Latvijas Universitāte
Fizikas un matemātikas fakultāte
Datorikas nodaļa

JAVAS UN MOBILO IERĪČU SADARBĪBAS ŠABLONI

Maģistra darbs

Autors:
Deniss Garavņovs

Vadītājs:
Dr. dat. Kalvis Apsītis, LU

Rīga, 2006

Anotācija

Šajā darbā tiek apskatītas lietojumprogrammas, kurās ietilpst gan Java 2 Enterprise Edition (J2EE) serveris, gan arī Java 2 Micro Edition (J2ME) lietojumi. Aprakstīti divu iepriekš minētu platformu sadarbības šabloni, kuri var atvieglot programmatūras izstrādi, veicināt koda atkalizmantojamību un standartizēt komunikāciju starp Javu un dažādiem mobilo ierīču tipiem. Darbā tiek apskatīti programmatūras paraugi, kas ilustrē divu platformas mijiedarbības iespējas un potenciālus risinājumus. Kā arī tiek piedāvātas dizaina un implementēšanas vadlīnijas, kuras var palīdzēt optimizēt esošo kodu, un novērst potenciālās un iespējamās kļūdas izstrādes laikā.

Darba praktiskajā daļā ir aprakstīts J2ME lietojums un tā mijiedarbība ar J2EE aplikācijas serveri. Lietojumā tiek realizēti divi darba aprakstīti šabloni.

Šis darbs var tikt izmantots, kā informācijas avots projektējot un izstrādājot bezvada lietojumus mobilajiem ierīcēm, kuru mērķis ir efektīvāk strādāt ar J2EE aplikāciju serveriem, izmantojot darbā piedāvātos šablonus, paraugus un vadlīnijas.

Darbā ir 40 attēli, 8 pielikumi un 15 informācijas avoti.

Annotation

There are considered programme applications in this work, where used as Java 2 Eneterprise Edition (J2EE) server, as Java 2 Micro Edition (J2ME) applications. There are also considered interaction patterns early mentioned two platforms, which can relieve application development process, facilitate source code reusing, and standardize communication between Java and different mobile device types. There are proposed application examples, which provide possibilities and two platform interaction potential solutions. There are suggested design and implementation guidelines, which can help the existing source code and prevent possible errors in the development period of time.

In the practical part of the work there is described J2ME application and its interaction with J2EE application server. There are applied two patterns in this application.

This paper can be used as information source for mobile devices wireless applications design and development, which have the target of effective work with J2EE application servers, using suggested in this paper patterns, examples un guidelines.

Course paper contains 40 illustrations and 8 appendices. Bibliography includes 15 sources.

Анотация

В этой работе рассмотрены програмные приложения, в которых задействованы как Java 2 Enterprise Edition (J2EE) сервер, так и Java 2 Micro Edition (J2ME) приложения. Рассмотренны шаблоны взаимодействия, ранее упомянутых двух платформ, которые могут облегчить разработку програмного обеспечения, способствовать повторному использованию кода и стандартизировать коммуникацию между Java и разными типами мобильных устройств. В работе рассмотрены примеры приложений, которые иллюстрируют возможности и потенциальные решения взаимодействия двух платформ. Также предложены советы по дизайну и реализации, которые могут помочь оптимизировать существующий код, и предотвратить возможние ошибки во время разработки.

В практической части работы описано J2ME приложение и его взаимодействие с J2EE сервером приложений. В приложении реализованы два рассмотренных в работе шаблона.

Эта работа может быть использована как источник информации для проектирования и разработки беспроводных приложений для мобильных устройств, у которых цель эфективнее работать с J2EE серверами приложений, используя предложенные в работе шаблоны, примеры и советы.

В работе: 40 рисунков, 8 приложений и 15 информационных источников.

Autoreferāts.

Darba autors ir izpētījis bezvada ierīču lietojuma risinājumu problemātiku, kura ir saistīta ar J2ME un J2EE platformu mijiedarbību. Pētījumu rezultāts norāda, ka vairumā problēmas ir attiecināmas uz J2ME mobilo ierīču pusi. Autors ir apskatījis divu platformu arhitektūru un to sastāvdaļu pienākumu sadali un atbildību par savstarpējo savienojumu izveidošanu.

Pētnieciskā darba veikšanās gaitā autors ir apkopojis un sastrukturizējis informāciju par ziņojumapmaiņās tipiem starp mobilām ierīcēm un J2EE aplikāciju serveriem. Mūsdienu tehnoloģiju apstākļos vairāki faktu dēļ ne visi ziņojumapmaiņas formāti var būt veiksmīgi pielietoti reālā projektā. Autors secina, ka šis fakts vedina uz domu par mobilo ierīču sadarbības šablonu definēšanas un standartizēšanas nepieciešamību sadarbojoties ar Java aplikācijas serveriem.

Dažreiz gadās, ka nav iespējams izmantot nepieciešamo ziņojumapmaiņas formātu, tādos gadi jomos, darba autors ir aprakstījis dažus parauga risinājumus, kuri var palīdzēt apiet šo problēmu.

Praktiskā daļā autors pats pārliecinājies par šablonu vieglu implementēšanu un pielietojis 2 šablonus no aplūkotiem šablonu paraugiem.

Saturs

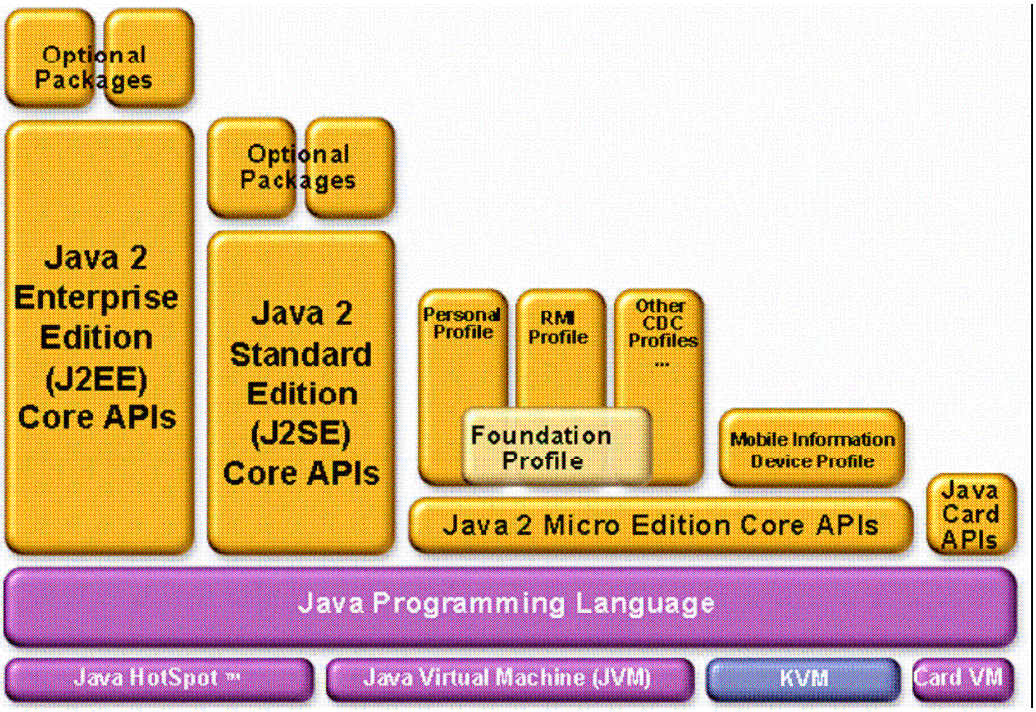
Ievads

Topošās bezvadu tehnoloģijas atver plašu un perspektīvu tirgu jaunajām lietojumu stilam un servisiem, kuri orientēti uz dažādām pircēju formām, piemērām uzņēmumiem, mazām kompānijām utt. Java tehnoloģija nodrošina stabilo tehnoloģisku pamatu, kurš ļauj nākamo paaudžu ierīcēm piedāvāt jaunas iespējas, tādas kā uzlabota mijiedarbība, daudzfunkcionāla lietotāja saskarne, vairāku procesu autonoma apstrāde, lokāla datu atmiņa, tīklu komunikācija jeb tīklošana. Izmantojot šīs jaunās iespējas, izstrādātāji un uzņēmumi varēs radīt jaunus, aizraujošus servisus bezvadu tehnoloģiju tirgū. [1]

Lai izstrādātu lietojumus izmantojot bezvadu Java tehnoloģiju ir nepieciešmas zināšanas divās Java 2 platformas tehnoloģijās – Java 2 Micro Edition un Java 2 Enterprise Edition. Turpinājumā darba autors aprakstīs šīs tehnoloģijas.

1999. gadā jūnijā kompānija *SUN Microsystems* pieņēma lēmumu attīstīt Java 2 platformu trijos virzienos:

- *Java 2 Standart Edition (J2SE)*
- *Java 2 Enterprise Edition (J2EE)*
- *Java 2 Micro Edition (J2ME)*



1.att. Java 2 platforma

J2SE platforma orientējas uz gala lietotāju sistēmām, savukārt J2EE platforma orientējas uz servera lietojumiem. J2ME ir lietojumprogrammu saskarnes (API) kopums, kurš fokusējas uz patērētajiem un pārnēsājamām ierīcēm, kuras var būt diapazonā no TV, telemātikas sistēmām līdz mobiliem telefoniem un personāliem ciparasistentiem (PDA). Kā ir redzams 1. attēlā zem katra Java 2 platformas virziena ir realizēta sava Java Virtual Machine (JVM) implementācija, kura optimizēta speciālam sistēmas tipam ar savām specifiskām īpašībām. Piemēram, K Virtual Machine (KVM) ir Java Virtual Machine, kura ir orientēta uz resursa ierobežotām ierīcēm, tādām kā mobilie telefoni vai PDA. [1]

Visiem trim virzieniem ir kopīgas šīs galvenās īpašības:

1) „Write Once Run Anywhere" – „Uzraksti Vienu reizi un Aktivizēt Visur";
2) Drošība;
3) Daudzfunkcionāla, grafiska lietotāju saskarne;
4) Tīkla iespējas.

Nākošā daļā autors apskatīs J2EE un J2ME platformas arhitektūru jau detalizētāk.

Bezvada Java tehnoloģijas būtība ir divu plaši pazīstamu vārdu krustojums: bezvada datu komunikācijas un Java platforma. Java mūsdienu mobilās ierīcēs ir plaši popularizēta, pieņemta un jau statistiski pieradīta, ka ir labākā vide mobilās ierīcēs. Ap 250 miljonu uz Java tehnoloģijām balstītu bezvada ierīču, no 31 mobilo ierīču ražotājiem, tiek izmantotas vairāk par 75 mobilo operatoru tīklos. Un pus biljonu Java (tm) Card tiek ieviestas viedkartēs (smart card) un mobilo telefonu vidē. Šie skaitļi liek domāt par to, cik pievilcīga ir Javas platforma mobilo risinājumu tirgū. Analītiķi prognozē, ka vismaz 80 procentu no mobiliem telefoniem ir gaidāmi ar Java atbalstu 2006. gadā un apmērām 450 miljonu mobilo ierīču tiks ekspluatēti 2007. gadā.

Vislielākie un visveiksmīgākie Javas tehnoloģijas panākumu faktori mobilo ierīču tirgū ir:

1) Plaša ierīču, tādu kā smartfoni, ar progresīvam iespējam un lietojumu vajadzību;
2) Platformneatkarīgu tehnoloģijas nepieciešamība multi platformu mobilā pasaulē;
3) Konfigurējamo ierīču radīšanas nepieciešamība, kuras ģenerēs unikālus servisus mobiliem operatoriem un dos iespēju lejupielādēt lietojumus bezvada tīklā.

Šķiet, ka Java nākotnē nodrošinās vienkāršāku mobilo telefonu programmēšanu, bagātākas spēles, un lielākus ienākumus mobiliem operatoriem.

Mobilas ierīces parasti ir nokomplektētas ar standarta skaitu iepriekšuzstādītu lietojumprogrammu, piemēram, kalendāru, pulksteni un dažādām spēlēm. Java tehnoloģija to izmaina pilnībā, ļaujot mobilas ierīces lietotājiem lejupielādēt savos tālruņos jaunas lietojumprogrammas, kuras var būt ne tikai spēles, bet arī biznesa lietojumprogrammas, kā arī kuras var projicēt dažādas funkcionalitātes no parastiem gala lietotāju lietojumiem. Tādējādi, mobilo telefonu lietotāji var baudīt lietojumprogrammu izstrādātāju radošā potenciāla augļus. Java tehnoloģija padara tālruņa lietošanu daudz aizraujošāku un ļauj personificēt ierīci, uzstādot tajā mobilā telefonā lietotāja izvēlētās lietojumprogrammas.

Gribu atzīmēt vienu svarīgu organizāciju, kuras Java platformas attīstība ir mērķis un organizācijas dibināšanas iemesls. Java Community Process (JCP) ir atvērta organizācija, kas sastāv no Java izstrādātājiem un no licenču turētājiem. To ir nodibinājusi Sun Microsystems, lai izstrādātu un attistīt Java tehnoloģiju tehniskos parametrus, paraug realizācijas un tehnoloģiskos komplektus. JCP vada divas izpildkomitejas. Viena izpildkomiteja atbild par J2EE un J2SE, bet otra par J2ME.

1. Mobilo tīklu lietojumu izstrādes sarežģītība un problēmas

Programmatūras izstrādātāji, kuri izstrādā speciālas lietojumus lielām, spēcīgām sistēmām ir kļuvuši ļoti nevērīgi pret virtuālas atmiņas ierobežotu resursu un uz apstrādes jaudu. Programmatūras izstrādātāji satiekas ar jaunu šķēršļu kopumu, kad viņiem ir jāstrādā ar mobilam bezvadu ierīcēm ar ierobežotu resursu pieejamību.

Izstrādātāji tāpat satiekas ar apkārtējās vides problēmām, kuras ierosina pašas ierīces. Kā arī mobilitāte bezvada tīklos, kura tipiski tiek piedāvāta ar pazemināt joslas platumu un mazāku drošumu nekā vada tīkls. Šie ierobežojumi piespiež izstrādātājus domāt savādāk par lietotāja saskarni, apstrādes jaudu, atmiņas pārvaldīšanu un kļūdu apstrādi. Izstrādājot lietojumus ierīcēm ar ierobežotiem resursiem, tiek prasīti zinoši izstrādātāji, kuriem ir velēšanās mācīt un saprast kā bezvada vide darbojas un būt spējīgiem pieņemt pamatotus lēmumus attiecība uz programmatūras arhitektūru un izpildes īpatnībām.

Straujš bezvadu ierīču tirgus pieaugums stimulē plaši izplatītos lietojumus adaptēt gandrīz jebkurai tehnoloģijai, pārstrādājot tos no galddatoriem izmantotajiem lietojumiem uz mobilo ierīču lietojumiem, kuri izmanto bezvada tīklu. Šī adaptācija nav vienkāršs.

Pirmkārt, bezvada lietojumiem jāstrādā lielos, stingros, "drakoniskos" ierobežojumos:

1) ***Atmiņa.*** Tādiem ierīcēm, ka mobilie telefoni un peidžeri, ir ierobežots atmiņas apjoms, kurš piespiež apdomāt atmiņas pārvaldīšanu, vairāk rūpīgi projektējot lietojumu objektus.
2) ***Apstrādes jauda.*** Kā arī bezvada ierīcēm ir ierobežota apstrādes jauda (16 bitu procesori tipiski). Izstrādātajiem vajag prast atpazīt, kuri komplicēti uzdevumi var izpildīties nepieņemami ilgu laiku.
3) ***Ievade.*** Potenciālās datu ievades iespējas ir ierobežotas. Vairums no mobiliem telefoniem nodrošina vienrocīgu papildtastatūru ar divpadsmit pogām: desmit ciparu pogas un zvaigznīte (*) un restīte (#).
4) ***Ekrāns.*** Ekrānam jābūt mazam, piemērām 96 pikseļu platam un 54 pikseļu augstam un 1 bitu krāsām (melns un balts). Kā arī informāciju apjoms, kurš var ievietoties šajā ekrānā ir īpaši ierobežots

No vienas puses skatoties, redzams ka pārāk vienkārši fokusējoties uz ierīču ierobežošanu var aizmirst, kā bezvada vide "piedāvā" arī turpmākus ierobežojumus:

- Bezvada tīkli ir nedroši un dārgi, kā arī joslas platums ir zems.
- Bezvada tīkliem rodas vairāk tīkla kļūdas nekā parasto vada tīkliem.
- Bezvadu ierīču pārāk plašā mobilitāte palielina risku, kad savienojums var tikt pārtraukts.

Lai izstrādāt efektīvas un drošas bezvadu lietojumus, vajag ņemt vērā šos iepriekš minētos ierobežojumus.

Kā darba autors minējis iepriekš, izstrādājot mobilo bezvadu ierīču lietojumus rodas dažādas problēmas, kuras izstrādātajiem jāņem vērā. Zemāk autors aprakstīs dažas, nozīmīgākās problēmas, ar kurām jāsaskaras un jāmēģina atrisināt mobilo tehnoloģiju lietojumu izstrādātājiem savos projektos.

Datu pārraides kļūdas.

Ziņojumi, kuri ir nosūtīt bezvada tīklā var būt bojāti ar traucējumiem, kuri var izmainīt saturu, kuru saņem cits lietotājs, ierīce vai serveris. Šādus gadījumus vajag paredzēt mobilo tehnoloģiju lietojumos. Pārraides kļūdas var rasties jebkurā vietā ziņojumu sūtīšanas vai saņemšanas laikā. Bezvada tīkla protokoli var atklāt un koriģēt šīs kļūdas, bet vajag prast lietot kļūdu apstrādes stratēģijas, kuras atrisina visu veidu pārraides kļūdas, kuras var rasties.

Ziņojuma gaidīšanas laiks.

Ziņojuma gaidīšanas laiks – tas ir laiks, kurā tiek nogādāts ziņojums. Pirmkārt, tas ir atkarīgs no sistēmas rakstura, kura apstrādā ziņojumus, nepieciešamā apstrādes laika un no kavēšanās, kura var rasties jebkurā vietā no sūtītāja līdz saņēmējām. Izstrādātājam ir jāņem vērā arī šo ziņojuma gaidīšanas laiku, un lietotāja saskarnē vajag paredzēt kādu informācijas paziņojumu par šo faktu. Svarīgi šeit atzīmēt, ka ziņojums var būt nogādāts lietotājam pēc kāda nenoteikta laika kopš tā brīža kad tas bija izsūtīts. Ilgs kavējums var būt tīkla seguma vai pārraides kļūdu dēļ, ka arī lietotāja ierīce var būt izslēgta vai tai ir beigusies baterija. Bet tomēr dažas sistēmas ir uzprojektētas tādā veidā, kā pēc kāda laikā mēģina pārraidīt vēl reiz šo ziņojumu, kamēr tas netiek nogādāts adresātam. Savukārt, citas sistēmas saglabā ziņojumus, un

ziņojumi tiek nogādāti kad ierīce būs savienojumā stāvoklī. Tāpēc jādomā arī par šo reāli potenciālu iespējamo problēmu.

Drošība.

Jebkura informācija, kura tiek pārraidīta bezvadu tīklā var nokļūt trešās pusēs rīcībā. Daža informācija varu būt slepena, tāda kā kredītkartes numuri vai kāda cita personiskā informācija. Risinājums ir atkarīgs no informācijas svarīguma un slepenības līmeņa. Lai piedāvātu galīgu drošības risinājumu, to vajag realizēt abās pusēs –klienta un servera pusē. Kā arī pārliecināties, ka visas komunikāciju iesaistītas puses ir vienlīdzīgi nodrošinātas. Viens no risinājumiem ir pielietot šifrēšanu. [2]

Autora viedoklis ir, ka eksistē vēl vienā liela problēmā – gatavo lietojumu pārnesamība no viena ražotāja ierīces uz citu ražotāju ierīci. Šajā gadījumā tiek noliegts apgalvojums un rodas šaubas par vienu no Java izvirzīto un popularizēto priekšrocību - „Write Once Run Anywhere”. Problēma rodas tāpēc, ka *katrs* mobilo ierīču ražotājs izveido *savu* bibliotēku darbam ar grafiskām, skaņu, komunikācijas vai citu veidu iespējām mobilā ierīcē. No vienas puses tas ļauj pilnīgi realizēt un izmantot visas ierīces funkcionalitātes un potenciālu, bet no citas puses tā ir šī pārnesamības problēma. Protams, var izveidot lietojumu, kura izmantos tikai standartus mobilo ierīču Java bibliotēkas, bet tad, šīs lietojums būs ļoti ierobežots un mobilo programmatūras tirgū tas zaudēs vietu citiem lietojumiem, kuri izmanto visu ierīču funkcionalitātes jaudu.

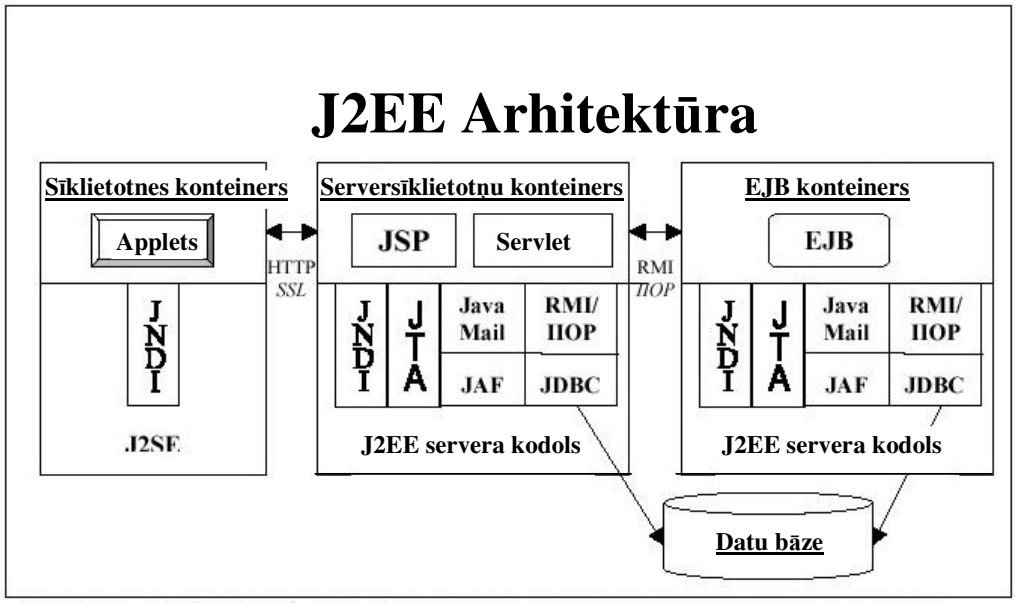
2. Java 2 platformu veidi

2.1. Java 2 Enterprise Edition

J2EE platforma piedāvā nepieciešamo lietišķo Java programmu interfeisu un servisu daudzumu korporatīvo lietojumu uzturēšanai. Visa platforma var būt realizēta vienā sistēmā vai sadalīta vairākās sistēmās, bet visu API saskarni jāiekļauj kopējā sistēmā.

2.1.1 J2EE arhitektūra

2. attēlā ir parādīta J2EE arhitektūra.



2.att. J2EE arhitektūra

Paskaidrojumi 2. attēlam:

- **Sīklietotnes konteiners** – pārvalda sīklietotņu izpildīšanu. Sastāv no Web pārlūka, un utilītprogrammām (Plug-in), kuras tiek aktivizētas klienta datorā vienlaicīgi.
- **Serversīklietotņu konteiners** – pārvalda JSP lappuses un serversīklietotņu komponentu izpildīšanu J2EE lietojumos. Web komponenti un Web konteiners tiek aktivizēti J2EE serverī.

- **EJB konteiners –** pārvalda EJB komponentu izpildīšanu J2EE lietojumos. EJB komponenti un tā konteiners tiek aktivizēts J2EE serverī.

Kā redzams, gan Servlet, gan EJB konteinerī jābūt realizētiem sekojošajiem API: JNDI, JTA, JavaMail, JAF, RMI, JDBC. Šo saskarņu paskaidrojums atrodams 2. pielikumā [3].

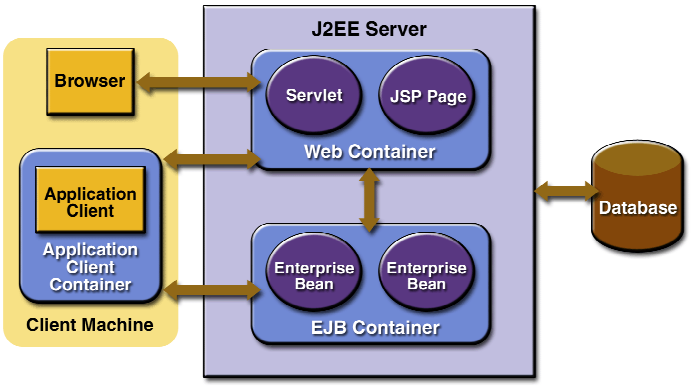
J2EE arhitektūras pamats bāzējas uz „Komponents - Konteiners" („*Component - Container*") struktūras, kura tiek realizēta visos 2.attēlā paradītajos konteineros.

2.1.2 „Client – Server" tehnoloģija

Objektīvi un vispārīgi var sadalīt J2EE platformas „Klients - Serveris" („*Client - Server*") tehnoloģiju 3 līmeņos:

- Klienta līmenis;
- Servera līmenis;
- Datu bāzes līmenis.

3.attēlā ir parādīta „Klients - Serveris" tehnoloģijas līmeņu sadarbība.



3.att. J2EE platformas „Klients - Serveris" struktūra

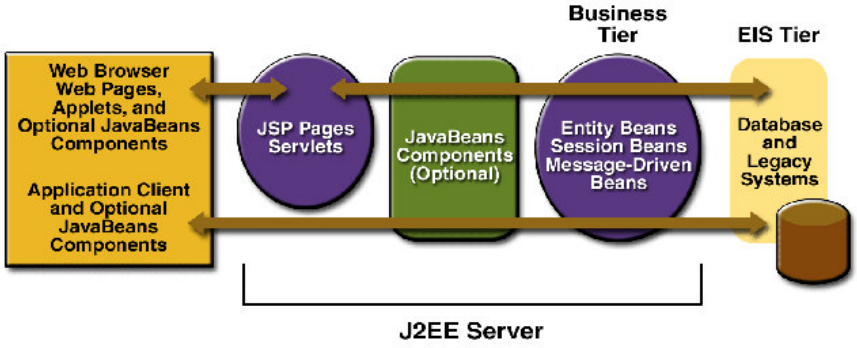
Parasti, daudz saistīto lietojumu, ir grūti programmēt, jo tas iekļauj daudz sarežģīta koda, kurš apstrādā: transakcijas, stāvokļa vadību, resursu pārvaldību un daudz citu sarežģītu apakš līmeņa uzdevumus. Neatkarīga no platformas J2EE arhitektūra, kura tiek bāzēta uz komponentiem, ļauj viegli izstrādāt J2EE lietojumus, jo izmantojot komponentus viegli rakstīt programmatūras kodu, tā kā visa uzmanība tiek fokusēta tikai uz biznesa loģiku. J2EE serveris piedāvā apakš līmeņa servisu konteinera formā katram komponenta tipam.

Konteineri ir interfeiss starp komponentu un platformas apakš līmeņa speciālu funkcionalitāti, kuru uztur šis komponents. Zemāk tiek paskaidrots 3. attēls.

- **Browser** – pārvalda sīklietotņu izpildīšanu. Sastāv no Web pārlūka, un utilītprogrammām, kuras tiek palaistas klienta datorā vienlaicīgi.
- **Web container** – pārvalda JSP lappušu un Servlet komponentu izpildīšanu J2EE lietojumos. Web komponenti un Web konteiners tiek palaisti J2EE serverī.
- **EJB container –** pārvalda EJB komponentu izpildīšanu J2EE lietojumos. EJB komponenti un tā konteiners tiek aktivizēts J2EE serverī.
- **Application client container** – pārvalda klienta lietojumu komponentu izpildīšanu. Lietojumu klients un EJB konteiners tiek aktivizēts klienta datorā [4].

2.1.3 Standarts

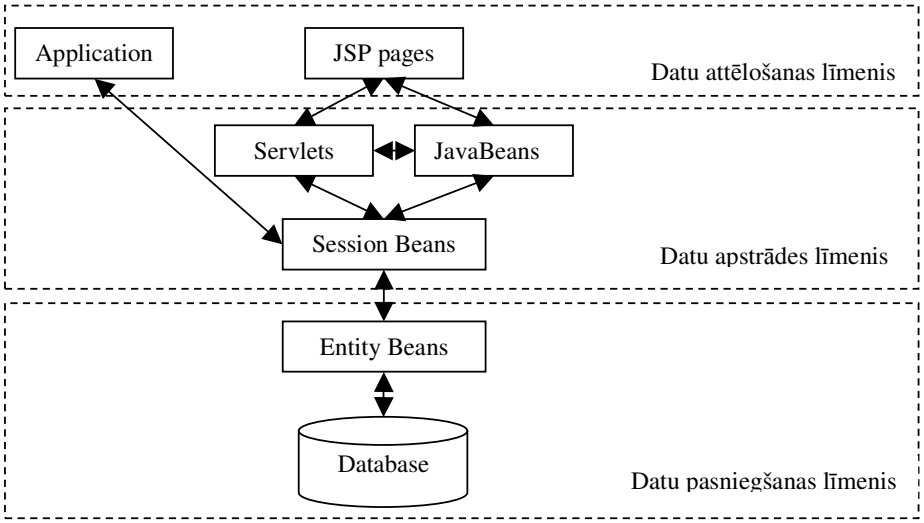
J2EE platforma definē standartu daudzlīmeņu korporatīvo lietojumu izstrādāšanai. 4.attēlā ir parādīta šo līmeņu saistība un mijiedarbība.



4.att. J2EE daudz līmeņu struktūra

J2EE atvieglo korporatīvo lietojumu izstrādāšanu pateicoties standartizētu moduļu komponentu izmantošanai, kura tiek apgādāta ar pilno nepieciešamo servisu. 2. attēlā bija parādīts šis serviss.

Apkopojot J2EE arhitektūras un struktūras aprakstu, šajā daļā 5.attelā ir paradīts J2EE konkretizēts standarts.



5.att. J2EE platformas standarts

Šajā attēlā ir parādīti trīs pamatlīmeņu savstarpējie sakari J2EE platformas standartā, kur katrā līmenī, savukārt, tiek izveidoti savi savstarpēji sakari starp līmeņa moduļiem – J2EE platformas daļām.

Protams, var izveidot saikni ar datu bāzi apejot kādu līmeni, vai tā satura līmeņa moduli, bet tad projekts zaudē savu vērtību daudzos parametros, tādos kā: drošība, vienkāršība, daudzfunkcionalitāte, projekta modificēšanas sarežģītība utt.

Datu attēlošanas līmenī jārealizē klienta saskarne, kur tiek attēloti dati no datu bāzes. Šajā līmenī klientam ir iespēja darboties ar datiem.

Datu apstrādes līmenī jārealizē visa biznesa loģika, kura ir noteikti definēta projektam.

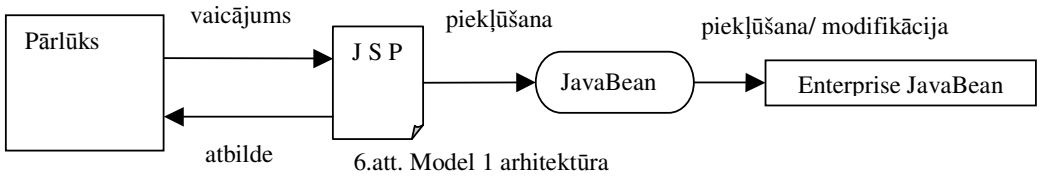
Datu pasniegšanas līmenī jārealizē tikai datu bāzes struktūras pasniegšana *Entity Beans* komponentēs.

2.1.4 JSP piekļūšanas modeļi

JSP tehnoloģijas specifikācija piedāvā divus pamata modeļus (Model 1, Model 2), ar kuru palīdzību var efektīvi izmantot biznesa objektu loģiku un piekļūšanu datu bāzei.

Model 1

Model 1 arhitektūra nosacīti ir parādīta 6.attēlā. Tas iekļauj pārlūku – klienta darbības vidi, JSP lappusi, JavaBean komponentes un biznesa objektus – EJB.



6.att. Model 1 arhitektūra

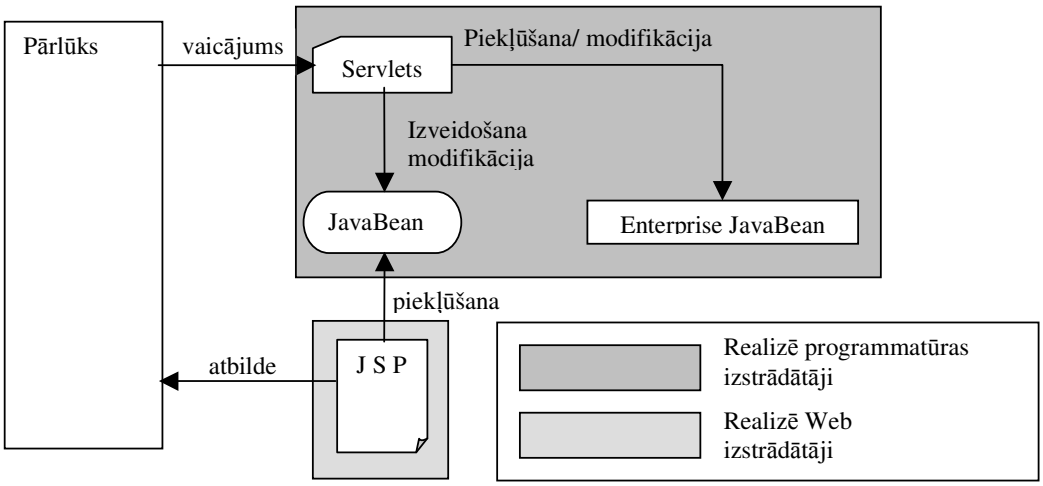
Saskaņā ar šo arhitektūru, vaicājumi tiek nodoti JSP lappusēm, kuras mijiedarbojas ar EJB komponentēm, izmantojot JavaBean komponentes. Šāda veida mijiedarbība samazina JSP atkarību no izmaiņām, kura var būt EJB komponentēs. EJB un JavaBean komponentes realizē programmatūras izstrādātāji, bet Web – lappušu izstrādātāji nodrošina JSP lappušu izveidošanu.

Teorētiski, tāda pienākumu sadale varēja nodrošināt paralēlu projekta izstrādi abām izstrādātāju grupām. Bet reāla dzīvē, praktiski tas nav iespējams.

Tā kā sadalīt pienākumus starp Web un programmatūras izstrādātājiem ir ļoti grūti, tad šo modeli var pielietot tikai nelielos projektos. Lielajos projektos nākas izmantot citu pieeju – Model 2. Tas būs apskatīts zemāk [5].

Model 2

Saskaņā ar Model 2 arhitektūru vaicājumi tiek nodoti serversīklietotnei, kura griežas pie EJB komponentiem un izveido saturu. Šis saturs saglabājas JavaBean komponentā, kuru izmanto JSP lappusēs, lai ģenerētu atbildi klientam. Model 2 arhitektūra ir parādīta 7. attēlā.



7.att. Model 2 arhitektūra

Iespēja sadalīt satura ģenerāciju un pasniegšanu ir ļoti svarīga, tā kā datu ģenerācija izmanto Java kodu. Java – koda inkapsulācija ļauj programmatūras izstrādātājiem

koncentrēties uz serversīklietotnēm un EJB komponentiem, bet tajā pašā laikā Web izstrādātāji nodarbojās ar JSP lappušu izveidošanu. Attēlā leģendā ir parādīta darbu sadale starp izstrādātājiem [5].

2.2. Java 2 Micro Edition

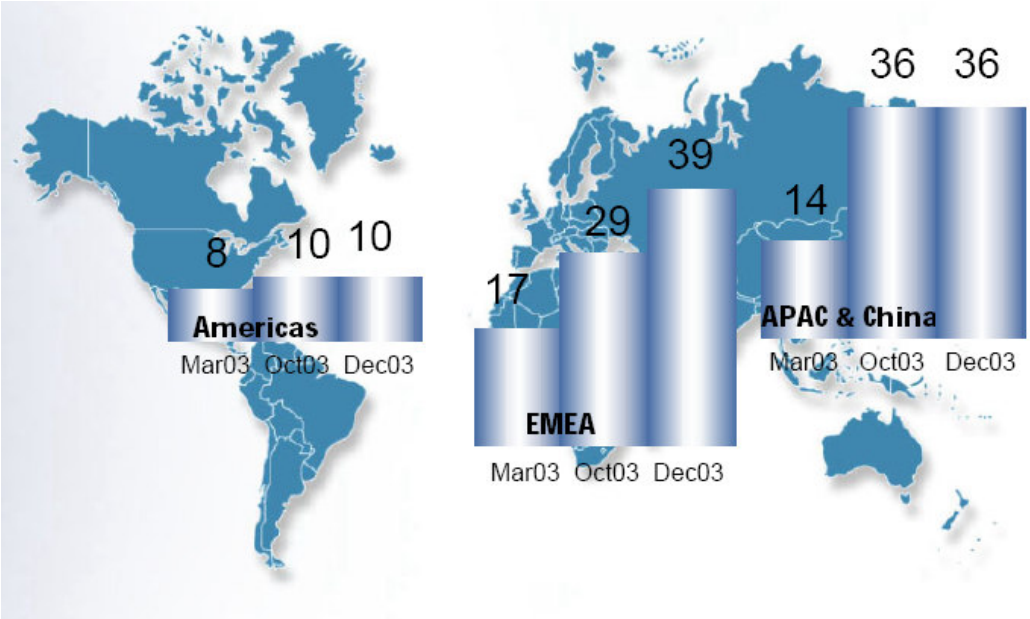
1999. gada 15. jūnijā Sun kompānija izsludina jaunu Java 2 Micro Edition platformu - "Jaunu tehnoloģiju, kura ienes Java platformas jaudu patērētāju elektronikas tirgū" Šo jaunievedumu var atrast Sun.com arhīvos. Pēc Sun kompānijas nodoma šī platforma vajadzēja ieinteresēt mobilo ierīču izstrādātājus. Principā to tā un arī pielieto. Kā pierādījumu, autors grib paradīt statistiku un dažus faktus, kurus prezentēja 2004. gada 20. februārī kompānija NOKIA.[6]

2003. un 2004. gada statistika:

- 2004. gadā tirgu bija parādījušies virs 250 miljonu ierīču ar Java atbalstu.
- Pasaule eksistē ap 4 miljoniem Java izstrādātajiem, no kuriem 350 000 fokusējas uz mobilo vidi.
- Finansu ienākumi no Java mobila biznesa 2003. gadā bija €2.7 biljonu.
- 2003. gadā apmēram 10 miljonu Java lejupielādes globāli mēnesī.
- 2004. gadā apmēram 15 miljonu Java lejupielādes globāli mēnesī.
- 100 mobilo operatoru visā pasaulē ieviesa Java™ pakalpojumus (2003. gada decembris)
- Mobilo ierīču ražotāji ap 80% nodrošināja J2ME atbalstu savā produkcijā
- Izlaisti 200 mobilo ierīču modeļi ar Java atbalstu no 27 ražotājiem (2003. gada oktobris).
- 2003. gadā oktobrī mobilo ierīču daudzums ar Java atbalstu tirgū sastādīja vairāk par 120 miljoniem (2003. gada oktobris).

Vairāk par 1 miljonu Java izstrādātāju rīku kopu tiek lejupielādēta no forum.nokia.com web-vietnes. [6]

8. attēlā ir paradīta triju (EMEA – Europe, Midle East, Africa; APAC – Asia Pacific) lielo pasaules reģionu mobilo operatoru pieaugums 2003. gada trijos mēnešos (marts, oktobris, decembris), kuri ieviesa Java pakalpojumus.

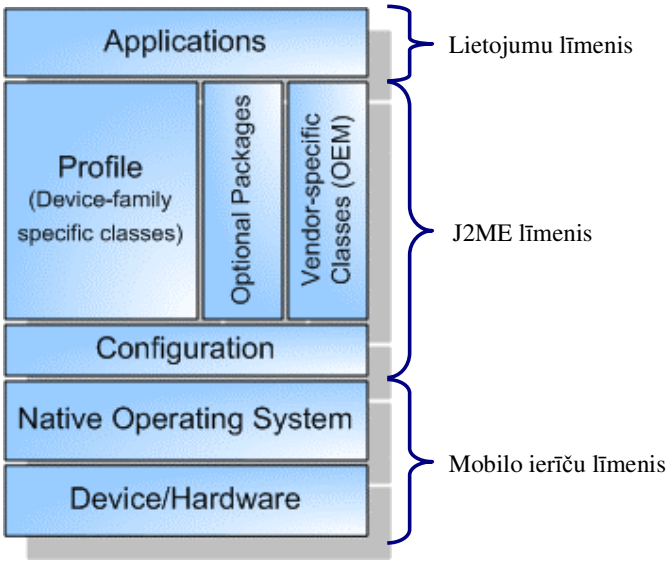


8.att. Mobilo operatoru skaits, kuri ieviesa Java pakalpojumus

2.2.1 J2ME arhitektūra

Java 2 Micro Edition (J2ME) platforma ir vesela tehnoloģiju un tehnisko parametru kolekcija, kas ir izstrādāta dažādām nelielu ierīču tirgus daļām. J2ME ļauj izstrādātajiem izveidot Java lietojumus dažādām tirgus daļas biznesa patērētājiem, izmantojot platformas komponentes.

9. attēlā ir paradīta vispārēja 3 līmeņu arhitektūra, kura realizētā jebkurā mobilā ierīcē ar J2ME platformas atbalstu.



9.att. J2ME platformas organizācija

Mobilo ierīču līmenī atrodas paša mobila ierīce *(Device/Hardware)* ar savu operētāj sistēmu *(Native Operating System)*. Piemērām NOKIA telefonu gadījumā tas var būt NOKIA operētājsistēma, vai Symbian operētājsistēma.

J2ME līmenī atrodas J2ME platformas komponentes. Vispār var sadalīt J2ME platformas komponentes divās daļās: pamat komponentes un neobligātas komponentes. Par neobligātām komponentēm autors uzskata, ka var būt vai var arī nebūt mobilā ierīcē. Pamat komponentes - *Profile* un *Configuration*, bet neobligātas – *Optional Packages* un *Vendor-specific classes(OEM)*.

- *Configuration* - J2ME platformas kodolu veido divas dažādas konfigurācijas: Connected Device Configuration (CDC) un Connected Limited Device Configuration (CLDC). CLDC – ir vismazākais no divām konfigurācijām. Tas bija izveidots ierīcēm ar intermitējošā tīkla savienojumiem, lēniem procesoriem ar ierobežoto atmiņu – ierīcēm kā mobilie telefoni, peidžeri un PDA. CDC – ir izveidots ierīcēm kuriem ir vairāk atmiņas, ātrāk ir procesori un efektīvāk ir realizētas tīkla savienojuma iespējas.
- *Profile* - nosaka funkcionalitāti noteiktā ierīču kategorijā. Mobile Information Device Profile (MIDP) – Mobilo informācijas ierīču profils ir profils mobilām ierīcēm ar CLDC konfigurāciju, kam ir informācijas pārraides iespējas. MIDP piedāvā lietojumu funkcionalitātes kodolu, kurā ietilpst lietotāja saskarne, pastāvīgā atmiņa, tīkla funkcijas, lietojumprogrammu modelis, lokālu datu atmiņa un lietojumu vadīšana.
- *Optional Packages* - J2ME platforma var būt paplašināta kombinējot dažādus neobligātas pakotnes ar konfigurācijām un viņu profiliem
- *Vendor-specific classes(OEM)* – Mobilā ierīcē var būt realizēta ražotāja specifiska lietojumprogrammas saskarne. Tas nav standarta daļa no J2ME platformas. Šī saskarne paplašina un piedāvā funkcionalitāti, kura ir specifiska noteiktai ierīcei.

Lietojumu līmenī atrodas paši lietojumi uzrakstīti Java valodā. [7]

Tīkla savienojumi ar serveru vai citu ierīci ir iespējamas izmantojot Generic Connection Framework (GCF). GCF ir aprakstīts 3.4 nodaļā. Izstrādātāji var izveidot tīkla lietojumu klientus, kuri realizē bezvada servisus izmantojot standarta tīkla protokolus. MIDP profilam obligāta prasība, kuru jārealizē visās J2ME ierīcēs, ir HTTP protokola atbalsts.

3. Mobilo ierīču komunikācija

3.1. Bezvadu datu pārraides standarti un to evolūcija

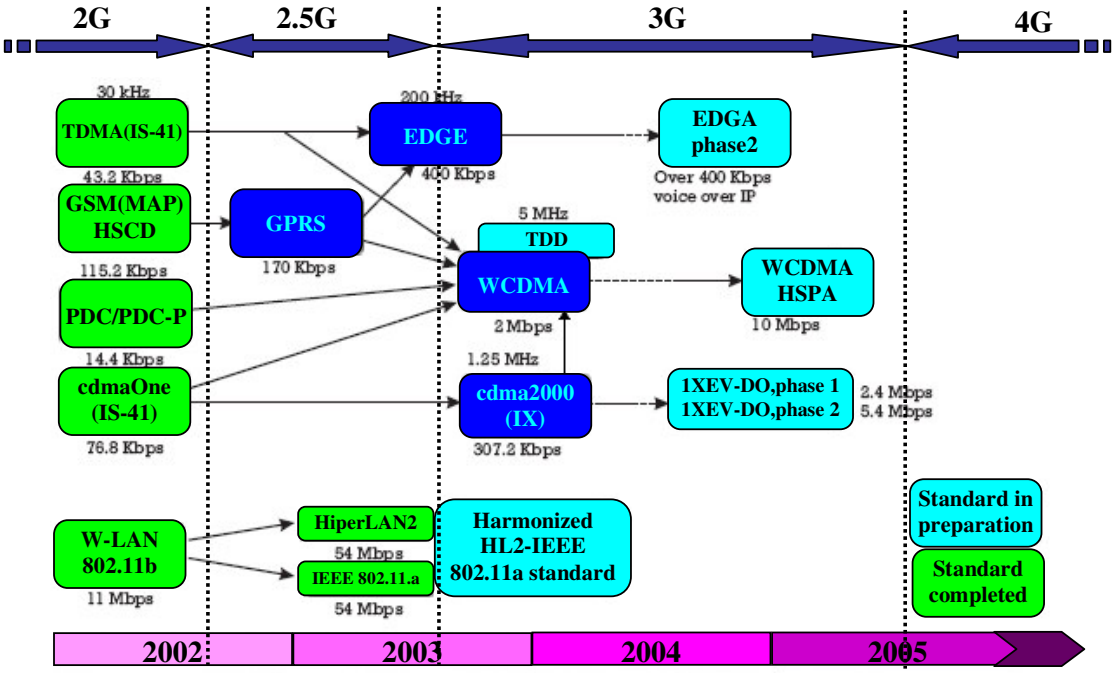
Tā, ka sadarbība starp mobilajiem ierīcēm un serveriem ir iespējama izmantojot bezvadu sakarus, tad šajā nodaļā būs neliels apskats par bezvadu datu pārraides standartiem un to evolūciju. Autors uzskata, ka mobilo ierīču J2ME platformas evolūcija ir cieši saistīta ar bezvadu sakaru tehnoloģijām un standartiem. Autoram šķiet, ka katrā konkrētā gadījumā un katrā konkrētā valstī mobilo lietojumu potenciāls ir ierobežots ar konkrētu mobilo operatoru bezvadu sakaru tehnoloģiju atbalstu. Jo plašāks ir mobilo operatoru jauno bezvadu standartu atbalsts, jo plašākas ir iespējas mobilo ierīču programmatūrā.

Atskatoties, nedaudz no vēstures, 80. gados bezvadu sakaru tehnoloģija bija attīstības pašā sakumā. Izmantojot analoģisko bezvadu tehnoloģiju, tipiskais mobilais telefons nozīmēja vairāk nekā šodienas standarta vada telefons. Eiropā katra valsts izstrādāja savu analoģisko bezvada sistēmu, kura bija ierobežota ar darbību tikai savā valsts teritorijā. Ceļot no valsts uz valsti bija ļoti ērti valstu atrašanas vietas tuvuma dēļ, bet bezvadu sakaru viesabonēšana bija neiespējama. 1983. gadā bija izstrādāts digitāls standarts, tā saucamais GSM (Global System for Mobile communications - globālā mobilo sakaru sistēma). Tuvāk 1993. gadam, lielākais Eiropas mobilo operatoru skaits migrēja uz GSM standartu. GSM nevarēja piedāvāt ātru datu pārraides savienojumu. Kad operatori nolēma migrēt uz ciparu tehnoloģiju, lai saņemt vēl lielu jaudu no tā, bija iespējami trīs standarti – TDMA, CDMA un GSM. Katrs no standartiem bija stipri atbalstīts ar šo standartu iesniedzējiem. Rezultātā operatori izvelējas dažādus standartus.

1999. gadā bija nodibināta International Telecommunication Union (ITU) organizācija, kuras radīšana bija nepieciešama, lai standartizētu nākotnes ciparu bezvadu sakarus un veidot globālu viesabonēšanu. Vēlāk šī organizācija bija nosaukta par 3G.

Vispār bezvada sakaru attīstību var sadalīt daudzos periodos, kur katrs periods var būt apzīmēts par kārtējo paaudzi G (generation). Proti, periodu secība ir sekojoša: 1G, 2G, 2.5G, 3G un nākotnes 4G standarts.

10. attēlā ir paradīta bezvadu standartu atkarība un evolūcija laikā un G periodu mērogā.



10.att. Bezvadu datu pārraides standarti un to evolūcija

1990. gads un turpmākie gadi bija 1G periods jeb sākums ciparu sistēmām. Bezvadu sakaru augstu līmeņu izmantošana un ierobežota frekvences sadale ierosināja lielāku efektivitātes spektru vajadzību. Un attiecīgi šīs prasības spēja realizēt tikai 2G paaudze. Tā kā mobilo telefonu izmantošana palielinājās un cilvēki kļuva vairāk mobili, parādījās jaunas iespējas izmantot mobilo telefonu datu pārraidei. Pirmais etaps sakaru migrācija bija ar mērķi nodrošināt vidējo datu pārraides ātrumu. Šīs sistēmas iekrīt 2.5G paaudzē. Tomēr beidzamais mērķis bija nodrošināt relatīvi ātru datu pārraides iespēju.

Principā nākamo paaudžu bezvadu tīklu prasības bija sekojošas:

- Uzlabota sistēmas jauda,
- Savienojamība ar iepriekšējo paaudzi,
- Multimediju atbalsts,
- Ātru pakešu datu servisi, kuri atbilst specifiskiem datu pārraides ātruma kritērijiem.

Kā arī bija tehniskas prasības uz datu pārraides ātrumu:

- 2 Mbps fiksētā vidē,
- 384 Kbps kājāmgājēja vai pilsētas vidē
- 144 Kbps plašā apgabalā mobilā vidē

Šīs prasības bija apmierinātas 3G paaudzē. Vispār 3G standartus tiek plānots ieviest mobilos operatoru tīklos no 2000. līdz 2009. gadam. Ja skatoties no mūsdienas, tad 4G standarts ir nākotnes paaudzes standarts. 4G datu pārraides ātruma prasības ir 100 Mbps kustībā un 1Gbps fiksēta stāvoklī. Ieviešanas periods var būt no 2010. gada līdz 2015. gadam. Būtībā 4G standarts piedāvās lielu datu pārraides ātrumu un augstāku video un audio kvalitāti pārraidēs režīmā.[8]

GPRS ir bāzes datu pārraides sakars starp J2ME un J2EE platformām. 10. attēlā ar tumšu zilu krasu ir atzīmēti standarti, kuri pašlaik ir aktuāli – GPRS, EDGE, WCDMA un cdma2000. Aktuāli ir tāpēc, ka Latvijā mobilie operatori uztur visus vai dažus iepriekšminētus standartus.

SIA "Latvijas Mobilais Telefons" (LMT) trešās paaudzes pakalpojumu sniegšanu komerciālā režīmā uzsāka 2004. gada 13. decembrī. LMT, izmantojot trešās paaudzes mobilos sakarus, sniedz šādus pakalpojumus: balss zvans augstākā kvalitātē, īsziņas, datu pārraide līdz 384 Kbps, papildpakalpojumi (kopš 2004. gada 13. decembra), video straumēšana „Mobilā TV" (kopš 2005. gada 9.jūnija) un video zvans (2006. gada 16. janvāra).

SIA „Telekom Baltija" mobilo sakaru nodrošināšanai izmanto divas no CDMA2000 saimes tehnoloģijām CDMA2000 1x (nodrošina bāzes 3G mobilos sakarus) un CDMA2000 1xEV-DO (nodrošina 3G mobilos sakarus pilnā apjomā. CDMA2000 1x komerceksplu atācijā sāka izmantot no 2005. gada 1.februāra, bet CDMA2000 1xEV-DO – no 2005. gada 1.decembra.

SIA "Tele2" 3G pakalpojumi ir pieejami kopš 2005. gada 29. novembra, izmantojot UMTS tīklu, kas ir 3G paaudzes standarts. EDGE ir GSM tehnoloģijas uzlabojums, UMTS, savukārt, ir pilnīgi jauna sistēma, par jaunu uzbūvēts tīkls.
Galvenā otrās un trešās paaudzes mobilo sakaru atšķirība ir ievērojami lielāks datu pārraides ātrums. Pašlaik, izmantojot trešās paaudzes mobilo sakaru tīklu, klientiem ir pieejami šādi pakalpojumi: video zvans, paātrinātas datu pārraides pakalpojumi, mobilā televīzija. [9]

3.2. Protokolu tipi

Praksē lielākiem MIDP lietujumu kopumam ir pieeja serveriem, un tāpēc tie reprezentē dalītos lietojumus. Īstenībā, daudzi mobilī lietojumi parāda savu svarīgo misiju un realizē nozīmīgu, perspektīvu biznesa ideju tikai savienojoties ar servera vidi. Savienojums var būt "vienmēr ieslēgts" vai tikai aktīvs, kad lietojums ir vajadzīgs komunikācija ar serveru. Mobilais korporatīvais risinājums var realizēties tikai mijiedarbībā starp J2EE un J2ME tehnoloģijām.

Risinājumiem, kuri bāzējas uz J2ME tehnoloģiju ir svarīgi pievērst uzmanību, ka tīkla savienojums un ierīces resursi ir aprobežoti un neatspoguļo parastas darba stacijas tipiskos augstos standartus. Ir daudz faktoru, kuri ir ļoti svarīgi izvēloties, savienojuma protokolu, realizējot mobilo lietojumu savienojumu ar J2EE serveri:

1) ierīces procesora jauda;
2) ierīces atmiņas iespējas;
3) ierīces un ierīču operatora tīkla ātrums (upload/download Kb/s);
4) ierīces un ierīču operatora 2G, 2.5G vai 3G paaudzes tehnoloģiju atbalsts;
5) potenciālo klientu lietojumu izmantošanas atrašanās vieta.

HTTP ir ideāls klient/serveris komunikācijas protokols Java mobiliem lietojumiem. Pēc specifikācijas katrai MIDP 1.0 saderīgai ierīcei jāatbalsta HTTP protokols. Citi protokoli, tādi kā TCP vai UDP tiek izmnatoti MIDP 2.0 versijā. Tā kā ne visas MIDP ierīces spēj uzturēt soketu un datagrammas bāzētu savienojumu, HTTP ieviešana ir elastīgais risinājums starp dažādiem mobilo ierīču ražotājiem.

J2ME platforma nepiedāvā standartizētu mehānismu kā Java Remote Method Invocation (RMI) un Java API XML-bāzētam Remote Procedure Calls (JAX-RPC). Izstrādātajam pašam jāizanalizē un jādefinē komunikācijas slāņi savā projektā.

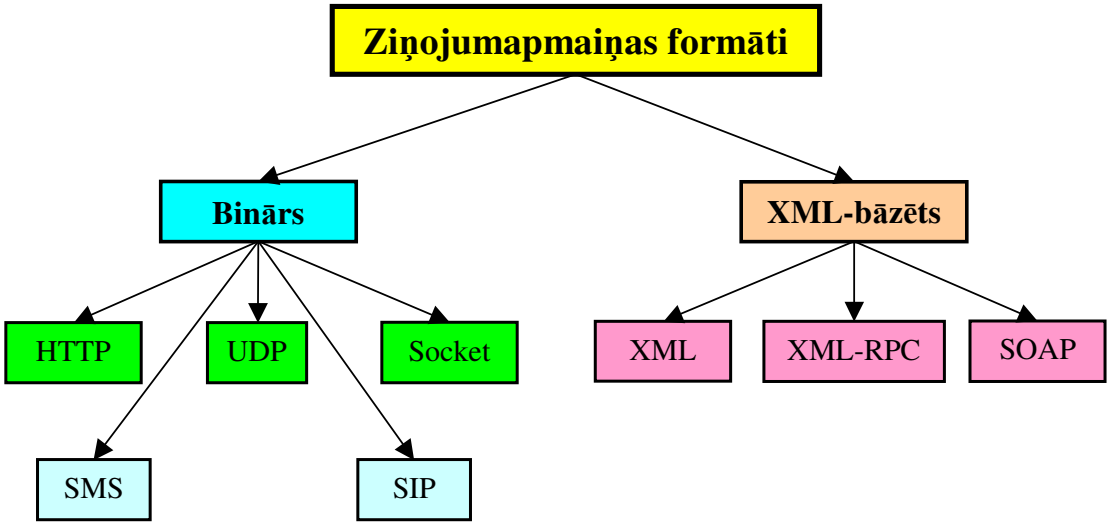
Darba ietvaros, autors izpētījis dažādus ziņojumapmaiņas formātus un nonācu pie secinājuma, kurš ir redzams 11. attēla. Tur ir paradīta ziņojumapmaiņas formātu hierarhija.

Galvenokārt, kā ir redzams šajā attēlā, eksistē divi ekstrēmi ziņojumapmaiņas hierarhijā:

1) Binārais formāts;
2) XML-bāzēts.

Pirmais formāts ir tīri vai optimizēti binārs, kurš garantē visaugstāko efektivitāti, kamēr citi sarežģīti XML-bāzēti formāti, tādi, ka SOAP, maksimāli garantē

pārnesamību un lasāmību, izpildās diemžēl diezgan vāji un lēni, īpaši mobilās ierīcēs, jo mobilās ierīcēs ir aprobežoti uz apstrādes jaudu un uz nepieciešamu resursu apjomu. Problēma, ar kuru sastopas izstrādātājs, ir atrast labāku risinājumu starp šiem diviem ekstrēmiem. [10]



11.att. Ziņojumapmaiņas formātu hierarhija

3.2.1 Binārs

Galvenais uzsvars šim darbam ir bināram formātam, bet 3.2.2. apakšnodaļā autors pastāstīja par XML-bāzētu formātu un to iespējamam realizācijām J2ME platformā. 11. attēlā ir redzams, ka binārs formāts sastāv no 5 protokoliem. Tā kā pamatā J2ME lietojumu savienojums ar J2EE serveriem vairumā notiek izmantojot HTTP, UDP un Socket protokolus, tad tos autors nolēma apskatīt detalizētāk 3.4 nodaļā.

SMS

SMS (Short Message Service) - īsziņu pakalpojums var būt ļoti plaši pielietots izstrādājot klienta/servera lietojumus. 80. gados parādījās ideja pielikt GSM pakalpojumos teksta sūtīšanu izmantojot mobilo telefonu. SMS pakalpojumā bija atrasts un realizēts plašs komercialpiedavajumu asortiments: reklāma, izklaide, visādu veidu notifikācijas, SMS, ka maksāšanas pakalpojums. Mūsdienas, SMS serviss ir ļoti plaši atbalstīts ar finansu iestādēm un īpaši ar bankām. Mobila banka ir viens no nozīmīgiem piemēriem, ka pielietojot SMS var veikt dažādas finansu operācijas. 4.4.

nodaļā autors pastāstīs par vienu paraugu, kur ir skaidri aprakstīta automatizēta SMS apstrāde un sūtīšana.

SMS pakalpojumam ir ļoti plašs pielietojumu spektrs, bet tā kā autora darba tēma ir saistīta ar mobilām ierīcēm un to komunikāciju ar serveriem, tad šajā gadījumā, autors uzskata, SMS pakalpojumu, ka vienu no ziņojumapmaiņas formātiem starp klientu (mobilo telefonu) un J2EE serveri.

2.2.1 nodāļā bija apskatīta J2ME platformas arhitektūra, tur bija pieminēta viena no 4 J2ME platformas sastāvdaļām - *Optional Packages*. J2ME platformā ir paredzēta neobligāta pakotne darbībai ar SMS. Tā ir Wireless Messaging API (WMA) bezvadu ziņojumapmaiņas lietojumprogrammu saskarne. Veicot pētījumus, ar ļoti interesantu pielietojumu saskāros WMA realizācijā J2ME platformas Push-Registry tehnoloģijā, kura ir aprakstīta 4.3. nodaļā.

SIP

Session Initiation Protocol (SIP) sesijas inicializācijas protokols bija definēts Internet tehniskā uzdevumgrupai (Internet Engineering Task Force (IETF)) sadarbībā ar 3GPP (Third Generation Parthership Project) grupai kā RFC 2543 standarts. 3GPP grupas galvenais mērķis ir trešās paaudzes bezvadu sakaru specifikāciju un standartu izstrāde un ieviešana. 3GPP grupa specificēja SIP, kā tīkla protokolu visiem IP pakešu komutācijas 3G paaudzes tiklam.

SIP ir signālu protokols, kurš tiek izmantots, lai dibinātu un kontrolētu multimedijau komunikāciju sesijas līdz etapam, kur tiek izmantots Interneta protokols (IP). Sesija var būt vienkārša, ka divu mobilo telefonu zvans, vai var būt sarežģītāk kā video konference. Ar SIP vispirms bija plānots dibināt, modificēt un nobeigt sesijas. Tas padara SIP protokolu plaši mērogojamo un paplašināmu, tas var būt viegli izmantots dažādos arhitektūras. SIP jau tiek izmantots visu veidu jauninājumos lietojumos un servisos. Īpaši gribu atzīmēt, ka SIP protokola svarīgums mobilos tīklos turpinās augt. Tā, ka MIDP 2.0 profils tagad arī uztur TCP/IP soketus un UDP/IP datagrammas, SIP kļūst svarīgs protokols IP-bāzētos mobilo telefonu vidēs.

JCP organizācija izstrādāja lietojumprogrammas specifikāciju JSR 180 (SIP API for J2ME), kura definē neobligāto pakotni (*optional package*) kā un iepriekš minētajā apakšnodaļā WMA gadījumā.

SIP protokols var piedāvāt jaunus servisus: balsu bāzēta e-komercija, izvietojuma servisi, tūlītēja ziņojumapmaiņa (instant messaging), multimedija servisi, online spēles un daudzi citi.

Ja runājam par SIP arhitektūru, tad šis protokols ir uzbūvēts uz diviem Internēt lietojuma protokoliem Simple Mail Transfer Protocol (SMTP) un Hypertext Transfer Protocol (HTTP). Papildus šim faktam, SIP protokols izmanto funkcijas, kuras nodrošina Real Time Transoprt Protocol/Real Time Control Protocol (RTP/RTCP), pateicoties kuram ir iespējama multimedijas satura straumēšana un izmantot Session Description Protocol funkcijas, kuras definē un specificē parametrus multimedija sesijām. SIP ir lietojuma protokols, kurš var izmantot dažādus transporta protokolus: Transport Control Protocol (TCP) vai User Datagram Protocol (UDP). [11]

3.2.2 XML-bāzēts

XML

XML reprezentē nākošo soli pašraksturojumu (self-description) ziņojumu formātā. XML nepieprasa automātiski izmantot RPC protokolu kā XML-bāzēts SOAP. Izstrādātājs var pats izveidot savu ziņojumapmaiņas formātu, kurš bāzējas uz XML. XML izšķiroša priekšrocība – ka tas ir standartizēts un augsti portatīvs. Turklāt, tas ir teksta-bāzēts un tāpēc strukturēti dati var būt aprakstīti pašizskaidrojoša formā. Uzņēmuma biznesa vidē XML sasniedzis priviliģētu statusu starp ziņojumapmaiņas protokoliem pamatā pateicoties labai uzturēšanai, kuru piedāvā J2EE platforma.

XML ziņojumapmaiņas formāts ir ar lielo datu apjomu teksta un XML-bāzētas specifisko tagu dēļ. J2EE platforma parasti uztur XML (vismaz viens XML pārsērs ir pieejams jebkuram servletam pēc noklusēšanas), bet MIDP 1.0 vide nepiedāvā XML parsinga uzturēšanu. Tādēļ, jebkuram XML-bāzētam risinājumam ir vajadzīgs XML pārsērs, kurš būs iekļauts MIDP klientā. Bet tomēr eksistē daudz open source parseru, tadi, ka NanoXML, TinyXML, un KXML, kuri speciāli izveidoti minimālo resursu izmantošanai.

XML bināru saspiešana

Wireless Binary XML (WBXML) bezvada binārs XML ir XML kompakta reprezentācija un Wireless Application Protocol (WAP) prezentācijas loģikas daļa.

WBXML nozīmīgi uzlabo XML pārraidīšanas efektivitāti aprobežotas joslas platuma tīklos, kur datu apjoms ir pats svarīgākais. WBXML formāts palīdz samazināt XML dokumenta izmēru, tāpēc dokumenta teksta formāts ir nokonvertēts binārā formā. Izmantojot šo formātu, apjoms ir samazināts izmantojot tagu, atribūtu un vērtību aizvietošanu ar iezīmes konfigurējamo kopu. Komunikācijas starp MIDP ierīci un J2EE platformu var notikt šifrējot un dešifrējot ziņojumus. WBXML parsētājs attiecīgi uzņemas šo šifrēšanu un dešifrēšanu.

XML – RPC

XML-RPC ir ārkārtīgi viegls ziņojumapmaiņas protokols, kurš ļauj izpildīt attālinātas procedūras pa tīklu izmantojot HTTP protokolu. Klients sūta serverim XML ziņojumu, izmantojot HTTP POST metodi, bet serveris savukārt parsē šo ziņojumu. Kad ziņojums ir noparsēts servera pusē, tiek izsaukta servera procedūra, kura atgriež atbildi, kura XML formā tiek atgriezta klientam. XML-RPC paņem nozīmīgas priekšrocības no XML. Tas datu tipus pārraida, ka parametrus izsaucot attālinātus procedūras platforma-neitrāla pieejā. Formātu sfēra ir saglabāta, cik iespējams maza aprobežojot, protokolu līdz sešiem primitīviem datu tipiem.

Trūkums šeit atkal paradās tādā pašā veidā kā ar XML, tas ir, ka J2ME platforma nepiedāvā integrētu uzturēšanu priekš XML-RPC, un tāpēc ir nepieciešami paļauties uz papild pakotnēm tādām kā KXML-RPC, ieviešanas laikā uz iecerēto ierīci ar lietojumu. KXML-RPC pakotne ir izveidota uz KXML projekta bāzes un tā pārvalda komunikāciju XML veidā, izmantojot tikai 24 Kb no ierīces resursiem iekļaujot KXML. Eksistē arī citi analoģiski open source projekti, tādi kā Apache XML-RPC.

SOAP

SOAP (Simple Object Access Protocol) – ir vienkāršs XML-bāzēts protokols, kurš ļauj lietojumiem apmainīties ar informāciju, izmantojot HTTP protokolu. SOAP ir protokols Tīmekļa Servisa (Web Service) piekļuvei.

SOAP –am ir tādi raksturojumi, kā:

- Komunikācijas protokols
- Komunikācija starp lietojumiem
- Formāts priekš ziņojumu sūtīšanas
- Tas izveidots, lai sazināties caur Internet
- Platforma neatkarīgs

- Programmēšanas valodas neatkarīgas
- Bāzēts uz XML
- Vienkāršs un paplašināms
- Izstrādāts kā W3C standarts.

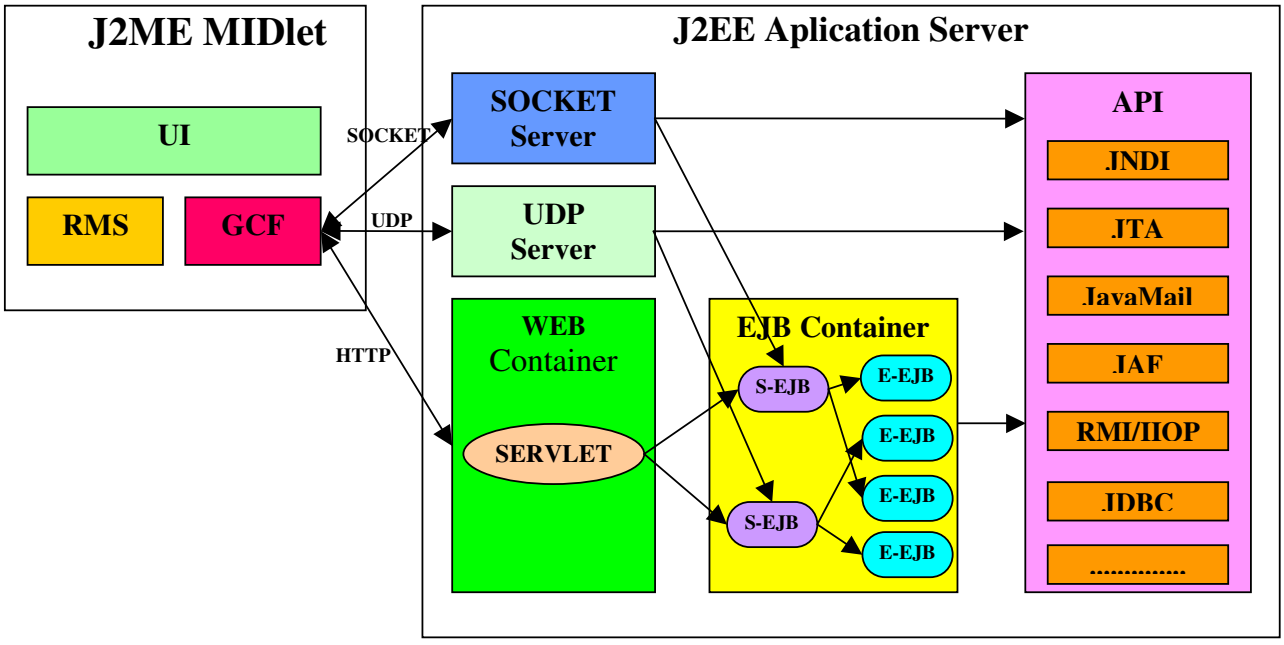
SOAP un pirmām XML-RPC versijām ir vienādas saknes. Tomēr atšķirīgi no SOAP, XML-RPC bija labots laiku pa laikam un tādā veidā protokola sarežģītība nav tik augsta kā SOAP. Lielāka sarežģītība noteikti padara protokolu ļoti elastīgu, un tieši tāpēc šis protokols bija konstatēts de facto standarts priekš attālināto procedūru izsaukumiem.

SOAP ir līdzīgs trūkums salīdzinot ar XML-RPC. J2ME platforma neatbalsta SOAP tehnoloģiju. Jebkuram klientam, kuram ir vajadzīgs SOAP atbalsts ir vajadzīgi papildus integrēt savā lietojumā SOAP funkcionalitāti.

Enhydra ME projekts atrisināja arī šo problēmu – kSOAP. KSOAP pakotne prasa vismaz 41 Kb atmiņas. SOAP jāatbalsta arī uz servera. Eksistē daudz risinājumu, viens no tiem ir AXIS projekts.

3.3. Programmēšanas modelis

12. attēlā ir paradīta tipiska struktūra izstrādājot J2ME lietojumu, kura savienojas ar J2EE serveru. Attēlā ir paradītas praktiski visas J2ME un J2EE platformas komponentes un tehnoloģijas, kuras piedalās divu platformu mijiedarbībā.



12.att. J2ME / J2EE augsto līmeņu arhitektūra

J2ME pusē par visiem tīkla savienojumiem atbild **GCF** (Generic Connection Framework). Par šo struktūru detalizētāk ir aprakstīts nākamā daļā. Pārējie J2ME puses komponentes ir **UI** (user interface) – lietotāja saskarne un **RMS (Record Management System)** –ierakstu pārvaldības sistēma. J2ME klients, izmantojot MIDP profilu, piedāvā lietotāja interfeisu mobilā ierīcē, kurš izmantojot GCF struktūru komunicējas, piemērām, ar Java servletu ar HTTP palīdzību, nepieciešamības gadījumā var pielietot drošu kanālu (HTTPS). Serlvets interpretē vaicājumus no midleta (J2ME lietojums) un taisa vaicājumu EJB komponentiem. Kad pieprasījums ir apstrādātas ar visu biznesa loģiku, izmantojot EJB komponentes, servlets ģenerē atbildi midletam. Tāda paša apstrādes loģika var būt izmantojot *WEB Container* vietā *SOCKET* vai *UDP* serverus.

J2EE aplikāciju serveru galā pamatā ir 3 komponentes, kuri mijiedarbojas ar J2ME platformas klienta pusi: **SOCKET Server, UDP Server, WEB Container**. **EJB Container** konteinerā atrodas *S-EJB* objekti un *E-EJB* objekti. *S-EJB* – tas ir Session EJB un *E-EJB* – attiecīgi Entity EJB. *API* blokā tiek piedāvāti visi J2EE platformas lietojumprogrammas saskarnes.

Pamatā lietojumu arhitektūra izmantojot bezvada klientus ir līdzīga kanoniskam J2EE lietojumam, kurš izmanto 2.1.3 apakšnodaļā aprakstītu standartu. Bet gribu atzīmēt, kā izmantojot UDP serveru vai SOCKET serveru var pa tiešu izmantot visas J2EE platformas API apejot EJB Container slāni, bet tad lietojums zaudē savu vērtību svarīgos parametros, tādos kā drošība, vienkāršība, modificēšanas ātrdarbība un daudzos citos.[7]

3.4. Generic Connection Framework

Generic Connection Framework (GCF) nodrošina visu J2ME tīkla savienojuma arhitektūru. GCF tas ir interfeisu un klāšu hierarhija, kura palīdz veidot tīkla konekcijas. GCF piedāvā vispārēju pieeju savienošanai. Tas ir vispārējs, tāpēc, ka tas nodrošina kopējo API pamatu visiem bāzes savienojuma tipiem:

- ❖ Pakešu-bāzētam (datu bloki);
- ❖ Plusmas-bāzētam (datu sekvence).

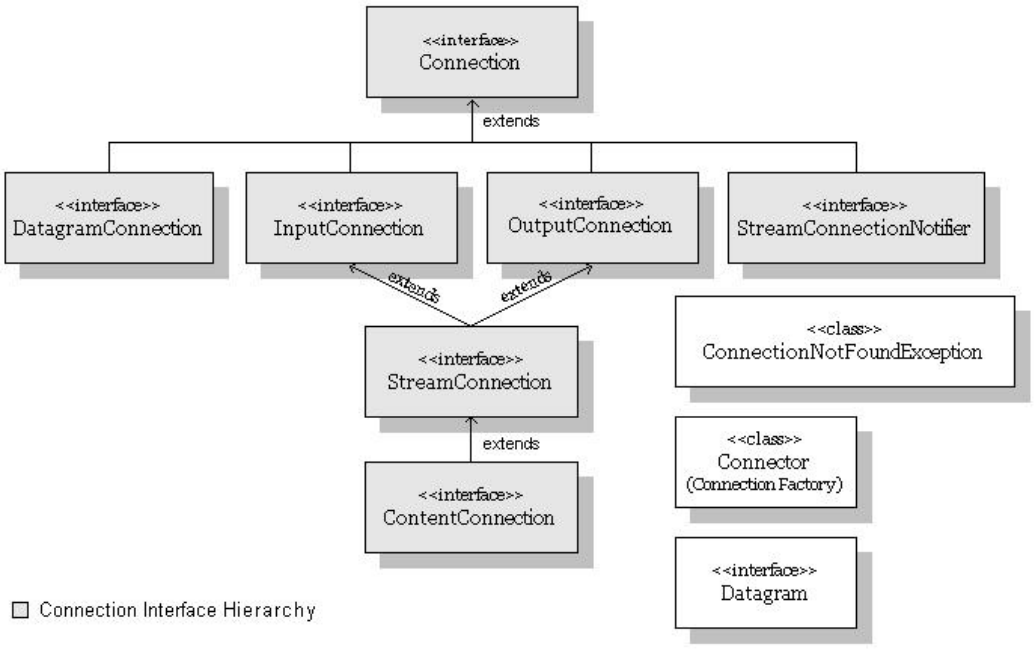
GCF vispārinājums ir iespējams izmantojot:

- ✓ Interfeisu hierarhiju, kuru var paplašināt;

- ✓ Konekcijas rūpnīcu (connection factory);
- ✓ Standart Uniform Resource Locators (URL) konekcijas tipu veidošanas nozīmēšanai.

GCF ir definēts J2ME arhitektūrā konfigurācijas līmenī. Realizējot šo framework-u Connector un pārējie interfeisi ir pieejami visos profilos. CDC un CLDC konfigurācijas atbalsta GCF.

13. attēlā ir paradīta GCF klašu un interfeisu hierarhija. [12]



13.att. Generic Connection Framework klašu un interfeisu hierarhija

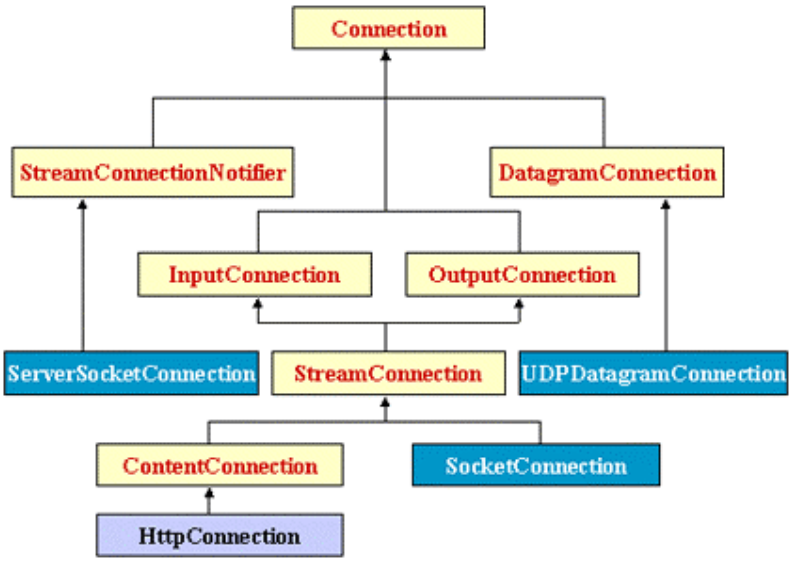
Kā ir redzams 13. attēlā hierarhijas pašā augšā atrodas *Connection* interfeiss, bāzes savienojuma tips. Visi konekcijas tipi paplašina šo interfeisu.

Priekš pakešu-bāzētam savienojuma tipam GCF definē *DatagramConnection*, un plūsmas-bāzētam tipam *InputConnection*, *OutputConnection*, *StreamConnection* un *ContentConnection*. Īpaši gribu atzīmēt *StreamConnection,* kurš paplašina uzreiz divus interfeisus *InputConnection* un *OutputConnection*. Hierarhijas paša apakšā atrodas *ContentConnection* – tas ir *StreamConnection* speciālais tips, kurš nodrošina satura-specifisku informāciju, tādu, ka: garums, satura tips, datu kodējums. Un

beidzot *StreamConnectionNotifier* veicina lietojumam gaidīt ienākošus plūsmas konekcijas asinhroni.

Papildinot konekcijas hierarhiju GCF nodrošina *Connector* klasi, kura ir augstāk minēta konekcijas rūpnīca (connection factory), un *ConnectionNotFoundException* klasi, kura tiek izmantota kļūdas gadījumā ja konekcija nevar būt izveidota. Pakešbāzētam konekcijām GCF definē *Datagram* interfeisu, kurš reprezentē datagram paketi.

3.2 nodaļā 11. attēlā bija paradīts *Binārs* ziņojumapmaiņas tips ar 3 formātiem zaļā fonā: HTTP, UDP un Socket. 14. attēlā zilā fonā ir paradītas Java klases, kuri realizē šos 3 formātus: ***HTTP*** formātu – realizē *HttpConnection* klase, ***UDP*** – *UDPDatagramConnection,* ***Socket*** formātu – *ServerSocketConnection* un *SocketConnection.*[13]



14.att. GCF klašu un interfeisu hierarhija ar triju savienojuma tipu realizācijām

14. attēlā dzeltenā fonā ir paradīti interfeisi, kuri tiek iekļauti CLDC 1.0 versijās konfigurācijā. *HttpConnection* interfeiss bija papildināts ar MIDP 1.0 versijas profilu. Bet tie interfeisi, kuri ir paradīti ar tumšu zilu krasu bija pielikti MIDP 2.0 versijā.

Connector klase tiek izmantota lai veidotu savienojuma protokola instances izmantojot vienu no *Connector* statiskam metodēm. Atgriezta instance tas ir *Connection* interfeisa realizētais pēctecis, ka ir redzams 14. attēlā. No *Connector* klases nav iespējams dabūt instanci. Visas *Connector* klases metodes ir statiskas (static). *Open()* metode atgriež *Connection* tipa instanci. Tas ir lietojumu atbildība, kuru instanci (*HttpConnection* vai *SocketConnection* vai *UDPDatagramConnection*)

sagaidīt izsaucot *open* metodi, jo *Connector* klase skatīsies tieši uz padoto parametru *name* atgriežot instanci. *Connector* definē trīs open() metodes variantus:

• *open(String name)*

• *open(String name, int mode)*

• *open(String name, int mode, boolean timeouts)*

Name īstenībā tas ir URI adrese, kura sastāv no 3 daļām: shēma, adrese un parametru saraksts. Vispārēja forma ir šāda:

```
<scheme>:<address>;<parameters>
```

scheme – identificē kā konekcija ir izveidota (http, datagram, socket un ttl.)

address – adrese kam konektēties (www.smsrelax.id.lv vai 159.148.178.24 un ttl.)

parameters – identificē citu informāciju kuru var nodot *name=value* veidā.

Adresē var būt arī noradīts porta numurs, pie kura notiek pieslēgšana. Zemāk ir paradīti piemēri:

HTTP savienojumam – izsaucot

Connector.open("http://www.smsrelax.id.lv /SomeServlet") metodi, tiek atgriezta *HttpConnection* klases instance.

UDP savienojumam – izsaucot *Connector.open("datagram://www.smsrelax.id.lv:3333")* tiek atgriezta *UDPDatagramConnection* klases instance.

Soketu savienojumam – izsaucot *Connector.open("socket://www.smsrelax.id.lv:7777")* tiek atgriezta *SocketConnection* instance.

Nākamajā daļā ir aprakstīts praktiskais darbs kur darba autors pielietojis dažas šajā darbā gan J2EE gan arī J2ME tehnoloģijas. [14]

3.5. HTTP, UDP, SOCKET protokolu ātrdarbība

Iepriekšējā nodaļā bija aprakstītas J2ME platformas Java klases, kuras ļauj realizēt midlet lietojumu savienojumus ar J2EE aplikācijas serveru izmantojot HTTP, UDP, vai SOCKET ziņojumapmaiņas formātus.

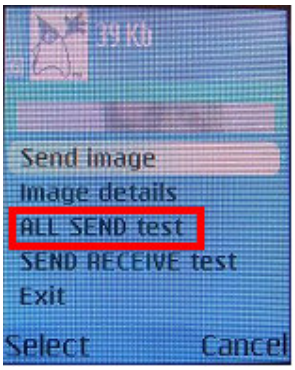
Šajā nodaļā autors nolēma pārbaudīt praksē HTTP, UDP un SOCKET protokola ātrdarbību. Pēc teorijas it kā UDP būs visātrākais protokols, jo tas neprasa atkārtotu sūtīšanu dažos gadījumos. Vai reāli UDP ir visātrākais?

Ātrdarbība būs pārbaudīta sekojoša veidā – tiek sūtīts serializētais objekts no J2ME midleta uz J2EE aplikācijas serveri 6 reizes pēc kārtas ar UDP,SOCKET,HTTP protokolu. Laiks, kurš būs starp diviem objektiem un skaitās par sūtīšanas laiku no midlet lietojuma uz J2EE aplikācijas serveru.

Par testēšanas ierīci autors paņēma NOKIA 3230 mobilo telefonu (7. pielikums)

Par J2EE aplikācijas serveri autors paņēma JBoss serveri.

15. attēlā ir paradīts midleta lietojuma ekrāns ar ALL SEND test pogu, kurš startē 3 protokolu testa sūtīšanu.



15. att. midlet lietojuma testa poga

16. attēlā ir paradīts JBoss servera konsole objektu saņemšanas laikā no J2ME.

```
INFO  [PhotoReceiver 55] @ @ @   Listening for client J2ME  accept:
INFO  [PhotoReceiveThread 190]  +++ INIT SESSION  for /212.93.97.131    +++
INFO  [PhotoReceiveThread 54] Remote address = /212.93.97.131
INFO  [PhotoReceiveThread 55] Remote port = 54932
INFO  [PhotoReceiveThread 110] ^^^^          THE USER VERSION = 1.0.0
INFO  [PhotoReceiveThread 490]  % % % % %     MOBILEPHONE USER Super man LOGINING =  7777777
INFO  [UdpServerThread 40]              +++  @@  UDP    test object sending   @@  +++
INFO  [UdpServerThread 43]             UDP    object received in  546 ms size = 1780
INFO  [UdpServerThread 43]             UDP    object received in  719 ms size = 1780
INFO  [UdpServerThread 43]             UDP    object received in  703 ms size = 1780
INFO  [UdpServerThread 43]             UDP    object received in  1141 ms size = 1780
INFO  [UdpServerThread 43]             UDP    object received in  797 ms size = 1780
INFO  [PhotoReceiveThread 97]           +++  @@  SOCKET test object sending   @@  +++
INFO  [PhotoReceiveThread 100]          SOCKET object received in  734 ms size = 1780
INFO  [PhotoReceiveThread 100]          SOCKET object received in  578 ms size = 1780
INFO  [PhotoReceiveThread 100]          SOCKET object received in  484 ms size = 1780
INFO  [PhotoReceiveThread 100]          SOCKET object received in  516 ms size = 1780
INFO  [HttpTestImageSendingServlet 53]  +++  @@  HTTP   test object sending   @@  +++
INFO  [PhotoReceiveThread 100]          SOCKET object received in  594 ms size = 1780
INFO  [HttpTestImageSendingServlet 55] HTTP   object received in  4406 ms size = 1780
INFO  [HttpTestImageSendingServlet 55] HTTP   object received in  3313 ms size = 1780
INFO  [HttpTestImageSendingServlet 55] HTTP   object received in  2812 ms size = 1780
INFO  [HttpTestImageSendingServlet 55] HTTP   object received in  3578 ms size = 1780
INFO  [HttpTestImageSendingServlet 55] HTTP   object received in  4110 ms size = 1780
```

16.att. JBoss aplikācijas servera konsole

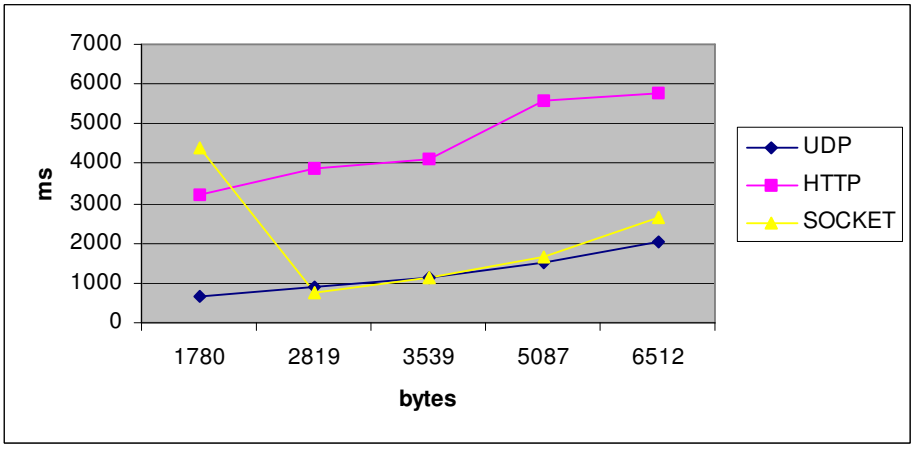
Lai būtu precīzāki mērījumi autors nolēma katru baitu izmēru notestēt 5 reizes .

Tas nozīmē, ka objektu kopējo sūtīšanu skaits būs – 5 reizēs X 3 protokolu tipus X 5 merītus objektus. Kopā vienāds ar 75 objektiem un attiecīgi būs 75 laika rezultāti, bet katram protokolam būs 25 laika rezultāti. Galīgais rezultāts būs paņemts vidējais aritmētiskais.

Zemāk ir redzama tabula kur kolonnas nosaukumi ir baitu izmērs.

	1780 b	2819 b	3539 b	5087 b	6512 b
UDP	673.4	916.92	1138.8	1534.4	2050.66
HTTP	3220.64	3888.68	4112.6	5596.379	5750.306
SOCKET	4406.25	775.56	1124.28	1677.067	2671.82

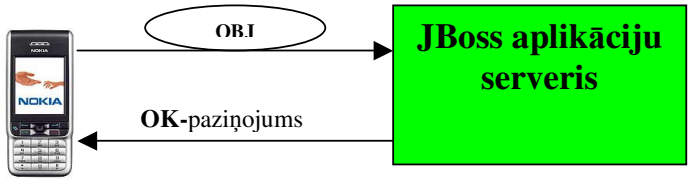
17. attēlā ir paradīts grafiks, balstoties uz tabulas datiem.



17.att. protokolu ātrdarbības grafiks

Šajā grafikā ir redzams UDP protokola līnija ap 3Kb iet viena līnija ar SOCKET līniju. HTTP krietni atpaliek no SOCKET un UDP protokola.

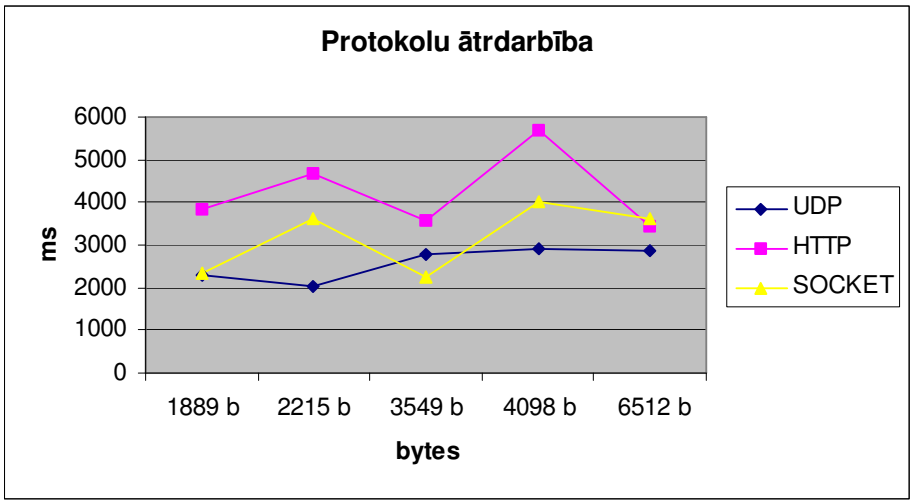
Autors nolēma vēl notestēt ātrdarbību izmantojot call-back metodi, cik aizņems pēc laika nosūtīt objektu uz J2EE aplikācijas serveri un sagaidīt no tā akceptu. Laika rezultāts būs starpība starp objektā sūtīšanu un akcepta saņemšanu midlet lietojumā. 18. attēlā ir paradīta testa sūtīšanas shēma.



18.att. testa sūtīšanas shēma

Rezultātu tabula ir redzama turpinājumā, bet grafiks 19. attēlā.

	1889 b	2215 b	3549 b	4098 b	6512 b
UDP	2281	2032	2765	2906	2859
HTTP	3828	4687	3578	5703	3453
SOCKET	2344	3625	2250	4000	3625



19.att. call-back testa resultātu grafiks

Šajā grafikā ir redzams kā tiešām UDP protokols ir ātrākais. Bet HTTP ir vislēnākais.

Testa veikšanās procesa bija novērots, ka dažreiz UDP objekti bija pazaudēti, uz konsoles nebija pilnīgi visu 5 objektu informācija. Protams daudzi faktori var ietekmēt "nejauši" uz ātrdarbību: mobila operatora tiklas paaudze, pašreizējo GPRS lietotāju skaits, signāla stiprums, atrašanas vieta, mobilas ierīces īpašības utt. Šo divu testu secinājuma autors var pateikt, kā UDP un SOCKET protokolu ātrdarbība gandrīz vienāda. Un ja ir vajadzīgi pārsūtīt datus tad drīkst izmantot SOCKET protokolu. UDP protokols nevar būt izmantots datu pārsūtīšanai. HTTP protokols atpaliek no diviem iepriekšējiem viduvēji gandrīz divreiz.

4. Projektēšanas paraugi un šabloni

Izstrādājot korporatīvus projektus kur komunicējas J2ME ar J2EE platformu, vai mobila ierīce ar J2EE platformu dažreiz rodas grūtības lietotāju saskarnes un komunikācijas protokolu izvelēs jautājumā. Dažreiz gadās, ka projekti ar līdzīgu funkcionalitāti atkārtojas, un šajā gadījumā būtu labi netērēt laiku uz citu savu risinājumu izgudrošanu, bet pielietot vienu no pārdomātiem risinājumiem, kurš tiešām atbilst prasībām un viegli ieviešami izstrādē. Šie risinājumi izstrādēs pasaulē ir zināmi kā dizaina šabloni (design patterns).

Pasaulē standartiem galddatoru serveriem, kuri bāzējas uz J2EE platformu tiek izgudroti ap 250 šablonu. Bet ir liela atšķirība starp J2EE un J2ME platformu. Mobilām ierīcēm ar J2ME platformu ir viens liels būtisks trūkums – resursu ierobežotība. Šis trūkums neļauj pilnīgi pielietot J2EE platformas šablonus J2ME vidē. Dažreiz tas prasa šablonu pārstrādi.

Šajā daļā autors grib aprakstīt dažus lietotāja saskarnes šablonus un dažus komunikācijas paņēmienus un risinājumus, kur var būt noderīgi kā paraugi kādam projektam ar līdzīgu funkcionalitātes vajadzību.

4.1. Model-View-Controller modelis J2ME platformā

Izstrādājot bezvadu lietojumus, kuri savienojas ar attālinātiem serveriem ļoti svarīgi pievērst uzmanību lietotāju saskarnes problemātiskām vietām: ātrdarbībai, vieglai modificēšanai un uzturēšanai. Šīs problēmās pasaulē tiek atrisinātas izmantojot Model View Controler (MVC) modeli. 2.1.4 apakšnodaļā bija aprakstīti divi MVC) modeli. Tie orientēti *tikai* uz J2EE platformu, parastiem galddatoru serveriem. Vienkārši pārkopēt tos J2ME platformā nav iespējams daudzu iemeslu dēļ.

Šajā nodaļā tiks paradīts J2ME platformas MVC modeļa šablonu, kuru autors pielietojis praktiskā darbā.

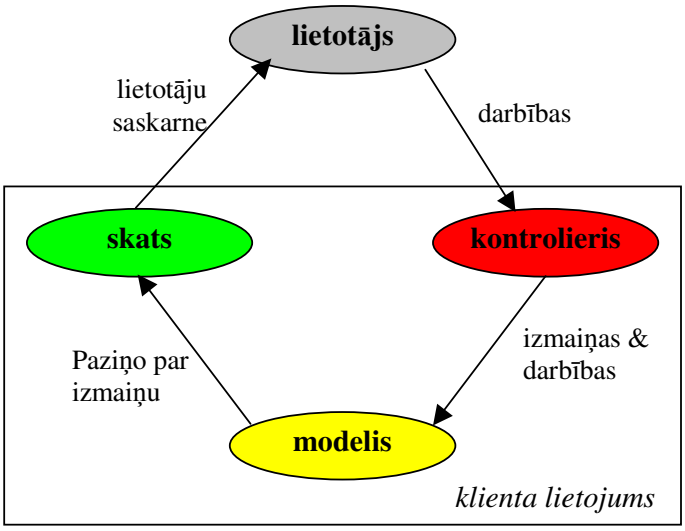
MVC šablons sastāv no trim daļām: modelis (model), skats (view) un kontrolieris (controller).

Modelis atbild par lietojuma biznesa loģiku. Modelis apstrādā datus izmantojot savu un lietojuma apakšējos slāņus biznesa loģiku.

Skats tiek domāts datu reprezentācijai lietotājam un datu savākšanai no lietotāja. Vienkārši tas ir lietotāju saskarne.

Kontrolieris apstrādā lietotāja darbības un nosūta tos tālāk modelim.

20. attēlā ir paradīts vienkāršots MVC šablons.

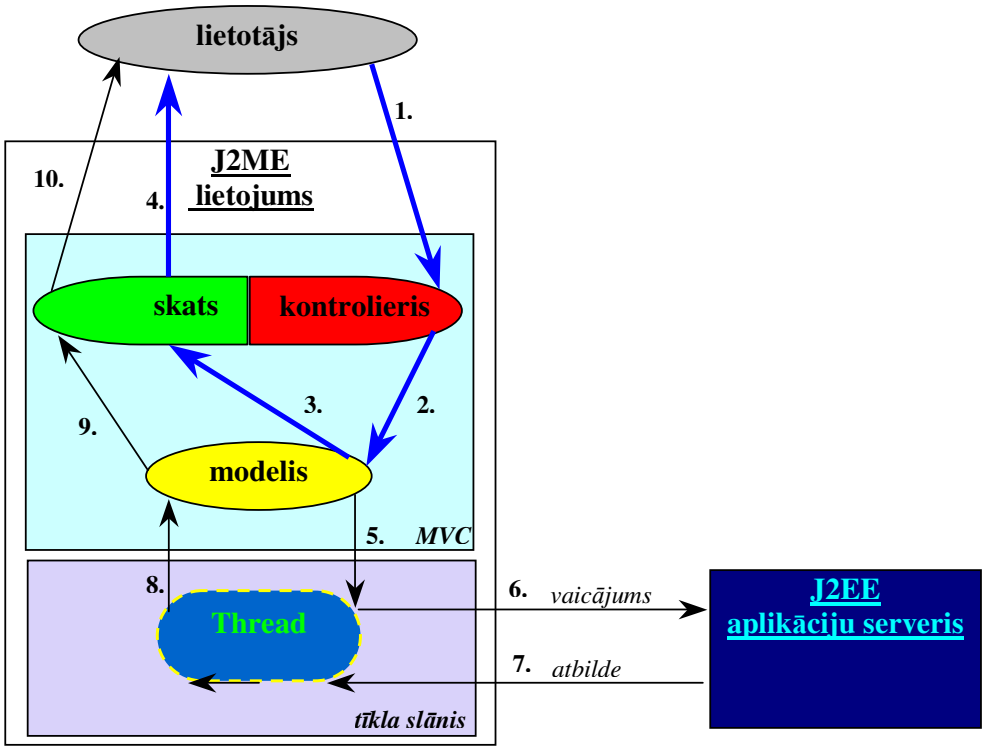


20.att. MVC vienkāršots modelis

Modeļa darbības secība ir sekojoša. Kad lietotājs izmaina datus vai nospiež pogas lietojumā, notiek dažas darbības, kuras nonāk kontrolierim. *Kontrolieris* noteic kāda veida izmaiņas ir bijušas un izsauc attiecīgu modeli. *Modelis*, pēc autora viedokļa, ir galvenā komponente šajā šablonā, īpaši ja biznesa loģika prasa savienojumu ar serveriem. Autors grib apskatīt gadījumu kad J2ME lietojuma ir vajadzīgi savienoties ar serveriem. Lietotāju darbība prasa noteiktu laiku uz visu MVC modeļa apstrādi. Visi tīkli savienojumi ir ļoti lēni, bet lietotājs negrib gaidīt un redzēt, ka lietojums it aizkārās. Uz *modeli* aiziet vislielākais laiks no visa darbības apstrādes. Tāpēc to jārealizē izmantojot Java Thread objektus, kuri apstrādās ilgākus tīkla procesus nebremzējot lietotāju saskarnes mijiedarbību ar lietotāju. Kad *modelis* ir apstrādājis izmantojot savu biznesa loģiku notiek datu nodošana *skatam,* kurš reprezentē šos datus lietotājam.

Tā kā darbā tematika saistīta ar mobilo ierīču un J2EE serveru mijiedarbību, tad darba autors grib piedāvāt J2EE MVC modeli, kurš ir orientēts tiešu uz tiklas savienojumu gadījumu.

21. attēlā ir paradīts MVC modeļa variants, kurš ir domāts mijiedarbībā ar J2EE aplikācijas serveri.



21.att. Tīkla MVC modelis

Kā bija iepriekš pateikts mobilas ierīcēs eksistē daudz ierobežojumu, arī uz J2ME lietojuma izmēru. Lai minimizētu koda izmēru attiecīgi uz Java klašu daudzumu autors nolēma skatu un kontrolieri savienot vienā komponentē. Šīs savienojums vienā komponentē realizējas vienā Java klases failā. No izmaiņu redzes viedokļa tur arī ir savas priekšrocības. Piemērām ja mainās lietotāju saskarne ar jaunu elementu izvietošanu, tad jauna elementa apstrādes pielikšana aizņem minimālu laiku, jo viss ir vienā failā, gan lietotāju saskarne, gan darbību apstrādātājs.

Tagad paskaidrošu 21. attēlu. Visas darbības autors sanumurējis lai vieglāk būtu izskaidrot attēlu.

Vispār šajā modelī lietotāja darbības apstrādi var sadalīt divos procesos:

1) Soļi – 1.2.3.4 (atzīmēts ar zilam bultiņām);
2) Soļi – 1.2.5.6.7.8.9.10.

Īstenībā divi procesi sākas jau tad kad beidzās 2. solis. Principā pirmā procesa apstrāde ir tāda pati kā un MVC vienkāršotā modelī, izņemot 3. soli.

Tagad pēc kārtas. 1. solī kontrolieris apstrādā lietotāja saskarnes notikumus un nodot apstrādei modelim (2. solis). Pieņemsim, ka notikums ir lietotāja autentifikācija. Ir vajadzīgi pārbaudīt lietotāja vārdu un paroli ar datu bāzes vērtībām serveru galā. Autentifikācijas process ir lēns sakarā ar J2ME lietojuma savienojuma izveidošanu ar J2EE serveru, datu nodošanu un atbildes saņemšanu. Bet lietotāja saskarne nevar

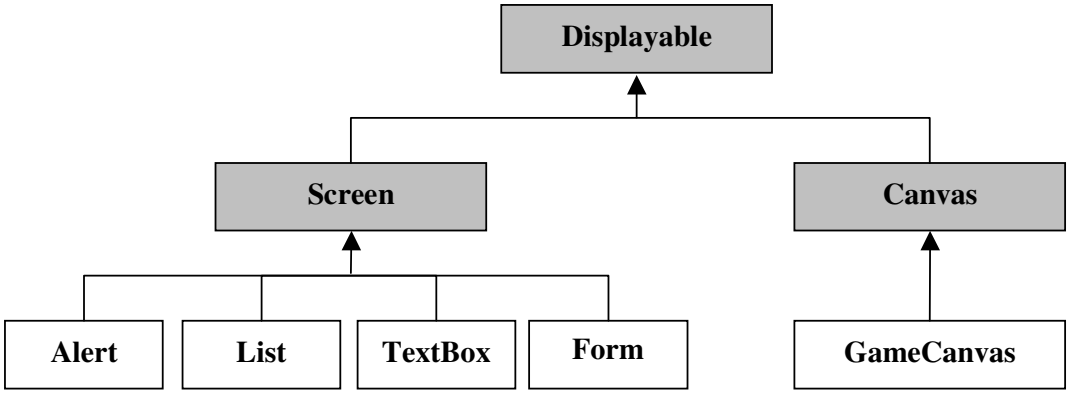
apstāties un gaidīt šo atbildi, jo ir vajadzīga pastāvīga interaktivitāte starp lietotāju un lietojumu. Tāpēc šeit sakas otrais process. Lai lietotājam paziņotu par autentifikācijas procesu stāvokli ir vajadzīgi sadalīties. Pirmais process izmantojot 3. un pēc tam 4. soli paziņos lietotājam ka viņa autentifikācija ir apstrādē. Bet otrais process ar 5. soli izveidos atsevišķu Java *Thread* objektu. *Thread* objekts inicializē savienojumu ar J2EE serveru un nosūtīs pieprasījumu (6. solis). Mūsu piemērā būs nosūtīti lietotāja vārds un parole. J2EE servera galā notiks vaicājumu saņemšana apstrāde un atbildes sūtīšana atpakaļ *Thread* objektam (7. solis). *Thread* objekts nodod to *modelim* (8. solis). Un modelis paziņo skatam kā autentifikācija ir veiksmīga (9. solis).Un skatam jāparada attiecīgu paziņojumu lietotājam (10. solis). Tādā veidā izmantojot divus procesus lietojums paliek interaktīvs ja pat otrais process aizņems daudz laiku.

Šo modeli autors pielietojis praktiskajā daļā – "Mobilais fotogrāfs" lietojumā.

Tā kā šis modelis tiek pielietots J2ME platformā, tad aprakstīsim dažas J2ME platformas Java klases kuri atbilst *skatam* un *kontrolierim* MVC modelī.

J2ME platformā ir tāda MIDP profila pakotne *javax.microedition.lcdui* kura atbilst par lietotāja saskarni.

22. attēla ir paradīta klašu diagramma, kura atbilst MVC modeļa *skata* komponentei.

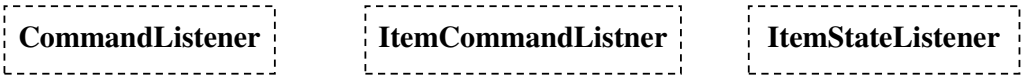


22.att. MVC Skata komponentes klašu diagramma

Šajā attēlā pelēkas ir Java abstrakta klases (*abstract class*). Vispār lietotāju saskarni var sadalīt uz augša līmeņa un apakšēja līmeņa. Saskarnes augstam līmenim atbilst *Screen* klase un viņa pēcteči, bet apakšēja atbilst *Canvas* klase un viņa pēctecis.

Praktiskā daļā *skata* komponentes gadījumā autors pielietoja *Form* un *Canvas* klases.

23. attēlā ir paradīti Java interfeisi (*interface*), kuras atbilst MVC modeļa *kontroliera* komponentei.



23.att. MVC Kontroliera komponentes klases

Praktiskā daļā kontroliera komponentes gadījumā autors pielietojis *ComandListener* un *ItemComandListener* klases.

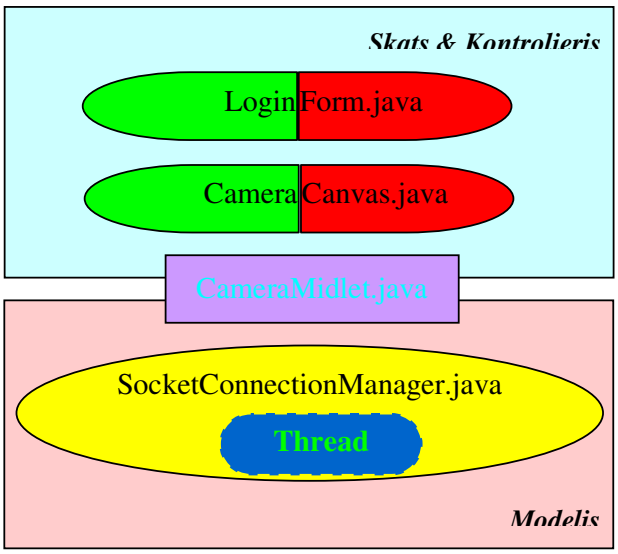
5. pielikumā atrodas praktiskas daļas pirmkods, kur Lietotāju saskarnes pakotne (camera) – tas ir pakotne ar skata un kontrolieris klasēm. Kur viena klases reprezentē gan skatu, gan kontrolieri. Bet Tīkla savienojumu pakotne (network) – pakotne ar trim modelim.

Lai viena Java klase varētu realizēt skatu un kontrolieri tad tai jāpaplašina vienu no Displayable klasēm un jāimplementē vienu no 23. attēla interfeisiem.

Tā kā praktiskā daļā autors šo MVC modeli realizējis, tad zemāk paradītas praktiski realizētas MVC modeļa Java klases.

24. attēlā ir redzamas 4 Java klases. CameraMidlet.java ir galvenā klase lietojumā. MVC gadījumā šī klase spēlē savienojuma lomu Skats&Kontrolieris ar Modelis slāni. LoginForm.java un CameraCanvas.java realizē skatu un kontrolieri.

SocketConnectionManager.java tas ir modelis, kurš atbild par socket savienojumu ar J2EE aplikācijas serveru un nodarbojas ar tīkla pieprasījumu nosūtīšanu un atbildes saņemšanu un nodošanu tālākai apstrādei *skatam*. Šajā klasē jau iekļauts *Thread* objekts.



24.att. MVC modeļa praktiskas daļas Java klases

LoginForm.java klases deklarācija ir paradīta zemāk.

```
public class LoginForm extends Form
        implements CommandListener, ItemCommandListener
```

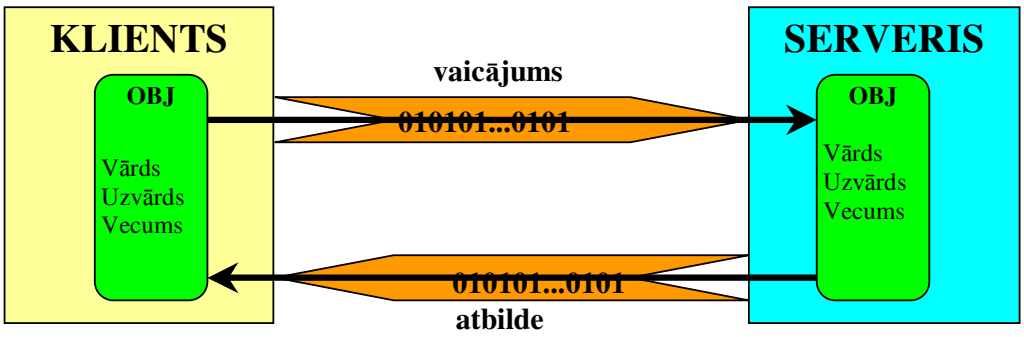
Deklarācijā ir redzams ka extends *Form* nozīme skatu realizāciju, bet implementēs *CommandListener* nozīme kontroliera realizāciju.

4.2. Objektu serializācija

Diemžēl MIDP profils neiekļauj sevī sarežģītus klient/server komunikācijas mehānismus, ka, piemēram, RMI (Remote Method Invocation) vai RPC (Remote Procedure Calls). Izstrādātājiem jāizvēlas kuru ziņojumapmaiņas formātu pielietot un kādā veidā. 3.2 nodaļa bija aprakstīti ziņojumapmaiņas formāti, un 3.4 nodaļā pamat akcents bija uzsvērts uz UDP, SOCKET, HTTP J2ME platformas klašu realizāciju. Neatkarīgi no izvelēta ziņojumapmaiņas formāta izvēles pastāv jautājums kādā struktūrā veikt vaicājums/atbilde operācijas starp klientu un serveru. Tā kā Java ir objektu orientēta valoda, tad standarta platformā J2SE bija paraudzēta iespēja apmainīties ar objektiem izmantojot objektu serializāciju. Objekta serializācija palīdz pārvaldīt un nodot objektus starp klientu un serveri vienkāršā veidā. J2SE vidē tas ir iespējams paplašinot *Serializable* interfeisu, un Java pati rūpējas par objekta serializāciju. Diemžēl J2ME platformā nebija paredzēta objektu serializācija un nav tur tāda *Serializable* interfeisa.

Šajā nodaļa autors grib paradīt kāda veidā var atrisināt šo problēmu. Praktiskā daļā autoram bija nepieciešami serializēt objektus, un viņš izmantoja tur šajā nodāļā aprakstītu šablonu.

25. attēlā ir paradīts J2ME platformas klients un J2EE platformas serveris.



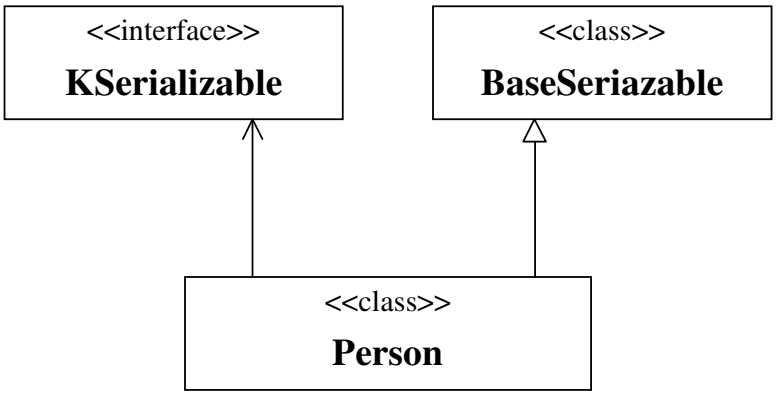
25.att. Objektu apmaiņa starp klientu un serveri

Klientā pusē tiek izveidots objekts OBJ ar trim laukiem. Šo objektu ir nepieciešami serializēt un nodot uz serveri. Servera pusē var notikt dažas objekta izmaiņas un pēc tam objekts var būt atgriezts atpakaļ klientam. Uzskatīsim, ka objekts OBJ ir Person.java fails.

Vispār lai jebkuru J2ME vides objektu varētu serializēt, ir vajadzīgi izdarīt divas darbības:

1) Implementēt KSerializable interfeisu
2) Paplašināt BaseSeriazable klasi

26. attēlā ir paradīta Objektu serializācijas klašu diagramma.



26.att. Objektu serializācijas klašu diagramma

Interfeiss *KSerializable* ir bāzes elements visā objektu serializācija. Tas definē divas metodes, kuras jāimplementē katrā objektā: *serialize* un *deserialize,* Zemāk ir paradīts *KSerializable* interfeisa kods.

```
public interface KSerializable {
    byte[] serialize() throws IOException;
    void deserialize(byte[] data) throws IOException;
}
```

Zemāk ir paradīts *BaseSeriazable* klases kods. Pamatā tas ir palīg klase serializētam objektam, kur var būt dažas objektu lauku vērtību pārbaudes – sarežģītas un triviālas.

```
public class BaseSeriazable {
 public String getValidValue(String s) {
        return s == null ? "" : s;
    }
}
```

Un beidzot zemāk ir paradīts klases *Person* kods.

```
public class Person extends BaseSeriazable implements
KSerializable {
    private String name;
 private String surname;
    private int age;

    public Person() {
    }
    public Person(String name, String surname) {
        this.name = name;
        this.surname = surname;
    }

// Šeit iet GET SET metodes katram laukam
    public byte[] serialize() throws IOException {
         ByteArrayOutputStream bout = new
        ByteArrayOutputStream();
        DataOutputStream dout = new DataOutputStream(bout);
        dout.writeUTF(getValidValue(name));
        dout.writeUTF(getValidValue(surname));
        dout.writeInt(age);
        dout.flush();
        byte [] result = bout.toByteArray();
        try {
            dout.close();
        } catch (IOException e) {
        }
        return result;
    }

    public void deserialize(byte[] data) throws IOException {
        ByteArrayInputStream bin = new
        ByteArrayInputStream(data);
        DataInputStream din = new DataInputStream(bin);
        name = din.readUTF();
        surname = din.readUTF();
        age = din.readInt();
        try {
            din.close();
        } catch (IOException e) {
        }
    }
}
```

Ar treknrakstu autors atzīmēja divus pašas galvenās metodes objektu serializācijā. Kā ir redzams serializācijā tiek pielietoti divas klases *ByteArrayOutputStream* un *DataOutputStream*. Īpaši gribu atzīmēt kārtību pēc kura tiek ierakstīti objekta lauki *DataOutputStream* objektā *serialize* metodē un kāda veidā tiek nolasīti no *DataInputStream* objektā *deserialize* metodē. Šis princips ir lādzīgi FIFO (first in first out) metodei.

Tātad jebkurā gadījumā starp klientu un serveri pa tīklu būs pārraidīts baitu masīvs. Izmantojot metodi *serialize* var dabūt šo baitu masīvu un jau sūtīt to izmantojot bināru ziņojumapmaiņas formātu. Zemāk ir paradīts klienta puses kods.

```
Person bruceLe = .....; // personas instance
try {
    byte[] byteArray = bruceLe.serialize();
} catch( java.io.IOException e ){
}
```

Pieņemejoša pusē saņemt baitu masīvu un ar metodi *deserialize* dabūt objektu tajā pašā stāvoklī, kurš bija pirms sūtīšanas. Zemāk ir paradīts servera puses kods.

```
byte[] bruceLeByteArray = .....; // saņemtais baitu masīvs
try {
     Person bruceLe = new Person();
    bruceLe.deserialize(bruceLeByteArray);
} catch( java.io.IOException e ){
}
```

Praktiskā daļā darba autors izmantojis objektu serializāciju. 5. pielikumā attēla kreisajā pusē var redzēt J2ME lietojuma Java pakotnes, kur *Palīg info objektu pakotne (util)* atrodas *Kserializable* un *BaseSeriazable* klases, bet *Info objektu pakotne(info)* atrodas serializētie info objekti.

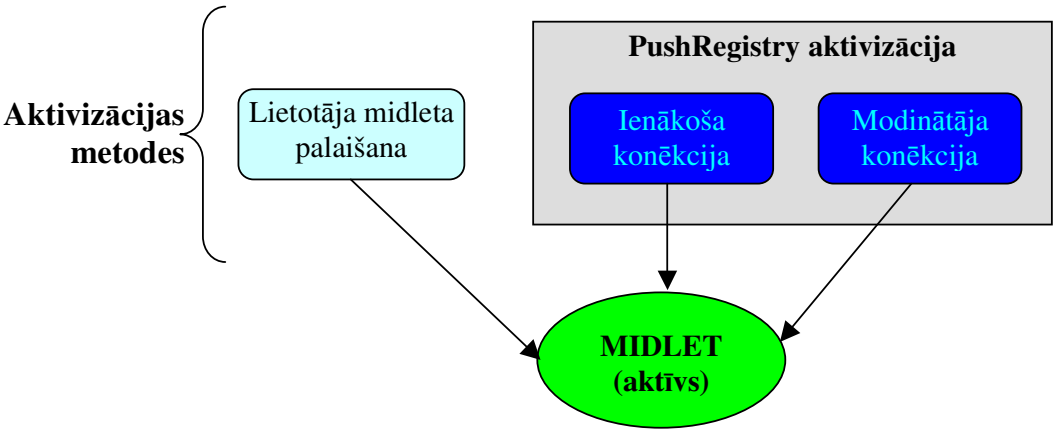
4.3. Push registry tehnoloģija

MIDP 1.0 profilā komunikācija ar serveri notiek tikai tad, kad klients(mobila ierīce) ar midlet lietojuma palīdzību iniciē savienojumu ar J2EE aplikācijas serveri. Sanāk, ka serveris nevar pirmais iniciēt komunikāciju ar midletu. Šīs ierobežojums bija atrisināts MIDP 2.0 versijā ar *Push registry* palīdzību. Šajā nodaļā autors pastāstīts par to mehānismu un kādas iespējas atver šīs risinājums.

Push registry ļauj midlet lietojumam uzstādīt savu automātisku palaišanu bez lietotāja darbības. Tas pārvalda tīkla un laika bāzētu midlet aktīvāciju. Midlet lietojuma aktivizācija notiek divos gadījumos:

1) ar ienākošo tīkla savienojumu;
2) kad nostrādā modinātājs mobilā ierīcē.

27. attēlā ir paradītas visi trīs iespējamie midleta aktivizācijas veidi.



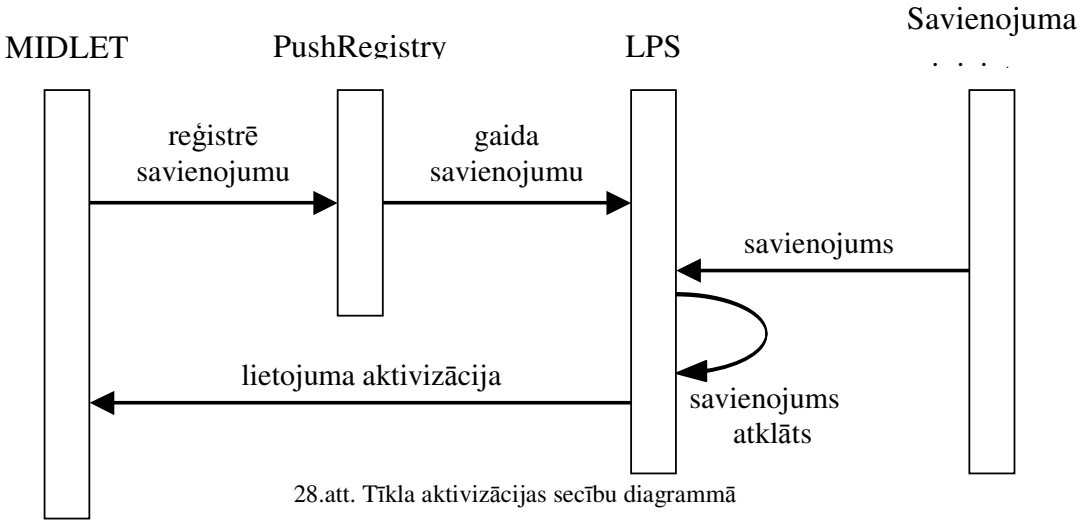
27.att. midlet lietojuma aktivizācijas veidi

Push registry atļauj ienākošus tīkla savienojumus un laika bāzētu pēc modinātāja darbības. Modinātāja bāzēta midleta palaišana var būt pielietotā dažādos sinhronizācijas veidos. Bet ienākošo tīkla savienojuma midleta palaišana atver plašu un lielu servera pakalpojuma spektru mobiliem klientiem. Tādā veidā gandrīz jebkurā mobilā ierīcē kļūst par mazo serveru ar ierobežoto funkcionalitāti! Ar šo iespēju mobila ierīce palielina savu lomu bezvadu lietojumu komunikācijā.

Push registry ir lietojumu pārvaldības sistēmas (LPS) komponente un daļa no GCF, kurš bija aprakstīts 3.4 nodaļā. LPS – ir mobilas ierīces programmatūra, kura atbild par katru lietojuma dzīves ciklu (uzstādīšana, palaišana, izpildīšana, dzēšana).

kurš bija aprakstīts 3.4 nodaļā. LPS – ir mobilas ierīces programmatūra, kura atbild par katru lietojuma dzīves ciklu (uzstādīšana, palaišana, izpildīšana, dzēšana).

MIDP 2.0 profilā atbildība par push-konekcijām sadalīta starp midlet lietojumu un LPS. LPS kontrolē reģistrētus push-notikumus izmantojot midlet lietojumu. Kad push-notikums tiek atklāts, LPS aktivizē attiecīgu lietojumu lai apstrādāt to. 28. attēlā ir paradīta tikla aktivizācijas secību diagramma.



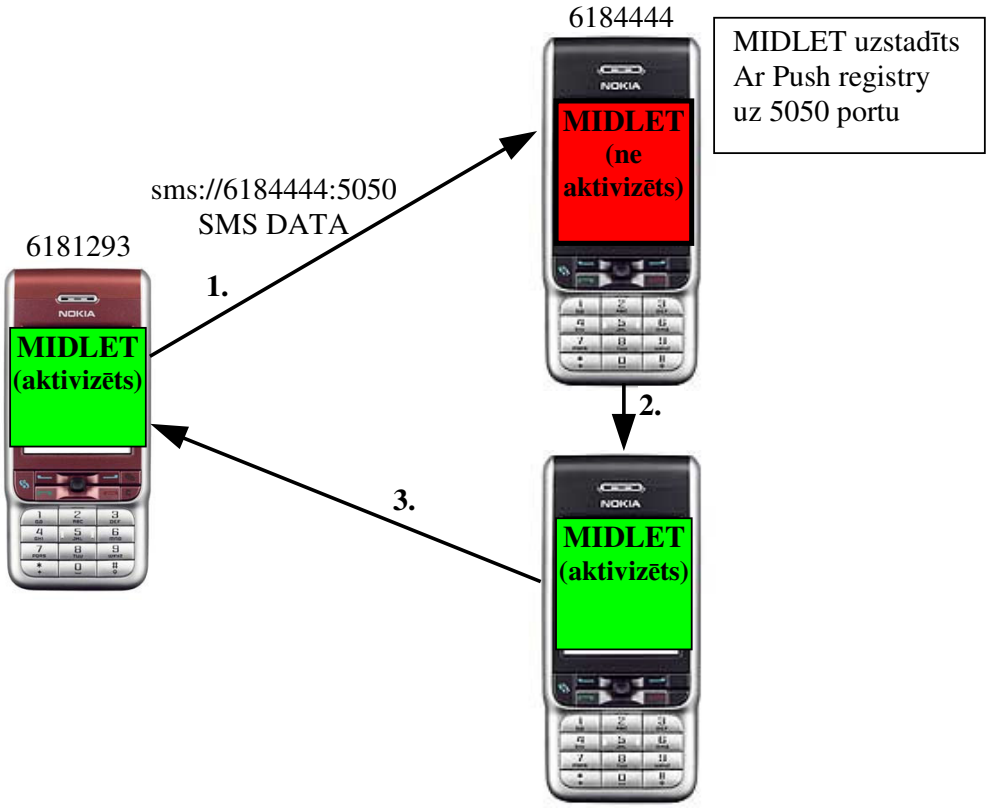
28.att. Tīkla aktivizācijas secību diagrammā

push-notikums tiek atklāts, LPS aktivizē attiecīgu lietojumu lai apstrādāt to. Ienākošam savienojumi var būt sekojoši: HTTP, SOCKET, UDP un SMS.

Es gribu pievērst uzmanību, ka SMS-bāzēta lietojumu aktivizācija ir viegli realizējamā salīdzinot ar citiem trim.

Ja mobilā ierīcē ir WMA lietojumprogrammu saskarne, tad LPS var piereģistrēt midletu, ka tam jāaktivizējas kad būs saņemts SMS uz noteikto portu ar noteiktu saturu.

29. attēlā ir paradīts piemērs starp diviem dažādiem mobiliem telefoniem ar numuriem 6181293 un 6184444. Melnajā mobilā telefonā ir uzstatīts midlet lietojums ar Push registry iespēju uz 5050 portu. Tas nozīmē, ja šim telefonam atnāks īsziņa uz šo portu, tad būs aktivizēts šis midlet. Nosūtīt īsziņu uz portu var izmantojot WMA neobligātas pakotni. Sarkanais telefons ar numuru 6181293 sūta SMS melnajam. Pēc kā LPS apstrādā to un noteic, ka tā ir SMS uz 5050 portu, aktivizē midlet un midlet uzsāc savas biznesa loģikas darbības.



29.att. midlet lietojuma aktivizācija izmantojot WMA API

4.4. SMS īsziņu apstrāde izmantojot Gammu un J2EE lietojumus

SMS pakalpojumi mūsdienas tiek izmantoti visur, kur ir iespējama informāciju apmaiņa. Dažreiz rodas nepieciešamība realizēt savos esošos projektos papild pakalpojumu SMS veidā, kur SMS kalpos sistēmai, ka pieprasījums kādai informācijai un atbilde uz pieprasījumu.

Šajā nodaļā autors grib piedāvāt paraugu, izmantojot kuru var ieviest jaunu SMS pakalpojumu relatīvi viegli esošai sistēmai.

Paraugā ļoti lielu lomu spēlē Gammu produkts. Gammu – tas ir projekts, kur izveidoti lietojumi, skripti un draiveri tiek izmantoti lai pārvaldītu visas iespējamas funkcijas mobilos telefonos un citās mobilās ierīcēs. Pašlaik tas piedāvā stabilu un gatavu bāzi daudziem mobilu telefonu modeļiem. Gara un ilgstoša izstrāde orientēta uz lietojumprogrammu saskarnes API izstrādi, nevis uz kādu noteiktu mobila telefona modeli uzturēšanu. Gammu ir uzprogrammēts izmantojot C++ valodu. [15]

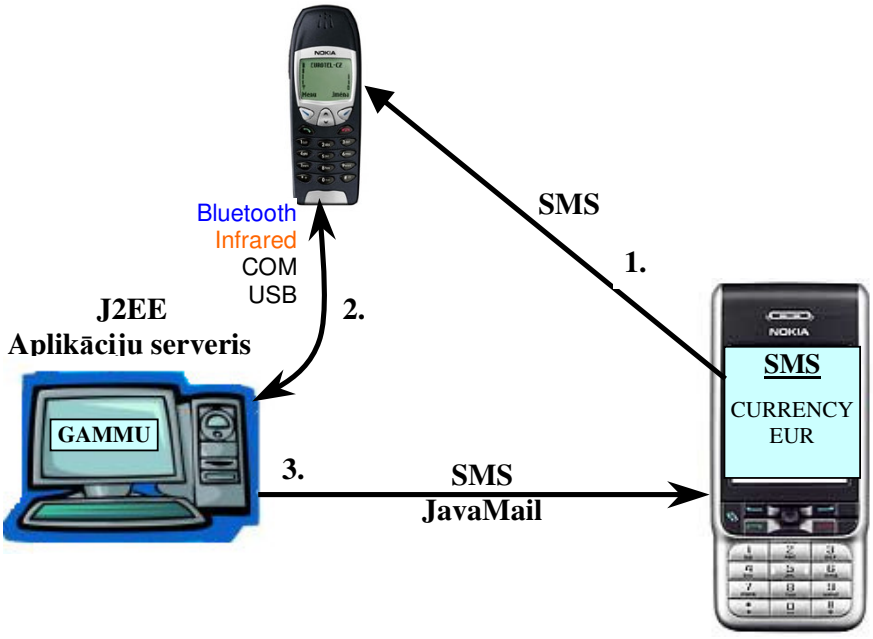
Paraugā pielietoja nepieciešamu Gammu produktu funkcionalitāti, kura saistīta ar SMS nolasīšanu no mobila telefona un ierakstīšanu datu bāzē.

Parauga nepieciešamas sastāvdaļas ir dators, kur tiek palaists J2EE aplikāciju serveris ar izstrādāto lietojumu un mobilais telefons, kurš pieslēgts datoram ar datu kabeli, vai ar Bluetooth, vai Infrared savienojumu. Lai izmantotu Gammu SMS funkcijas, ir vajadzīgi datorā uzstādīt My SQL datu bāzi un palaist Gammu sagatavotu skriptu, kurā ir sadefinēti visas nepieciešamas tabulas.

Vispār SMS vaicājuma/atbildes procesu var sadalīt trijos posmos.

1) Pieprasījums SMS-veidā;
2) SMS apstrāde izmantojot Gammu;
3) SMS atbilde izmantojot JavaMail API.

Visi 3 posmi ir paradīti 30. attēlā.



30.att. SMS apstrāde izmantojot Gammu

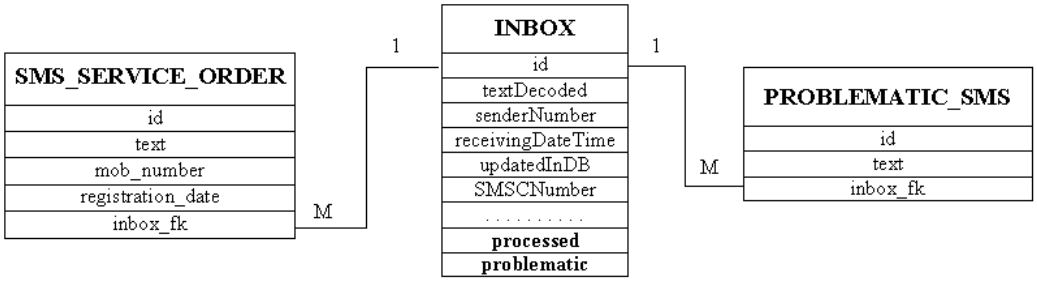
Detalizētāk autors grib apskatīt 2. posmu. Šis posms sastāv no divām daļām. Vienu daļu izpilda Gammu, otru daļu izpilda Java lietojums. No sākuma pastāstīšu par pirmo daļu.

Gammu produkta kodols viens *exe* fails un divi konfigurācijas faili. Viena konfigurācijas failā atrodas uzstādījumi: pieslēgta mobila telefona modelis, pieslēguma veids, un dažādi sinhronizācijas parametri. Otrajā konfigurācijas failā atrodas uzstādījumi, kas saistīti ar SMS funkcijām un datu bāzes parametri.

Palaižot *exe* failu Gammu skatās jaunus SMS īsziņas mobilā telefonā un automātiski saglabā tos datu bāzē un nodzēs no mobila telefona. SMS konfigurācijas failā var uzstādīt iterācijas periodu sekundēs, pēc kura tiek pārbaudīti jaunie SMS mobilā telefonā(piemēram ik pēc 5 sekundēm).

Otrajā daļā nostrādā Java lietojums, kurš arī ar noteiktu konfigurējamo periodu apstrādā datu bāzi. Lietojumā nostrādā Thread objekti ar savi biznesa loģiku, kuri cikliski atkārtojas pēc noteiktā perioda. Sakarā ar Java lietojumu biznesa loģiku nākas pielikt divas laukus esošai Gammu INBOX tabulai un izveidot divas citas tabulas.

31. attēlā ir paradītas tabulas un viņu attiecības.



31.att. SMS apstrādei nepieciešamas datu bāzes tabulas

Šajā attēlā INBOX tabula ir izveidota ar Gammu piedāvāto skriptu, bet SMS_SERVICE_ORDER, un PROBLEMATIC_SMS ir Java lietojumam divas izveidotas tabulas. INBOX tabulā ar treknrakstu ir redzami 2 jaunie lauki: *processed* un *problematic*. Šie lauki ir *boolean* tipa lauki. Kad Gammu nostrādā un ierakstā jaunu SMS INBOX tabulā, tad *processed* laukā vērtība ir 0 (false). Pēc Java lietojuma apstrādes lauka vērtība mainās uz 1 (true). Kad SMS īsziņa ir apstrādāta, tā tiek ierakstīta tabulā SMS_SERVICE_ORDER ar referenci uz INBOX ierakstu. Lai apstrādātu SMS ir vajadzīgi specificēt SMS saturu. Piemērām CURRENCY EUR nozīmē – klientam ir vajadzīgi uzzināt eiro valūtu kursu. Var būt gadījumi kad SMS neatbilst satura specifikācijai. Tad šis SMS būs atzīmēts kā kļūdains *problematic* laukā ar vērtību 1 (true). Ja SMS īsziņas saturs atbilst SMS satura specifikācijai tad vērtība ir 0 (false). Ja SMS ir kļūdaina tad, pēc apstrādes tā tiek ierakstīta PROBLEMATIC_SMS tabulā ar referenci uz INBOX tabulu.

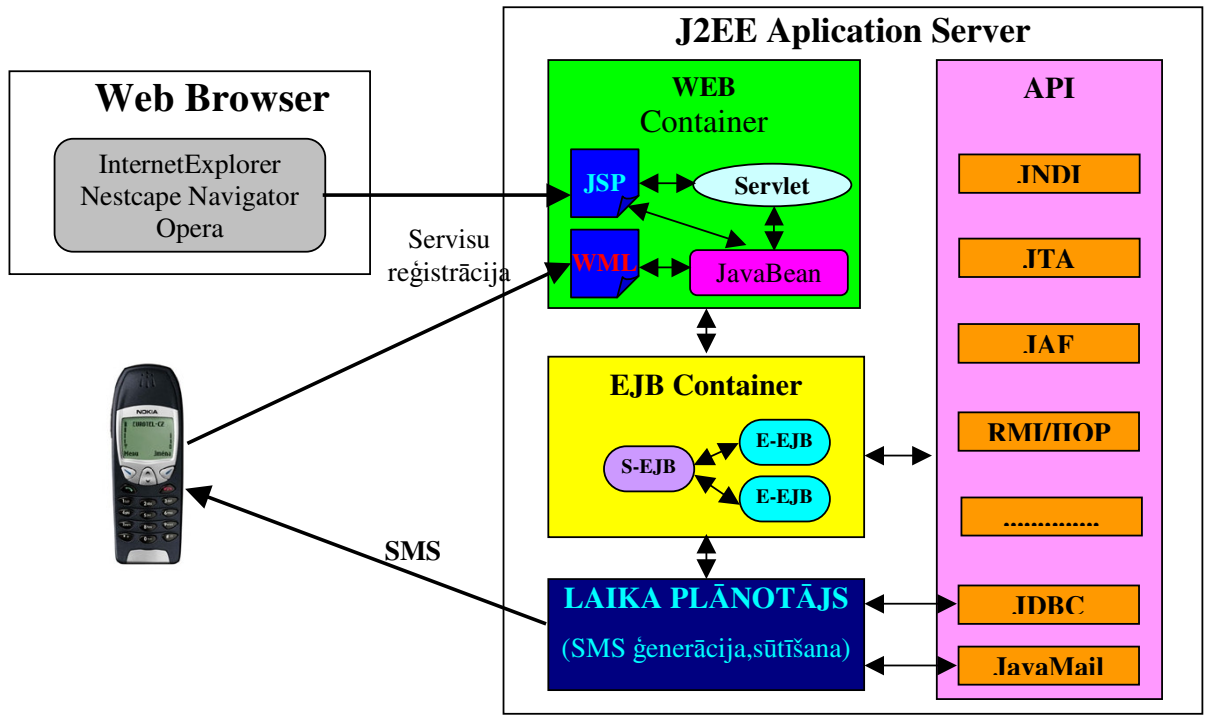
Kad īsziņa ir apstrādāta un atbilst SMS satura specifikācijai, tad izmantojot biznesa loģiku ģenerējas SMS atbilde. Gammu nodrošina SMS sūtīšanu izmantojot pieslēgto mobilo telefonu. Mūsu mobilie operatori piedāvā īsziņas sūtīšanu izmantojot e-mail. J2EE aplikācijas serveris nodrošina JavaMail API, izmantojot kuru var sūtīt e-mail, tāpēc izmantojot JavaMail API var nosūtīt atbildes īsziņu.

4.5. Automatizēta SMS sūtīšana izmantojot J2EE lietojumu

Iepriekšējā nodaļā bija apskatīts paraugs kad SMS atbilde atnāk tikai pēc klienta pieprasījuma. Tas ir ērti tajā gadījumā kad klientam ir nepieciešami uzzināt informāciju. Bet dažreiz klientam šī informācija ir vajadzīga ar noteikto biežumu, piemērām reizi dienā vai tikai noteiktajās dienās. Lai katru reizi netaisīt vaicājumu, būtu labi automatizēt šo SMS sūtīšanas procesu.

Šajā nodaļa autors paradīs paraugu, izmantojot kuru var automatizēt SMS sūtīšanu izmantojot JavaMail API.

32. attēlā ir paradīti parauga sastāvdaļas un to darbības princips.



32.att. Parauga sastāvdaļas un to mijiedarbība

Paraugs sadalīts 4 sastāvdaļās:

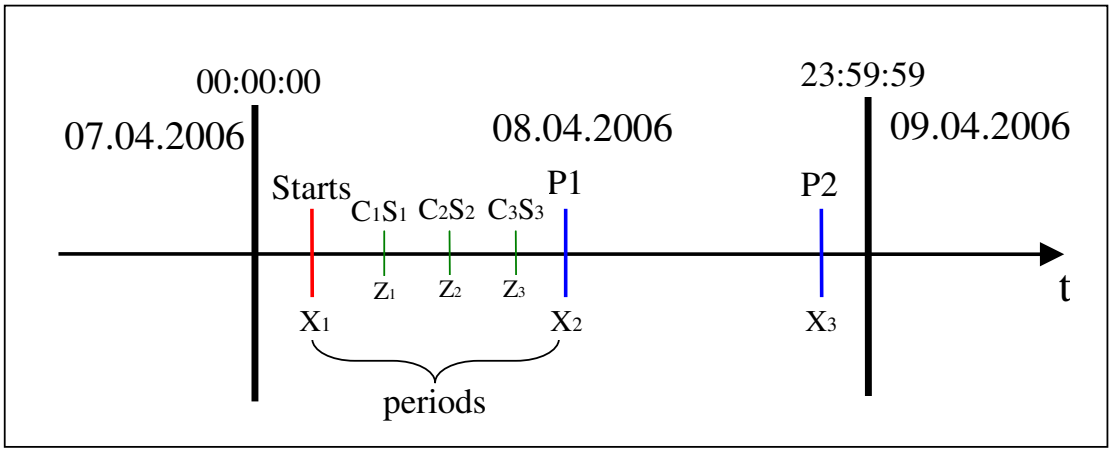
1) Pārlūkprogramma;
2) J2EE lietojums;
3) J2EE lietojuma laika plānotājs;
4) Mobilais telefons.

Izmantojot pārlūkprogrammu lietotāji var aiziet Internetā uz Web-vietni un piereģistrēt servisus. Servisi – tas ir pakalpojumi, kuri būs nosūtīti lietotājam SMS veidā. Arī ir iespēja piereģistrēt servisus izmantojot mobila telefona pārlūkprogrammu. Servera galā to nodrošina Wireless Markup Language (WML) - bezvadu pārraides iezīmēšanas valoda.

3. pielikumā var redzēt J2EE lietojuma galvenās lappuses. J2EE lietojums pilnīgi atbilst 2.1.3 apakšnodaļā aprakstītam standartam. Lietojumā bija pielietots Model 2 MVC modelis, kurš bija aprakstīts 2.1.4 apakšnodaļā. Tur pielietoti visas standarta sastāvdaļas. J2EE lietojuma galvenā nozīmē ir jauno klientu reģistrācija un esošo klientu servisus reģistrācija un apkalpošana. 3. pielikumā apakša ir INBOX lappuse

kur notiek klienta apkalpošana. 4. pielikumā var apskatīt J2EE lietojuma datu bāzes shēmu ar tabulas paskaidrojumiem.

Svarīgo un galveno lomu J2EE lietojumā spēlē Laika Plānotājs (LP). LP darbs ir automatizēta SMS ģenerācija un sūtīšana. Java programmēšanas valodā eksistē objekts *Timer*, kurš ļauj darboties ar laiku. Galvenā procedūra šajā objektā ir *schedule,* uzdevuma plānošana noteiktajā laikā. Kad šis laiks pienāks, tad tiks iedarbināts uzdevums, kurš bija saplānots. Laika plānotāja darbs pilnīgi balstās uz *Timer* objekta darbību. Laika plānotāja darbs laika formātā ir redzams 33.attēlā.



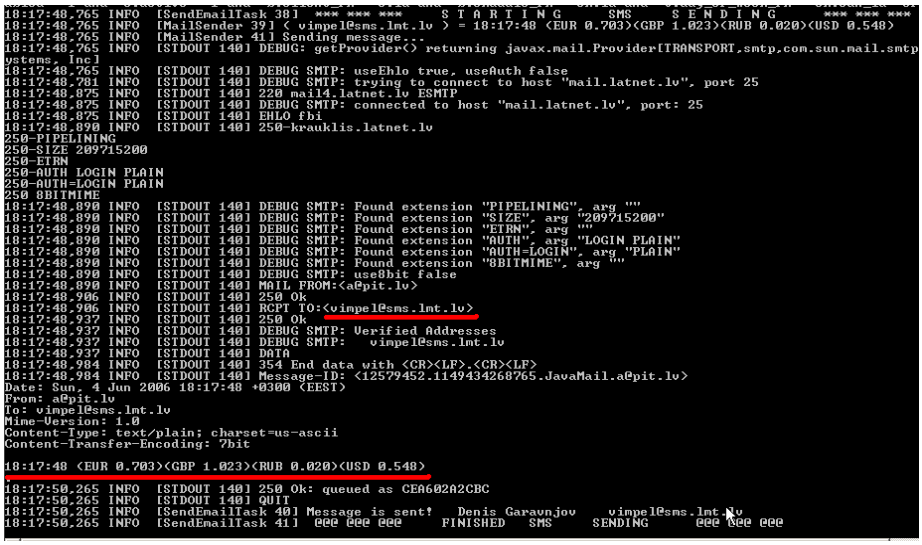
33.att. Laika plānotāja darbs

Šajā attēlā ir redzamas trīs dienas: 07.04.2006, 08.04.2006, 09.04.2006. X_1, X_2, X_3 nozīmē laika momentus: laika planētāja starts (Starts), 1. pārplānojums (P1), 2. pārplānojums (P2) – formātā: stundas:minūtes:sekundes (HH:MM:SS). Z_1, Z_2, Z_3 nozīmē laika momentus, kad laika plānotājs saplānojis sūtīt: 1. klientam (C_1) servisu S_1; 2. klientam (C_2) servisu S_2; 3. klientam (C_3) servisu S_3. *t* burts nozīme laiku. *periods* ir tabulas ***CONFIG*** *shadulePeriod* lauka vērtība. 4. pielikumā ir redzama datu bāzes struktūra. Šajā laika *periodā* tiek saplānoti uzdevumi uz izpildīšanu.

Paņemsim par tekošo dienu 8. aprīli. Ar sarkano svītriņu ir apzīmēts laika plānotāja starts – tas ir laika moments X_1, kad tika palaists laika plānotāja darbs. X_1, X_2, X_3 laika momentā laika plānotājs izsauc procedūru, kura meklē datu bāzē lietotājus, kuri bija pasūtījuši servisus laika periodā starp X_1 un X_2. Ja tādi lietotāji ir atrasti, tad katram no tiem (C_1, C_2, C_3) tiek saplānots savs uzdevums, kurā izriez tiek ielikts saģenerētais īsziņas teksts. Kad uzdevuma laiks pienāks, SMS būs nosūtīta klientam. Šī procedūra ir galvenā laika plānotāja darbā. Šīs procedūras beigās laika plānotājs

plāno to pašu procedūru izsaukšanu pēc noteikta *perioda*. Un tas atkārtojas tik ilgi, kamēr nebūs apturēts laika plānotājs. Tieši tāpēc laika plānotājs darbojas autonomi un nepārtraukti.

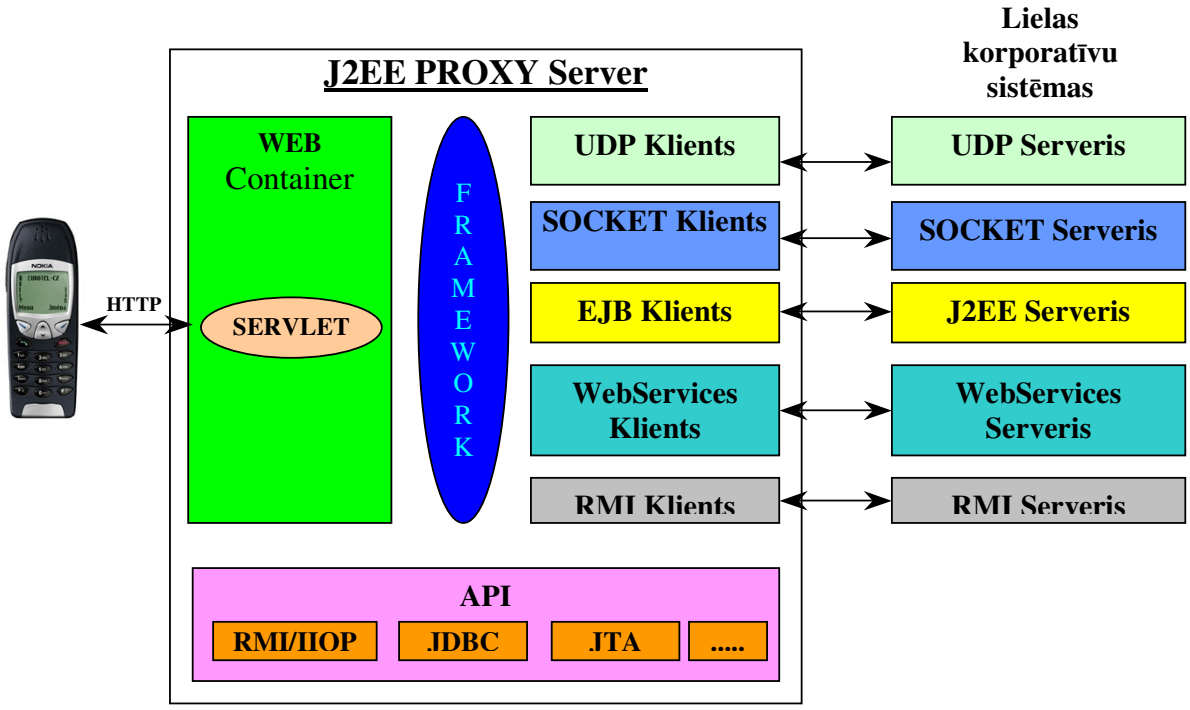
34. attēlā ir paradīta J2EE servera konsoles fragments, kur ir paradīts moments kad laika plānotājs sūta īsziņu e-mail veidā. Pirmā sarkana līnija norāda uz e-mail adresi kurš atbilst LMT mobila telefona numuram (vimpel@sms.lmt.lv atbilst 6181293 numuram). Ar otro sarkano līniju ir paradīts īsziņas teksts.



34.att. J2EE servera konsole ar nosūtītu elektronisko vēstuli

4.6. J2EE Proxy servera paraugs

Bezvadu ierīču programmatūra var paplašināt uzņēmuma patērētāju vai pircēju loku. Dažreiz uzņēmumiem eksistē jau esošas IT sistēmas, kuras tiek ieviestas un strādā ilgu laiku. Tas stabili nostiprinājās uzņēmumā un apmierina uzņēmuma biznesa prasības. Bet var gadīties, ka uzņēmumu valde nolēma attīstīt biznesu izmantojot mobilas ierīču korporatīvus risinājumus, bet negrib tērēt laiku vai/un naudu uz veselu klient/serveris projektu izstrādi. Tā kā HTTP protokolu atbalsta ***visi*** mobilie telefoni ar Java atbalstu. Tad 35. attēlā ir paradīts risinājums šim gadījumam.



35.att. J2EE servera Proxy paraugs

Attēlā ir redzams, ka Proxy serveris var būt risinājums starp mobiliem ierīcēm un jau esošām jau sen izstrādātām lielām korporatīvām sistēmām. 3.2.2 apakšnodaļā bija apskatīti XML-bāzētie ziņojumapmaiņas formāti. Tur bija pateikts, ka jau eksistē J2ME risinājumi, kuri ļauj izmantot WebServices pakalpojumus mobilās ierīcēs. Ja runājot par mobiliem telefoniem, tad tādi mobilie telefoni, kuri ļauj apstrādāt XML-bāzētus formātus ir ļoti dārgi un ne visiem pircējiem tie ir vai būs, bet mūsdienas gandrīz visi telefoni ir ar Java un attiecīgi ar HTTP protokola atbalstu. Attēlā ir paradīti 5 gadījumi ar lielām korporatīvu sistēmām, bet protams tos var būt vairāk. Visus nav iespējams aprakstīt. Galvenā sarežģītība Proxy servera realizācija būs saistīta ar *Framework* moduļa izstrādi. Jo šajā modulī notiks konvertācija starp HTTP vaicājumu un socket, RMI vai EJB izsaukumu.

5. Dizaina un implementācijas vadlīnijas

Starp standartu J2SE vai J2EE platformas izstrādātājiem pastāv problēma izstrādājot J2ME platformas lietojumus. Vairuma gadījumos tas ir projekta lietotāju saskarnes dizaina un izstrādes problēmā. Programmētāji ir pieradījusi izmantot jaudu un resursu potenciāli izstrādājot korporatīvus lietojumus parastiem galddatoriem, un kad ir nepieciešami parorientieties uz J2ME platformu, tad rodas daudz problēmas ar projektēšanu, tehnoloģiju izvēli izstrādēs fāzē un pamatā ar komunikācijas modeli izvēli starp mobilām ierīcēm un serveriem. Jo mobilās ierīces resursu ziņā nav tik jaudīgas ka parasti galddatori. Tāpēc šajā daļā autors grib aplūkot dažas vadlīnijas un paņēmienus izstrādājot korporatīvus risinājumus ar mobilām ierīcēm, kuri var būt noderīgi programmētajiem pariešanas periodā no J2EE vai J2SE uz J2ME platformu. Autors mēģinās šos paņēmienus izklāstīt soļu veidā pēc kārtas.

5.1. Implementācijas vadlīnijas

Vairāk aprēķinus servera pusē.

Vajag izvairīties lielas informāciju apstrādes mobilā ierīcē. Atrast pareizo balansu jautājumā – kuras darbības veikt servera pusē un kuras mobilā ierīcē ir ļoti grūts uzdevums. Šajā jautājumā jāņem vērā ne tikai mobilas ierīces CPU jaudu, operatīvo atmiņu, bezvadu sakaru atbalstu tehnoloģijas, bet vēl savienojuma kvalitāti ar serveru (mobila operatora bezvadu sakaru paaudze 2G vai 3G, kāds reālais datu pārraides ātrums, kādas standartus tas nodrošina).

Vienkāršot lietojumu.

Šo soli vislabāk taisīt lietojuma projektēšanas fāzē. Acīmredzama darbība šajā solī ir nevajadzīgas funkcionalitātes dzēšana vai izvairīšana. Vajag *katru* lietojama funkcionalitāti rūpīgi izpētīt. Vai lietotājs varēs apieties bez tas savā lietojumā?
Ja funkcionalitāte ir brīžiem vajadzīga, tad to var mēģināt iznest otra palīg lietojumā. Lietotājs pēc tam pats novērtēs vai ir šī palīg lietojums viņam vajadzīgs, ja nav tad pats vienkārši nodzēsīs to. Vajag mēģināt grupēt biežāk izmantotas lietojuma

funkcijas tādā veidā, lai tie būtu ātri izsaucami nospiežot dažas pogas vai izveidojot "karstas taustiņus" (hot keys).

Vairākkārtēja lietotāju saskarnes izmantošana.

Ja lietotāja saskarnē tiek novērota atkārtota pogu vai formas radīšana, tad jāprojektē tāda veidā lai atkārtota lietotāju saskarnes daļa būtu inicializēta vienreiz, bet izmantota vairākas vietās. Tas ne tikai samazina lietojuma koda apjomu, bet arī padara lietojumu vieglāk lietotājam apmācības gaitā.

Lietojuma izmēru jātaisa mazāk.

Mazs lietojuma izmērs ņem mazāk mobilas ierīces atmiņu, un tas prasa mazāk laiku lietojuma uzstādīšanai. Vēl pie tā palaižot lietojumu mazāk laika jāgaida kad tas pielādēs visas nepieciešamas klases Java virtuāla mašīnā. Un no cenas viedokļa tas arī izdevīgāk. Tā kā pagaidām GPRS trafiks iet pēc megabaitiem, tad neliela izmēra lietojumu pielādējot izmantojot bezvadu sakaru maksās mazāk.

Jāizmanto mazāk operatīvas atmiņas izpildes laikā.

Kā iepriekš bija pieminēts, mobilajiem ierīcēm ir daudz resursu ierobežojumu un tajā skaitā uz operatīvo atmiņu. Mobilām ierīcēm tapāt kā galddatoriem pamatā ir divas atmiņas: operatīva un fiziska. Fiziska atmiņa var būt liela, bet pamats ierobežojums iet uz operatīvo atmiņu. Ja lietojums izpildes laikā prasīs daudz operatīvas atmiņas, tad būs izmantota fiziska atmiņa, kura prasa vairāk laika uz apstrādi. Rezultātā pazemināsies lietojuma ātrdarbība.

Jāizmanto skalārus tipus.

Katru izmantojamu objektu ir vajadzīgi iedalīt no operatīvas heap atmiņas. Objekta konstruktors tiek palaists automātiski iedalīšanas laikā. Tāpēc katru objekts, kurš tiek izmantots paņem noteikto atmiņas apjomu, kuru tas pieprasa. Tas ietekmē uz kopējo operatīvo atmiņas sadali un attiecīgi uz lietojuma ātrdarbību. Lai izvairīties no objekta skaita pēc iespējas jāizmanto skalārus tipus, tā saucamos primitīvus tipus (int, boolean, long u ttl.) objektu vietā.

Nedrīkst pilnīgi paļauties uz Garbage Collector.

Java virtuāla mašīnā ar operatīvas atmiņas tīrīšanu nodarbojas Garbage Collector (GC). Ar objektu pārvaldību nodarbojas šī svarīga virtuālas mašīnas komponente. Programmētājiem nav jādomā par "mirušiem" referencēm uz objektiem. No vienas puses tas ir priekšrocība programmētājiem, bet no citas rodas viena problēma. Lai palaist savu darbību GC izmantot tāpat, ka un jebkurš cits mobilas ierīces lietojums procesoru. Ja lietojumā ir izveidots daudz objektu, GC var parādīties problēmas taisot savus tiešus pienākumus. Rezultāta lietojums var laiku pa laika taisīt pauzes, kurā laikā GC veidos savus darbus. Iespējami, ka lietojums šajos gadījumos var uzskarties. Lietojuma izpildes laikā būs pieprasīts vairāk un vairāk operatīvas atmiņas no sistēmas lai iedalīt jaunus objektus atmiņā. Un rezultātā lietojums var uzkārties un tas atmiņas apjoms, kuru GC mēģināja atbrīvot paliks uz laiku "mirušais" un citi lietojumu nevarēs izmantot šo atmiņu. Tāpēc ja lietojumā tiek izmantotas tīkla savienojumi, iet darbības ar failiem un lokālu glabatuvi (RMS) tad kodā obligāti vajag aizvērt visas šis resursus uzreiz pēc tas izmantošanas. Ja tiek izmantots *try catch*, tad resursu aizvēršana notiek *finally* blokā. Zemāk ir koda gabals kur ar treknrakstu ir paradīts piemērs.

```
HttpConnection c = null;
InputStream is = null;
try {
  c =
      (HttpConnection) Connector.open("http://url");
  is = c.openInputStream ();
} catch (Exception e) {
} finally {
   try {
      if ( c != null ) c.close();
      if ( is != null ) is.close();
   } catch (IOException ioe) { }
}
```

Jāizmanto "slinku" inicializāciju.

Cita metode kā samazināt operatīvas atmiņas palielinājumu ir izmantot objektus tika tad kad tie reāli ir vajadzīgi. Parasti šo tehniku nosauc par "slinku" inicializāciju. Šajā pieejā galvenā ideja pirms objekta izmantošanas vajadzīgi pārbaudīt objektu uz null. Zemāk ir paradīts klases *SomeClass* kods, kur metodē *getVector()* tiek realizēta šī pieeja.

```
public class SomeClass {
   private Vector v;
   public Vector getVector(){
   if( v == null ) v = new Vector();
   return v;
   }
}
```

Jāizmanto lokālus mainīgus.

Vispār pieeja klases mainīgiem ir lēnāk nekā pieeja lokāliem mainīgiem. Dažreiz gadās izmantot klases mainīgus ciklā, tad izdevīgāk piešķirt vērtību īslaicīgam mainīgam un izmantot to klases mainīgā vietā. Bet vajag būt uzmanīgam ja šie dati var būt pieejami daudziem thread objektiem. Īpaši šī pieeja ir svarīga darbībā ar masīviem. Katrā reizē kad notiek kāda piekļuve elementam masīvā, Java interpretators veic robeža pārbaudes vai piekļuves indekss ir pareizs. Ja šī piekļuve atkārtojas vairākas reizes tad izdevīgāk būtu piešķirt indeksa vērtību kādam īslaicīgam mainīgam un pēc tam izmantot to.

Jāizvairās no String konkatenācijas.

Java ir ļoti viegli izveidot virknes izmantojot konkatenācijas procedūru. Piemēram:

```
String someStr = "some string " + "concatination";
```

No izpildes ātruma viedokļa tas nav labs stīls. Šī konkatenācija Java iekšējā realizācijā izveido *StringBuffer* objektu, izmanto šī objekta *append* metodi un vēl daudz citas darbībās. Ja konkatenācija notiek ciklā tad visas iepriekšējas darbības jāreizina uz cikla reizes skaitu. Tas viss ietekmē uz ātrdarbību. Tāpēc labāk uzreiz izmantot *StringBuffer* objektu un apiet pēc iespējas tiešo konkatenāciju.

Jāizmanto Thread, bet jāizvairās no sinhronizācijas.

Threads ir ļoti savtīgi Java valodā. Vajag vienmēr ņemt priekšrocības no šiem procesiem. Vispār ir tāds likums, kā jebkura operācija, kura izpildās vairāk nekā 0.1sekundes jāiznes atsevišķa thread procesā. Pamatā tas notiek kad tiek izmantota lietotāju saskarne, kur operāciju izpildes laiks bremzē interaktīvu lietojuma darbību. Lietotāju saskarne mobilajā ierīcē īpaši ir svarīga ātruma ziņā. Bet ja tiek izmantoti threads tad dažreiz ir vajadzīgi sinhronizēt resursus ar *synchronized* vardu kodā. Nepārdomāta sinhronizācija var būt avots dažiem blokiem thread procesu izpildes laikā.

5.2. Mobilo lietojumu izstrādes rīki

J2ME platforma relatīvi jauna. Labi un ērti izstrādes rīki īpaši izstrādes vides (IDE) ievērojami var palīdzēt bezvada lietojumu izstrādātajiem. Izstrādes rīku pasaule ir ļoti plaša savā piedāvājumā. Tā kā mobilo ierīces lietojumi kļūst vairāk un vairāk populāri, tad daudz gan mobilo ražotāju kompānijas, gan lielas IT kompānijas piedāvā savas izstrādes rīkus realizācijas.

Viens no populārākiem izstrādes rīkiem ir kompānijas Sun *Wireless Toolkit*. Skaitās ka tas ir moderns izstrādes rīks, kurš bāzējas uz J2ME platformas CLDC konfigurācijām un MIDP profiliem. Šīs rīks iekļauj emulēšanas vides, izpildes optimizāciju, dokumentāciju un piemērus.

Pamatā jaudīgie rīki balstās uz soļu (step-by-step) iterācijām. Balstoties uz savu pieredzi autoram liekas ka šis pakāpeniska iterācijas ir svarīgas tikai pašā sakumā, kad tehnoloģija vai platforma nav zināma cilvēkam, ir jauna priekš viņam. Tad svarīgi saprast kas notiek katrā solī lai būtu skaidra saprašana kas kur atrodas. Bet kad jau ir kompetence tehnoloģijā un ir vajadzīgi ātri iteratīvi izstrādāt lietojumu, tad būtu svarīgi automatizēt šo pakāpenisku procesu. Tā kā autoram ir kompetence J2EE un J2ME platformā tad izstrādājot praktisko daļu autors pieturējiss pie otra varianta – izstrādes automatizācija. Šajā nodaļā apraksti rīki, kurus autors pielietojis izstrādes laikā.

Pamatā tas ir 4 rīki:

- Izstrādes vide – Intelij IDEA;
- Projektu veidošanas rīks – Ant;
- J2ME platformas specifisks veidošanas rīks – Antenna;
- J2ME koda optimizēšanas rīks.

Izstrādes vide – Intelij IDEA

Eksistē daudz Javas koda izstrādes vides, bet autoram personiski patīk Itelij IDEA.

Tā reāli fokusējas uz programmētāja produktivitāti. Tā piedāvā veselīgu izstrādes rīku kombināciju iekļaujot refactoring, J2EE platformas automatizācijas servisus, Ant rīku atbalstu, JUnit testēšanu un koda versiju kontroles integrāciju. Savākta ar gudru Java koda redaktoru, kodēšanas palīdzību un automatizētas kodēšanas iespējas Intelij

IDEA palīdz Java programmētājiem palielināt produktivitāti un samazināt rutīnas darbības kodēšanas procesā. 5. pielikumā ir paradīts šīs vides attēls.

Šī vide vinnējusi daudzos konkursos un saņēmusi apbalvojumus.

Piemērām pēdējie no tiem ir:

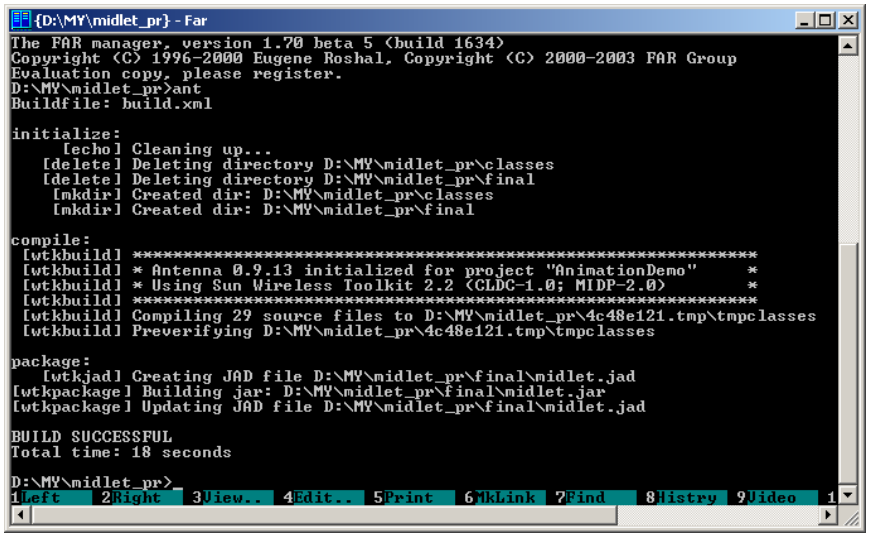
- 17. martā 2006. gadā 16. Jolt award ceremonijā;
- "Java Developer's Journal" žurnāla lasītāji balsoja par 2005. gada kategorijā "Best Java IDE";
- "Software Development Magazine" žurnāla lasītāji balsoja par 2005. gada kategorijā "Best Technical Support".

Projektu veidošanas rīks – Ant

Ant – ir platformneatkarīgs projektu veidošanas rīks, kas palīdz lietotājam ar XML veida skripta failu nodrošināt bieži lietojamu darbību izpildi. Ant ir ļoti jaudīgs rīks dažu darbību automatizācijai, piemēram:

- Koda kompilācija
- Darbības ar failiem: kopēšana, dzēšana, pārnešana
- Arhivācija
- Ieviešana uz aplikācijas serveru.

Pamatā Ant galvenā būtība ir predefinēto Ant komandu kopums (tā saucamas tasks), kuru var paplašināt ar savām uzrakstītam komandām, kas un bija izdarīts Antenna gadījumā. 6. pielikumā atrodas skripta faila saturs, kurš palīdzēja veidot "Mobilais fotogrāfs" projektu. Bet 36. attēlā ir paradīts Ant rīka darbību izpildes kārtība un rezultāts.



36.att. Ant izstrādes rīka izpildes rezultāts

J2ME platformas specifisks veidošanas rīks – Antenna

Iepriekšējā punktā bija pastāstīts par Ant izstrādes rīku, kur bija pieminēts ka Ant uzdevumus var paplašināt ar saviem uzrakstītiem *task-iem*. Antenna uzdevumu kopa ir viena no Ant uzdevumu paplašinājumiem, kura ļauj automatizēt J2ME platformas lietojuma veidošanu. 6. pielikumā ir paradīta visa Ant skripta faila saturs. Tur arī ir redzams kāda veidā tiek definētas Antenna taski. Zemāk ir paradīts kādā veidā tiek definēti Antenna taski.

```
<taskdef name="wtkjad"
classname="de.pleumann.antenna.WtkJad"/>
<taskdef name="wtkbuild"
classname="de.pleumann.antenna.WtkBuild"/>
<taskdef name="wtkpackage"
classname="de.pleumann.antenna.WtkPackage"/>
<taskdef name="wtkmakeprc"
classname="de.pleumann.antenna.WtkMakePrc"/>
<taskdef name="wtkrun"
classname="de.pleumann.antenna.WtkRun"/>
<taskdef name="wtkpreverify"
classname="de.pleumann.antenna.WtkPreverify"/>
<taskdef name="wtkobfuscate"
classname="de.pleumann.antenna.WtkObfuscate"/>
```

Midlet lietojums pamatā sastāv no diviem failiem JAR un JAD. JAR ir lietojuma klašu arhīvs, kurš tiek uzstādīts mobilā ierīcē. JAD ir uzstādīšanas aprakstīšanas fails, kurš satur dažas konfigurācijas. Galvenais Antenna rīka darbības rezultāts ir midlet lietojuma divi faili: saģenerēts JAD fails un izveidots JAR arhīvfails. 36. attēlā bija paradīts Ant skripta izpilde. Izpildes rezultātā ir 2 “Mobilais fotogrāfs” lietojuma faili: midlet.jar un midlet.jad, kura saturs ir paradīts zemāk.

```
MIDlet-Jar-URL: midlet.jar
MIDlet-Jar-Size: 61262
MIDlet-Name: midlet
MIDlet-Vendor: Denis Garavnjov
MIDlet-Version: 1.0.0
MIDlet-
1:midlet,/icons/welcome.png,sms.midlet.camera.CameraMIDlet
Servlet-URL: http://www.smsrelax.id.lv/http_test
Send-Receive-Servlet-URL:
http://www.smsrelax.id.lv/http_send_receive_test
Socket-URL: socket://www.smsrelax.id.lv:7777
JAD-Version: 1.17
MicroEdition-Profile: MIDP-2.0
MicroEdition-Configuration: CLDC-1.0
```

J2ME koda optimizēšanas rīks

Mobilo ierīču lietojuma izmērs ir ļoti svarīgs daudzos iemeslos. 5.1 nodaļā apskatījis šos iemeslus. Lai samazināt lietojuma izmēri ir vajadzīgi pielietot speciālu rīku, kurš angliski saucās obfuscator. Vēl pie lietojuma samazināšanas šīs rīks padara class failus ļoti grūti dekompelēšanai pirmkodā. Kā ir zināms jebkuru parasto nokompilēto class failu var dekompilēt pirmkodā, piemērām ar jad programmu. Tadā veidā jebkurai programmai kurai ir gatavas class failu pakotnes, var uzzināt pirmkodu. Obfuscator rīks izmaina tādā veidā class failus, lai dekompilēt vispār nav reāli. Daudzi obfuscator rīki vēl var dzēst nevajadzīgus vai nelietotus metodes vai pat klases.

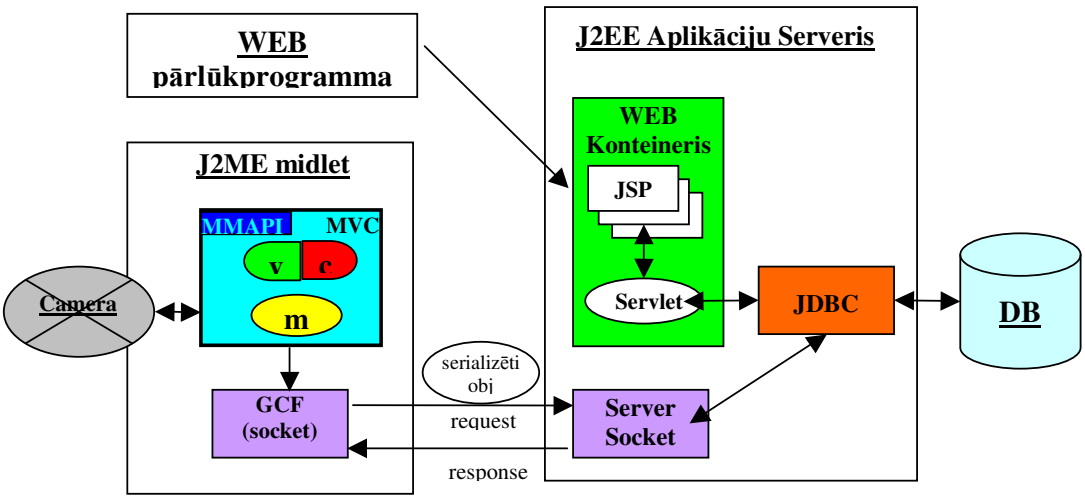
6. “Mobilais fotogrāfs” lietojums

Praktiskā darbā darba autors uzrakstījis J2ME lietojumu – midletu, kurš savienojas ar J2EE serveri. Šajā praktiskā darbā viņš pielietojis divas šablonus no 4. daļas. Midleta lietojumā autors norealizēja J2ME platformas MVC modeli, kurš bija aprakstīts 4.1 nodaļā. Datu apmaiņas atvieglošanai starp klientu un serveri autors izmantojis strukturēti organizētus objektus. Pārsūtot objektus bija vajadzīgi izmantot objektu serializācijas šablonu, kurš bija aprakstīts 4.2. nodaļā. Izstrādājot midlet lietojumu, kā testēšanas ierīci autors izmantojis NOKIA 3230 mobilo telefonu. 7. pielikumā atrodas mobila telefona tehniskie raksturojumi. MVC modeļa *skata* komponente tiek paradīta 8. pielikumā.

Tagad tiks paskaidrots kāpēc autors nosauca realizētu projektu par “Mobilais fotogrāfs”.

Ja mobilam telefonam ir iebūvēta fotokamera un tas atbalsta GPRS, tad izmantojot šo lietojumu var nofotografēt un uzreiz nosūtīt foto uz serveri. Servera galā atrodas Java lietojums, kurš pilnīgi realizē visu J2EE standartu, kurš bija aprakstīts 2.1.3. apakšnodaļā. Šī servera lietojums saņem foto no midletiem, saglabā tos datu bāzē. Lietotājs Internētā var aiziet pēc noteiktas adreses (www.smsrelax.id.lv) uz WEB-vietni un apskatīt visas fotogrāfijas, kādi bija nosūtīti ar midlētiem.

37. attēlā ir paradīta lietojumu darbības shēma.



37.att. Lietojuma “Mobilais fotogrāfs” darbības shēma

Kā ziņojumapmaiņas formātu, kuri ir aprakstīti 3.2. nodaļā, autors izvēlējies Bināru tipu - Socket veidu. Tā kā Socket bija realizēts MIDP 2.0 versijā, tad telefonam arī jāatbalsta šo 2.0 versiju.

Midlet lietojumā ir iespēja nofotografēt izmantojot "dzimto" mobila telefona programmatūru. Šo iespēju realizē Mobile Media API (MMAPI). MMAPI paplašina J2ME platformas funkcionalitāti piedāvājot audio, video un laika-bāzētu multimedija atbalstu resursa ierobežotām ierīcēm. Tas realizācija ir vienkārša un "viegla" (lighweight) neobligāta pakotne, kura dod Java izstrādātajiem pieeju iebūvētiem multimedija servisiem, kuras atrodas mobilā telefonā. Tāpēc mobilām telefonam lai izmantotu "Mobilais fotogrāfs" midletu ir vajadzīgs MMAPI atbalsts.

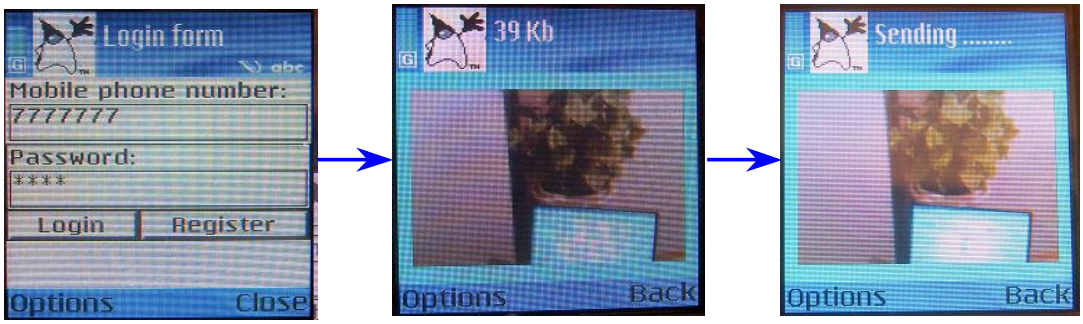
Kā ir redzams 37. attēlā komunikācija starp midletu un aplikācijas serveru notiek izmantojot Socket palīdzību. Autors ir izstrādājis savu vienkāršu vaicājums/atbilde protokolu kur vaicājums varbūt iniciēts tikai no midlet-a, bet atbilde tiek atgriezta pēc biznesa loģikas izpildīšanas.

Kad startējas J2EE aplikācijas serveris tad uzreiz startējas lietojums, kurš izmantojot ServerSocket objektu klausās definēto portu uz ienākošiem savienojumiem. Midlet, startēšanas brīdī, izveido savienojumu ar Socket objekta palīdzību. Kad savienojums ir izveidots starp midletu un serveri, tad midlet izmantojot iebūvēto kameru ar MMAPI palīdzību nofotografē un iegūtu foto, serializēto objektu veidā, nosūta serverim uz saglabāšanu. Serveris saņem serializēto objektu, izņem no tā foto datus un izmantojot JDBC saglabā tos datu bāzē. Midletam ir iespēja dabūt jebkuru veco foto no datu bāzes. Šajā gadījumā darbības secība ir otrāda, izņemot MMAPI un kameru izmantošanu, jo kad serializētais objekts nodots midletam, tiek iegūta foto, kura vienkārši attēlojas uz mobila telefona displejā. 8. pielikumā ir paradīta lietotāju saskarnes formu secība.

Lietotājs izmantojot parasto darba staciju un pārlūkprogrammu var aiziet uz Internet adresi, kura var būt aplikācijas servera ārēja IP adrese vai domēna nosaukums (domain name). Servera galā lai skatītos nofotografētas bildes darba autors realizējis WEB daļu ar JSP lappusi, kur izmantojot Servlet ar JDBC palīdzību tiek dabūta fotogrāfija un pārraidīta lietotāju pārlūkprogrammā.

Turpinājumā autors grib paradīt ar dažiem no 8. pielikuma attēliem midleta lietojuma darbības secību un to rezultātu.

No sākuma lietotājs startē midlet lietojumu, autentifikācijas un sūta foto uz J2EE aplikācijas serveri. Visa šī secība ir paradīta 38. attēlā.

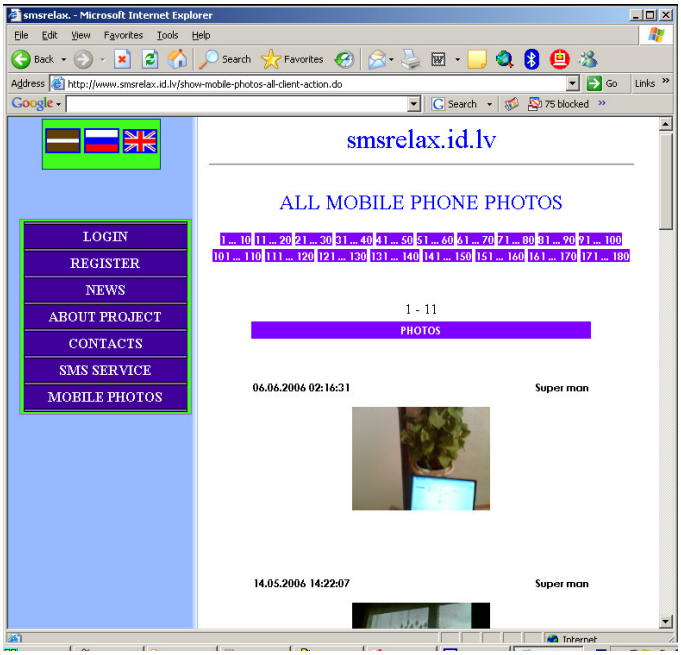


38.att. Midlet lietojuma attēla sūtīšana

Pēc dažām sekundēm parādās paziņojums par veiksmīgu saglabāšanu datu bāzē, kurš ir redzams 39. attēlā.

39.att. Paziņojums par veiksmīgu saglabāšanu datu bāzē

Un tagad jebkurš lietotājs var aiziet pēc noteiktas adreses un apskatīt nofotografētu attēlu (40. attēls).



40.att. WEB-vietne ar nofotografētiem attēliem

Nobeigums

Darbā tika aprakstīti divu Java 2 platformu sadarbības šabloni, paņēmieni un paraugi korporatīviem lietojumiem, kuros var būt iesaistītas: J2ME un J2EE platformas. Tika apskatīts divu platformu komunikācijas modelis, lietojumu izstrādes sarežģītības un problēmas, kuras rodas bezvadu lietojumu programmatūras projektēšanā. Kā arī tika aprakstītas dizaina un implementācijas vadlīnijas, sekojot kurām izstrādātāji un projektētāji var izvairīties no problēmām, kuras var potenciāli rasties nākotnē, izmantojot izstrādātos bezvadu korporatīvus risinājums.

Pati par sevi J2EE platforma ir ļoti jaudīga korporatīvo sistēmu risinājumos, savukārt sadarbībā ar J2ME platformu tā vēl vairāk paplašina klient – serveris tipa lietojuma segmentu. Problemātika, kura saistīta ar divu platformu mijiedarbību ir saistīta ar to, ka J2EE platformas vidē eksistē dizaina šabloni (design patterns), bet J2ME vidē tādu stingri definētu šablonu nav, jo J2ME platforma salīdzinot ar citiem Java 2 platformām ir samēra jauna. Plašs J2ME platformas ierīču asortiments, J2ME lietojumu pārnesamības problēmas, mobilo ierīču resursu ierobežojums neļauj standartizēt un atvieglot unificētu šablonu izstrādi un specificēšanu kā J2EE platformas gadījumā. Bet tā, ka J2ME platforma attīstās izmantojot JCP organizāciju, tiek sagaidīts, ka nākošās MIDP profilu versijās būs atrisinātas šīs problēmas.

Mobilo ierīču daudzums pasaulē ir krietni vairāk nekā parasto galddatoru skaits, kā arī mobilo ierīču jauda un potenciāls strauji attīstās. Šīs attīstības iespaidā, mobilo lietojumu vide kļūst par pievilcīgu ienākumu avotu daudzās biznesa sfērās.

3.1. nodaļā bija aprakstīti bezvadu datu pārraides standarti. Pēc autora domām, mobilo ierīču tīklu lietojumu tehnoloģiskā evolūcija ir pilnīgi atkarīga no bezvada tīkla evolūcijas. Autors prognozē, ka nākotnes bezvada ierīču un bezvada tīklu evolūcija atrisinās un novērsīs visus šķēršļus un problēmas, kuras saistītas ar J2ME platformas šablonu izveidi.

Ņemot vērā iepriekš minētas divu tehnoloģiju problēmas, pēc autora domām, var apkopot un norādīt J2ME šablonu ieviešanu labās (priekšrocības) un sliktās (trūkumus) puses.

Priekšrocības:

- Tiek samazināts projekta izstrādes laiks;
- Atvieglota turpmāka projekta uzturēšana;
- Plaša šablonu attīstība un ceļš uz standartizāciju.

Trūkumi:

- Palielinās J2ME lietojumu koda izmērs;
- Katram risinājumam jāoptimizē šablonu, kur optimizācijas aizņem noteiktu laika periodu;
- Šablonam trūkst universalitātes īpašības.

Literatūra

[1] *"Building Compelling Services for the Wireless Market Using Java Technolobgy"*
Internet adrese: http://developers.sun.com/techtopics/mobility/getstart/articles/whyjava/

[2] *"Wireless Software Design Techniques"*
Internet adrese: http://developers.sun.com/techtopics/mobility/midp/articles/uidesign/

[3] Группа Патрисии Сейбольд: *"Java. 2 Platform, Enterprise Edition Обеспечение совместимости, переносимости и возможности сетевого взаимодействия"*
Эн Томас Июнь 1999, Internet adrese: http://ru.sun.com/win/java/j2ee/index.html.

[4] Stephanie Bodoff, Dale Green, Kim Haase, Eric Jendrock, Beth Stearns: The J2EE ™ tutorial Internet adrese: http://ac.cs.nstu.ru/~mrc/docs/java/j2eeTutorial/doc/Overview3.html#63961, 2002. gada 24. aprīlis.

[5] Дэвид М. Гери. Библиотека профессионала Java Server Pages. – Москва: „Вильямс". – 2002. – 443 lpp.

[6] *"Mobile Services Market-Status, Evolution and Forecasts"*
http://ncsp.forum.nokia.com/download/?asset_id=11991

[7] *"A Survey of J2ME Today"*
Internet adrese: http://developers.sun.com/techtopics/mobility/getstart/articles/survey/

[8] *"A Brief History of 3G"*
Internet adrese: http://www.cdmatech.com/download_library/pdf/QCOM_3G_Overview.pdf

[9] *KONKURENCES PADOME,Lēmums 2006.gadā 15. martā*
Internet adrese: http://www.competition.lv/uploaded_files/2006/R16_1503.pdf

[10] *"Optimizing the Client/ServerCommunication for Mobile Applications Part 1"* Internet adrese:
http://www.forum.nokia.com/info/sw.nokia.com/id/d7268f17-d6f5-4a52-a599-7b5175879a5c/Optimizing_client_server_part1.pdf.html

[11] *Getting Started with SIP API for J2ME (JSR 180)*
Internet adrese: http://developers.sun.com/techtopics/mobility/apis/articles/sip/

[12] "*The Generic Connection Framework*"
Internet adrese: http://developers.sun.com/techtopics/mobility/midp/articles/genericframework/

[13] *"J2ME Low-Level Network Programming with MIDP 2.0"*
Internet adrese: http://developers.sun.com/techtopics/mobility/midp/articles/midp2network/

[14] *"The network connection"*
Internet adrese: http://developers.sun.com/techtopics/mobility/midp/chapters/j2mewhite/chap13.pdf

[15] *"Mygnokii2/Gammu HomePage and mailing list"*
Internet adrese: http://www.mwiacek.com/gsm/soft/gammu.html

PIELIKUMI

1. pielikums

Terminu vārdnīca

JSP – *Java Server Pages* – Java servera lappuses
EJB – *Enterprise JavaBeans* - Korporatīvie JavaBeans komponenti
J2EE – *Java 2 Enterprise Edition* - Java 2 korporatīvais izlaidums
J2ME – *Java 2 Micro Edition* - Java 2 mikro izlaidums
J2SE – *Java™ 2 Standard Edition* **–** Java 2 standarta izlaidums
HTML – *HyperText Markup Language* – hiperteksta iezīmēšanas valoda
XML – *Extensible Markup Language* – paplašināmās iezīmēšanas valoda
API – *Application Programming Interface* –lietojumprogrammu saskarne
JNDI – *Java Naming and Directory Interface* – Java vārdu un katalogu interfeiss
RMI – *Remote Method Invocation* – attālinātu metožu izsaukšana
Java IDL – *Java Interface Definition Language* – Java interfeisa definēšanas valoda
JTA – *Java Transaction API* – Java transakcijas lietojumprogrammu saskarne
JTS – *Java Transaction Service* –Java transakcijas serviss
JDBC – *Java database connectivity* – Java datu bāzes savienojamība
JAF – *JavaBeans ™ Activation Framework* – JavaBeans aktivizācijas karkass
JMS – *Java Message Service* –Java ziņojumu serviss
WBXML – *Wireless Binary Extensible Markup Language* – bezvada bināra paplašināmās iezīmēšanas valoda
WAP – *Wireless Application Protocol* - bezvadu lietojumu protokols
EDGE - *Enhanced data Rates for GSM Evolution* – tehnoloģija EDGE
GPRS - *General Packet Radio Service* - vispārējais pakešu radio pakalpojums
UMTS - *Universal Mobile Telecommunications System* - Universālā mobilo telesakaru sistēma
SOAP - *Simple Object Access Protocol* - vienkāršais objektu piekļuves protokols
MMAPI - *Mobile Media API* – mobilas medījas API

2. pielikums

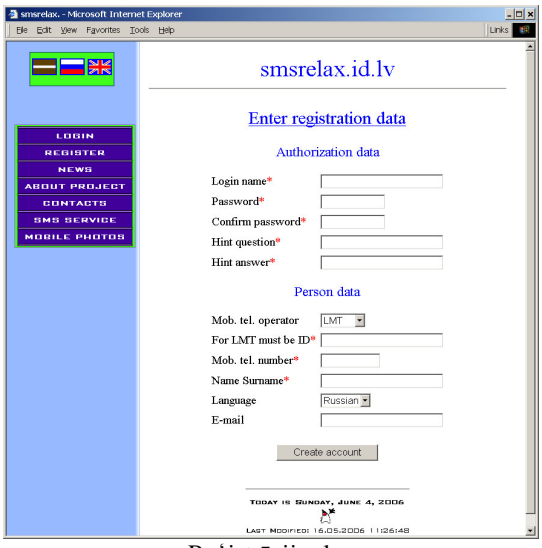
Enterprise Java API apraksts

API nosaukums	API apraksts un paskaidrojumi
EJB	Servera komponentu modelis, kurš nodrošina pārnesamību starp lietojuma serveriem un realizē automātiskus servisus no lietojuma komponentu puses.
JNDI	Piedāvā piekļūšanu kataloga un vārdu piesavināšanas servisiem: DNS, NDS, LDAP un Corba Naming.
RMI/IIOP	Attālinātu vaicājumu metode dibina attālinātus interfeisus priekš Java – Java
Java IDL	Dibina attālinātus interfeisus Java-COBRA paziņojuma uzturēšanai.
Servlets and JSP	Java serversīklietotnes un Java Server Pages ir servera komponenti, kuri izpildās Web-serverī, kurš uztur dinamisku HTML lappuses ģenerāciju un pārvalda seansa savienojumu ar pārlūka sistēmas lietotājiem
JTA	Transakcijas interfeiss.
JTS	Definē attālinātu transakcijas servisa pārvaldību, kura ir pamatota uz *COBRA Object Transaction Service*
JDBC ™	Piedāvā unificētu piekļūšanu pie relāciju datu bāzēm.
JavaMail	Piedāvā no protokola neatkarīgu struktūru *e-mail* lietojuma projektēšanai.
JAF	Piedāvā standarta interfeiss attiecīgai JavaBeans komponentu ieviešanai datu apstrādei.

3. pielikums

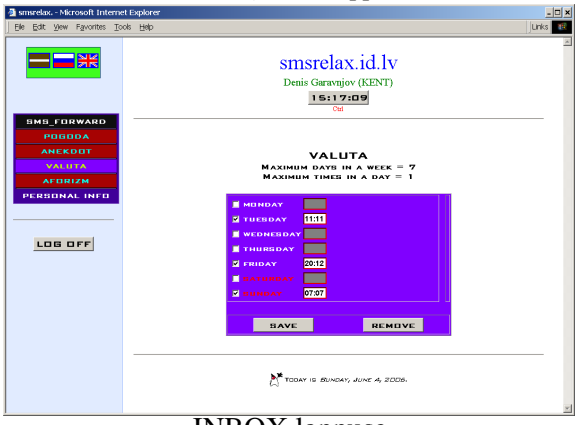
Parauga galvenās JSP lappuses

Šajā pielikumā atrodas attēli, kuros ir parādītas galvenās lappuses, kurās notiek visbiežākā un svarīgā datu apstrāde. Reģistrācijās lappusē klienti reģistrējas. Pēc reģistrēšanas viņi automātiski apmeklē piekļūšanas lappusi. Pēc savas identifikācijas datu ievades viņi apmeklē savu personīgu INBOX-u. Tas viss ir parādītas hronoloģiskā secībā zemāk.



Reģistrācijas lappuse

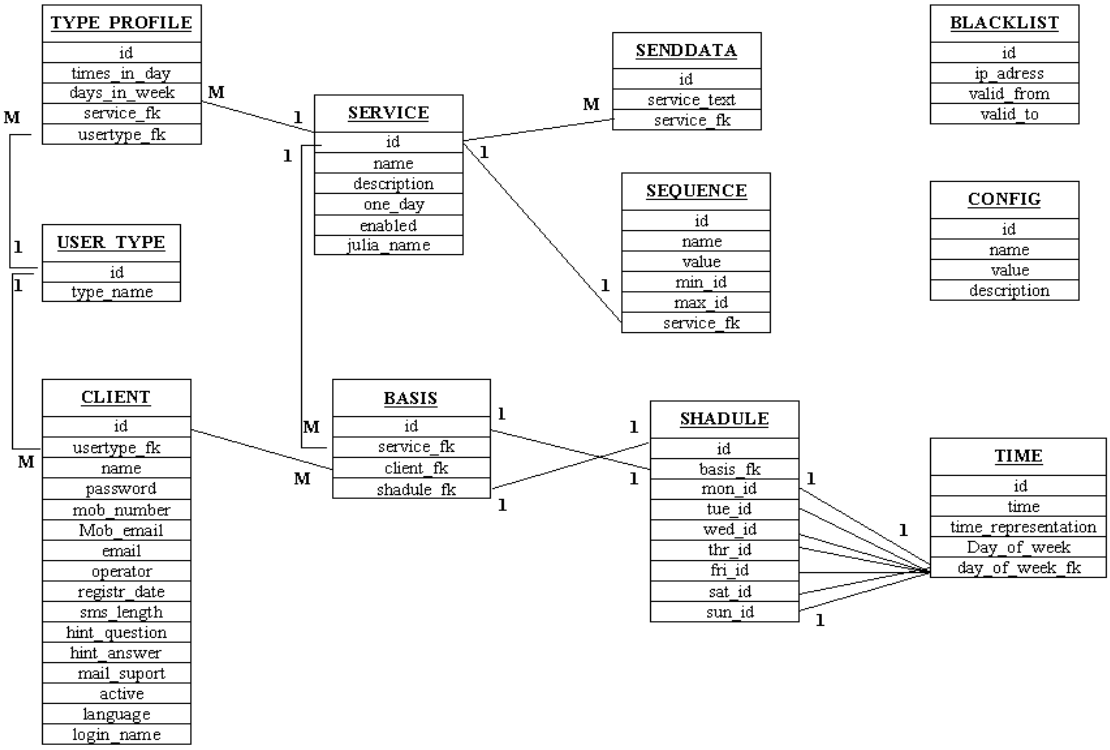
Piekļūšanas lappuse



INBOX lappuse

4. pielikums

Parauga datu bāzes shēma



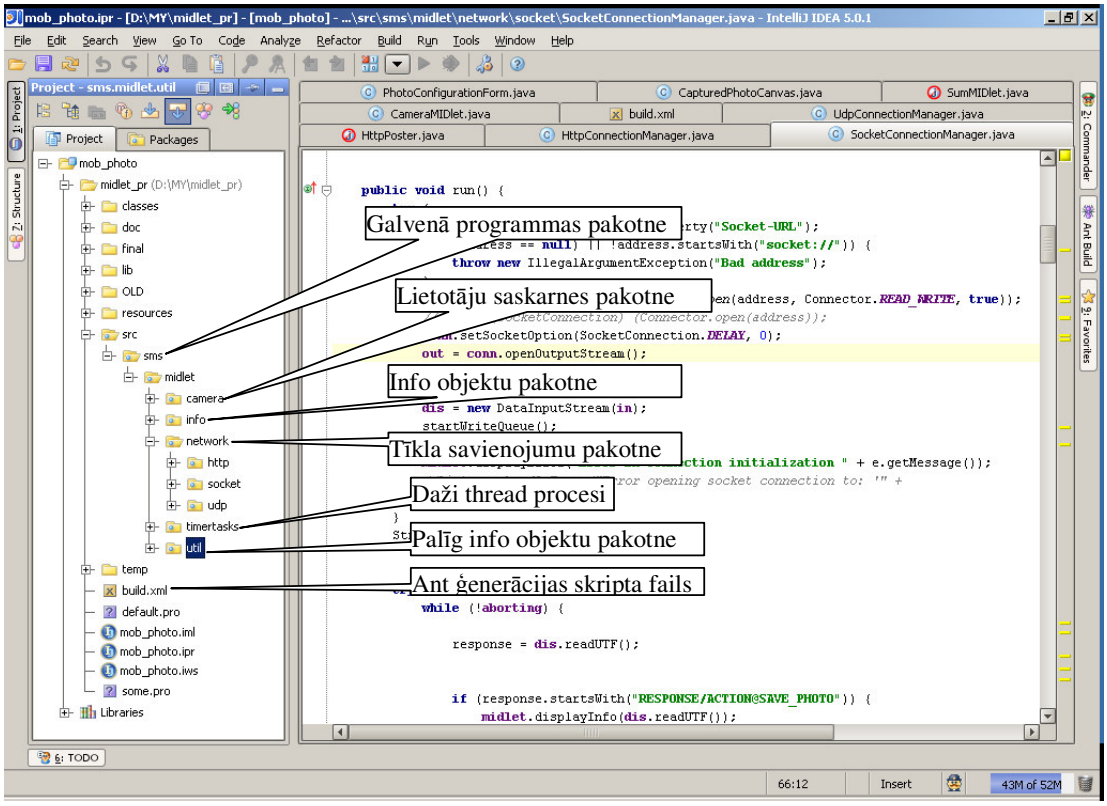
Datu bāzes shēma sastāv no 11 tabulām:

1. **USER_TYPE** – tabula, kur ir informācija par klienta tipu.
2. **TYPE_PROFILE** – tabula, kur ir klienta tipa profila konfigurācija.
3. **CLIENT** – tabula, kur ir visa informācija par klientiem.
4. **BASIS** – centrāla tabula, kura savieno 2 pamat tabulas un 1 starp tabulu.
5. **SERVICE –** tabula, kurā ir ievadīti visi pakalpojuma nosaukumi, kuri uz šo brīdi ir domāti aizsūtīšanai klientiem.
6. **SENDDATA –** tabula, kurā ir ievadīts pakalpojumu saturs.
7. **SHADULE** – nedēļas dienas identifikatoru sarakstu tabula.
8. **TIME** – tabula ar laiku, kuru var definēt katrai nedēļas dienai.
9. **CONFIG** – tabula, kur glabājas dažādi konfigurācijas parametri.
10. **BLACKLIST** – tabula, kur glabājas IP adreses, no kurienes mēģināja uzlauzt sistēmu.
11. **SEQUENCE** – tabula, kur glabājas ID parējām tabulām.

5. pielikums

Vizuāla izstrādes vide

Šajā attēlā ir parādīta Intelij IDEA 5.0.1 vizuāla izstrādes vide. Attēla kreisajā pusē ir parādītas midleta "Mobilais fotogrāfs" Java pakotnes. Blakus ir redzami paskaidrojumi.



6. pielikums

Ant izstrādes rīka skripta fails

Zemāk ir paradīts *build.xml* faila saturs, kurš definē Ant uzdevumu izpildes kartību veidojot projektu.

```
<project name="AnimationDemo" default="package" basedir=".">
   <taskdef resource="proguard/ant/task.properties"
         classpath="C:/Java/proguard3.2/lib/proguard.jar" />
   <property name="MIDletSuite" value="midlet"/>
   <property name="MIDlet_1_name" value="midlet"/>
   <property name="MIDlet_1_class" value="sms.midlet.camera.CameraMIDlet"/>
   <property name="MIDlet_1_icon" value="/icons/welcome.png"/>
   <property name="MicroEdition-Profile_version" value="MIDP-2.0"/>
   <property name="MicroEdition-Configuration_version" value="CLDC-1.0"/>
   <property name="JADversion" value="1.17"/>
   <property name="MIDlet_version" value="1.0.0"/>

   <!-- Directories of file locations -->
   <property name="src.dir" location="src"/>
   <property name="classes.dir" location="classes"/>
   <property name="final.dir" location="final"/>
   <property name="resources.dir" location="resources"/>
   <!-- Location of Antenna Tasks -->
   <property name="wtk.home" value="C:\Java\wtk22"/>
   <property name="wtk.cldc.version" value="1.0"/>
   <property name="wtk.midp.version" value="2.0"/>

   <taskdef name="wtkjad" classname="de.pleumann.antenna.WtkJad"/>
   <taskdef name="wtkbuild" classname="de.pleumann.antenna.WtkBuild"/>
   <taskdef name="wtkpackage" classname="de.pleumann.antenna.WtkPackage"/>
   <taskdef name="wtkmakeprc" classname="de.pleumann.antenna.WtkMakePrc"/>
   <taskdef name="wtkrun" classname="de.pleumann.antenna.WtkRun"/>
   <taskdef name="wtkpreverify" classname="de.pleumann.antenna.WtkPreverify"/>
   <taskdef name="wtkobfuscate" classname="de.pleumann.antenna.WtkObfuscate"/>

   <!--_____Compile & Preverify_____-->
   <target name="compile" depends="initialize">
      <wtkbuild srcdir="${src.dir}"
         destdir="${classes.dir}"
         classpath = "lib/hmidp80.zip"
         preverify="true"
         source="1.3">
      </wtkbuild>
   </target>
   <!--_____Package_____-->
   <target name="package" depends="compile">
      <!-- Create JAD file -->
      <wtkjad jadfile="${final.dir}/${MIDletSuite}.jad"
         jarfile="${final.dir}/${MIDletSuite}.jar"
         name= "${MIDlet_1_name}"
         vendor="Denis Garavnjov"        version="${MIDlet_version}"  >
```

```
        <!-- Add user-defined attributes to the JAD -->
        <attribute name="Servlet-URL" value="http://www.smsrelax.id.lv/http_test"/>
        <attribute name="Send-Receive-Servlet-URL"
value="http://www.smsrelax.id.lv/http_send_receive_test"/>
        <attribute name="Socket-URL" value="socket://www.smsrelax.id.lv:7777"/>
        <attribute name="JAD-Version" value="${JADversion}"/>
        <attribute name="MicroEdition-Profile" value="${MicroEdition-Profile_version}"/>
        <attribute name="MicroEdition-Configuration" value="${MicroEdition-
Configuration_version}"/>

        <midlet name="${MIDlet_1_name}"
          icon="${MIDlet_1_icon}"
          class="${MIDlet_1_class}">
        </midlet>
      </wtkjad>
      <!-- Create JAR file, obfuscating if desired -->
      <wtkpackage basedir="${classes.dir}"
        jarfile="${final.dir}/${MIDletSuite}.jar"
        jadfile="${final.dir}/${MIDletSuite}.jad"
        config="${MicroEdition-Configuration_version}"
        profile="${MicroEdition-Profile_version}"
        obfuscate="false">
        <fileset dir="${resources.dir}"/>
      </wtkpackage>
    </target>

    <target name="obfuscate" depends="package">
      <java fork="yes" classname="proguard.ProGuard"
          classpath="C:/Java/WTK22/bin/proguard.jar">
          <arg line="-libraryjars 'C:/Java/wtk22/lib/midpapi20.jar'"/>
          <arg line="-libraryjars 'C:/Java/wtk22/lib/cldcapi10.jar'"/>
          <arg line="-injars final/${MIDletSuite}.jar"/>
          <arg line="-outjar final/${MIDletSuite}_A.jar"/>
          <arg line="-keep 'public class * extends javax.microedition.midlet.MIDlet'"/>
       </java>
    </target>
    <target name="run">
      <!-- _______Run _______ -->
          <exec executable="C:\Java\wtk22\bin\emulator.exe">
                        <arg line="-Xdescriptor final\${MIDletSuite}.jad"/>
                    </exec>
    </target>
    <!--_________Initialize_________-->
    <!-- Delete class file before starting -->
    <target name="initialize">
      <echo message="Cleaning up..."/>
      <!-- Delete all output directories -->
      <delete dir="${classes.dir}"/>
      <delete dir="${final.dir}"/>
      <!-- Create empty directories -->
      <mkdir dir="${classes.dir}"/>
      <mkdir dir="${final.dir}"/>
    </target>
</project>
```

7. pielikums

Praktiskas daļas izmantotājs mobilais telefons.

Izstrādājot praktisko daļu, darba autors izmantojis NOKIA 3230 mobilo telefonu, kā testēšanas ierīci.

NOKIA 3230 tehniskās īpašības:

- Java tehnoloģija:
  - CLDC 1.0
  - Wireless Messaging API (JSR-120)
  - Mobile Media API (JSR-135)
  - Bluetooth API (JSR-82 No OBEX)
  - MIDP 2.0
  - Nokia UI API
- Tīkla atbalsta tehnoloģijas:
  - CSD
  - GPRS
  - HSCSD
- 1.3 megapixel camera
- Ekrāna īpašības:
  - krasas: 16 bit
  - izšķirtspēja: 176 x 208

8. pielikums

Praktiskas daļas lietotāju saskarnes formas

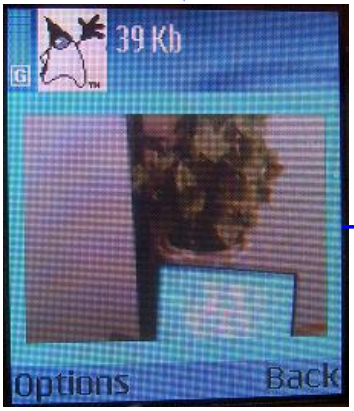
mobila telefona izvēlne
ar midleta ikonu

piekļūšanas forma
inicializācijas stāvoklī

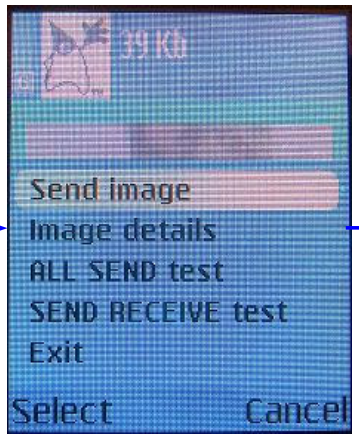
piekļūšanas forma

videokamera

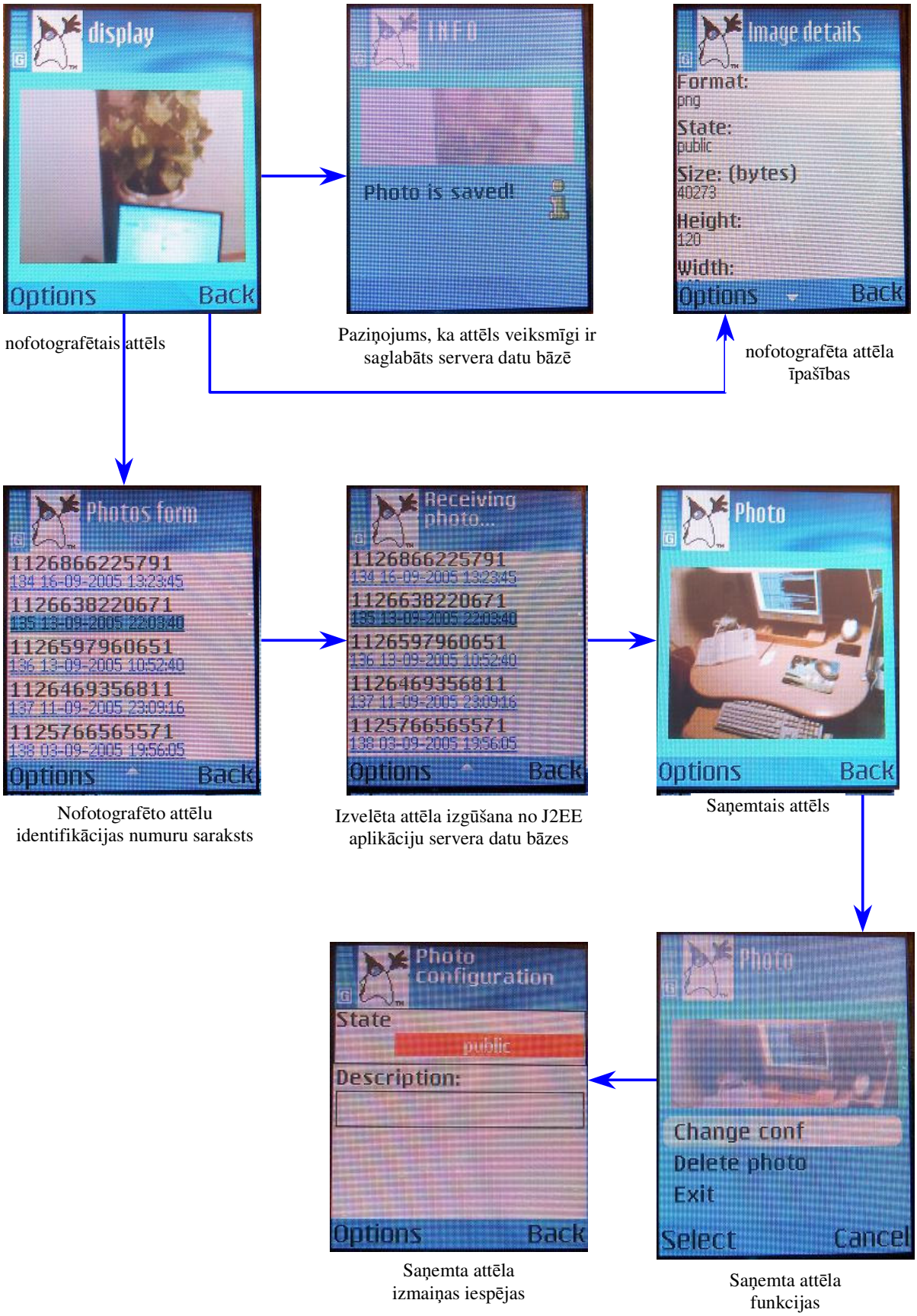
reģistrācijas forma



nofotografētais attēls



attēls sūtīšanas procesā
uz J2EE serveru



nofotografētais attēls

Paziņojums, ka attēls veiksmīgi ir saglabāts servera datu bāzē

nofotografēta attēla īpašības

Nofotografēto attēlu identifikācijas numuru saraksts

Izvelēta attēla izgūšana no J2EE aplikāciju servera datu bāzes

Saņemtais attēls

Saņemta attēla izmaiņas iespējas

Saņemta attēla funkcijas

Apliecinājums

Ar šo es apliecinu, ka šodien iesniegto maģistra darbu es esmu veicis pašrocīgi un esmu izmantojis tikai tajā norādītos palīglīdzekļus.

Rīgā, ( ) Paraksts:

Reģistrācijas lapa

Maģistra darbs izstrādāts
LU Datorikas nodaļā

Autors:
Fizikas un matemātikas . . . . . .
fakultātes students Deniss Garavņovs
St. apl. Nr. DatZ000112 2006. g. . jūnijā.

Darba vadītājs
Dr. dat. Kalvis Apsītis . . . . . .

Recenzents
Dr. dat Jānis Kalniņš . . . . . .

Darbs iesniegts maģistrantūras sekretariātā 2006. g. .jūnijā.

Pieņēma sekretāre . . . . . .

Aizstāvēts datorzinātņu maģistra pārbaudījumu komisijas sēdē

2006.g. . . . . .ar atzīmi. . . . . .

Protokols Nr. _____

Maģistra pārbaudījumu
komisijas sekretārs . . . . . .