SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ZAVOD ZA TELEKOMUNIKACIJE

DIPLOMSKI RAD br. 2674

AGENTSKI SUSTAV ZA UGOVARANJE SASTANAKA

Marko Skomeršić

Zagreb, travanj 2006.

Sadržaj

Uvod

Programski agent (u daljnjem tekstu agent) je program koji predstavlja korisnika u mreži i obavlja poslove u njegovo ime. Više agenata tvori agentski sustav. Konceptualno su programski agenti inovativna tehnologija za učinkovitu i inteligentnu realizaciju složenih, raspodijeljenih i izrazito interaktivnih heterogenih sustava te aplikacija. Agentska tehnologija ima potencijal da postane ključna tehnologija u izradi virtualnih organizacija (automatizacijom svakodnevnih procesa), inteligentno upravljanje računalnim i telekomunikacijskim mrežama, upravljanje sve većom količinom informacija, itd.

Da bi iskoristili ovaj potencijal, agenti moraju moći komunicirati, pregovarati i surađivati u „otvorenom“ okruženju. FIPA[1] (*Foundation of Intelligent Physical Agents*) osigurava programske standarde omogućujući interoperabilnost agenata u takvim okruženjima. Agentska platforma osigurava smještanje i upravljanje agentima. Jedna takva, javno dostupna, agentska, platforma je JADE[2] (*Java Agent Development Framework*). Uz standardiziranost i otvorenost, agentske platforme moraju podržavati i heterogene uređaje kao što su mobilni telefoni, PDA (*Personal Digital Assistant*), prijenosna računala, osobna računala, itd. Agentske platforme kao što je JADE ne podržavaju rad na malim uređajima i mobilnu komunikaciju. U tu svrhu je razvijena JADE/LEAP[3] (JADE/*Lightweight Extensible Agent Platform*) agentska platforma koja se temelji na JADE platformi.

JADE-LEAP je prva agentska platforma, usuglašena sa FIPA standardima, koja se izvršava na malim uređajima. Pošto se i JADE i LEAP distribuiraju s LGPL licencom, mogu se slobodno koristiti, mijenjati i nadograđivati. Postoji više verzija LEAP biblioteka za izvršavanje na J2SE, J2ME ili za izvršavanje na *Personal Java* sustavima.

---

[1] FIPA je neprofitno standardizacijsko tijelo koje se bavi standardizacijom programskih agenata i agentskih platformi. FIPA standardizira arhitekturu agentskih sustava, upravljanje agentima, njihovu komunikaciju, itd. http://www.fipa.org/

[2] JADE je raspodijeljena agentska platforma kreirana prema FIPA standardima. JADE je razvijena u programskom jeziku Java (J2SE) od strane *TILabs Turin and University of Parma*, Italija te je dostupna kao *open source* platforma s LGPL licencom. http://jade.tilab.com/

[3] LEAP je knjižnica (*library*) koja, u kombinaciji s komponentama JADE platforme, osigurava *run-time* okruženje za izvršavanje na malim uređajima. LEAP platforma je razvijena od strane LEAP konzorcija u sklopu europskog projekta IST-199-10211. http://leap.crm-paris.com/

1. Mali uređaji

1.1. Općenito

Pod terminom mali uređaj (*small device*) podrazumijevamo ručna računala (PDA – Personal Digital Assistent), mobilne telefone (cell/mobile phone), pametne telefone (*smartphone*) i sl. Da bi takav uređaj imao uporabnu vrijednost mora omogućavati bežičnu povezivost (GPRS, WLAN, Bluetooth, ...) ostalim računalnim mrežama te mora sadržavati API (*Application Programming Interface*) za razvoj programske podrške. Mali uređaji su prilično ograničeni u odnosu na osobna računala, poslužitelje, itd., a ta ograničenja se očituju u:

1. ograničenoj procesorskoj snazi koja se najčešće sastoji od 16-bitnih procesora brzine oko 200 MHz,

2. ograničenoj količini radne memorije koja varira od 64KB na SIM karticama do manje od 16MB na mobilnim uređajima ili 64MB na PDA računalima,

3. ograničenom prostoru za stalnu pohranu podataka koja, najčešće, nije dostupna na mobilnim telefonima,

4. ograničeni izvor energije za napajanje uređaja,

5. ograničenom pokrivenošću signalom za komunikaciju uređaja (ruralna i nenastanjena područja),

6. visoka latencija mreže i mala širina prijenosnog pojasa (tipično 9.6 kbps za GSM, 128 kbps za GPRS),

7. mala veličina ekrana za prikaz podataka (ekran) i

8. ograničeni mehanizam za unos podataka (numerička tipkovnica, ...).

Točke od 1. do 4. su „konstante“ koje treba uzeti u obzir prilikom razvoja aplikacija za pojedinu kategoriju malih uređaja, te se njihov utjecaj može ublažiti „pametnim“ odabirom algoritama i načina komunikacije. Točke od 5. do 8. su, u biti, skup zahtjeva na pružatelja usluge koje on mora osigurati da bi sustav, uopće, bio iskoristiv. Na te elemente ne možemo utjecati, ali možemo sustav dizajnirati tako da barem neke efekte ublažimo (npr. u slučaju da mobilni telefon izgubi vezu prema mreži, pametno bi bilo aplikaciju privremeno zaustaviti tako da nepotrebno ne troši resurse i energiju).

Velika uporabna vrijednost malih uređaja, unatoč gore navedenim nedostacima, se očituje u činjenici da ih korisnik, gotovo, stalno ima uz sebe te ih svakodnevno koristi. Zbog toga su idealni za proširenja dodatnim funkcionalnostima koje korisnik želi svakodnevno koristiti kako bi mu pojednostavile svakodnevnicu kako osobnu tako i poslovnu.

1.2. J2ME

Gotovo u potpunosti, današnji, mali uređaji imaju podršku za razvoj aplikacija u programskom jeziku Java™[4]. U svrhu razvoja Java aplikacija kreirane su četiri, osnovne, Java 2 platforme (slika 1.2.1.):

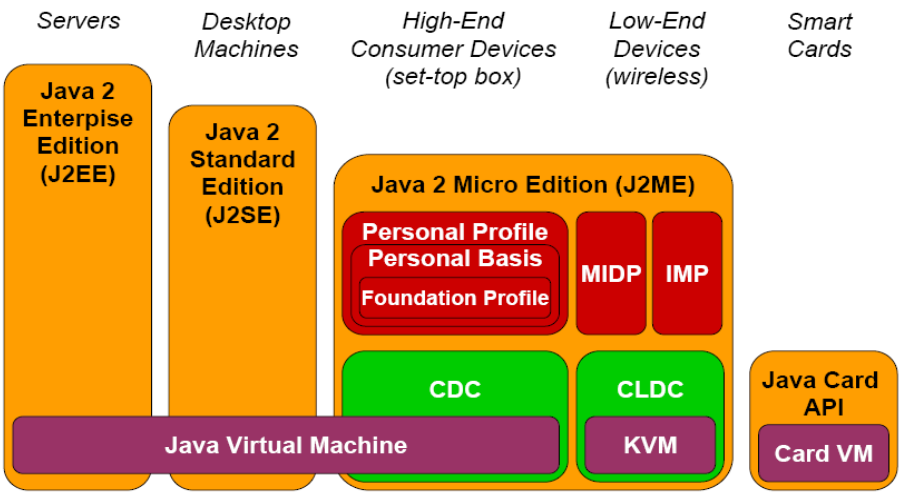
1. **J2EE** (*Java 2 Enterprise edition*) koja je namjenjena korištenju pri izradi poslužiteljskih aplikacija. J2EE obuhvaća potpuni spektar funkcionalnosti.

2. **J2SE** (*Java 2 Standard edition*) namjenjena korištenju na stolnim računalima. J2SE ima nešto ograničeniji spektar funkcionalnosti u odnosu na J2EE.

3. **J2ME** (*Java 2 Micro edition*) namjenjena je korištenju na malim uređajima s ograničenom procesorskom snagom, memorijom, ekranom i tipkovnicom. J2ME ima, zbog gore navedenog, još ograničeniji skup funkcionalnosti od J2SE. Podjeljena je na profile i konfiguracije čijom se kombinacijom može optimalno prilagoditi određenoj vrsti uređaja na kojoj se koristi.

4. **Java CARD API** je Sun-ova platforma posebno razvijena za plaćanje pametnim karticama , transakcije, identifikaciju i sigurnost. Može i ne mora biti sastavni dio J2ME. U kombinaciji s OCF (*Open Card Framework*) Java Card API bi mogla postati otvoreni standard za razvoj sustava pametnih kartica (*smart card*).

[4] JAVA – objektno orijentirani programski jezik razvijen od strane Sun Microsystems (www.java.sun.com)



Slika 1.2.1.

Pregled Java2 platforme s profilima

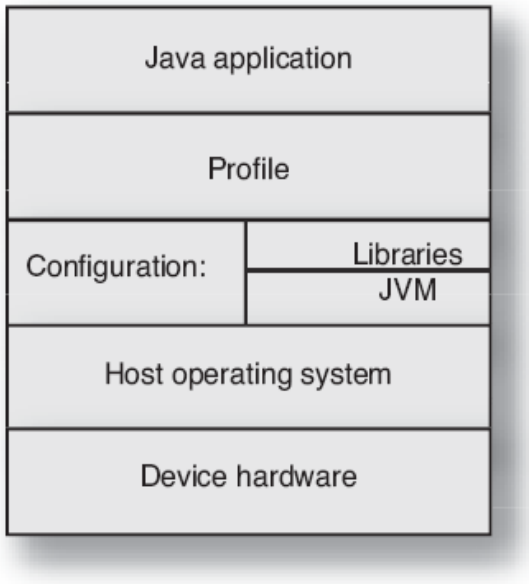
Kada pogledamo veličinu i kompleksnost J2SE (*Java 2 Standard Edition*) platforme, bilo je za očekivati da će se, porastom tržišta malih uređaja, pojaviti potreba za modificiranom Javinom platformom namijenjenom uređajima s ograničenom memorijom, procesorskom snagom te video mogućnostima. J2ME (*Java 2 Micro Edition*) je kreirana kako bi popunila novonastalu prazninu.

Čak i među „malim“ uređajima postoje značajne razlike u mogućnostima i performansama. Na primjer, tipični PDA (*Personal Digital Assistant*) ima više memorije, procesorske snage, ekran višestruko veće rezolucije od tipičnog mobilnog telefona. Kako bi J2ME podržala sve te uređaje i varijacije u njihovim sposobnostima kreiran je koncept konfiguracija i profila.

U osnovi, konfiguracije se odnose na skup mogućnosti određenog uređaja (memorija, mogućnosti komunikacije, ekran, itd.), dok su profili API-ji (*Application Programming Interface*) koji se koriste u određenoj konfiguraciji, omogućujući specifičan skup programskih funkcija za konkretnu vrstu uređaja.

Na slici 1.2.2. vidimo organizaciju J2ME platforme isključujući specifične API-je koje specificiraju pojedini proizvođači.

J2ME platforma je organizirana kao složaj programske podrške. Nove funkcionalnosti se dodaju kako se uzlazno pomičemo po slojevima arhitekture i to osnovne funkcionalnosti preko Java biblioteka (*Java libraries*) i JVM (*Java virtual machine*), skup sistemskih API-ja uključenih u konfiguraciju i skup aplikacijskih API-ja uključenih u pojedine profile.



Slika 1.2.2.
Organizacija J2ME platforme

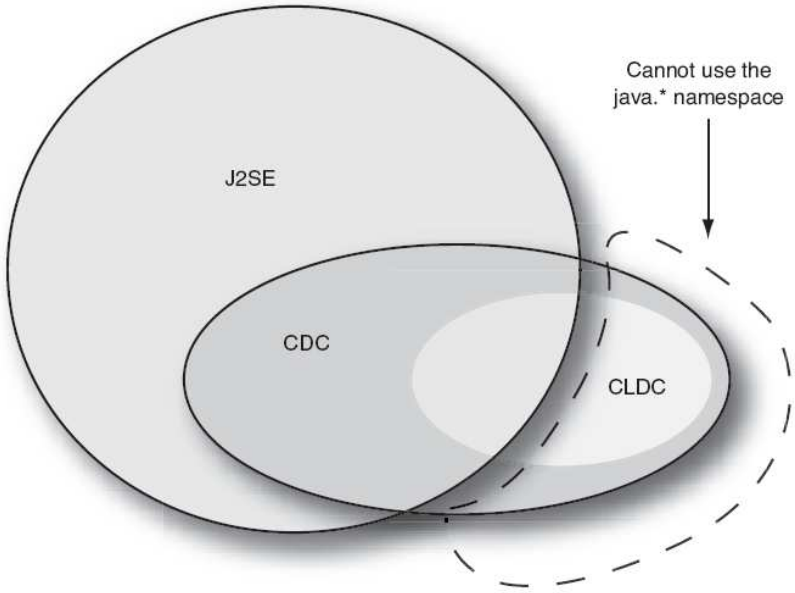
1.2.1. Konfiguracije

J2ME konfiguracija definira Java platformu za određenu klasu malih uređaja (PDA, mobilni telefoni i sl.). Svaka konfiguracija u sebe uključuje osnovne funkcije Java jezika i osnovne Java biblioteke. Svaka J2ME aplikacija se može realizirati tako da koristi jedna od dvije osnovne konfiguracije:

a) CDC (*Connected Device Configuration*)
   - definiran u JSR 36 (*Java Specification Request* - 36)
   - koristi se u uređajima koji implementiraju barem:
     - 512 kB memorije za pokretanje Java aplikacija,
     - 256 kB alocirane radne memorije,
     - mogućnost spajanja na mrežu, poželjno stalnu vezu velike protočnosti.

b) CLDC (*Connected Limited Device Configuration*)
   - CLDC verzija 1.0 definirana je u JSR 30, a verzija 1.1 u JSR 139.
   - CLDC 1.1. u konfiguraciju implementira Float i Double tipove podataka te implementira podršku za slabe reference .
   - koristi se u uređajima koji implementiraju barem:
     - 128 kB memorije za pokretanje Java aplikacija,
     - 32 kB alocirane radne memorije,
     - ograničeno korisničko sučelje (tipkovnica, ekran, ...),
     - koristi energiju iz baterija.

Iz slike 1.2.1.1. vidi se da je konfiguracija CLDC pravi podskup konfiguracije CDC što znači da CLDC ne dodaje niti jednu, novu, funkcionalnost već samo ograničava iskoristivost CDC konfiguracije. Iz slike se još očituje da niti CDC i CLDC nisu pravi podskupovi J2SE platforme. Drugim riječima ove dvije konfiguracije dodaju nove funkcionalnosti u odnosu na J2SE platformu. Te nove funkcionalnosti se odnose na specifične klase potrebne za adekvatan razvoj aplikacija za male uređaje, poštujući sva njihova ograničenja.



Slika 1.2.1.1.

Odnos J2SE platforme te CDC i CLDC konfiguracija

1.2.2. Profili

J2ME profil je proširenje konfiguracije. Profili definiraju biblioteke dostupne razvijatelju aplikacije za specifičnu vrstu uređaja (PDA, mobilni telefoni, dojavljivači, itd.). Profili su kreirali mnogi učesnici kroz Sun-ovu Java zajednicu (*Java Community Process - JCP,* http://jcp.org/). Svakom proizvođaču malih uređaja omogućeno je da se opredijeli hoće li prihvatiti neki od ponuđenih profila ili će kreirati vlastiti.

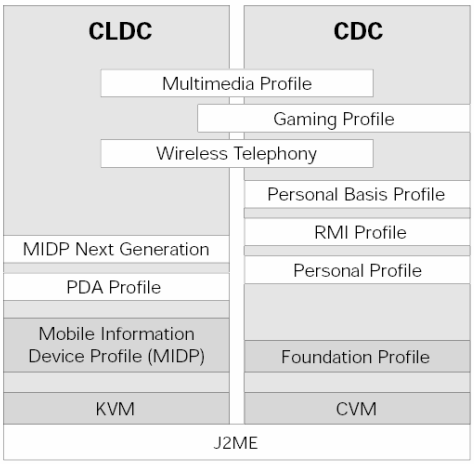
Točno definiranje profila (tablica 1.2.2.1.) osigurava kompatibilnost svih uređaja koji podržavaju taj profil. Na primjer, J2ME aplikacija napisana koristeći određeni profil znači da će se uspješno moći izvoditi na svim uređajima koji podržavaju taj profil. Naravno, bez ikakvih modifikacija. Jedan uređaj može podržavati više različitih profila. Pravilan odabir jednog ili više profila, jedna je od najvažnijih odluka prilikom kreiranja aplikacija.

Profil	**Konfiguracija / VM**	**Virtual Machine**	**Ciljna skupina uređaja**
MIDP	CLDC	KVM	Mobilni telefoni i dojavljivači
PDAP	CLDC	KVM	PDA
Foundation	CDC	CVM	Primarily a foundation for Personal Profile
Personal	CDC	CVM	Pocket PC, tablet, komunikatori
RMI	CDC	CVM	Svi
Personal Basis	CDC	CVM	Svi
Multimedia	CDC/CLDC	CVM/KVM	Svi
Gaming	CDC/CLDC	CVM/KVM	Svi
Telephony	(WTCA) CDC/CLDC	CVM/KVM	Mobilni telefoni

Tablica 1.2.2.1.

Trenutno definirani profili u J2ME

Definiranje budućeg razvoja više vrsta različitih profila omogućuje razvijateljima i proizvođačima da mogu pravovremeno implementirati te profile u svoje aplikacije i uređaje. Na taj se način osigurava stalni napredak platforme i njezina buduća primjena u novim uređajima.



Slika 1.2.2.1.

Plan za budući razvoj profila

Na slici 1.2.2.1. ilustrirani su elementi od kojih se J2ME sastoji. Profili *MIDP* i *Foundation profile* su dostupni dok su ostali u fazi razvoja

MIDP (*Mobile Information Device Profile*) profil je najpoznatiji profil, a ujedno i najstariji. MIDP proširuje CLDC i definira API-je za korisničko sučelje, ulazno/izlazne funkcije, itd., uzimajući u obzir ograničenja ekrana, memorije i procesorske snage mobilnih uređaja koji koriste CLDC konfiguraciju.

1.2.2.1. Usporedba i pregled MIDP 1.0 i MIDP 2.0 profila

J2ME MIDP 1.0 je razvijen 2001. godine te je specificirao osnovnu podršku za korisničko sučelje (*UI – User Interface*) funkcionalnost, primitivnu pohranu podataka i podršku za HTTP. Nedostaje mu podrška za napredniju grafiku, sigurnost te mrežne protokole što su temelji za širu primjenu u razvoju komercijalnih aplikacija. Također jedan od velikih nedostataka MIDP 1.0 specifikacije jest pomanjkanje *provisioning*[5] specifikacije. MIDP 1.0 profil je u industriji mobilnih uređaja bio veoma dobro prihvaćen s više od 75 milijuna uređaja širom svijeta koji su podržavali MIDP 1.0. Kao izravnu posljedicu dobili smo situaciju da su operateri počeli ulagati kapital u razvoj aplikacija temeljenih na novoj tehnologiji. Unatoč tome, MIDP 1.0 je bio prilično ograničen u funkcionalnosti tako da je bila nužna evolucija profila u MIDP 2.0 koji nije samo donio nove funkcionalnosti nego i sveukupno novi model sigurnosti te dodatne API-je za izradu igara i višemedijskih sadržaja pomoću kojih će biti omogućeno korištenje punog potencijala J2ME platforme u pažljivo kreiranom okruženju.

MIDP 2.0 specifikacija ispravlja mnoge nedostatke iz prethodne verzije koju ujedno nadograđuje novim idejama i funkcionalnostima uključujući i *provisioning*. MIDP 2.0 predstavlja novi sigurnosni koncept MIDlet-a od povjerenja (trusted MIDlet) koji koriste „zaštićene domene“ koristeći X.509 PKI (*public key infrastructure*). Takvi MIDlet-i mogu biti digitalno potpisani (*digitally signed*). MIDP 2.0 specifikacija podržava i HTTP i HTTPS protokole, utičnica (*socket*), te konekcija poslužiteljskim utičnicama (*server socket connections*). Ugrađen je i dodatni API za podršku spajanju serijskim priključkom na samom uređaju što je značajno prilikom spajanja na GPS (*Global Positioning System*) prijemnike, BAR kod čitače i slično. To, ujedno, znači da jednostavne J2ME aplikacije mogu omogućiti lokacijske (*location-based*) usluge.

[5] Provisioning - termin u industriji pružatelja bežičnih usluga koji podrazumijeva mogućnost naplate nove aplikacije ili nadogradnje postojeće koja je na uređaj prebačena bežičnim putem

Nova specifikacija omogućuje *push* mehanizam tako da MIDlet-i mogu biti postavljeni i „aktivirani“ od strane vanjskog izvora. *MIDP 2.0 Media API* (`javax.microedition.media`) omogućuje veoma bitnu funkcionalnost generiranja i reprodukciju zvuka s podrškom za MIDI i WAV zvučne formate te reprodukcije video zapisa (*video streaming*). Novom specifikacijom je olakšan i razvoj igara za male uređaje (*MIDP Game API* - `javax.microedition.lcdui.game`). Naravno, MIDP 2.0 specifikacija je u potpunosti kompatibilna s MIDP 1.0 specifikacijom.

1.3. PIM (Personal Information Managment)

PIM (*personal information managment*) je skraćeni termin koji označava upravljanje osobnim podacima kao što su osobni adresar, raspored, imenik, itd. Upravljanje privatnim podacima je prilično osjetljivo zbog zaštite privatnosti podataka i specifičnog formata pohrane. PIM funkcionalnosti je veoma korisno implementirati u male uređaje iz više razloga posebice zato jer se podrazumijeva da takve uređaje korisnik gotovo stalno nosi uz sebe.

PIM API je u J2ME specificiran opcionalnim paketom JSR-75 (*Optional Packages for the J2ME™ Platform*) koji uključuje dvije Java platforme: J2SE platformu i J2ME dodatne pakete usmjerene da podrže mogućnosti specifične za PDA i slične uređaje. Ti dodatni paketi omogućuju pristup upravljanju osobnim podacima u bazama (*PIM API*) i mogućnost upravljanja lokalni datotečni sustav (*FileConnection API*). Ta dva paketa su međusobno nezavisna tako da uređaji mogu podržavati jednog od njih ili oba. Minimalni zahtjev za upotrebu PIM API-ja je da uređaj podržava barem CLDC verziju 1.0.

PIM API je opcionalni J2ME paket koji omogućuje pristupanje i modificiranje PIM baza koje postoje u uređajima koji podržavaju MIDP. Svrha stvaranja PIM API-ja je standardizacija sučelja za pristup tim bazama koje se mogu tada koristiti u mnogim vrstama uređaja na siguran način.

Trenutno je podržan pristup i modifikacija tri različite baze (liste): liste kontakata (*Contact lists*), liste događaja (*Event lists*) i liste poslova (*TO-DO lists*). Te tri liste ne moraju, nužno, biti podržane u svim uređajima, ali zahtjev specifikacije je da barem jedna bude podržana. Neke implementacije mogu sadržavati više listi

jednog tipa, na primjer, mobilni telefon može podržavati kontakt listu smještenu u memoriji samog uređaja i još jednu kontakt listu koja se nalazi na SIM kartici telefona.

PIM API se odnosi prema objektima u bazi kao elementima (*PIMItems*) u listi (*PIMList*). Kao što je već rečeno postoje tri podržane liste: : *ContactList*, *EventList*, i *ToDoList*. PIM klasa sadrži dvije openPIMList metode za pristup danoj PIMlist-i. Listama pristupamo na tri predefinirana načina:

READ_ONLY - isključiva mogućnost čitanja podataka iz liste
WRITE_ONLY – isključiva mogućnost pisanja u listu
READ_WRITE – omogućeno čitanje i upisivanje u listu

Svaka se baza identificira jedinstvenim imenom (i zamjenskim imenima) koje se specificira kod kreiranja baze. U API je uključena metoda `listPIMLists()` koja kao rezultat poziva vraća listu svih imena baza za dani tip baze. Na primjer, kontakt lista može imati ime „Kontakti“, „Contacts,”, „Contactos,” ili „Puhelinluettelo” ovisno o jeziku koji se koristi.

1.3.1.Pregled osnovnih metoda u PIM API-ju

PIMList sadrži nekoliko `items()` metoda za pristup cjelovitom sadržaju PIMItems kao i za odabir određenih elemenata iz liste. Te se metode trebaju koristiti s posebnom pažnjom jer se često izvršavaju prilično sporo (ovisno o veličini baze).

PIMList, također, podržava stvaranje, modificiranje i brisanje elemenata (*PIMItems*). Treba obratiti pažnju da prilikom kreiranja ili modificiranja elemenata, promjene u bazi očituju tek nakon izvršavanja metode `commit()`. Za razliku od stvaranja i modifikacije elemenata, brisanje se očituje odmah po pozivu metode za brisanje elemenata. Da bi, uspješno, izvršili gore navedene akcije, bazu moramo otvoriti u WRITE_ONLY ili READ_WRITE modu.

Svaka PIMList – a ne mora, nužno, podržavati sva polja opisana u API-ju. Svaka implementacija definira podržana polja. Zbog toga se preporuča, prije čitanja polja, provjeriti je li određeno polje podržano u PIMList-i pozivom metode `isSupportedField()` ili `getSupportedFields()`.

Za prebrojavanje elemenata u listi moramo koristiti metodu `countValues()` inače nam se može dogoditi iznimka `IndexOutOfBounds`.

Svaki unos može imati i dodatne atribute koji ga opisuju. Na primjer, telefonski imenik može sadržavati više telefonskih brojeva za isti kontakt s atributima FAX, kućni telefonski broj, itd. U tim slučajevima je nužno provjeriti podržava li PIMList te atribute. Provjera se vrši korištenjem metoda `isSupportedAttribute()` i `getSupportedAttributes()`.

PIM API sadrži i metode za serijalizaciju i deserijalizaciju elemenata (*PIMItems*) što omogućuje MIDletima da komuniciraju s vanjskim aplikacijama. Format za serijalizaciju je standardni vCard (*The Electronic Business Card*) za kontakte (*Contacts*), vCalendar (*The Electronic Calendaring and Scheduling Exchange Format*) za događaje (*Events*) i ToDo elemente. Točan podržani standard se može vidjeti pozivom metode `PIM.supportedSerialFormats()`.

2. Agentski sustav na malim uređajima

Agentski sustav je programska (softverska) infrastruktura koja služi kao okruženje za isporuku, izvršavanje i upravljanje (kontrolu) programskim agentima. Agentski sustavi imaju ugrađene i ostale (dodatne) funkcionalnosti kao što su sigurnosni sustav, bijele i žute stranice, itd. Agentski sustavi se klasificiraju s obzirom na broj agenata kojima mogu upravljati, stupnjem inteligencije te stupnjem pokretljivosti (programskih agenata) koji omogućuju.

2.1. Programski agenti

Programski agent je još jedan oblik programske apstrakcije, u istom smislu kao što su i metode, objekti, i sl. Na primjer, objekt je programska apstrakcija koja opisuje metode i atribute programske komponente. Agent je softverska apstrakcija na još višoj razini apstrakcije koja omogućuje jednostavan i veoma moćan alat za izgradnju kompleksnih programskih entiteta. Za razliku od objekata koji su definirani atributima i metodama, programski agent je definiran svojim ponašanjem (*behaviour*). Ta činjenica je veoma važna jer svodi programiranje agentskih sustava, u biti, na specificiranje ponašanja pojedinih agenata. Ovakav pristup je puno intuitivniji za razvijatelja, ali zahtjeva promjenu načina razmišljanja kao što je bilo potrebno prilikom prelaska iz proceduralnog u objektni način razmišljanja.

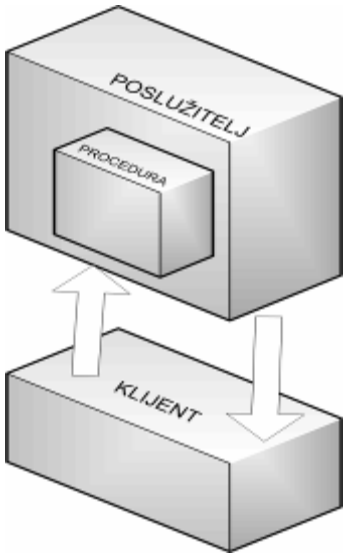
Programski agent ima svojstva:

- **autonoman**
  - sposobnost samostalnog kretanja i rada u sustavu bez potrebe za uplitanjem korisnika u rad agenta,
  - nužno svojstvo,
- **pokretljiv**
  - sposobnost kretanja unutar agentskog sustava,
- **imati cilj**
  - agent mora imati svoj konačni cilj, a agentska platforma treba omogućiti mehanizme za uspješno ispunjavanje poslova za koje je agent namijenjen,
- **postojan**
  - prilikom premještanja agent mora zadržati trenutno stanje izvođenja. Stanje agenta obuhvaća trenutne vrijednosti varijabli sadržanih unutar agenta, programski kôd, te liniju kôda od koje se agent počinje izvršavati kad stigne na odredište,
- **komunikativan**
  - da bi izvršili postavljene im zadatke, agenti u pravilu moraju razmjenjivati podatke, bilo međusobno, bilo sa okolinom u kojoj se izvršavaju,
  - nužno svojstvo, kako bi mogao komunicirati s ostalim agentima, sustavom ili korisnikom i
- **inteligentan**
  - podrazumijeva sposobnost učenja te sposobnost odlučivanja na temelju podataka koje agent dobiva iz okoline.

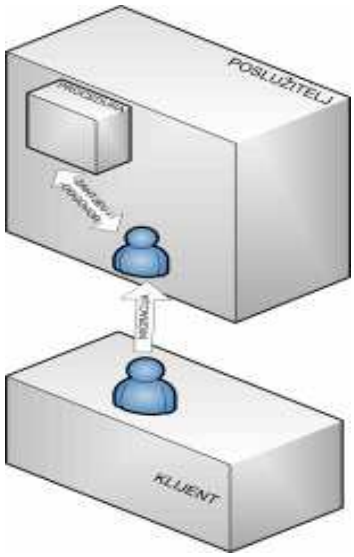
Programski agenti, kao i ljudi, mogu posjedovati različite razine kompetencija za obavljanje određenog zadatka. Ta razina kompetencije se naziva inteligencijom agenata. Npr. programski agent A koji ima zadatak obrade e-pošte može svoj zadatak odraditi jednostavnim primanjem i slanjem e-pošte, dok će drugi, „inteligentniji“ agent imati i „moć“ prepoznavanja neželjene e-pošte ili malicioznih dodataka.

Programski agenti najefikasnije ispunjavaju svoje zadatke u slučaju kada mogu surađivati s ostalim agentima u okruženju. Skupina agenata koji komuniciraju i surađuju se naziva agencija. Dizajneri sustava baziranih na programskim agentima trebaju implementirati višeagentske sustave uzimajući u obzir mogućnosti svakog pojedinačnog (specijaliziranog) agenta te iskoristiti njegove mogućnosti u cilju efikasnijeg izvršavanja zajedničkog zadatka. Npr. Aplikacija za elektroničko poslovanje agente specijalizirane za kupnju, prodaju, burzu novca i dobara, upravljanje bazama podataka, e-poštu, itd. Svi ti agenti moraju biti u mogućnosti međusobno komunicirati u cilju postizanja zajedničkog cilja.

U osnovi, programske agente dijelimo na pokretne i stacionarne. Stacionarni programski agenti nemaju mogućnost kretanja u sustavu. Oni stoga svoje aktivnosti u višeagentskom sustavu isključivo koordiniraju primjenom agentskih konverzacijskih protokola. S druge strane, pokretni programski agenti imaju mogućnost preseljenja (fizičko preseljenje) s jedne lokacije na drugu te se na taj način izvršavati na raspodijeljenim sustavima. Na taj se način klijent seli na poslužiteljsko računalo i tako se udaljeno međudjelovanje[6] (Slika 2.1.1.) pretvara u lokalno (Slika 2.1.2.).



Slika 2.1.1.
Udaljeno međudjelovanje



Slika 2.1.2.
Međudjelovanje primjenom pokretnih programskih agenata

[6] Danas se većina raspodijeljenih aplikacija temelji na klijent-poslužitelj arhitekturi u kojoj klijentska aplikacija s poslužiteljem komunicira putem telekomunikacijske mreža i to razmjenom upita i odgovora. Vrijedi i obrnuti slučaj

2.2. Agentska platforma

Da bi agenti mogli izvršavati svoje zadatke, mora postojati okruženje koje će prihvatiti agenta i omogućiti mu izvođenje. Takvo okruženje se naziva agentska platforma. Platforma upravlja životnim ciklusima agenata (stvaranje i uništavanje agenata) te omogućuje njihovu migraciju i komunikaciju. Agentska platforma je smještena na poslužitelju u mreži. Više takvih, međusobno povezanih, poslužitelja (s agentskim platformama) čini agentsko okruženje unutar kojeg se agenti kreću i izvode operacije.

Agentska platforma (slika 2.2.1.) mora pružati mehanizme za:

- upravljanje agentima (agentski sloj),
    - kreiranje, uništavanje, identifikacija, kretanje
- sigurnost (sigurnosni sloj) i
    - zaštita privatnosti, informacija, zaštita od neovlaštenog pristupa
- komunikaciju među agentima (komunikacijski sloj)
    - isporuka poruka koje agenti, međusobno, razmjenjuju

Slika 2.2.1.

Osnovni prikaz arhitekture agentske platforme

JADE (*Java Agent DEvelopment Framework*) je razvojni okružje za više agentske sustave u skladu sa specifikacijama FIPA za inteligentne agente. Sadrži dva dijela: FIPA sukladnu agentsku platformu i paket za razvoj Java agenata. JADE je izveden u Java programskom jeziku pa se i agenti programiraju u njemu. Platformom JADE se želi postići jednostavan razvoj, koji će osigurati sukladnost sa standardima kroz sustav usluga i agenata. JADE ima sljedeće karakteristike:

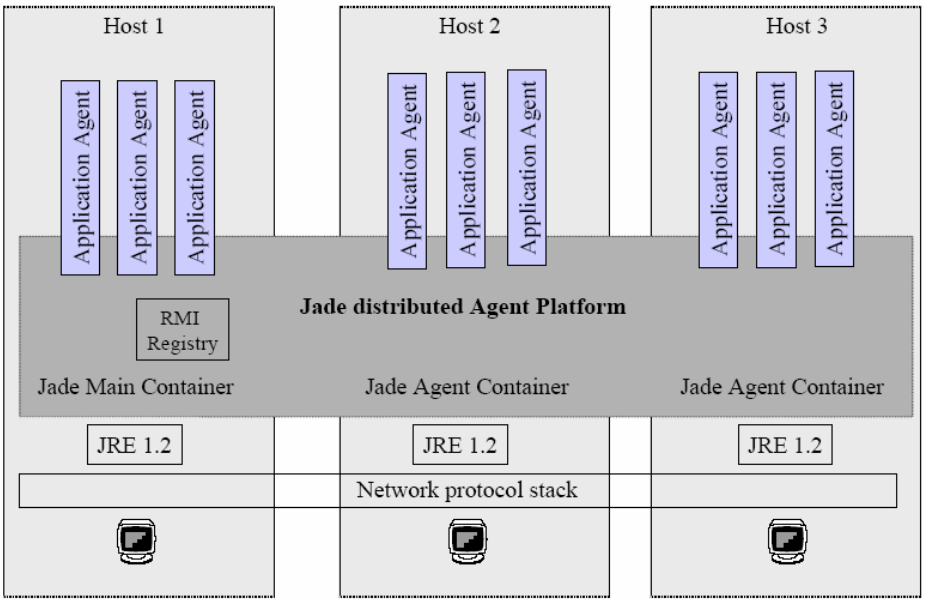
- FIPA sukladna agentska platforma, sadrži sustav za upravljanje agentima (*Agent Management System* – AMS), DF (*Directory Faciliator*) služi kao imenik agenata i kanal za komunikaciju agenata (*Agent Communication Channel* – ACC). Ova se tri agenta automatski aktiviraju kod pokretanja platforme,
- distribuirana agentska platforma. Agentska platforma može biti podijeljena između nekoliko računala ako među njima nema vatrozida. Agenti su implementirani kao Java nit, a događaji se koriste za komunikaciju između agenata na istom računalu. Paralelni zadaci mogu se izvesti jednim agentom, a platforma JADE raspoređuje te zadatke na efikasan način,
- nekoliko DF agenata može biti pokrenuto kako bi se implementirala više-domenska aplikacija,
- programirano sučelje za pojednostavljenu registraciju agentskih usluga unutar jedne ili više domena,
- transportni mehanizam i sučelje za slanje/primanje poruka prema/od drugih agenata,
- FIPA97 sukladni IIOP (*Internet Inter ORB Protocol*) protokol za spajanje s drugim platformama,
- transport ACL (*Agent Communication Language*) poruka unutar iste agentske platforme. Poruke se šalju kao Java objekti. Kada pošiljatelj ili primatelj ne pripadaju istoj platformi, poruka se automatski pretvara u/iz FIPA sukladnog formata niza znakova,
- skup FIPA protokola za interakciju spremnih za korištenje,

- automatska registracija agenata s agentskim upravljačkim sustavom AMS,
- FIPA sukladno pridjeljivanje imena i
- grafičko korisničko sučelje za nadgledanje i upravljanje pojedinom platformom.

Platforma JADE se sastoji od nekoliko spremnika (*container*). Mogu se nalaziti na raznim terminalima (računalima, dlanovnicima i pokretnim telefonima). Na terminalu na kojem je pokrenuta platforma uvijek se nalazi glavni spremnik, a drugi mogu biti raspoređeni po terminalima. Time se platforma raspoređuje na nekoliko spremnika kao na slici

JADE je međuoprema koja služi za razvoj više agentskih sustava. Ono sadrži:

- izvršno okružje u kojem žive JADE agenti, a ono treba biti aktivno na terminalu prije nego se agenti mogu pokrenuti,
- skup klasa koje programeri koriste za razvoj agenata i
- grafičke alate koji omogućavaju upravljanje i nadgledanje aktivnosti pokrenutih agenata



Slika 2.2.2..

Agentska platforma JADE distribuirana preko više spremnika

Svaki spremnik može sadržavati nekoliko agenata. Glavni spremnik mora biti stalno aktivan, a ostali spremnici se trebaju prijaviti prilikom pokretanja. Prvi spremnik koji se pokreće treba biti glavni spremnik, a ostali trebaju biti obični spremnici koji znaju gdje se nalazi glavni spremnik. Drugi glavni spremnik pokrenut na drugoj lokaciji, stvara novu platformu na koju se mogu prijavljivati novi spremnici.

2.3. Pregled JADE-LEAP agentske platforme

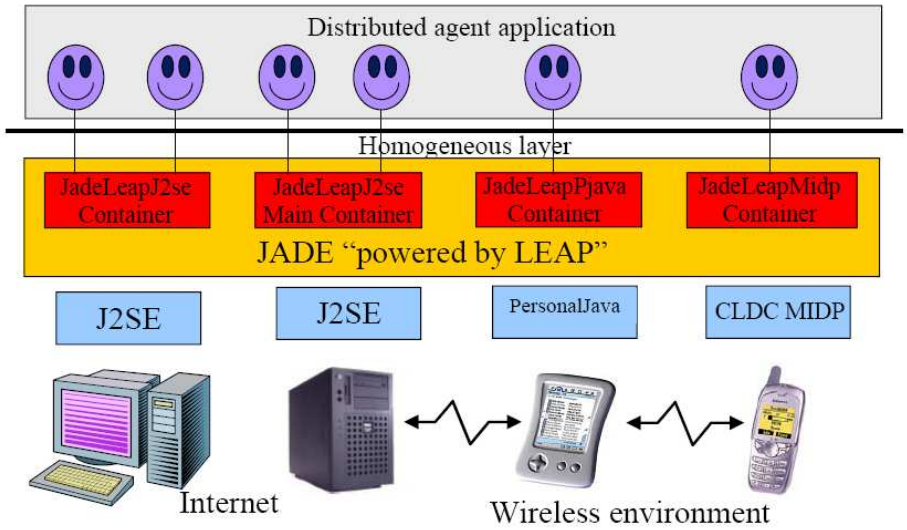
Kao što je u uvodu rečeno JADE-LEAP je prva agentska platforma, usuglašena s FIPA standardima, namjenjena izvršavanju na malim uređajima. Ona je zamišljena (arhitekturalno) kao JADE platforma s dodatkom LEAP biblioteka (*libraries*). Samostalna JADE platforma se ne može izvršavati na malim uređajima iz nekoliko osnovnih razloga :

1. cjelovito JADE okruženje ima memorijske zahtjeve od nekoliko MB, što je veoma zahtjevno za male uređaja,
2. JADE zahtjeva Java JDK 1.4 ili novije, dok su mali uređaji ograničeni na PersonalJava ili MIDP profil i
3. bežične veze imaju drugačije karakteristike u odnosu na fiksne mreže.

LEAP dodatak u kombinaciji s JADE platformom, zamjenjuje neke dijelove JADE jezgre tvoreći modificirano izvršno okruženje (slika 2.1.) koje nazivamo JADE-LEAP (JADE powered by LEAP). JADE-LEAP se može koristiti sa tri različite okoline:

1. J2SE
   - za izvršavanje na osobnim računalima, poslužiteljima,
2. pJAVA
   - za izvršavanje na ručnim računalima koji podržavaju *PersonalJava* okruženje (većina današnjih PDA računala),
3. MIDP
   - za izvršavanje na malim uređajima koji podržavaju MIDP (pretežito mobilni telefoni s podrškom za Javu).

Iako se, arhitekturalno, razlikuju, sve ove tri varijacije osiguravaju podjednaki skup API-ja kako bi razvijateljima omogućili homogenost pri razvoju aplikacija bez obzira na uređaje za koje ih razvijaju.



Slika 2.1.

Prikaz JADE-LEAP izvršno okruženja

Unatoč tome, ipak, postoje neke funkcionalnosti koje se mogu koristiti na J2SE i pJAVA platformama, a ne na MIDP platformi. To je izravna posljedica nemogućnosti korištenja određenih klasa koje nisu dostupne na MIDP platformama. Prednosti ovakvog pristupa su, generalno, u tome što nije potrebno posebno učenje i navikavanje na novu platformu jer se većina dokumentacije kreirane za JADE platformu primjenjuje i na JADE-LEAP platformu. Dok je , s druge strane, nužno imati na umu da se JADE kontejneri i JADE-LEAP kontejneri ne smiju, zajedno, koristiti na jednoj platformi. Također postoje još neka ograničenja, specifična za određenu Java platformu.

Tako za MIDP imamo :

- svi JADE administracijski alati imaju GUI (*Graphical User Interface*) temeljeno na Java SWING paketima tako da se ne mogu izvršavati na pJava i MIDP baziranim uređajima. Isto se odnosi i na `jade.gui` pakete,

- nije moguće koristiti „*sniffer*“ i „*introspector*“ agente u samostojećim kontejnerima, dok je moguće u podijeljenim kontejnerima,

- usluge „*MainReplication*“ i „*PersistentDelivery*“ nisu omogućene,

- nije omogućena pokretljivost agenata,

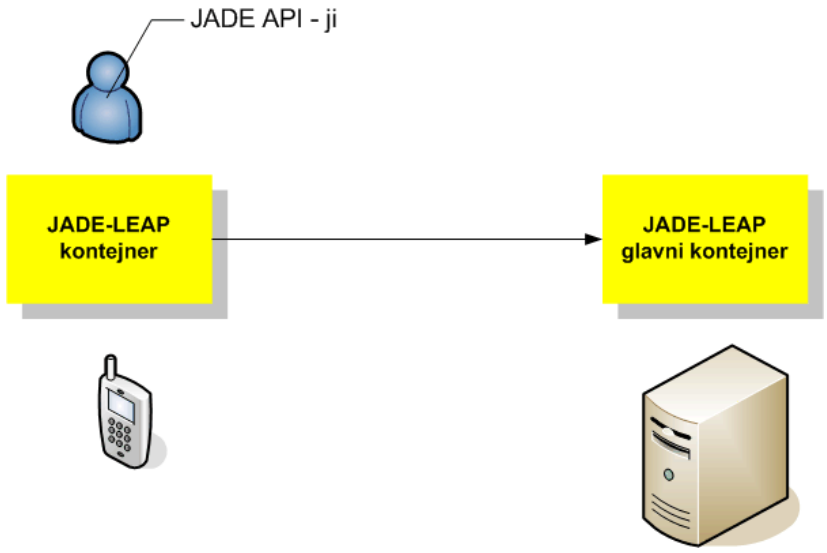
- nije moguće klonirati agente,

- reflektivni introspektor agent (`jade.content.onto.ReflectiveIntrospector` i `jade.content.onto.BCReflectiveIntrospector`) nisu podržani i

- nije dostupan `jade.wrapper` paket i metode unutar `jade.core.Runtime` kase s iznimkom `StartUp()` i `ShutDown()` metoda.

JADE-LEAP izvršno okruženje se, na malim uređajima, može izvršavati na dva načina:

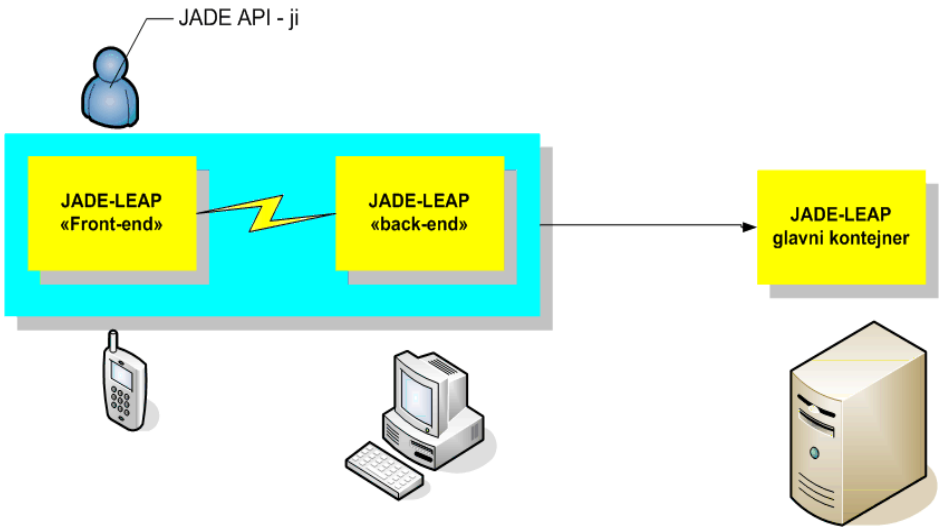
1. „*Stand-alone*“ ili samostojeći način (Slika 2.2.) gdje se cjelokupni kontejner izvršava na malom uređaju (*personal Java*). Ovakav model se može primijeniti kod uređaja koji imaju više procesorske snage kako bi mogli u potpunosti izvršavati sve procedure.



Slika 2.2.

Samostojeći (*stand alone*) model izvršavanja

2. „*Split*“ ili podijeljeni način izvršavanja (Slika 2.3.) gdje je kontejner podijeljen u dva dijela: prednji (*front end*), koji se izvršava na malom uređaju i stražnji (*back end*) dio, koji se izvršava na J2SE poslužitelju te su međusobno povezani stalnom vezom. Ovakav model je pogodan za bežične uređaje s veoma ograničenim resursima te od verzije JADE-LEAPa 3.4. isključivi način izvođenja za MIDP profil.



Slika 2.3.
Podijeljeni (*split*) model izvođenja

JADE-LEAP platforma se sastoji, kao i kod JADE platforme, od nekoliko osnovnih elemenata:

1. **AMS** (*Agent Management System*)
   - obavezna komponenta sustava,
   - osigurava bijele stranice (*white papers*), te pruža usluge praćenja životnog ciklusa agenata i usluge direktorija agenata,
   - osigurava kontrolu pristupa, kreiranje, brisanje, migraciju, prijavu i odjavu na sustav,
   - ...

2. **DF** (*Directory Facilitator*)

- obavezna komponenta sustava,
- osigurava podatke o agentima, uslugama koje pružaju,...
- omogućuje prijavu i odjavu novih usluga za pojedine agente (žute stranice),
- ...

3. **ACC** (*Agent Communication Channel*)

- osigurava interakcija agenata putem protokola za prijenos poruka (MTS),
- prosljeđuje i usmjerava poruke,
- ...

4. **MTS** (*Message Transport System*)

- osigurava komunikaciju agenata na drugim platformama slanjem i primanjem ACL (*Agent Communication Language*) poruka

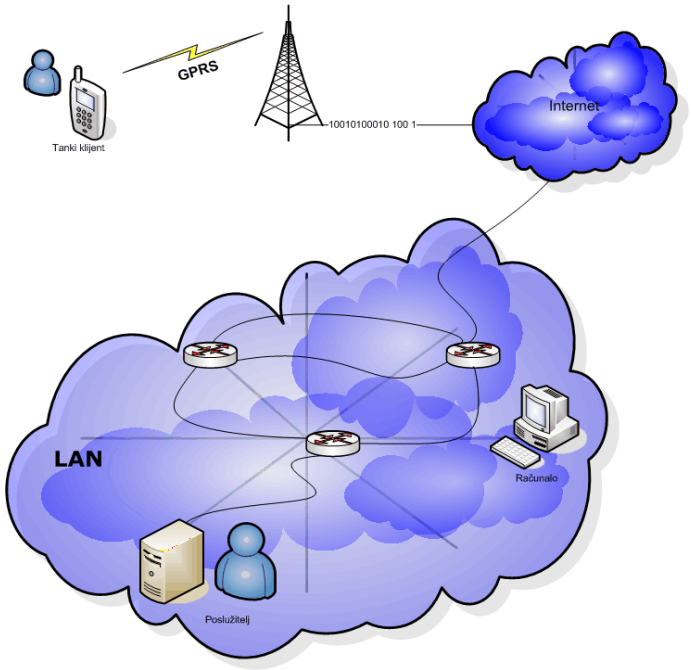
Razvojno okruženje uključuje i grafičko sučelje (u daljnjem tekstu GUI – *Graphical User Interface*) koje olakšava kontrolu i upravljanje kreiranim programskim agentima. To su sučelja RMA (*Remote Monitoring Agent*), *Dummy Agent*, *Sniffer Agent*, *Introspector Agent* i *GUI-DF Agent.*

2.4. Pokretni programski agenti na malim uređajima

U današnjem društvu znanja građani zahtijevaju brz, pouzdan, jeftin i fleksibilan način pristupa svim vrstama informacija. Također žele imati pristup tim informacijama s bilo kojeg mjesta i u svakom trenutku koristeći najnovije tehnologije kao što su mobilni telefoni, PDA, PC računala, itd. U posljednje vrijeme vidljiv je trend objedinjavanja funkcionalnosti tih uređaja u jedan, jedinstveni, uređaj. Uz postojanje bežične mrežne infrastrukture (UMTS, GPRS, WiFi, IrDa, ...) omogućeno je implementirati programske agente da, umjesto svojih korisnika, pronalaze i upravljaju informacijama pružajući im široki spektar zahtijevanih funkcionalnosti.

2.4.1. Primjena pokretnih programskih agenata na malim uređajima

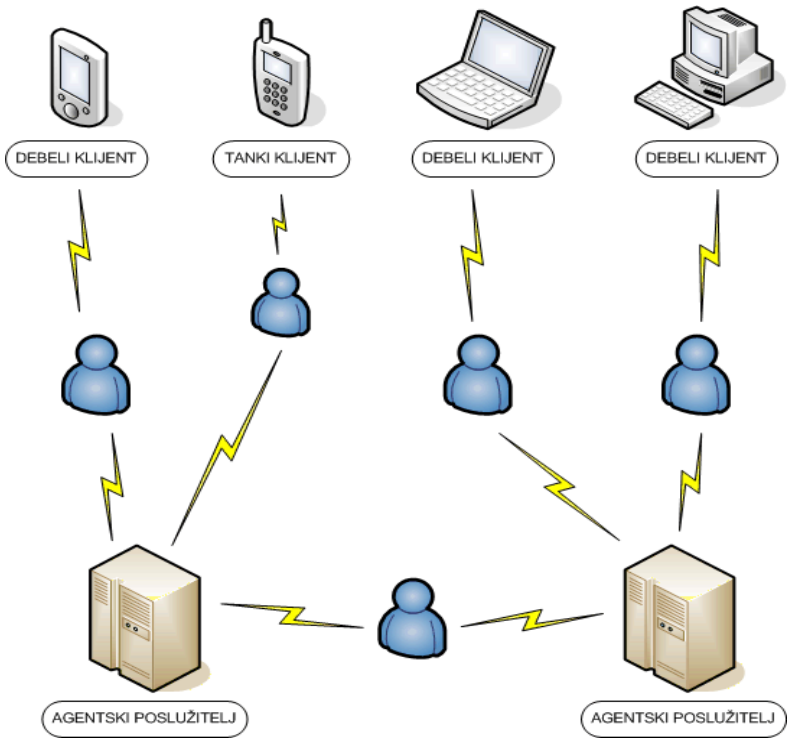
Najčešća primjena agenata na „malim uređajima“ je pri izradi aplikacija za prikupljanje, prikaz i sinkronizaciju informacija. Prvenstveno se primjena temeljila na implementacije vezane za elektronički prikaz informacija sa burze, lokacijske usluge (rezervacija i pozivanje taksija, potraga za lokacijskim informacijama – gdje je najbliži restoran, parkirna mjesta, ...). Realni prikaz mrežne topologije u primjeni malih uređaja koji pružaju usluge s dodatnom vrijednošću dan je na slici 2.4.1.1.



Slika 2.4.1.1.
Realni prikaz mrežne topologije

2.4.2. Principi arhitekture agentskog sustava za primjenu na malim uređajima

Pokretni agenti se na „malim uređajima“, najčešće primjenjuju kao „tanki klijent“ (*thin client*) na kojemu neko drugo, jače, računalo samo inicijalizira, pokreće i upravlja agentima čineći mobilni uređaj platformom za stacionarne agente kao što je prikazano na slici 2.1.1.1. Ponekad se primjenjuje i drugi model , tzv. „debeli klijent“ (*thick client*), kada mali uređaj vrši sve te funkcije (npr. u zrakoplovima kada je veoma skupo koristiti mrežnu infrastrukturu ili kada uređaj raspolaže sa respektabilnom procesorskom snagom, ..).



Slika 2.1.1.1.

Prikaz najčešćeg slučaja upotrebe agenata na malim uređajima

U načelu treba izbjegavati opterećivanje malog uređaja te je preporučljivo graditi takve implementacije sustava koje većinu „logike“ i komunikacije izvršavaju na agentskim poslužiteljima (*Agent server*). Arhitekture sustava slične onom na slici 2.1.1.1. ujedno omogućuju primjenu agenata u heterogenim mrežnim okruženjima.

Maksimalni broj instanci agenata koje se mogu inicijalizirati na pojedinom uređaju, kao i njihove performanse, direktno ovisi o performansama i specifikaciji samog uređaja (prvenstveno količina slobodne memorije i procesorska snaga uređaja), ali i o samoj implementaciji pojedinog programskog agenta. Najveći utjecaj na performanse samog agenta imaju implementirani algoritmi za izvršavanje agentskih zadataka (interne performanse) i performanse korištene agentske platforme (vanjske performanse). U slučaju da je platforma „svjesna“ performansi svakog agenta, može se podesiti tako da optimalno upravlja agentima te na taj način osloboditi određenu količinu resursa na samom uređaju.

2.4.3. Modeli komunikacije

Osobni agenti koji se pokreću na pokretnim uređajima moraju moći komunicirati (bežično) s agentima koji pružaju usluge i koji se izvršavaju na nepokretnim računalima (PC, poslužitelji, ...). JADE-LEAP radi na vrlo visokoj razini komunikacije, stvarajući TCP/IP konekcije između kontejnera, bez da vodi brigu o fizičkoj vezi na kojoj će se ta veza temeljiti. Stoga aplikacijski programeri moraju pronaći način na koji će pružiti vezu. Tu se postavljaju dva osnovna pitanja: koji komunikacijski protokol će se koristiti (npr. point-to-point protocol (PPP)) i koju vrstu veze će se koristiti. Gledajući sa stajališta izbora bežične veze nudi nam se široki spektar; najpopularnije su GPRS, UMTS, WLAN, Wi-Fi, Bluetooth i infracrvena veza.

GPRS

- U današnje vrijeme, najčešći, oblik komunikacije malih uređaja (posebice mobilnih telefona). GPRS vezu osigurava pružatelj usluge mobilne telefonije te za to prima određenu naknadu koja se obračunava u odnosu na količinu prenesenih podataka (u kB). Veoma je praktična jer je dostupna na veoma širokom području (gdje god je dostupan GSM signal).

WiFi

- Komunikacija WiFi mrežom je veoma praktična jer je prilično jeftina (ponegdje čak i besplatna), ali postoje prilična ograničenja i to prvenstveno u tome što većina malih uređaja (posebice mobilni telefoni), još uvijek, nema ugrađeni modul za korištenje WiFi mreža, a i same mreže nisu dostupne na širem području (uglavnom su zastupljene u zatvorenim prostorima ili u visoko urbanim područjima)

Bluetooth

- Koristeći Bluetooth dobiti ćemo besplatnu vezu (ukoliko imamo potrebnu opremu koja je danas gotovo standardno ugrađena u male uređaje) dobre kvalitete, no osnovni nedostatak ove veze je što moramo biti u krugu od oko 10 m od pristupne točke. Ovakav tip veze nikako ne bi odgovarao radu u pokretu prema kojem se teži, te bi se ovakva vrsta veze morala zamijeniti nekom drugom koja ipak ima veći domet. No kako je već spomenuto, kod agenata nije bitna vrsta veze, tako da se sustav može koristiti s bilo kojim tipom fizičke veze bez potrebe za modifikacijom.

2.4.4. Ad-Hoc arhitektura

Ad-Hoc mreže nemaju infrastrukturu koja bi podržavala njihovo funkcioniranje. Takve mreže se sastoje od promjenjivog broja pristupnih točaka (peer-ova) koje, međusobno, dijele svoje mrežne resurse tvoreći mrežnu, kolaboracijsku, okolinu kako bi omogućili komunikaciju uređaja koji nisu u direktnoj vezi. Klasični model autorizacije i autentifikacije, nije moguć, zbog nepostojanja centralnog upravljačkog tijela u mreži tako da se komunikacija oslanja, isključivo na povjerenje.

Kada se sve navedeno uzme u obzir dolazimo do zaključka da je implementacija agentskog sustava u ovakvim mrežama prilično kompleksna, ali ne i nemoguća. Da bi izgradili agentski sustav temeljen na Ad-Hoc mrežama moramo izvršiti modifikacije na samoj agentskoj platformi.

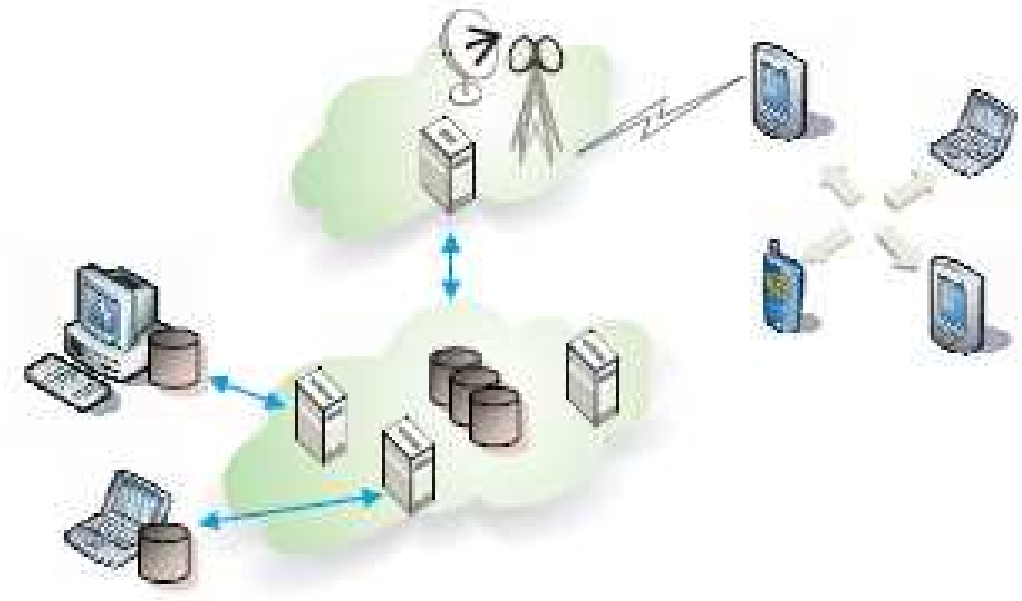
Osnovne potrebne modifikacije su :

- kreiranje DA (Discovery Agent-a koji će moći pronalaziti susjedne agente), uklanjanje koncepta podijeljenog kontejnera kako bi Ad-Hoc pristupne točke mogle kreirati nezavisne kontejnere na jedinstvenoj platformi,

- također je nužno osigurati mehanizam obavješćivanja sudionika o promjenama u topologiji mreže i

- potrebno je ukinuti postojanje DF-a i AMS-a kao obaveznih elemenata agentskog sustava već je nužno omogućiti njihovo pokretanje na zahtjev (*on-demand*).

Očito je da mrežni sustavi, posebice na malim uređajima, evoluiraju prema Ad-Hoc mrežama (slika 2.4.4.1.) tako da je razrada ovakvog pristupa sve interesantnija i aktualnija.



Slika 2.4.4.1.

Prikaz heterogene mrežne arhitekture s Ad-Hoc dijelom

3. Arhitektura agentskog sustava za ugovaranje sastanaka

3.1. Općenito

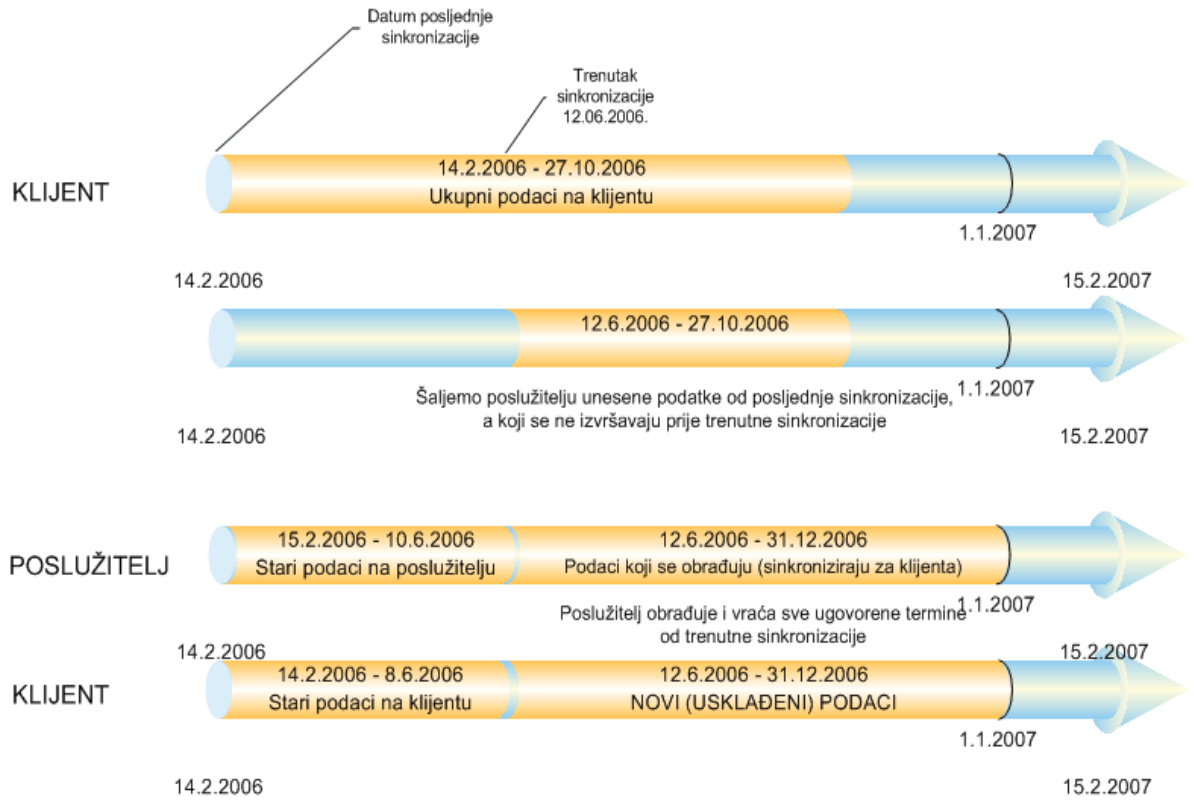
U današnje vrijeme, sve bržega života, od velike je važnosti, pravilno i redovito, voditi evidenciju o dogovorenim sastancima i planiranim obvezama. Vođenje takve evidencije se sastoji od nekoliko osnovnih funkcionalnosti:

1. unos novih podataka (obaveza),
2. brisanje informacija,
3. modifikacija unesenih informacija i
4. usklađivanje obveza sa sustavima višeg ili nižeg prioriteta (poslužitelji, ...).

Pošto vođenje brige o gore navedenim funkcionalnostima oduzima veoma mnogo vremena, čak toliko da je takav sustav upitno koristan, treba omogućiti automatizaciju što većeg broja traženih funkcionalnosti. Na taj način smanjujemo potrebu za intervencijom korisnika te im na taj način ostavljamo više vremena za ispunjavanje ostalih obveza. Posao automatizacije možemo prepustiti programskim agentima koji su idealni da taj posao obave u korist svojih „vlasnika“ (korisnika sustava). Priroda programskih agenata se gotovo u cijelosti može iskoristiti za usklađivanje (sinkronizaciju) podataka sa ostalim sudionicima sustava te djelomično za modifikaciju, unos novih i brisanje starih, već postojećih podataka.

Usklađivanje (sinkronizacija) podataka je najkritičnija od funkcionalnosti zbog više načina na koji se može realizirati. Naime, nema smisla usklađivati podatke (sastanke) koji su starijeg datuma (i sata) od trenutnog jer su ti sastanci već održani te bi njihovom obradom samo opterećivali sustav (klijenta, mrežu i poslužitelj). Također, radi optimizacije rada, može se uvesti dodjeljivanje prioriteta svakom pojedinom sastanku te na taj način izbjeći korisnikovu intervenciju za većinu odluka koje su nužne ako su dva ili više sastanka planirana u isto vrijeme (intervencija je i dalje nužna u slučaju da se preklapaju dva ili više sastanaka istog prioriteta).

Usklađivanje preporučujem izvršavati samo za podatke koji su nastali od trenutka posljednje sinkronizacije za koje termin izvršavanja nije stariji od trenutka sinkronizacije koja je u tijeku tj. nisu već istekli (Slika 3.1.1.). Na ovaj način pokušavamo rasteretiti sustav i ubrzati rad (prijenos i obradu podataka).



Slika 3.1.1.

Primjer usklađivanja sastanaka između klijenta i poslužitelja

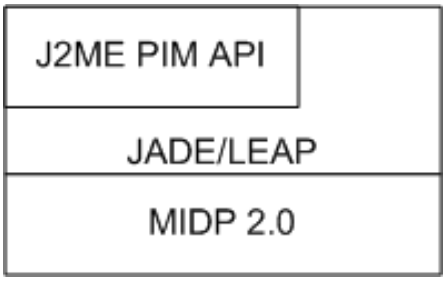
Slika 3.1.1. prikazuje primjer usklađivanja podataka za hipotetski period od 14. veljače 2006. godine do 01. siječnja 2007. godine. Na klijentskom uređaju su uneseni svi podaci od strane korisnika te su im nadodani i podaci od strane poslužitelja prilikom posljednje sinkronizacije. Hipotetski trenutak sinkronizacije je 12. lipnja 2006. godine u 0:00 sati. U ovakvom primjeru, klijent poslužitelju šalje sve podatke koje sadrži (posljednji rezervirani termin je 27. listopada 2006. godine), a da su nastali od trenutka posljednje sinkronizacije i da im je termin izvršavanja nakon 12. lipnja 2006. godine u 0:00 sati. Na taj način optimalno koristimo mrežne resurse.

Poslužitelj prima poslane podatke te ih uspoređuje s onima koji su trenutno aktualni u zajedničkom popisu sastanaka. Nakon usporedbe, poslužitelj šalje klijentu sve novonastale ili izmijenjene termine sastanaka koje klijent dodaje u svoj popis sastanaka. Moguća je i izvedba da klijent šalje samo one podatke koji u interesnom intervalu imaju noviji datum revizije od trenutka posljednje sinkronizacije, ali bi, zbog obilnog korištenja resursa (uzrokovanog velikim brojem usporedbi i odluka – *if ... then* ...) malog uređaja, doveli u pitanje njegov normalan rad, pa zbog toga preporučam mali ustupak na račun povećanja količine mrežnog prometa.

Algoritam za optimalno izvođenje sinkronizacije nije dio aplikacije ovog diplomskog rada pa ga preporučam kao jednu od mogućnosti za buduću razradu aplikacije.

3.2. Pregled korištene arhitekture

Klijentska aplikacija je realizirana korištenjem J2ME PIM API-ja (JSR-75) u kombinaciji s JADE/LEAP agentskom platformom. Pošto je aplikacija razvijana za primjenu na mobilnim telefonima, bilo je nužno proširiti JADE platformu LEAP bibliotekama. PIM API je korišten radi jednostavnije i standardizirane realizacije unosa, modifikacije i brisanje osobnih podataka, dok je za usklađivanje korišten agentski sustav JADE-LEAP posebno razvijen za upotrebu na malim uređajima s ograničenim resursima. Kako mobilni telefoni, u većini slučajeva, imaju isključivo podršku za MIDP 1.0 ili MIDP 2.0 J2ME profil, bilo je potrebno primijeniti podijeljeni model agentske arhitekture (*Split mode* – Slika 2.3.).



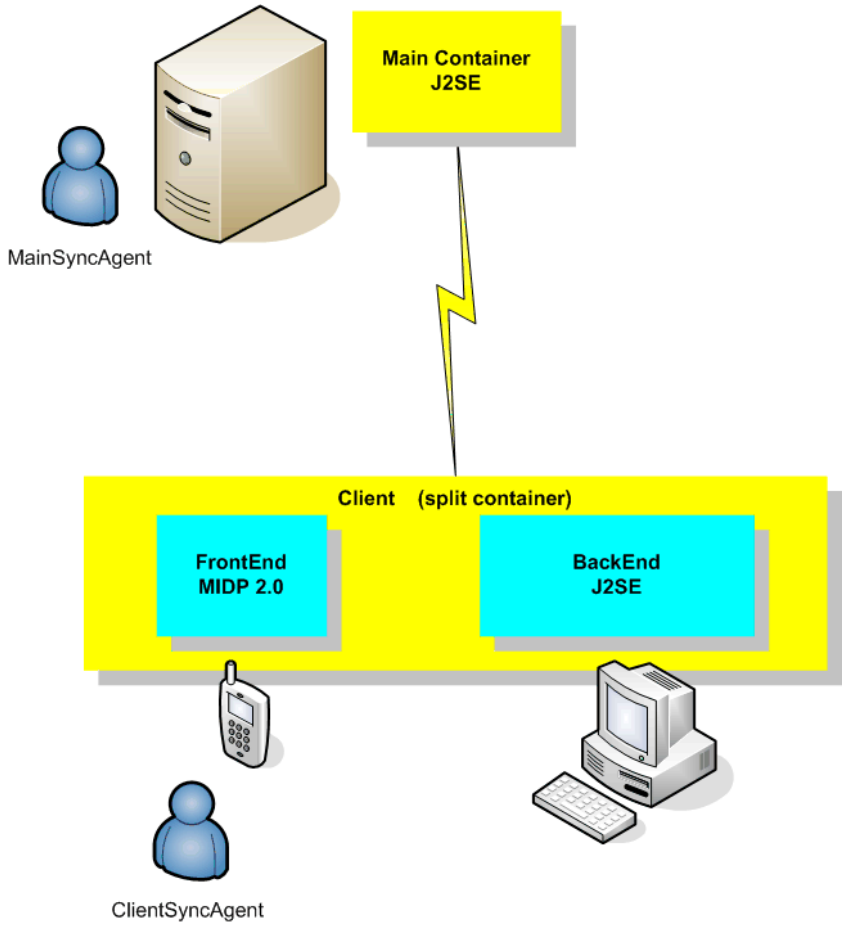
Slika 3.2.1.

Prikaz klijentske arhitekture

Također su, na klijentskom dijelu, primijenjeni principi arhitekture tankog klijenta s što manje kompleksnih algoritama i programskih ispitivanja, kako bi u što većoj mjeri rasteretili mali uređaj. Veći dio takvih algoritama koristi se u agentima koji se izvršavaju na poslužitelju i ostalim računalima u mreži.

4. Implementacija

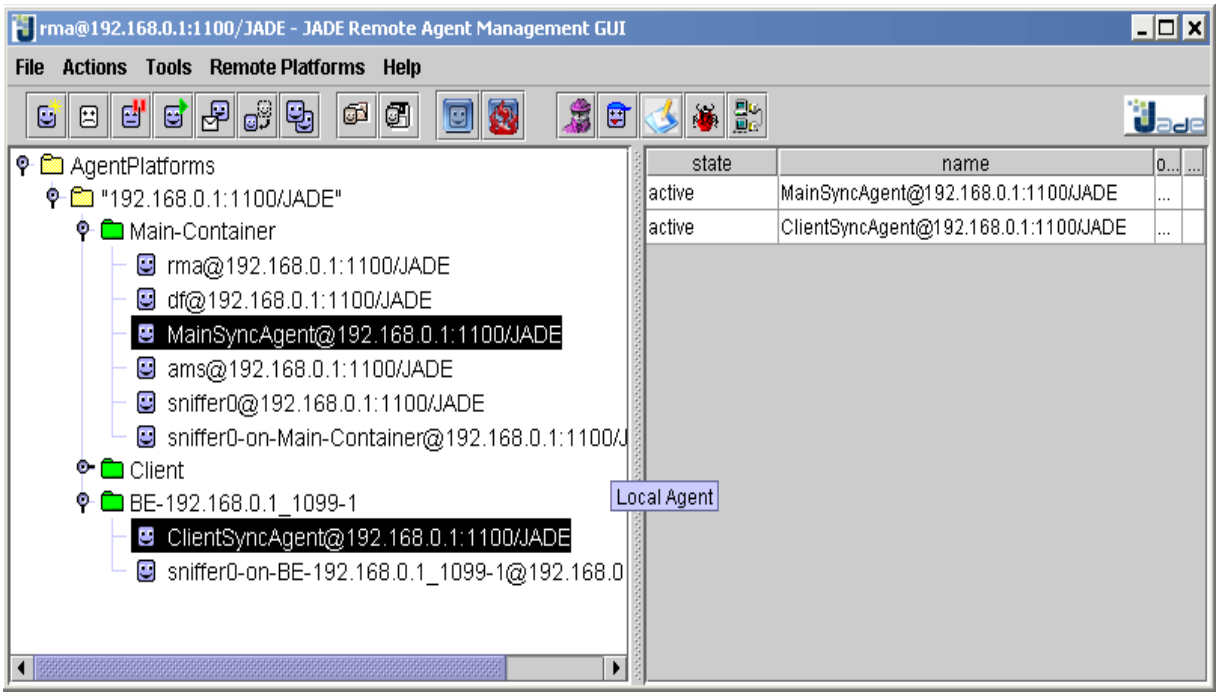
Aplikacija izrađena u sastavu ovog diplomskog rada služi za prikaz osnovnih funkcionalnosti agentskog sustava za ugovaranje sastanaka. U aplikaciji nisu implementirane sve mogućnosti koje se mogu očekivati od takvog sustava već je naglasak dan na upotrebu agenata na malim uređajima i to korištenjem JADE/LEAP agentske platforme. Temeljni zahtjevi su bili da se sustav implementira korištenjem MIDP 2.0 profila (na malom uređaju) što je impliciralo korištenje podijeljenog kontejnera (*JADE/LEAP Split container*) na klijentskoj strani i glavnog kontejnera na strani poslužitelja (*J2SE Stand alone container*). Arhitektura sustava dana je na slici 4.1.



Slika 4.1.

Arhitektura agentskog sustava za ugovaranje sastanaka

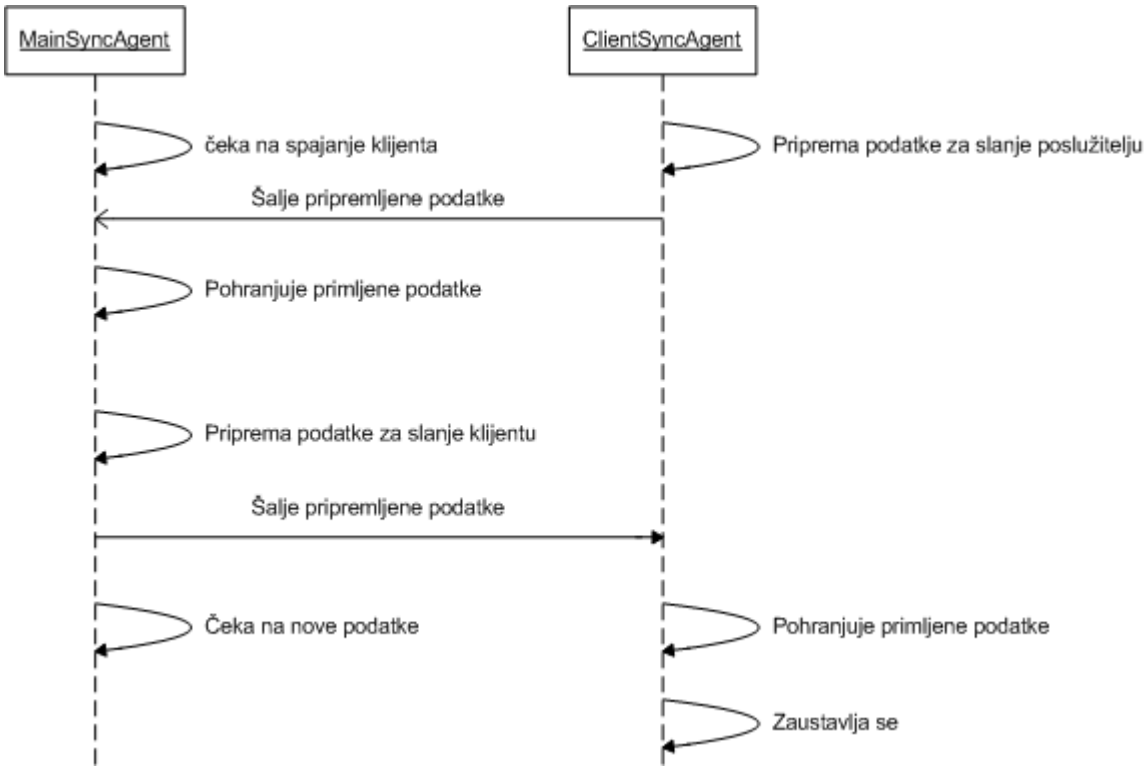
Komunikacija između agenata se odvija ACL porukama kojima se i prenose podaci. Svaki agent, prilikom svoje inicijalizacije, registrira svoje usluge u žutim stranicama agentske platforme tako da ga svi ostali agenti, zainteresirani za njegove usluge, mogu pronaći. Nakon registracije usluga agenti pronalaze ostale agente s kojima trebaju komunicirati. Prikaz stvorenih kontejnera i pripadajućih agenata prikazan je na slici 4.2.



Slika 4.2.

Prikaz kreiranih kontejnera i pripadajućih agenata

Po završetku inicijalizacijskih procedura MainSyncAgent (agent u glavnom kontejneru) prelazi u stanje čekanja do primitka poruke s podacima za sinkronizaciju. Nakon primitka svih podataka, kao odgovor, šalje ClientSyncAgentu, koji se nalazi na prednjem kraju (*Front End*) podijeljenog kontejnera, sve spremljene podatke. ClientSyncAgent sprema primljene podatke koji se u svakom trenutku mogu pregledati, modificirati ili brisati putem priložene aplikacije. Dijagram komunikacije prikazan je na slici 4.3.



Slika 4.3.

Tijek komunikacije prilikom sinkronizacije podataka

Pri izradi cjelokupne aplikacije korišteno je više tehnologija kao što su:

- J2ME
  - pri izradi klijentskog dijela aplikacije koji se izvršava na malom uređaju (*FrontEnd*). Korišten je MIDP 2.0 profil (CLDC 1.1.) proširen s PIM API-jem (JSR-75)
- J2SE
  - pri izradi klijentskog dijela aplikacije koji se izvršava na osobnom računalu koje je stalno spojeno na mrežu i povezano s poslužiteljem (glavni kontejner).
  - pri izradi cjelokupnog poslužiteljskog dijela.
- ANT build
  - prilikom prevođenja izvornog kôda same platforme (Jade proširena s LEAP dodatkom) i to za J2SE i MIDP profil
- JADE/LEAP agentska platforma verzije 3.4.
  - korištena je kao temelj za implementaciju agentskog sustava

Cjelokupna aplikacija je izrađena koristeći posljednje dostupne verzije alata i tehnologija što uključuje programski editor Eclipse SDK verzije 3.1.2. (Build id: M20060118-1600) koji je, zbog izrade MIDlet-a, proširen EclipseME (verzija 1.5.0.) dodatkom. Također su korišteni *Java Wireless Toolkit* verzije 2.3. Beta i JDK verzije 1.4.2_05[7].

Prije početka izrade same aplikacije potrebno je prevesti Jade izvorni kôd s LEAP dodatkom. Za to je korišten, priloženi, ANT kod te je uz određene modifikacije uspješno kreirano tri *JadeLeap* biblioteke (`JadeLeap.jar`) i to po jedna za upotrebu sa J2SE, MIDP i pJava. JadeLeap za MIDP profil je specifična jer prilikom pokretanja platforme kreira podijeljeni kontejner s prednjim dijelom koji se izvršava na malom uređaju (J2ME) i stražnjem dijelu koji se pokreće na kontejneru „smještenom“ na PC računalu (J2SE). Korištenje podijeljenog kontejnera je nužno iz razloga što nam omogućuje privremeno odspajanje prednjeg dijela što u klasičnoj izvedbi nije moguće.

[7] Postoji novija verzija (JDK 1.5.), ali je zbog preporuke razvijatelja Jade/Leap agentske platforme (nije kompatibilna s JDK 1.5.) korištena posljednja verzija JDK 1.4.2_05

4.1. Klijentska aplikacija

Klijentski dio aplikacije se sastoji od MIDlet-a koji se pokreće na malom uređaju i aplikacije na osobnom računalu. Oni zajedno kreiraju Client kontejner u podijeljenom modu rada koji komunicira s glavnim kontejnerom smještenim na udaljenom poslužitelju.

Da bi se podijeljeni kontejner, na malom uređaju, mogao uspješno pokrenuti nužno je osnovnu klasu proširiti s `MicroBoot` klasom sadržanom u *JadeLeap.jar* biblioteci te eksplicitno pozvati super konstruktor.

```
import jade.MicroBoot;
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class PimStart extends MicroBoot {
          .    .    .
}
public void startApp() throws MIDletStateChangeException {
super.startApp();
           ...
}
```

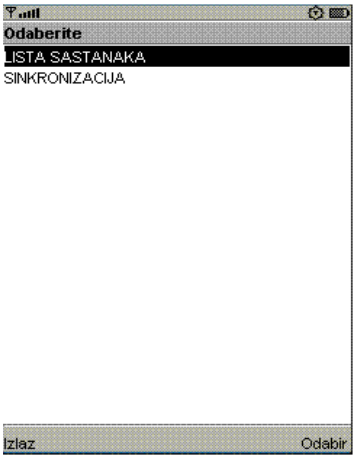
MIDlet omogućuje funkcionalnosti PIM-a kao što su vođenje liste kontakata, događaja i praćenje obaveza. Ovaj dio aplikacije je razvijen u sklopu seminara kao uvod u izradu diplomskog rada te je u svrhu aplikacije ograničen isključivo na praćenje obaveza (sastanaka) iako je ostavljena mogućnost „uključivanja“ ostalih opcija jednostavnim uklanjanjem komentara u izvornom kôdu na jednom mjestu.

```
private static final String EVENT_TYPE = "LISTA SASTANAKA";
//private static final String CONTACT_TYPE = "KONTAKTI";
private static final String SYNC_TYPE = "SINKRONIZACIJA";
//private static final String TODO_TYPE = "TO-DO";

public ListSelectionMain(PimStart midlet) {
      super("Odaberite", List.IMPLICIT);
      //append(CONTACT_TYPE, null);
      append(EVENT_TYPE, null);
      //append(TODO_TYPE, null);
      append(SYNC_TYPE, null);
```

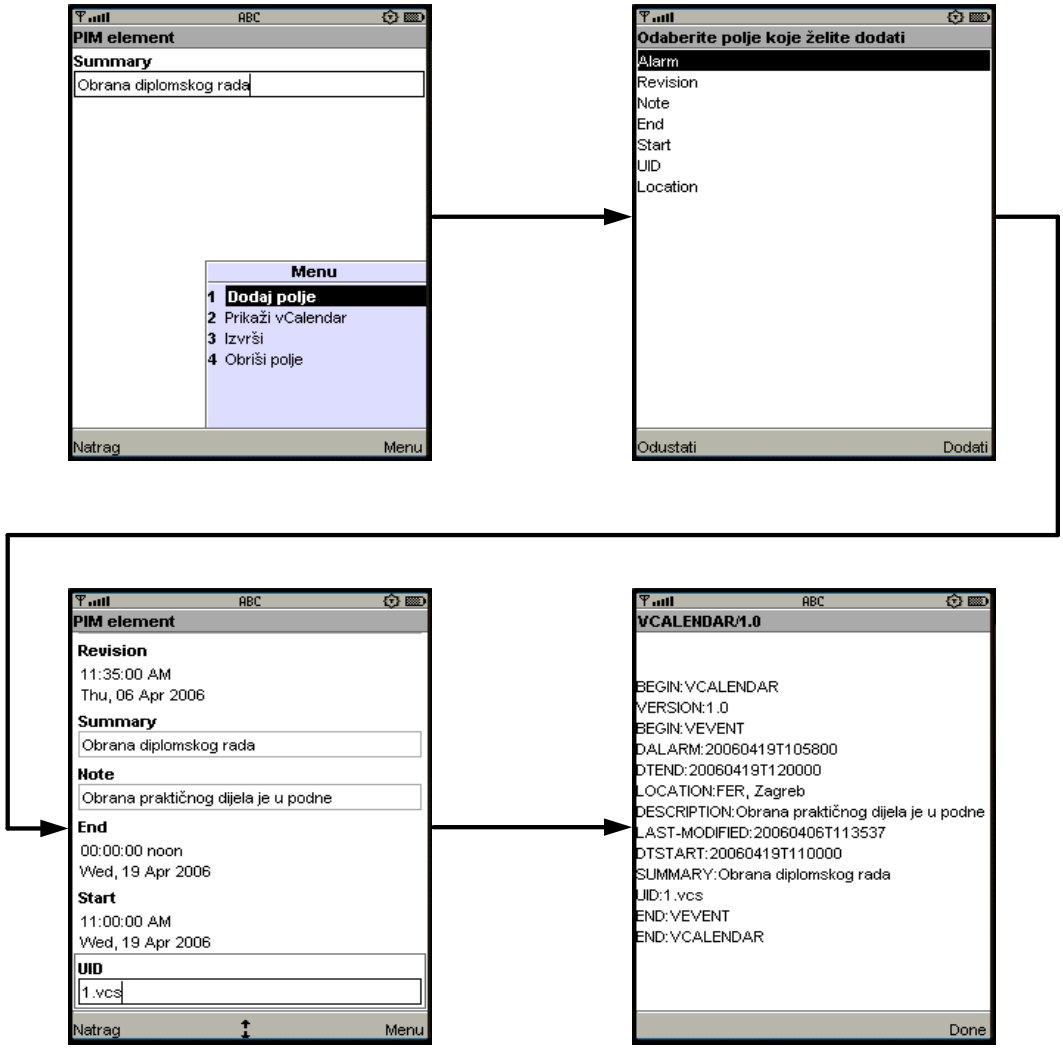
Prilikom pokretanja MIDlet-a, nakon pozdravnog ekrana, korisniku se prikazuje izbornik u kojemu je moguće odabrati rad sa sastancima (kreiranje novih, modifikacija i brisanje postojećih sastanaka) ili se može odabrati pokretanje sinkronizacije (slika 4.1.1.).

Odabirom prve opcije (LISTA SASTANAKA) pojavljuje se novi ekran u kojemu se prikazuju svi, dosada, definirani sastanci (slika 4.1.2.). Odabirom izbornika otvara se mogućnost odabira već definiranog sastanka, koji se može pregledati ili izmijeniti. Također je moguće kreirati novi ili obrisati postojeći sastanak (slika 4.1.2.). Prikaz cjelokupne procedure kreiranja novog sastanka prikazan je na slici 4.1.3.



Slika 4.1.1.
Prikaz glavnog izbornika

Slika 4.1.2.
Prikaz liste sastanaka

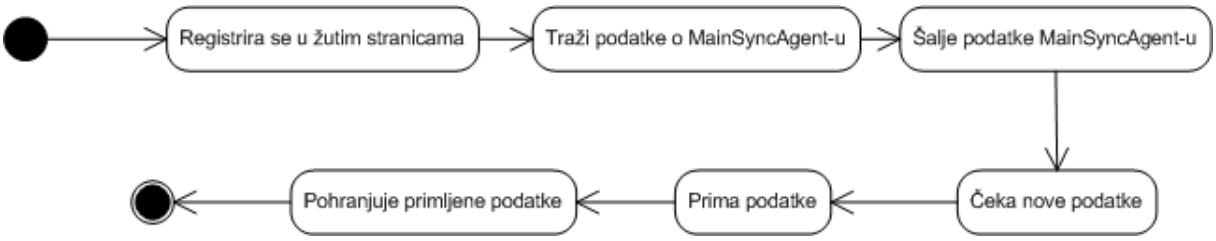


Slika 4.1.3.

Prikaz procesa dodavanja novog sastanka

Uz odabir liste sastanaka, u glavnom izborniku, je moguće odabrati i pokretanje sinkronizacije (SINKRONIZACIJA). Prilikom pokretanja sinkronizacije, kreira se posebni agent (ClientSyncAgent) na prednjem kraju podijeljenog kontejnera, a da bi, uopće, mogli kreirati podijeljeni kontejner, prethodno moramo kreirati, uz glavni kontejner i još jedan dodatni kontejner iz razloga što se glavni kontejner ne može dijeliti.

ClientSyncAgent u žutim stranicama (DF – *directory facility*) registrira svoje usluge i u istima traži MainSyncAgent agenta s kojim će razmjenjivati podatke, što je prikazano u kodu koji slijedi. Nakon završene razmjene podataka agent se zaustavlja da bi čim manje opterećivao mali uređaj (slika 4.1.4.).



Slika 4.1.4.

Dijagram stanja ClientSyncAgent-a

```
//registracija usluge u zutim stranicama

DFAgentDescription dfd = new DFAgentDescription();
dfd.setName(this.getAID());
System.out.println("DFD: "+dfd.getName().toString());

ServiceDescription sd = new ServiceDescription();
sd.setType("FEAgent");
sd.setName(getLocalName());
dfd.addServices(sd);

try {
     DFService.register(this, dfd);
} catch (FIPAException e) {
     System.out.println("\nNe mogu registrirati agenta \n");
     e.printStackTrace();
}
```

```
//Trazenje agenata u zutim stranicama

DFAgentDescription template = new DFAgentDescription();
ServiceDescription sdTemp = new ServiceDescription();
sdTemp.setType("ServerAgent");
template.addServices(sdTemp);

ACLMessage msgSend = new ACLMessage(ACLMessage.INFORM);
DFAgentDescription[] result;
try {
      result = DFService.search(myAgent, template);
      receiverAID = new AID[result.length];
      System.out.println("Br. primatelja " + result.length);
      for (int i = 0; i < result.length; i++) {
            receiverAID[i] = result[i].getName();
            System.out.println("\nPrimatelji "
            +receiverAID[i].getLocalName());
            msgSend.addReceiver(new
            AID(receiverAID[i].getLocalName(),
            AID.ISLOCALNAME));
      }
} catch (FIPAException e) {
      e.printStackTrace();
}
```

Na samom početku ClientSyncAgent prikuplja sve kreirane sastanke (koji se nalaze na malom uređaju) te ih priprema i šalje, putem ACL poruka, MainSyncAgent-u. Također ClientSyncAgent od MainSyncAgent-a prima sve podatke o sastancima koji su kreirani na poslužitelju. Podaci o sastancima su zapisani u standardiziranom formatu (vCalendar) sljedećeg oblika.

```
BEGIN:VCALENDAR
VERSION:1.0
BEGIN:VEVENT
DTSTART:20060419T120000Z
DTEND: 20060419T150000Z
LOCATION:Fakultet elektrotehnike i računarstva
CATEGORIES:Privatno
DESCRIPTION;ENCODING=QUOTED-PRINTABLE:Poruka uz podjetnik=0D=0A
SUMMARY:Obrana diplomskog rada
PRIORITY:1
UID:1.vcs
END:VEVENT
END:VCALENDAR
```

Ti su podaci kodirani UTF-8 standardom tako da se mogu čitati na svim platformama u pravilnom obliku. Bitno je primijetiti da je format datuma zapisan u US formatu (YYYYMMDD), a da je vrijeme zapisano u formatu HHMMSS. Cjelokupni prikaz aplikacijskog sustava koji se izvršava na malom uređaju prikazan je u obliku dijagrama klasa u dodatku A.

4.2. Poslužiteljska aplikacija

Poslužiteljski dio aplikacije sastoji se od samo jednog agenta (MainSyncAgent) čija je glavna zadaća komunikacija s ClientSyncAgent-om smještenom u Client kontejneru. Još jedna, bitna, zadaća ovog agenta je i pohranjivanje svih podataka o sastancima koje su kreirali klijenti, ali i svi ostali, zainteresirani, za obavješćivanje klijenata o sastancima. Npr. ako bi ovakav sustav postavili u nekoj kompaniji, zainteresirane strane bi bile: uprava kompanije, koja bi putem definiranog sučelja (*web services, stand-alone aplikacija, ...*) mogli kreirati novi sastanak ili napraviti izmjenu postojećeg te bi agent prilikom sinkronizacija te podatke prenio svim klijentima (zaposlenici kompanije koji koriste klijentsku aplikaciju na svojim mobilnim telefonima i putem nje kreiraju sastanke).

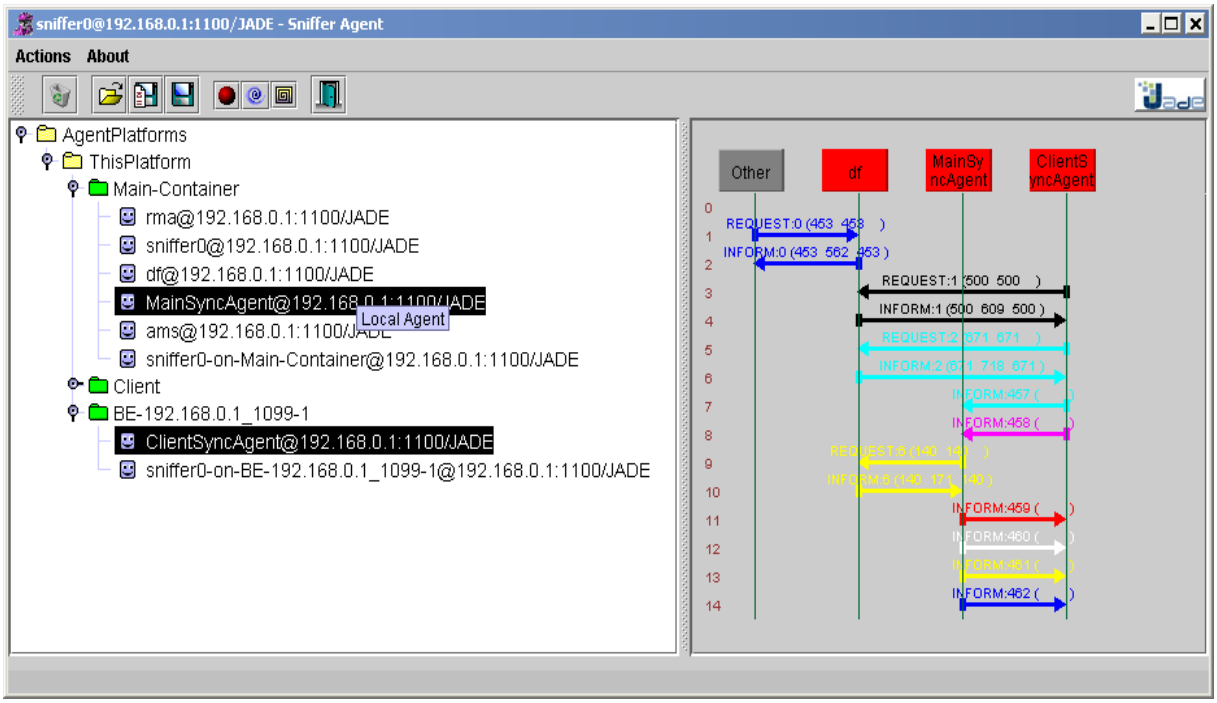
U ovom dijelu aplikacije mogu se implementirati mnogobrojna proširenja u obliku različitih algoritama po kojima se izvršava sinkronizacija, moguće je preoblikovati sustav tako da on koristi tehnologiju web usluga te se može integrirati s CRM (Customer Relationship Management) aplikacijama integrirajući se na taj način u veoma kompleksne sustave (pozivni i kontaktni centri, ...).

Prilikom pokretanja poslužiteljske aplikacije pokreće se agentska platforma i njezin glavni kontejner (*Main container*) na kojemu se kreira MainSyncAgent. Nakon tih, inicijalnih, procedura, MainSyncAgent registrira svoje usluge u žutim stranicama te čeka na upit o sinkronizaciji. Po primljenom zahtjevu, pošiljatelju šalje sve kreirane sastanke koji se nalaze u datotečnom sustavu (*filesystem*) poslužitelja i prima sve novonastale sastanke koje je klijent kreirao te ih pohranjuje na datotečni sustav i vraća se u stanje čekanja novih zahtjeva za sinkronizacijom.

Ovaj agent je proširen i metodama koje omogućuju pristupanje svim, pohranjenim, sastancima, kako bi se omogućio njihov pregled, modifikacija ili brisanje. Ovaj dio aplikacije razvijen je prvenstveno kako bi se olakšala daljnja proširenja i integracije s drugim sustavima.

Trenutno nije implementiran, nikakav poseban, algoritam za efikasnije izvršavanje sinkronizacije, što je ostavljeno za budući razvoj (primjer jednog takvog algoritma je dan u poglavlju 3.1.).

Prikaz komunikacije između MainSyncAgent-a i ClientSyncAgent-a prikazan je na slici 4.2.1. Na slici se mogu vidjeti upiti i registracija oba agenta u žutim i bijelim stranicama. Također je prikazan slijed slanja i primanja poruka. ClientSyncAgent šalje dvije poruke glavnom agentu, a ovaj mu vraća četiri nove poruke koje su prethodno bile smještene u njegovom datotečnom sustavu.



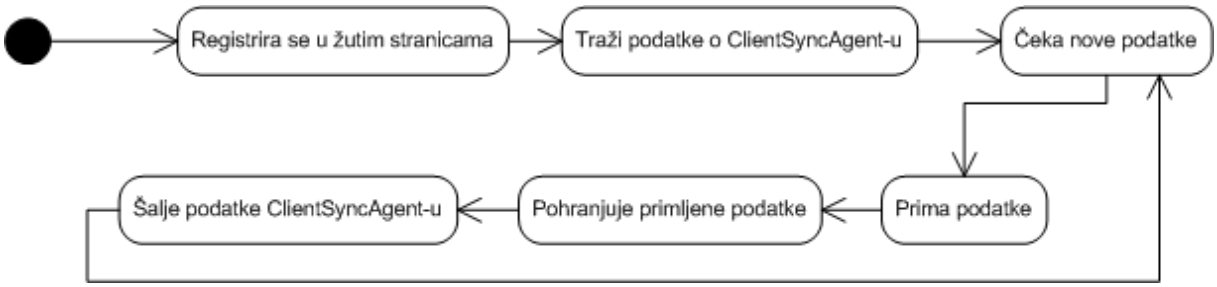
Slika 4.2.1.

Tijek komunikacije i prijenos poruka između MainSyncAgent-a i ClientSyncAgent-a

ClientSyncAgent, pohranjuje primljene podatke, koji se mogu u svakom trenutku, pregledati, izmijeniti ili obrisati.

Naime, jedna od karakteristika malih uređaja je i ta da ne moraju biti stalno vidljivi u mreži (prekid veze, različite smetnje, itd.) što može uzrokovati probleme s agentskom platformom pa je u tu svrhu razvijena arhitektura podijeljenog kontejnera koja je imuna na takve ispade sustava.

Na slici 4.2.2. dan je dijagram stanja MainSyncAgent-a iz kojeg je vidljivo da se taj agent nikada ne zaustavlja već je ili u stanju čekanja na nove podatke ili u jednom od stanja primanja, odnosno, slanja podataka nakon čega se vraća u stanje čekanja.



Slika 4.2.2.
Dijagram stanja MainSyncAgent-a

4.3. Primjena

Ovakva aplikacija ima široku primjenu u većim okruženjima koja imaju potrebu za globalnim obavješćivanjem ili vođenjem, određenog oblika, kolaboracijske okoline kao što su veće kompanije koje se bave visoko organiziranim poslovima pa im je veoma bitno imati usuglašen sustav obavješćivanja o zauzetosti pojedinih zaposlenika. Sustav se, također, može iskoristiti za obavješćivanje o zauzetosti nepokretnih resursa u kompaniji (sobe za sastanke, konferencijske sale, i sl.).

Arhitekturu ovog sustava možemo iskoristiti i za nadogradnje u obliku CRM-a što bi kao posljedicu imalo sustav za kolaboraciju vidljiv na svim resursima u mreži (posebice ako bi koristili tehnologiju web usluga za modifikaciju i kreiranje novih sastanaka).

Još jedan oblik primjene ovakve aplikacije vidim i u novoj usluzi pružatelja usluga mobilne komunikacije koja bi takvu uslugu mogla ponuditi svojim poslovnim korisnicima. Telekom bi, u ovom slučaju, kod sebe pohranjivao sve podatke koje su zainteresirane strane kreirale, administrirao sustav te primljene podatke prosljeđivao svim korisnicima kojima su oni namijenjeni. Ovakav pristup bi, vjerujem, bio prilično interesantan poslovnim korisnicima jer bi dobili globalan i robustan sustav za ugovaranje sastanaka, a telekom bi povećao mjeru privrženosti svojih poslovnih korisnika i povećao profit.

ZAKLJUČAK

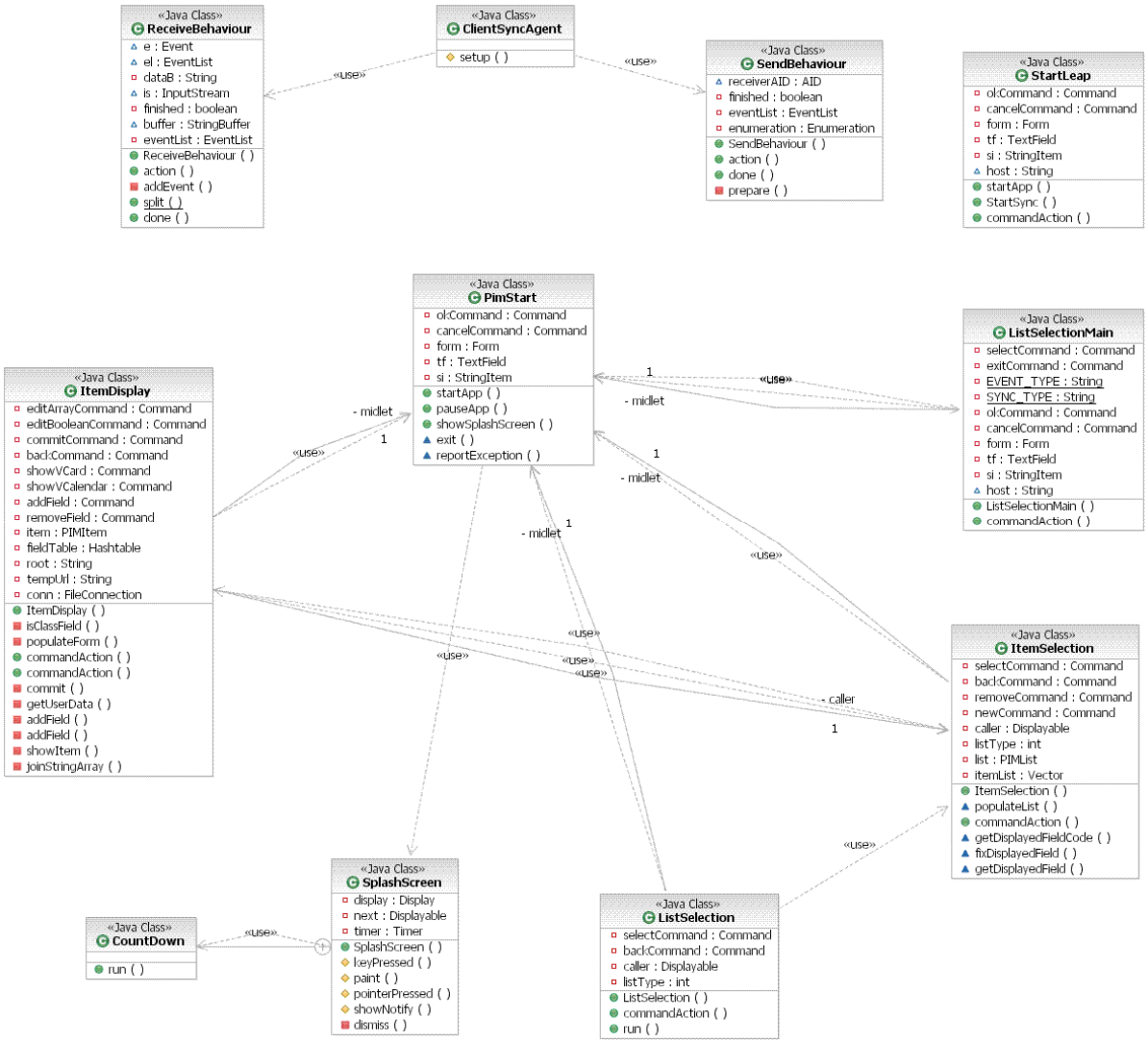
Rast broja korisnika malih uređaja uvjetuje i porast usluga koje se na tim uređajima mogu pružati. Posebno je interesantna usluga upravljanja osobnim podacima kao što su kalendar, adresar, planer i sl. Činjenica da svaki korisnik, neprestano, ima mali uređaj u neposrednoj blizini te na njega u svakom trenutku može pohraniti nove podatke ukazuje na potrebu za usklađivanjem takvih podataka s ostalim sustavima. Proces usklađivanja, zbog rasterećenja korisnika, možemo prepustiti programskim agentima.

U ovom diplomskom radu razrađena je ideja primjene agentskog sustava na malim uređajima u primjeru ugovaranja sastanaka. Posebna pažnja dana je na implementaciju agenata na malim uređajima pri čemu se moralo paziti na sva ograničenja koja su, na njima, prisutna. Zbog tih ograničenja korištena je JADE/LEAP agentska platforma koja omogućuje kreiranje podijeljenog kontejnera što osigurava optimalan rad takvih uređaja. Ostvarena je, dvosmjerna, komunikacija malog uređaja i poslužitelja u obliku ACL poruka kojima se prenose podaci vezani za svaki, pojedini, sastanak.

Implementacija predloženih algoritama kao i proširenja u smislu sinkronizacije ostalih PIM elemenata prepuštena je budućim razradama za koje je napravljena priprema u obliku stvaranja novih PIM elemenata na klijentskoj strani. Daljnja razrada trebala bi ići u smjeru proširenja poslužiteljskog dijela i to integriranjem usluge u CRM sustave korištenjem web usluga, što je od posljednje inačice JADE/LEAP (inačica 3.4.) platforme omogućeno, te u proširenju klijentskog dijela implementiranjem sigurnosnih mehanizama i modifikacijom sinkronizacijskih algoritama.

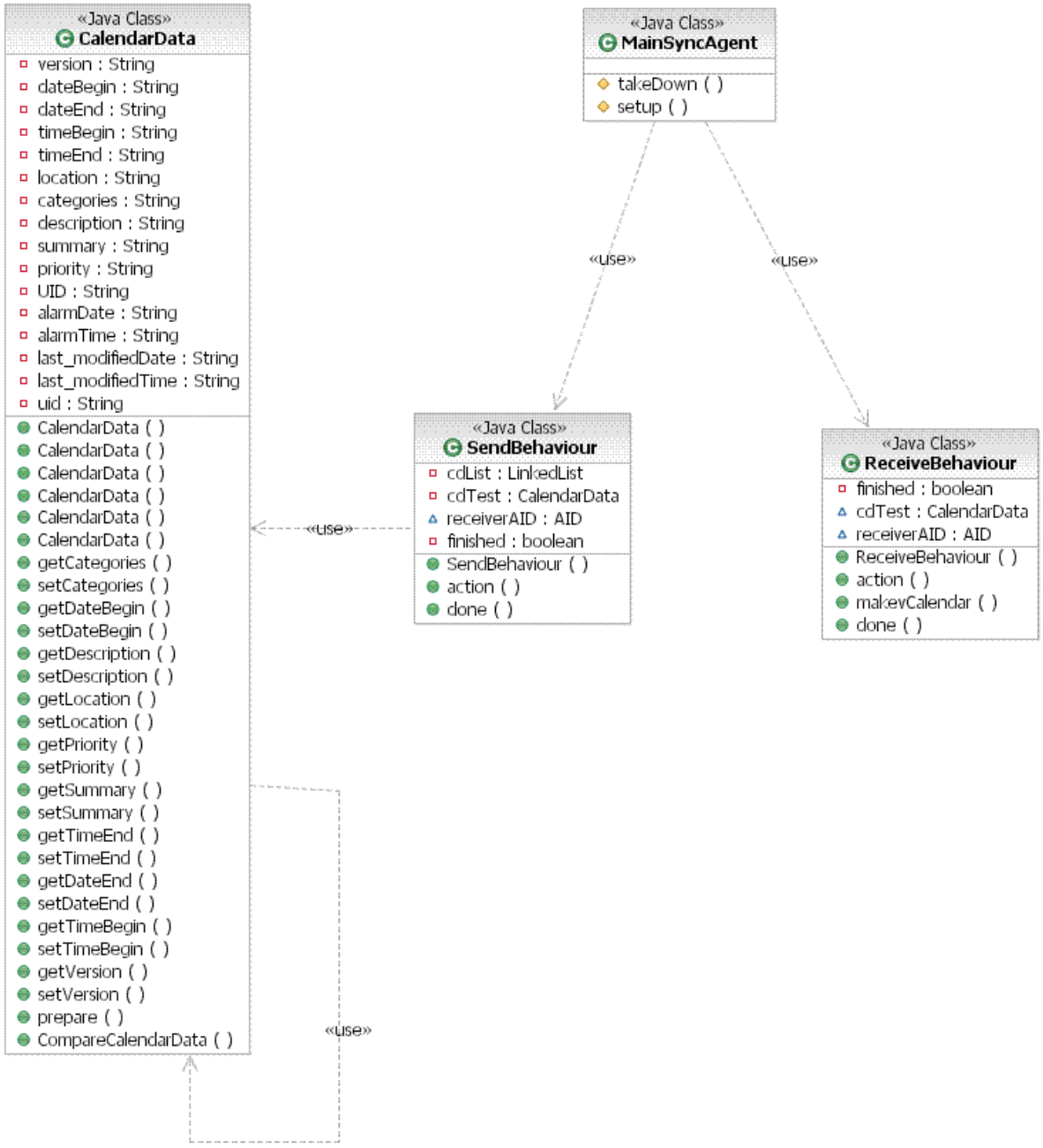
DODATAK A

Dijagram klasa aplikacijskog sustava koji se izvršava na malom uređaju (ClientSyncAgent).



DODATAK B

Dijagram klasa MainSyncAgent-a



LITERATURA

[1] Giovanni Caire, *JADE tutorial – JADE programming for beginners*, 2003.

[2] John Muchow, *Core J2ME Technology and MIDP* , Prentice Hall, 2002.

[3] C. Enrique Ortiz, Članak „*A Survey of J2ME Today*“, studeni 2002. http://wireless.java.sun.com/getstart/articles/survey/

[4] *Wireless J2ME Platform Programming*. ožujak 2002, Prentice Hall

[5] James White, David Hemphill, *Java in Small Things*, Manning Publications Co. 2002.

[6] Jonathan Knudsen, *Sun Technical Articles, What's New in MIDP 2.0*, studeni 2002.

[7] *MIDP Style Guide for the Java™ 2 Platform, Micro Edition*, Addison Wesley

[8] *Introduction To The PIM API, Version 1.0*; 25. studeni 2004, Forum NOKIA, http://www.forum.nokia.com/documents

[9] JSR-*75: PDA Optional Packages for the J2ME™ Platform , Java Community Process*, 2004, http://jcp.org/aboutJava/communityprocess/final/jsr075/index.html

[10] *Mobile Information Device Profile 2.0, Java Community Process*, 2002. http://jcp.org/aboutJava/communityprocess/final/jsr118/index.html

[11] Alf Inge Wang, Carl-Fredrik Sørensen, Eva Indal, *A Mobile Agent Architecture for Heterogeneous Devices*

[12] Niranjan Suri, Marco Carvalho, Robert Bradshaw, and Jeffrey M. Bradshaw, *Small Mobile Agent Platforms*

[13] Thomas Strang, Melanie Meyer, *Agent-Environment for Small Mobile Devices*

[14] Joseph Kinry, Daniel Zimmerman, *A hands-on look at JAVA mobile agents*

[15] M. Berger, S. Rusitschka, M. Schlichte, D. Toropov, M. Watzke, *Porting Agents to Small Mobile Devices – The Development of the Lightweight Extensible Agent Platform*

[16] Giovanni Caire, Federico Pieri, *LEAP USER GUIDE*, TILAB ex CSELT, 2006.

[17] Pedro Cuesta, Alma Gómez, Juan C. González, Francisco J. Rodríguez, *Developing a Multiagent System for Mobile Devices*

[18] Cosmin Carabelea, Olivier Boissier , *Multi-Agent Platforms on Smart Devices : Dream or Reality?*

[19] Michael Berger, Bernhard Bauer, Michael Watzke, *A scalable agent infrastructure*

[20] A. Moreno, A. Valls, A. Viejo, *Using JADE-LEAP to implement agents in mobile devices.*

[21] Jamie Lawrence, *LEAP for Ad-hoc Networks*, Media Lab Europe, 2002.

[22] Miko Laukkanen, *Agents on mobile devices*, Sonera Corp., 2003.

[23] Gordan Ježić, Mario Kušek, Krunoslav Tržec, Saša Dešić, *Daljinsko upravljanje programskom podrškom pomoću agenata,* Ericsson Nikola Tesla REVIJA 18, 2005.

[24] Fabio Bellifemine, Giovanni Claire, Tizziana Trucco, Giovanni Rimassa, JADE PROGRAMMER'S GUIDE, 2005.