16]. Prieiga per internetą: <paper.ijcsns.org/07_book/200607/200607A06.pdf>;

[13] Norman H. Cohen, *A Java Framework for Mobile Data Synchronization*: tarptautinės konferencijos pranešimų medžiaga. IBM Thomas J. Watson Research Center, 2000. 12 p. [Žiūrėta 2009-05-16]. Prieiga per internetą: <http://www.haifa.il.ibm.com/coopis2000.html>;

[14] Bellavista P., ir Corradi, A. *The Handbook of Mobile Middleware*. JAV, 2007. 1410 p. ISBN 9780849338335;

[15] Kathuria M., et al. *Designing and Implementing Java Databases for Mobile Computing*. JAV, 2003;

[16] Mallick, M. *Mobile and Wireless Design Essentials*. JAV, 2003. 454 p. ISBN 9780471214199;

[17] B., Reza *Mobile Computing Principles: Designing and Developing Mobile Applications with UML and XML*. UK, 2005. 878 p. ISBN 9780521817332;

[18] H., Mikael *Wireless XML Developer's Guide*. JAV, 2002. 525 p. ISBN 9780072195361

[19] Microsoft, *Windows Mobile Official Site*. [Žiūrėta 2009-05-18]. Prieiga per internetą: <http://www.microsoft.com/windowsmobile/en-us/default.mspx>;

[20] CreativeCorp, *Pocket PC Application development projects*. [Žiūrėta 2009-05-18]. Prieiga per internetą: <http://www.pocketpccreations.com/projects.htm>;

[21] Symbian Foundation, *Symbian OS Community Offical Site*. [Žiūrėta 2009-05-18]. Prieiga per internetą: <http://www.symbian.org>;

[22] Symbian Foundation, *Symbian Developer network*. [Žiūrėta 2009-05-18]. Prieiga per internetą: <http://developer.symbian.com>;

[23] Sun Microsystems, *Java ME Official Site*. [Žiūrėta 2009-05-18]. Prieiga per internetą: <http://java.sun.com/j2me>;

[24] Sun Microsystems, *Mobile Information Device Profile (MIDP); JSR 118 Official Site*.
[Žiūrėta 2009-05-18]. Prieiga per internetą:
<http://java.sun.com/products/midp>;

[25] Sun Microsystems, *Connected Limited Device Configuration (CLDC); JSR 139 Official Site*.
[Žiūrėta 2009-05-18]. Prieiga per internetą:
<http://java.sun.com/products/cldc>;

[26] Mobile Linux Community, *Mobile Linux Information and News site*.
[Žiūrėta 2009-05-18]. Prieiga per internetą:
<http://www.mobilelinuxinfo.com>

[27] Palm, *Palm Official Site*.
[Žiūrėta 2009-05-18]. Prieiga per internetą:
<http://www.palm.com>;

[28] Apple, *iPhone developer site*.
[Žiūrėta 2009-05-19]. Prieiga per internetą:
<http://developer.apple.com/iphone/>;

[29] Google, *Android Official site*.
[Žiūrėta 2009-05-19]. Prieiga per internetą:
<http://www.android.com/about/>;

[30] RIM, *BlackBerry developer site*.
[Žiūrėta 2009-05-19]. Prieiga per internetą:
<http://na.blackberry.com/eng/developers/javaappdev/javadevenv.jsp>;

[31] Pranevičius, H. *Sudėtingų sistemų formalizavimas ir analizė*. Kaunas, 2008. ISBN 978-9955-591-51-1

[32] Štuikys, V.; Damaševičius, E. *Modelių ir programų abstrakčiosios transformacijos*. Kaunas, 2008. ISBN 978-9955-591-50-4

[33] Sun Microsystems, *JDBC overview*. [Žiūrėta 2009-05-19]. Prieiga per internetą: <http://java.sun.com/products/jdbc/overview.html>;

[34] OMG, *UML official site*. [Žiūrėta 2009-05-19]. Prieiga per internetą: <http://www.uml.org>;

[35] Microsoft, *ODBC overview*. [Žiūrėta 2009-05-19]. Prieiga per internetą: <http://support.microsoft.com/kb/110093>.

8 TERMINŲ IR SANTRUMPŲ ŽODYNAS

API (Application programmable interface) – Susitarimų ir procedūrų rinkinys ryšiams tarp atskirų programų ir operacinės sistemos realizuoti.
Yra komponentinių ir modulinių sistemų projektavimo ir veikimo pagrindas.
Operacinės sistemos turi programų sąsajas, kurias programuotojai gali naudoti bendroms funkcijoms (pvz., duomenų įvedimui ir išvedimui, failų tvarkymui) vykdyti. Pavyzdžiui, „Windows“ operacinės sistemos programų sąsaja gali pasinaudoti operacinės sistemos standartiniais grafinės sąsajos elementais (išskleidžiamaisiais sąrašais, slankikliais, dialogo langais ir kt.).
Programų sąsaja panaši į protokolą tuo, kad apibrėžia komponentų sąveiką. Tačiau programų sąsajos sąvoka dažniau vartojama tada, kai kalbama apie tame pačiame kompiuteryje vykstančią sąveiką, o protokolo sąvoka – kalbant apie kompiuterių tinklus. [7]

CLDC (Connected Limited Device Configuration) – aprašo pagrindinį API ir virtualę mašiną įrenginiam su ribotais resursais, kaip mobilieji telefonai, peigeriai, ir delniniai kompiuteriai. [25]

Duombazė – duomenų rinkinys, susistemintas ir sutvarkytas taip, kad juo būtų galima patogiai naudotis. Duomenys gali būti įvairūs: tekstai, paveikslai, garsai. Juos tvarko duomenų bazių valdymo sistema. Santrumpa DB. [7]

Įrašas – Duomenų struktūra, sudaryta iš duomenų laukų. Pavyzdžiui, duomenų bazės įrašą sudaro duomenys, apibūdinantys tam tikrą vieną objektą. Sąryšinės duomenų bazės įrašą atitinka lentelės eilutė. Laukų duomenys gali būti įvairių tipų. [7]

JDBC (Java Database Connectivity) – tai specialus API, kuris jau yra tapęs industriniu standartu Java kalbos sujungimui su įvairaus tipo duombazėmis. JDBC API suteikia iškvietimo lygio API, kuri paremta SQL kreipiniais į duombazę. JDBC technologija leidžia naudotis principu, kuomet programa parašoma vieną karta, tačiau veikia visur. [33]

MIDP (Mobile Information Device Profile) – leidžia kurti parsiunčiamas aplikacijas ir paslaugas, mobiliem įrenginiam, kurie turi galimybę prisijungti prie tinklo. [24]

MVC (Model-View-Controller) – tai objektiškai orientuotas projektavimo šablonas, kurio paskirtis atskirti aplikacijos, kurios priima vartotojų duomenis, dalis atsakingas už skirtingą funkcionalumą. [17]

ODBC (Open Database Connectivity) – tai kompanijos Microsoft strateginė sąsaja, kuri naudojama duomenų prieigai skirtingose aplinkose prie reliacinių ar ne reliacinių duomenų bazių valdymo sistemų. [35]

OS (Operating system) – Programų komplektas, valdantis kompiuterio, prie jo prijungtų įtaisų klaviatūros, spausdintuvo ir kt.) ir jame esančių programų darbą.

Vykdo kompiuteriui pateiktas užduotis, paleisdama joms reikalingas programas, skirsto joms kompiuterio išteklius (atmintį, procesoriaus laiką ir kt.), tvarko duomenų persiuntimą tarp kompiuterio įrenginių. Tai terpė, kurioje veikia visos kitos, į ją įdiegtos, programos.

Vienu metu gali vykdyti daugelį užduočių. Atlieka sąsajos tarp kompiuterio ir jo naudotojo vaidmenį. Yra operacinių sistemų, aptarnaujančių vieną arba daugelį naudotojų.

Pirmųjų operacinių sistemų sąsajos buvo tekstinės, vėliau paplito naudojimui patogesnės grafinės. Populiariausios asmeninių kompiuterių grafinės operacinės sistemos yra „Windows“ (pirmoji iš dalies sulietuvinta versija „Windows XP“, 2002 m.), „Linux“ (lietuvinti pradėta 1999? m.), „Mac OS“, mobiliųjų įrenginių (kišeninių kompiuterių, delninukų) – „Palm“, „Windows CE“, kompiuterių tinklų – „Unix“, „Linux“. [7]

PDA (Personal digital assistant) – Mažas, į delną telpantis įtaisas, atliekantis ribotų galimybių kompiuterio, mobiliojo telefono, ryšio su internetu ir kitas mobiliems skaitmeniniams įtaisams būdingas funkcijas.

Turi mažą klaviatūrą (arba visai jos neturi). Dažniausiai neturi diskų skaitymo įtaisų. Programos ir duomenys laikomi vidinėje atmintyje. Paprastai turi lizdą išorinei laikmenai įdėti. [7]

SQL (Sequel Query Language) – Sąryšinių duomenų bazių užklausų kalba. Užklausos pavyzdys: SELECT VARDAS, PAVARDĖ FROM LENTELĖ WHERE AMŽIUS < 35

Ši užklausa išrenka iš duomenų bazės lentelės LENTELĖ vardus ir pavardes (duomenų bazės laukus) visų asmenų (duomenų bazės įrašų), jaunesnių kaip 35 metų. [7]

UML (Unified Modeling Language) – yra OMG organizacijos labiausiai naudojama specifikacija, kuri naudojama plačiai pasaulyje ne tik aplikacijų struktūrų, veikimo ir architektūrų modeliavimui, bet ir verslo procesų ir duomenų struktūrų kūrimui. [34]

9 PRIEDAI

9.1 Modelio H operatorių programos kodas

Vartotojo (User) agregato programos kodas

```
public partial class CUser : Aggregate
    {
        private double p_lamda;
        void Init()
        {
            p_lamda = Program.f1.Lamda;
            double t1 = lamda.NextExp(p_lamda);
            st1.Note(t1);
            w1.CreateInternalEvent(t1);
        }
        void H_X1_Query(AgEventArg av, double tm)
        {
            double t1 = lamda.NextExp(p_lamda);
            st1.Note(t1);
            w1.CreateInternalEvent(tm + t1);
            st2.Note(av.DeltaS);
        }
        void H_w1(AgEventArg av, double tm)
        {
            Y1.Output();
        }
    }
```

Kliento aplikacijos (PDA) agregato programos kodas

```
public partial class CPDA : Aggregate
   {
        private double[] localArray = new double[100];
        private ControlSequence p1 = new ControlSequence();
        void Init()
        {
            for (int i = 0; i < 100; i++)
            {
                localArray[i] = 0.0D;
            }
        }
        void H_X1(AgEventArg av, double tm)
        {
            int id = p1.Next(0, 99);
            Y1_Query.Output(new Query(id, localArray[id]));
        }
        void H_X2_Query(Query av, double tm)
        {
            if (av.timeStamp > 0)
            {
                localArray[av.id] = av.timeStamp;
            }
            Y2_Query.Output(av);

        }
   }
```

Serverio (Server) agregato programos kodas

```
public partial class CServer : Aggregate
{
    private double[] centralArray = new double[100];
    private double p_miu;
    private ControlSequence p1 = new ControlSequence();

    void Init()
    {
        p_miu = Program.f1.Miu;
        for (int i = 0; i < 100; i++)
        {
            centralArray[i] = 0.0D;
        }
    }
    void H_X1_Query(Query av, double tm)
    {
        if (av.timeStamp < centralArray[av.id])
        {
            double t1 = miu.NextExp(p_miu);
            st1.Note(t1);
            av.timeStamp = centralArray[av.id];
            w.CreateInternalEvent(tm + t1, av);
        }
        else
        {
            av.timeStamp = 0.0D;
            Y1_Query.Output(av);
        }
    }
    void H_X2(AgEventArg av, double tm)
    {
        int id = p1.Next(0, 99);
        centralArray[id] = tm;
    }
    void H_w(AgEventArg av, double tm)
    {
        Y1_Query.Output(av);
    }
}
```

Išorinių duomenų (ExternalData) agregato programos kodas

```
public partial class CExternalData : Aggregate

{
    private double p_tau;
    void Init()
    {
        p_tau = Program.f1.Tau;
        double t1 = tau.NextExp(p_tau);
        st1.Note(t1);
        w2.CreateInternalEvent(t1);
    }
    void H_w2(AgEventArg av, double tm)
    {
        Y1.Output();
        double t1 = tau.NextExp(p_tau);
        st1.Note(t1);
        w2.CreateInternalEvent(t1 + tm);
    }
}
```

9.2 Modelio transformuoti rezultatai

	Number of Observation	Min Value	Max Value	Mean Value	Standart Deviation
SimApplication.CUser					
Lamda	1998	0,2839256	3129,325	496,5471	478,0041
DeltaQ	1997	0	50,17088	4,109186	5,043643
SimApplication.CServer					
MIU	1646	0,001620894	50,17088	4,985445	5,14733
SimApplication.CExternalData					
TAU	9960	0,0008139792	1304,122	100,4111	101,2444

	Number of Observation	Min Value	Max Value	Mean Value	Standart Deviation
SimApplication.CUser					
Lamda	1927	0,1870907	4222,588	515,4365	526,2642
DeltaQ	1926	0	42,18444	3,555456	4,804166
SimApplication.CServer					
MIU	1377	0,006042604	42,18444	4,972991	5,023347
SimApplication.CExternalData					
TAU	4917	0,0106847	1599,159	203,4092	200,4494

	Number of Observation	Min Value	Max Value	Mean Value	Standart Deviation
SimApplication.CUser					
Lamda	1725	1,349494	5786,449	574,0935	589,7279
DeltaQ	1724	0	59,50039	6,001887	9,091528
SimApplication.CServer					
MIU	983	0,009019856	59,50039	10,5262	9,866825
SimApplication.CExternalData					
TAU	2532	0,01535408	2524,81	395,0143	387,7747

	Number of Observation	Min Value	Max Value	Mean Value	Standart Deviation
SimApplication.CUser					
Lamda	1315	0,3479939	6206,568	754,4988	787,8416
DeltaQ	1314	0	67,36604	6,154566	9,098334
SimApplication.CServer					
MIU	822	0,0164063	67,36604	9,83832	9,803107
SimApplication.CExternalData					
TAU	2535	0,01691038	3334,852	395,1544	391,1474

	Number of Observation	Min Value	Max Value	Mean Value	Standart Deviation
SimApplication.CUser					
Lamda	987	0,9502197	6926,36	1008,793	1001,277
DeltaQ	986	0	82,86386	7,303336	10,52056
SimApplication.CServer					
MIU	684	0,02150173	82,86386	10,52791	11,20823
SimApplication.CExternalData					
TAU	2535	0,0707266	3087,812	394,5139	390,3328

	Number of Observation	Min Value	Max Value	Mean Value	Standart Deviation
SimApplication.CUser					
Lamda	610	6,282297	10411,07	1632,458	1660,163
DeltaQ	609	0	71,02793	7,732476	10,62086
SimApplication.CServer					
MIU	460	0,008595288	71,02793	10,23713	11,12315
SimApplication.CExternalData					
TAU	2460	0,158342	3774,811	407,2454	423,0738

	Number of Observation	Min Value	Max Value	Mean Value	Standart Deviation
SimApplication.CUser					
Lamda	569	4,026166	8938,412	1755,78	1615,678
DeltaQ	568	0	60,24934	8,045925	9,857721
SimApplication.CServer					
MIU	440	0,01570697	60,24934	10,38656	10,05692
SimApplication.CExternalData					
TAU	2470	0,03818195	3373,342	404,9143	406,6388

	Number of Observation	Min Value	Max Value	Mean Value	Standart Deviation
SimApplication.CUser					
Lamda	501	3,280667	25006,23	1992,316	2155,997
DeltaQ	500	0	59,42829	8,07268	9,364568
SimApplication.CServer					
MIU	398	0,0251697	59,42829	10,14156	9,444145
SimApplication.CExternalData					
TAU	2495	0,3695179	3797,126	400,9177	391,6829

	Number of Observation	Min Value	Max Value	Mean Value	Standart Deviation
SimApplication.CUser					
Lamda	403	1,149445	17389,26	2476,146	2478,208
DeltaQ	402	0	58,53399	8,876224	9,884092
SimApplication.CServer					
MIU	337	0,004952999	58,53399	10,58825	9,920347
SimApplication.CExternalData					
TAU	2486	0,05781454	3063,979	402,4474	400,3344

	Number of Observation	Min Value	Max Value	Mean Value	Standart Deviation
SimApplication.CUser					
Lamda	405	11,76078	17331,36	2477,177	2569,739
DeltaQ	404	0	50,07813	8,309871	9,248788
SimApplication.CServer					
MIU	345	0,04397257	50,07813	9,730979	9,292051
SimApplication.CExternalData					
TAU	2626	0,1048736	2841,051	380,8263	365,3697