UNIVERSITATEA POLITEHNICA BUCUREŞTI
FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

PROIECT DE DIPLOMĂ

Platforma Android pentru discuţii pro şi contra.
Implementarea scenariului şi a interfeţei.

Coordonatori ştiinţifici:
Prof. Dr. Ing. Adina Magda Florea
As. Dr. Ing. Andrei Olaru

Absolvent:
Miruna Popescu

BUCUREŞTI
2012

Abstract

Pentru a evidenţia câteva aspecte fundamentale ale Inteligenţei Ambientale, cum ar fi acelea de flexibilitate şi omniprezenţă, am dorit extinderea platformei tATAmI, pentru dezvoltarea de sisteme multi-agent pe dispozitive mobile. Cu scopul de a permite pe viitor crearea de scenarii în care agenţii software să fie găzduiţi pe dispozitive mobile, am ales integrarea în platformă a tehnologiei Android. Am încercat ca această integrare să fie una transparentă prin crearea unor interfeţe comune ce separă dezvoltarea aplicaţiilor multi-agent de constrângerile impuse de mediul de execuție. În această lucrare este descrisă o aplicaţie al cărei rol este de a exemplifica şi demonstra contribuţia adusă în dezvoltarea platformei tATAmI. Am implementat un scenariu care foloseşte trei tipuri de agenţi, ce sunt găzduiţi atât pe PC cât şi pe dispozitive mobile şi care comunică între ei şi migrează de la un dispozitiv la altul. Aplicaţia este destinată grupurilor de persoane care doresc să dezbată un subiect de discuţie comun şi să gestioneze cu uşurinţă opiniile fiecărui participant. În dezvoltarea sa am îmbinat tehnologia Android, frameworkul JADE pentru dezvoltarea de sisteme multi-agent si limbajul S-CLAIM de proiectare de agenţi inteligenţi.

Cuprins

Capitolul 1. Introducere

1.1 Contextul proiectului

Inteligența ambientală sau AmI (Ambient Intelligence) reprezintă o viziune a interacțiunii dintre om și tehnologie în viitor, care presupune că omul își va desfășura activitățile zilnice utilizând în mod natural informația și inteligența unor dispozitive ascunse, conectate printr-o rețea și integrate în obiectele din jur.[1] Conceptul de AmI a fost introdus la sfârşitul anilor 1990. El se referă la un mediu electronic omniprezent, non-intruziv si pro-activ, care ajută oamenii, într-o manieră personalizată şi dependenta de context, în sarcinile ce fac parte din rutina zilnică. El este integrat atât de mult în viaţa de zi cu zi, încât devine "invizibil".[2]

Cele mai multe implementări de sisteme de inteligență ambientală realizate în ultimul deceniu s-au concentrat pe realizarea de sisteme complexe care servesc unor scopuri specifice. Acestea încearcă să atingă toate nivelurile implicate într-un mediu inteligent: hardware, interconectivitate, aplicație, interfată.[2] Proiectul tATAmI (towards Agent-based Technology for Ambient Intelligence) se concentrează doar asupra nivelului aplicație al unui sistem AmI, păstrându-și astfel genericitatea.[2] Proiectul a fost dezvoltat în urma unei colaborari dintre Laboratorul AI-MAS (Artificial Intelligence and Multi-Agent Systems) din cadrul Universității Politehnica București si Laboratorul de Informatică al Paris 6 (LIP6) din cadrul Universității Pierre et Marie Curie.

Platforma tATAmI dezvoltă un sistem multi-agent cu rol de middleware pentru viitoare aplicații din domeniul Inteligenței Ambientale. Scopul său este de a implementa un model de sistem care să dispună de auto-organizare, dependență de context și anticipare. Platforma se bazează pe utilizarea de agenți software cognitivi care gestionează și schimbă în mod natural informații de context. Folosește frameworkul JADE pentru dezvoltarea de sisteme multi-agent şi limbajul S-CLAIM de proiectare de agenţi inteligenţi.

Pentru a evidenţia aspecte fundamentale ale Inteligenţei Ambientale (cum ar fi acelea de flexibilitate şi omniprezenţă), am ales extinderea platformei tATAmI pe dispozitive mobile. Acestea ocupă astăzi un rol esenţial în vieţile noastre. Conform estimărilor Cisco, anul acesta, numărul de dispozitive mobile cu acces la Internet (telefoane, laptopuri şi tablete în principiu) va depăşi ca număr populaţia la nivel global.[3] Am ales adaptarea sistemului la tehnologia Android, platformă proiectată special pentru utilizarea pe dispozitive mobile. Aceasta este cea mai răspândită tehnologie ce se regăseşte la momentul actual pe telefoane mobile inteligente şi tablete. Totodată, pune la dispozitia dezvoltatorilor o serie de unelte si framework-uri pentru crearea rapidă de aplicaţii în limbajul Java.

Android a avut întâietate în faţa celorlalte tehnologii de pe piaţă, precum Windows Phone de la Microsoft, sau iOS de la Apple, acestea din urmă fiind construite pe sisteme de operare proprietare, care restricţionează comunicarea între aplicaţiile software şi sistemul nativ al telefonului, ingreunând dezvoltarea de către terţi a aplicaţiilor ce ţintesc platformele lor. Android este o platformă open-source, ce oferă dezvoltatorilor acces la intreg codul sursă al proiectului şi posibilitatea de a aduce modificări în funcție de propriile necesități.

1.2 Ideea și scopul proiectului

Acest proiect cuprinde două aspecte: portarea platformei tATAmI pe tehnologia Android și crearea unei aplicații orientate agent care să valideze integrarea cu succes a tehnologiilor folosite și totodata a principiilor de AmI pe care platforma le implementează.

Platforma tATAmi foloseste JADE (Java Agent Development Framework) pentru dezvoltarea de sisteme multi-agent. JADE oferă cadrul în care agenţii software pot exista, opera şi comunica. Structura JADE conţine un mediu special numit Container ce trebuie activat pentru a putea lansa agenţii în execuţie. Un container poate conține unul, mai mulți sau niciun agent. Un agent nu poate fi creat în exteriorul unui Container. Toate containerele active formează o Platformă.[4] Această arhitectură este detaliată în Capitolul 3.

Se doreste integrarea în proiectul tATAmI a posibilitații de conectare la platforma JADE a unui dispozitiv mobil ce folosește Android și de creare de containere și agenți software, care să ruleze pe acesta. Pentru implementarea acestei funcționalități a fost folosit un add-on, JADE-LEAP, special conceput pentru lucrul cu dispozitive mobile. S-a încercat ca această integrare să fie una transparentă, să nu diferențieze execuția unui agent pe PC de cea pe un PDA. În acest scop au fost create interfețe si încapsulări care să separe dezvoltarea de aplicații multi-agent de constrângerile impuse de mediul de execuție.

Pentru a exemplifica şi valida functionalitatea adusă proiectului tATAmI, am implementat o aplicatie care îmbină tehnologia Android, frameworkul JADE si limbajul S-CLAIM de proiectare de agenţi inteligenţi. Rolul său este de a demonstra contribuţia adusă în dezvoltarea platformei tATAmI. Am implementat un scenariu care foloseşte trei tipuri de agenţi inteligenți, ce sunt găzduiţi atât pe PC cât şi pe dispozitive mobile şi care comunică între ei şi migrează de la un dispozitiv la altul. Aplicaţia este destinată grupurilor de persoane care doresc să dezbată un subiect comun şi trebuie să gestioneze cu uşurinţă opiniile fiecărui participant. Ea centralizează si gestionează opiniile participanţilor în cadrul unui grup de discuție. Aplicația a fost testată pe dispozitive smartphone Samsung Galaxy S Plus și pe emulatorul Android Virtual Device. Aceasta aplicație demonsrează că în viitor, platforma va permite crearea de scenarii în care agenţii software să fie găzduiţi atât pe computere personale cât și pe dispozitive mobile.

Acest proiect a fost implementat în colaborare cu Iulia Moscalenco, sub supravegherea și îndrumarea as. dr. ing. Andrei Olaru, unul dintre dezvoltatorii proiectului tATAmI.

1.3 Structura lucrării

Voi face o prezentare sumară a fiecărui capitol din această lucrare.

Capitolul 1 „Introducere" descrie în prima parte contextul în care a fost dezvoltată această lucrare, apoi prezintă o descriere generală a proiectului şi a aplicaţiei implementate, evidenţiind scopul în care a fost concepută lucrarea şi rolul său în cadrul dezvoltării proiectului tATAmI.

Capitolul 2 „Aspecte teoretice" prezintă contextul teoretic în care a fost concepută lucrarea, defineşte următoarele noţiuni: "Inteligenţa ambientală", "scenariu", "agent software", „sistem multi-agent” şi prezintă limbajul S-CLAIM pentru dezvoltarea de agenți, care a fost folosit în cadrul proiectului, limbaj mai puțin cunoscut publicului larg. Prima parte trasează direcţiile în care Inteligenţa Ambientală doreşte să evolueze, descrie pricipalele caracteristici ale acesteia şi le evidenţiaza printr-un exemplu de utilizare, prezentat sub formă de scenariu. Totodată sunt prezentate

şi o parte din tehnologiile ce trebuie îmbinate pentru a crea cu succes un mediu inteligent şi e descris rolul pe care îl are platforma tATAmI în domeniul inteligenţei ambientale. Al doilea subcapitol descrie principalele caracteristici şi funcţionalitaţi ale unui agent software şi ale unui sistem multi-agent. Este descris comportamentul unui agent şi tipurile de agenţi reactivi şi cognitivi, şi este prezentată importanţa dependenţei agenţilor de contextul în care se află. În finalul capitolului este descrisă semantica şi sintaxa limbajului S-CLAIM.

In **Capitolul 3 „Tehnologii folosite”** sunt prezentate tehnologiile pe care le-am folosit în dezvoltarea acestei lucrări. Prima parte descrie platforma JADE (Java Agent Development Framework) utilizată pentru crearea agenţilor şi implementarea scenariului. Este prezentat modelul conceptual al platformei, arhitectura, funcţionalitaţile oferite şi conformitatea acesteia cu specificaţiile FIPA (Foundation for Intelligent Physical Agents). În a doua parte a acestui capitol este prezentat un scurt istoric al tehnologiei şi arhitecturii sistemului Android. Ultima parte prezintă modulul LEAP, extinderea framework-ului JADE peste tehnologia Android.

Capitolul 4 „Arhitectura sistemului” descrie arhitectura celor doua proiecte folosite: „tATAmI-Android” şi „tATAmI-PC” şi modul în care acestea sunt interconectate. Sunt prezentate modulele folosite în proiecte şi functionalitaţile lor.

Capitolul 5 „Detalii de implementare” oferă informaţii despre modul în care a fost implementată aplicaţia. În prima parte este descris scenariul de utilizare al aplicaţiei şi detaliată implementarea sa. Apoi este detaliată implementarea agenţilor în limbajul S-CLAIM şi a interfeţelor grafice ale agenţilor de pe PC şi de pe Android. Este specificat şi modul de conectare din aplicatia Android la platforma JADE şi diferenţele dintre folosirea unui emulator şi a unui dispozitiv real.

Capitolul 6 „Descrierea aplicației și instrucţiuni de utilizare” prezintă un scurt tutorial despre cum se poate folosi aplicaţia, despre modul de pornire, rulare şi utilizare, despre diferenţele rulării pe un dispozitiv real şi un emulator.

Ultimul capitol – **Capitolul 7 „Concluzii”** sintetizează contribuțiile acestei lucrări şi trasează posibilele îmbunătăţiri ce pot fi aduse pe viitor.

Capitolul 2. Aspecte teoretice

2.1 Inteligenţa ambientală (AmI)

Inteligența ambientală sau AmI (Ambient Intelligence) reprezintă o viziune a interacțiunii dintre om și tehnologie în viitor, care presupune că omul își va desfășura activitățile zilnice utilizând în mod natural informația și inteligența unor dispozitive ascunse, conectate printr-o rețea și integrate în obiectele din jur.[1] Conceptul de AmI a fost introdus la sfârşitul anilor 1990. El se referă la un mediu electronic omniprezent, non-intruziv si pro-activ, care ajută oamenii, într-o manieră personalizată şi dependentă de context, în sarcinile ce fac parte din rutina zilnică. El este integrat atât de mult în viaţa de zi cu zi, încât devine "invizibil".[2]

Viziunea AmI este caracterizată de două trăsături cheie: inteligenţă şi integrare. **Inteligenţa** se referă la faptul că mediul digital este capabil să analizeze contextul, să se adapteze oamenilor şi obiectelor, să înveţe din comportamentul acestora şi eventual să îi recunoască şi să exprime emoţii. **Integrarea** se referă la faptul că dispozitivele miniaturizate vor deveni din ce în ce mai mult parte componenta în fundalul activităţilor umane, într-un prim plan aflându-se interacţiunea socială şi funcţionalitatea . Dispozitivele vor deveni invizibile, prin integrarea lor în toate obiectele şi materialele din jur. Acest lucru va face ca totul să devină "inteligent" şi, prin comunicare să colaboreze cu noi în scopul de a oferii funcţii complexe şi rezultate utile.[1]

Este o viziune în care mediul înconjurător ne va înţelege dorinţele şi nevoile fără a fi nevoiţi să le explicităm folosind limbaje şi reguli specifice, iar utilizarea unor astfel de servicii nu va impune cunoştinţe tehnice avansate ci interacţiunea dintre utilizator şi mediu ambiental se va realiza într-o manieră naturală, intuitivă, asemeni interacţiunii dintre două persoane.

2.1.1 Conceptul de scenariu

In domeniul Inteligenţei Ambientale apare conceptul de scenariu. Acesta reprezintă descrierea unei situaţii în care oamenii utilizează şi interacţionează cu sistemul de inteligentă ambientală. Scenariile ne ajută să extragem caracteristicile şi funcţionalităţile AmI. Pentru a înţelege mai bine utilitatea scenariilor şi modul de organizare şi funcţionare al unui sistem AmI voi prezenta un exemplu de scenariu.

Elena se întoarce acasă după o zi lungă. La uşa din faţă ea este recunoscută de o cameră de supraveghere inteligentă, alarma de securitate se opreşte, iar uşa se deschide şi se deblochează. După ce intră în casă, harta casei indică faptul că soţul ei Peter este la un târg de artă din Paris şi că fiica ei Charlotte este în camera copiilor, unde se joacă cu un ecran interactiv. Serviciul de supraveghere la distanţă a copilului a notificat că Elena a ajuns acasă, şi se opreşte automat. Când intră în bucătărie, dispozitivul de inştiinţare al familiei se luminează pentru a indica faptul că există mesaje noi. Lista de cumpărături, pe care o compusese are nevoie de confirmare înainte de a fi trimisă pentru livrare la supermarket . Există şi un mesaj de atenţionare că sistemul de informaţii a găsit noi informaţii pe Internet despre case de vacanţă economice cu vedere la mare, localizate în Spania. Se conectează la camera de joacă a fiicei sale pentru a o saluta, iar pe ecranul folosit de copil apare instantaneu imaginea sa video. Se conectează apoi la Peter, la târgul de artă, iar acesta îi arată prin camera video din lentilele de contact câteva din sculpturile ce intenţionează să le cumpere. Ea îi confirmă alegerile. Între timp, selectează una din opţiunile de pregătire a mâncării ce poate fii preparată cu alimentele disponibile din frigider şi din cămară.

Observăm că obiectele folosite zilnic în locuinţa noastră ar trebui înzestrate cu noi funcţionalităţi, pentru a renunţa la utilizarea de interfeţe dedicate pentru interacţiunea cu sistemul. Astfel computerele vor deveni „invizibile” iar interacţiunea cu sistemul AmI se va face în mod natural şi intuitiv. Totodată nu trebuie neglijat faptul că în spatele interfeţei naturale pe care utilizatorul o vede şi simte, se ascunde un sistem complex ce gestionează cantitaţi mari de date.

2.1.2 Caracteristici AmI

AmI trebuie să fie **omniprezentă si pervazivă**. Arhitectura sa trebuie să suporte un număr mare de dispozitive mobile care să partajeze neîncetat cantităţi mari de informaţii, fără ştirea utilizatorului (dar nu şi împotriva preferinţelor lui). În acelaşi timp AmI ar trebui să fie **distribuită şi să lucreze la un nivel local**.[2]

AmI trebuie să fie **naturală**, prin utilizarea interfeţelor avansate, intuitive, multi-modale. AmI trebuie să se adapteze tuturor cerinţelor şi să necesite numai atenţia pe care utilizatorul este dispus să o investească.[2]

AmI trebuie să fie **previzibilă si transparentă**, astfel încât utilizatorul să înţeleagă informaţiile şi serviciile şi modul în care funcţionează sistemul, cel puţin în principiu. Principiile de bază după care lucrează AmI ar trebui să fie simple şi uşor de înţeles de către oricine, aceasta făcând ca AmI să fie **generică şi adaptivă**.[2]

AmI trebuie să fie, de asemenea,**pro-activă şi inteligentă**. Ea trebuie să adopte măsuri adecvate, fără să deranjeze. Acţiunea trebuie să fie luată doar în cazul în care utilizatorul înţelege cauzalitatea, aprobă acţiunea. Apare **dependenţa de context** care are un rol principal. În acest caz vorbim de **trasabilitatea** AmI. Trasabilitatea îmbunătăţeşte semnificativ acceptarea de către utilizator, utilizatorii fiind mult mai toleranţi faţă de acţiunile incorecte făcute de aplicaţiile dependente de context, atunci când înţeleg că au o bază raţională. AmI trebuie să furnizeze un flux de informaţii previzibil, natural.[2]

Acelaşi sistem de AmI va trebui să suporte mai mult de un utilizator la un moment dat (spre deosebire de cum vedem de obicei în scenarii). În spaţiile publice există un număr mare de utilizatori pe care AmI trebuie să fie capabilă să-i notifice şi asiste, fără a pierde confidenţialitatea (sau informaţii importante), faţa de alţi utilizatori.

Aplicațiile AmI sunt diverse și se regăsesc în multiple domenii ale activității umane (monitorizarea sănătății, transport, activități casnice, divertisment, etc.). Câteva direcţii de dezvoltare care sunt legate de AmI: percepţie, raţionament, decizie, interacţiunea om-calculator, şi viaţa privată şi securitate.

2.1.3 Principii AmI

Inteligenţa ambientală se bazează pe trei tehnologii cheie:

- calculul omniprezent (ubiquitous computing) ce realizează integrarea microprocesoarelor în obiecte de zi cu zi cum ar fi mobilă, îmbrăcăminte, jucării, electrocasnice, clădiri etc.;
- comunicarea omniprezentă (ubiquitous communication) ce permite acestor obiecte să comunice unele cu altele şi cu utilizatorul prin intermediul reţelelor ad-hoc şi fără fir;
- interfeţele inteligente cu utilizatorul (intelligent user interfaces) ce permit utilizatorilor din

mediile inteligente să controleze şi să interacţioneze cu aceste medii într-o manieră cât mai intuitivă şi naturală (voce, gesturi) şi într-un mod personalizat (în funcţie de preferinţe sau context).[1]

AmI este construită cu ajutorul tehnologiilor avansate de rețea, ce permit asocierea ad-hoc a rețelelor de dispozitive mobile și a altor obiecte. Prin adăugarea de metode adaptive de interacțiune utilizator-sistem, bazate pe analize ce se referă la modul în care oamenii interacționează cu sistemele de calcul, mediile digitale pot fi proiectate pentru a îmbunătăți calitatea vieții oamenilor, acționând în locul acestora. Aceste sisteme care se adaptează la context combină informația omniprezentă, comunicațiile și divertismentul cu personalizarea la cel mai înalt nivel, interacțiunea naturală și inteligența.[1]

Spațiul AmI poate fi văzut ca un integrator de funcții la nivel local în diverse medii și permite un dialog natural și intuitiv al utilizatorului cu aplicațiile și serviciile aferente diverselor medii – inclusiv mediul virtual – permițând organizarea și procesarea de conținut.[1]

Pentru dezvoltarea AmI este nevoie ca mai multe tipuri de tehnologii să conlucreze:

- hardware non-invaziv – conţinând dispozitivele din mediul AmI: nanotehnologii, senzori wireless, etc.;
- infrastructură mobilă/fixă de comunicații și de calcul: rețele fixe sau wireless care interconectează dispozitivele;
- un strat de interoperabilitate care asigură protocoale uniforme SOA – Arhitecturi Orientate pe Servicii) - nivelul aplicaţie / servicii inteligente, oferind servicii dependente de semantică şi context;
- interfețe prietenoase cu utilizatorul (agenți inteligenți, interacțiune multimod, sensibilitate la context, sisteme biometrice, etc.) care permit comunicarea naturală între utilizator şi sistem.

2.1.4 Viziune platformă

AmI este un domeniu vast, de aceea, este greu pentru aceeaşi echipă să dezvolte funcţionalităţi de conectivitate, de dependenţa de context, nivele de interoperabilitate şi interfeţe noi, în acelaşi timp. Acesta este motivul pentru care majoritatea implementărilor, încercând să abordeze toate aceste probleme, au ca rezultat un sistem care nu este flexibil sau scalabil, sau care este specific numai unui domeniu de aplicaţie.[2]

Platforma tATAmI, pe care o folosim în proiectul nostru, pentru a o adapta la tehnologia Android, în scopul creării de containere şi agenţi pe dispozitive mobile, reprezintă un sistem multi-agent pentru implementarea de aplicaţii AmI.

Platforma se concentrează pe un singur nivel al unui sistem AmI: nivelul care lucrează cu informaţii la nivel semantic, şi care este esenţial pentru dependenţa de context a întregului sistem. Sistemul foloseşte paradigma orientată agent pentru modelare şi implementare, aceasta oferind caracteristici importante pentru AmI, în special autonomia şi pro-activitatea. Platforma păstrează genericitatea deoarece se doreşte a fii un nivel de mijloc (middleware) pentru o gamă mai variată de sisteme AmI viitoare.[2]

2.2 Conceptele de agent software şi sistem multi-agent

Agentul software este o unitate de procesare care operează în paralel şi /sau în coordonare cu alţi agenţi, formând un sistem complex, interactiv de procesare a informaţiei. Un agent software se află la un moment dat într-o stare, posedă o bază de cunoştinţe (expertiză), şi este capabil să iniţieze sau să răspundă la acţiuni în cadrul sistemului multi-agent. Spunem că sistemele multi-agent modelează un sistem interactiv printr-o colecţie de agenţi specializaţi care produc şi reacţionează la stimulii existenţi în cadrul sistemului. Agentul software este deci o unitate ce poate executa un set de sarcini independent de activitatea altor agenţi, ca o consecinţă firească a arhitecturii bazate pe agenţi,şi care permite activitaţi simultane de interpretare şi manipulare a informaţiei.[8]

Capacitatea unui sistem de a interpreta şi manipula informaţii variază dinamic în funcţie de variabilele considerate într-un context dat. Variabilele contextuale se comportă ca nişte filtre cognitive. Ele formează un set de parametrii interni de stare folosiţi în procesele de reprezentare pentru a controla interpretarea şi manipularea informatiei.

Un sistem multi-agent poate fi privit ca un sistem în evoluţie, în care fiecare agent desfăşoară o activitate independentă. Pentru specificarea completă a unui sistem multi-agent este necesară definirea cunoştinţelor şi comportamentului intern al agenţilor şi a modului de interacţiune cu ceilalţi agenţi cu care coexistă în cadrul sistemului multi-agent.

Un agent este format din mecanisme de emisie-recepţie a mesajelor, memorie (pentru menţinerea unor stări), şi un procesor care analizează evenimentele de tip input, actualizează starea prezentă şi execută o serie de acţiuni, interacţionând cu ceilalţi agenţi.[8]
În domeniul platforme de inteligenţa ambientală bazate pe agenţi se dezvoltă două direcţii principale: o direcţie care foloseşte agenţi cu funcţionalităţi de învăţare şi de raţionament având componente centralizate (baze de cunostinţe, ontologii, etc.), orientaţi spre asistarea utilizatorului, şi o direcţie bazată pe coordonarea de agenţi asociaţi dispozitivelor, eventual mobili, într-un context de funcţionalitate mai simplă, luând în considerare, de asemenea controlul distribuit şi toleranţa la erori.[2]

Utilizarea de agenţi complecşi în număr mai mic din prima direcţie, ce provin din domeniul asistenţilor personali inteligenţi, este mai aproape de interfeţele inteligente cu utilizatorul şi anticiparea locală a intenţiilor utilizatorului. În acest caz agenţii se folosesc de componente centralizate pentru a regăsi date externe, sunt complexi şi utilizează algoritmi de învăţare . Comunicarea între agenţi este limitată, exceptând cazul în care componentele centralizate sunt desemnate ca unul sau mai multi agenţi.[2]
Utilizarea mai multor agenţi mai simpli, din a doua direcţie, implică rezolvarea diferitelor probleme cum ar fi mobilitatea utilizatorului, controlul distribuit, auto-organizarea şi toleranţa la erori, şi are o perspectivă mai globală privind modul în care o platformă AmI ar trebui să funcţioneze.[2]

2.2.1 Dependenţa de context

O problemă centrală în domeniul inteligenţei ambientale o constituie dependenţa de context. Înţelegerea adecvată din partea sistemului de AmI a contextului utilizatorului face posibil comportamentul pro-activ dar non-intruziv . Acţiunile sistemului trebuie să fie naturale şi corect integrate în situaţia actuală.[2]

Contextul este definit în mai multe feluri. Dey a dat în 2001 o definiţie care este citată mai mult decât altele în domeniul inteligenţei ambientale şi al aplicaţiilor dependente de context:

„Contextul este orice informaţie care poate fi folosită pentru a caracteriza situaţia unei entităţi. O entitate este o persoană, loc sau obiect care este considerat relevant pentru interacţiunea dintre utilizator şi aplicaţie, inclusiv utilizatorul şi aplicaţia inşăşi" [Dey 2001].„În ceea ce priveşte dependenţa de context, este proprietatea unei aplicaţii de a se adapta automat la contextul perceput, prin schimbarea comportametului aplicaţiei în consecinţa"[Chenand Kotz 2001].
Pentru a fi in masură să decidă cu privire la acţiunea corectă, o aplicaţie dependentă de context şi o aplicaţie Ami în special trebuie să poată accesa informaţia de context şi trebuie să inţeleagă contextul, să înţeleagă relaţiile dintre diverse fapte din context .

Contextul are mai multe aspecte: spaţiul fizic, timp, activitate(curentă, trecută sau planificată) relaţii sociale şi resurse computaţionale. Există mai multe tipuri de context: computaţional, temporal, social. Exemplu: unde se află utilizatorul, ce oră este, ce temperatură este afară, ce capabilitaţi are conexiunea la reţea şi asociaţii între diverse fapte care se referă la utilizator.

Există anumite principii ale schimbului de informaţii, dependent de context, între un agent şi vecinii săi într-un sistem de agenţi, ce pot fi utile în implementarea aplicațiilor de inteligență ambientală. Un agent trebuie să facă schimb de informaţii doar cu un alt agent care consideră informaţiile ca relevante. Această situaţie e valabilă numai în cazul în care agenţii au în comun unele tipuri de context. De exemplu în cazul în care agenţii fac parte din aceeaşi activitate sau în cazul în care utilizatorii lor sunt situaţi în acelaşi spaţiu. Când doi agenţi nu împărtăşesc nici un context nu au nici o informaţie care să fie relevantă pentru ambii. Deci topologia sistemului este indusă de context: dacă doi agenţi împărtăşesc contex, aceştia trebuie să fie vecini.[2]

2.2.2 Comportamentul agenţilor software

Sistemele multi agent sunt sisteme care au un număr mare de entităţi ce interacţionează intens şi formează bucle de feedback, sunt sisteme complexe în care o implementare ideală de AmI implică un număr mare de dispozitive ce schimbă continuu informaţii şi se adaptează la mediu. Aceste sisteme sunt caracterizate prin proprietăţi şi fenomene emergente care sunt neaşteptate în raport cu proiectarea elementelor individuale şi de aceea sunt greu de controlat, fiind greu de prezis rezultatul acţiunilor.

Un aspect important în cazul unui sistem multi agent pentru AmI, îl constituie comportamentul agenţilor, mai precis, modul în care trebuie să se facă schimb de informaţii între agenţi, astfel încât informaţiile interesante să ajungă la agenţii interesaţi fără un control centralizat. Trebuie creat un mediu de inteligenţa ambientală cu un număr mare de dispozitive care servesc nevoilor utilizatorilor respectivi. Dispozitivele vor gestiona în majoritatea timpului diverse informaţii: furnizarea de informaţii relevante pentru utilizatorii interesaţi, agregarea, filtrarea şi raţionamentul despre informaţii fiind „transportoare de informaţii". Trebuie soluţionată problema ca fiind dată o anumită informaţie, cum se va livra acea informaţie către agenţii interesaţi, agenţi care folosesc această informaţie pentru a ajuta utilizatorii. Aceasta este o problemă care se adresează nivelului aplicaţie al unui mediu AmI.

Agenţi reactivi

În domeniul sistemelor multi-agent, cele mai multe exemple care demonstrează proprietăţi emergente folosesc agenţi reactivi [Serugendo et al., 2006]. În domeniul sistemelor multi-agent proprietăţile emergente sunt mai uşor de implementat şi de controlat şi sunt potrivite pentru dispozitive mici, cu putere de calcul redusă. Sistemele de agenţi reactivi sunt mai uşor de anticipat şi de proiectat astfel încât ele să manifeste proprietăti de auto-organizare .[2]

Agenţi cognitivi

Pe lângă agenţii reactivi avem agenţi cognitivi. Deşi sistemele de agenţi reactivi pot fi foarte utile, există multe avantaje pe care un agent cognitiv le are fată de un agent reactiv. Un agent cognitiv este proactiv. El poate acţiona de la sine şi lua măsuri în conformitate cu obiectivele sale, chiar dacă nu există semnale, percepţii sau stimuli din mediu. Un agent cognitiv poate folosi experienţa şi informaţiile despre mediul inconjurător, precum şi consecinţele acţiunilor trecute pentru a dezvolta un plan mai bun de fiecare dată când apare o situaţie similară. El este conştient de situaţia sa şi poate raţiona despre ea.[2]

2.2.3 Caracteristicile unui agent software

Putem enumera câteva caracteristici ale agenţilor, ce sunt esenţiale rolului lor în cadrul sistemelor AmI:

- **autonomia**, reprezintă proprietatea unui agent de a funcţiona corect în situaţiile în care componente ale sistemului din care face parte devin nefuncţionale sau inaccesibile, asigurând astfel o funcţionare neîntrerupta a sistemului;[5]
- **anticipatia** şi **pro-activiatea**,sunt proprietaţi ce permit agenţilor să poată recunoaşte anumiţi factori, din care să reconstituie situaţii, şi pe baza cărora să poată acţiona luând propriile decizii, fără intervenţia utilizatorului;[5]
- **deschiderea**, este mai mult o caracterisitică a sistemului din care agenţii fac parte, ce oferă acestora dreptul de a se înscrie, a părăsi sistemul sau a fuziona, fără a aduce atingere rezultatului final;[5]
- **comportamentul contextual**, se schimbă în funcţie de situaţia prezentă la un moment dat, nedepinzând de starea sau existenţa unui element central de control. Primul aspect esenţial este colectarea informaţiilor, celălalt fiind modelarea şi prelucrarea acestora.[5] Necesitatea de structurare a modului de lucru în diverse contexte duce la împărţirea acestora în categorii precum:
    - context computaţional: resurse hardware disponibile, calitatea conexiunii la reţea;
    - context utilizator: profil al utilizatorului, situaţie socială, prieteni prezenţi în sistem;
    - context ambiental fizic: iluminare, temperatură, condiţii de trafic;
    - context de timp: data, moment al zilei.

Una dintre metodele de utilizare a agenţilor pentru implementarea sistemelor de inteligenţă ambientală foloseşte un numar redus de agenţi. Aceştia reţin preferinţele utilizatorului, şi totodată memorează informaţii despre contextul în care acestea apar. Această abordare nu este una generică şi nici foarte uşor scalabilă.

O altă abordare este concentrarea atenţiei în special pe modul de coordonare a agenţilor şi pe metode de auto-organizarea a acestora. În aceste cazuri devine mai dificil de modelat o acumulare flexibilă de cunostinţe din partea agenţilor, cât şi de reprezentare precisă a contextului.

În proiectul tATAmI s-a încercat dezvoltarea unui sistem scalabil, dar în acelaşi timp capabil să lucreze la un nivel avansat cu reprezentarea cunoştinţelor. Accentul cade pe nivelul aplicaţie şi pe schimbul de informaţii între agenţi. Analiza contextului s-a realizat prin două metode: modelarea implicită, folosindu-se o structură ierarhică de agenţi adaptată diferitelor situaţii contextuale şi o reprezentare grafică a informaţiei, ce permite identificarea datelor de interes şi rezolvarea posibilelor probleme date de lipsa anumitor detalii.

2.3 Limbajul S-CLAIM

Agenţii folosiţi în proiectul nostru au fost creaţi cu ajutorul limbajul S-CLAIM. Acesta este un limbaj special creat pentru proiectarea de agenţi inteligenţi. Limbajul reprezintă o imbunătăţire a limbajului CLAIM, denumirea sa venind de la „Smart CLAIM". A fost dezvoltat de echipa de sisteme multi agent a Laboratorului de Informatică Paris 6 (LIP6), în colaborare cu echipa AI-MAS de la Universitatea Politehnică Bucureşti. Principalele îmbunătăţiri aduse se referă la reducerea complexităţii sintaxei. S-a dorit oferirea unui limbaj orientat-agent, simplu de utilizat, care să reprezinte un bun instrument de programare chiar şi pentru cei care nu sunt familiarizaţi cu alte limbaje de programare. Acesta este scopul pentru care a fost conceput şi limbajul CLAIM (Limbaj computaţional pentru agenţi autonomi şi mobili) – de a fii un limbaj accesibil tuturor, simplu de utilizat de către proiectanţii de agenţi.[2]

Limbajul CLAIM reprezintă un bun instrument pentru crearea de sisteme multi-agent. El este conceput peste limbajul Java şi permite accesarea resurselor Java în mod direct atunci când acest lucru este necesar. Agenţii proiectaţi în limbajul CLAIM sunt rulaţi folosind platforma Sympa. Aceasta este responsabilă de ciclul de viaţă şi mobilitatea unui agent. Deoarece execuţia unui agent pe platforma Sympa nu foloseşte metode standard de comunicaţie şi mobilitate a agenţilor, dezvoltarea şi aducerea de noi funcţionalităţi este dificilă.[2]

Deşi semantica limbajului CLAIM încearcă să acopere o plajă largă de funcţionalităţi, nu permite descrierea componentelor grafice şi nici partea de algoritmică a agenţilor. Acestea vor trebui implementate direct în Java. Pentru o descriere completă a unui agent de complexitate medie programatorul va fi nevoit să îmbine limbajul CLAIM cu Java.

Pentru o mai uşoară întrebuinţare, limbajul S-CLAIM simplifică semantica, care nu este tocmai uşor de parcurs în cazul limbajului CLAIM. O altă îmbunătăţire semnificativă este că renunţă la utilizarea platformei Sympa în favoarea JADE. Reimplementarea limbajului peste platforma JADE permite îmbunătăţirea funcţionalităţilor.

Semantica limbajului S-CLAIM este strâns bazată pe semantica CoCoMo dezvoltată de Abdelkader Behdenna, în timpul stagiului de la Master la LIP6 în 2009. Ea este o simplificare a semanticii limbajului CLAIM, reducând lista de primitive.

2.3.1 Sintaxa limbajului S-CLAIM

Sintaxa limbajului S-CLAIM foloseste paranteze asemeni limbajului LISP. Variabilele sunt precedate de semnul „?". Un agent se defineşte prin nume, parametrii şi lista sa de comportamente: (agent [className] [params] [behaviors]). Definirea unui comportament se realizează astfel: (behavior [type] [statements]). Există două tipuri de comportamente definite:

- **iniţial:** se execută la crearea agentului;
- **reactiv:** se execută în urma primirii unui mesaj şi optional la îndeplinirea unor condiţii.

S-CLAIM folosește structuri de date specifice. Acestea sunt definite prin cuvântul cheie *struct* și lista de membrii a structurii. Primul membru reprezintă de obicei tipul structurii. Comunicația între agenți și lucrul cu cunostiințe foloseste scructuri predefinite de tipul *message* și respectiv *knowledge*. Limbajul se folosește foarte mult de recunoașterea de șabloane. Parametrii primitivelor pot fi legati, sau nu, de o anumită valoare în momentul apelului. Parametrii legați vor impune anumite restricții, iar cei liberi se vor lega de valorile regăsite pe poziţiile în cauză. S-CLAIM nu foloseşte tipuri de date predefinite, acesta va presupune tipul parametrilor în momentul apelului unei primitive. De exemplu primitivele *readK* şi *forAllK* folosesc parametrii legaţi pentru a selecta din baza de cunoştinţe a agentului doar anumite cunoştinţe, iar selecţia este data de valorile ce se vor lega de restul parametrilor. În cazul primitivei *receive* parametrii legaţi impun un anumit format al mesajului primit, iar cei liberi se vor lega la informaţiile primite prin mesaj.[2]

2.3.2 Semantica limbajului S-CLAIM

Limbajul S-CLAIM permite utilizarea următoarelor primitive:

- **primitive pentru comunicare:**
  - **send** - trimite un mesaj către alt agent, sau accesează un serviciu web; (send [to] (strunct message [content-items]));
  - **receive** – primeşte un mesaj de la alt agent sau primeşte o cerere pentru a accesa un serviciu web; (receive [content-items]);
- **primitive pentru mobilitate:**
  - **in** - devine fiul unui agent; agentul se va muta în containerul respectiv cu toată subierarhia sa de agenţi; (in [new-parent]);
  - **out** - părăseşte contextul părintelui său;
- **primitive pentru manipularea agenţilor:**
  - **open** – cere dizolvarea unui agent fiu, acumulând toate capabilităţile şi cunoştinţele agentului fiu;
  - **acid** - agentul se dizolvă în părintele său, părintele acumulând toate capabilităţile şi cunoştinţele agentului fiu;
  - **new** - crează un agent nou, care devine fiu al său;
- **primitive pentru manipularea cunoştinţelor:**
  - **addK** - adaugă o nouă informaţie în baza de cunoştinţe a agentului; (addK (strunct knowledge [content-items]));
  - **removeK** - sterge o informaţie din baza de cunoştinţe a agentului, în funcţie de corespondenţa acesteia cu un şablon; (removeK (strunct knowledge [content-items]));
  - **readK** - citeşte prima cunoştinţa din baza de cunoştinţe a agentului care corespunde şablonului, neţinând cont de informaţiile nespecificate în şablon; (readK (strunct knowledge [content-items]));
  - **forAllK** - iterează printre cunoştinţele din baza agentului care corespund unui anumit şablon; la fiecare iteraţie se poate executa şi o serie de alte operaţii care să folosească cunoştinţa selectată; (forAllK (strunct knowledge [content-items]) [statements]);
- **primitive de control:**
  - **condition** - condiţionează activarea unui comportament în funcţie de o valoare logică; (condition [condition]);
  - **if** - condiţionează execuţia unui bloc de operaţii; (if [condition] then [statements] else [statements]);
  - **wait** - întrerupe comportamentul unui agent pentru o perioadă specificată de timp; (wait [time]).

Putem observa că semantica S-CLAIM nu cuprinde şi partea de algoritmică, funcţii de procesare, operaţii aritmetice sau logice. Acestea pot fi implementate în limbajul de programare Java şi pot fi invocate asemeni primitivelor clasice descrise mai sus.

Capitolul 3. Tehnologii folosite

3.1 Framework-ul JADE pentru dezvoltarea de agenţi software

Pentru crearea agenţilor şi implementarea scenariului am folosit platforma Jade. În continuare voi prezenta modelul conceptual al platformei, arhitectura si funcţionalităţile acesteia. Un alt aspect important pe care l-am menţionat este încadrarea tehnologiei în standardele de dezvoltare FIPA (Foundation of Inteligent Physical Agent), care asigură interoperabilitate între proiecte.

JADE (Java Agent Development Framework) este o platformă care facilitează dezvoltarea aplicaţiilor distribuite multi-agent, conform specificaţiilor FIPA (Foundation of Inteligent Physical Agent). Framework-ul a fost dezvoltat de catre CSELT (Centro Studi e Laboratori Telecommunicazioni, cunoscut şi ca Telecomm Italia Lab) împreună cu Computer Engineering Group de la Universitatea din Parma.[6]

FIPA (Foundation for Intelligent Physical Agents) este o organizaţie având standardele IEEE Computer Society, care promovează tehnologia bazată pe agent şi interoperabilitatea propriilor standarde cu alte tehnologii. Standardul FIPA se bazează pe principiul că ar trebui specificat numai comportamentul extern al componentelor sistemului, lăsând arhitectura internă şi detaliile de implementare dezvoltatorilor de platforme individuale. Acest lucru asigură interoperarea între platforme conforme.[7] Platforma JADE oferă compatibilitate completă cu specificaţiile FIPA: comunicare, management şi arhitectură. Există o serie de servicii predefinite pe care aceasta le oferă şi o serie de interfeţe standard ce sunt implementate.[6]

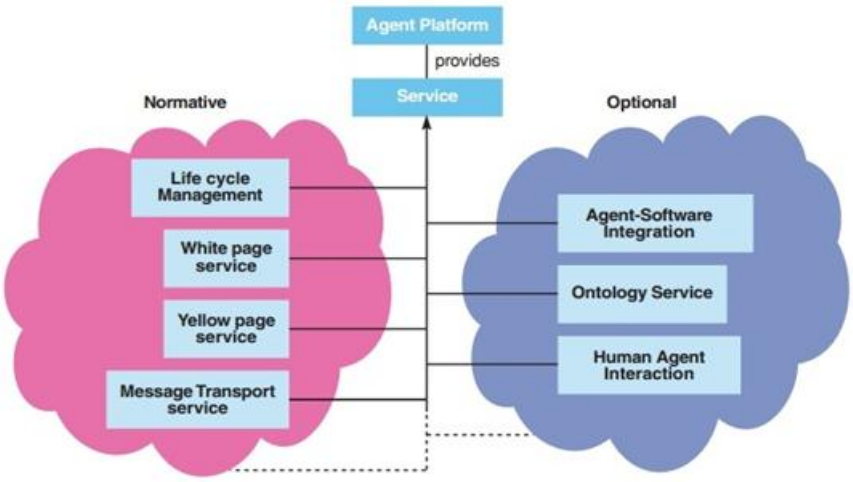

Figura 1. Standardul FIPA de servicii[6]

JADE reprezintă un middleware de dezvoltare şi execuţie a aplicaţiilor peer-to-peer peste reţea, ce folosesc paradigma orientată agent. Acestea pot intercomunica atât în mediu de reţea ethernet cat şi wireless. Modelul peer-to-peer este caracterizat de faptul că orice instanţa (nod) poate avea atât rol de server cât şi de client. Adică orice nod poate iniţia comunicaţia şi trimite cereri către o altă instanţă, în acelaşi timp ascultând cereri venite din exterior. Nu mai există o centralizare ca în cazul modelului client server, rolurile de client şi de server îmbinându-se într-unul singur. Fiecare nod poate părăsi sau se poate alătura topologiei în orice moment, şi poate descoperii singur restul

nodurilor din retea.[6]

Acest model se potriveşte foarte bine cu paradigma orientată agent peste sisteme distribuite. Fiecare agent este reprezentat de către un nod din reţea. Acesta trebuie să fie capabil să iniţieze trimiterea de mesaje către alţi agenţi, această caracteristică fiind una dintre diferenţele majore ale comunicării inter-agent şi RPC (remote procedure call).

Comunicarea joacă un rol foarte important în sistemele multi agent. Acestea trebuie să pună la dispoziţie o suită de funcţionalităţi:

- comunicarea trebuie să se realizeze asincron;
- transmiţătorul unui mesaj nu trebuie să rămână blocat până când mesajul va fi primit;
- receptorul are dreptul de a analiza mesajele primite, de a le filtra şi de a le oferi priorităţi diferite de răspuns;
- trimiterea şi primirea unui mesaj se realizează total independent în timp;
- receptorului îi este permis să nu fie disponibil la momentul respectiv sau chiar să nu existe; cu toate acestea el va primii mesajul atunci când va fi disponibil;
- mesajele pot avea ca destinaţie un grup de agenţi ce au în comun diferite proprietaţi sau interese;
- acţiunea de a comunica trebuie pusă pe acelaşi nivel de interes ca şi restul acţiunilor unui agent;
- comunicarea între agenţi trebuie să respecte un protocol comun astfel încât conţinutul mesajelor să poată fii interpretate la fel de către toţi agenţii.

3.1.1 Modelul Arhitectural

Platforma JADE oferă cadrul în care agenţii pot exista, opera şi comunica. JADE conţine, pe lângă biblioteca de clase pentru dezvoltarea şi manipularea agenţilor, un runtime environment ce trebuie activat pentru a putea lansa agenţii în execuţie. Fiecare instanţa a acestui runtime environment este numită *Container*. Fiecare container poate conţine mai mulţi agenţi. Toate containerele active formează o Platformă, în care trebuie să existe obligatoriu un singur container special numit *Main Container*. Această platforma are rolul de a face „invizibile”, faţă de agenţi şi de utilizatori, nivelurile de dedesubt (nivelul hardware, sistemul de operare, nivelul retea, JVM).[4]

JADE a fost implementat integral în limbajul Java, şi poate rula peste următoarele maşini virtuale: J2EE (Java Platform Enterprise Edition), J2SE (Java Platform Standard Edition), J2ME (Java Platform Micro Edition – pentru aplicaţii embedded). Totodată, JADE a fost integrat în platforma .NET.

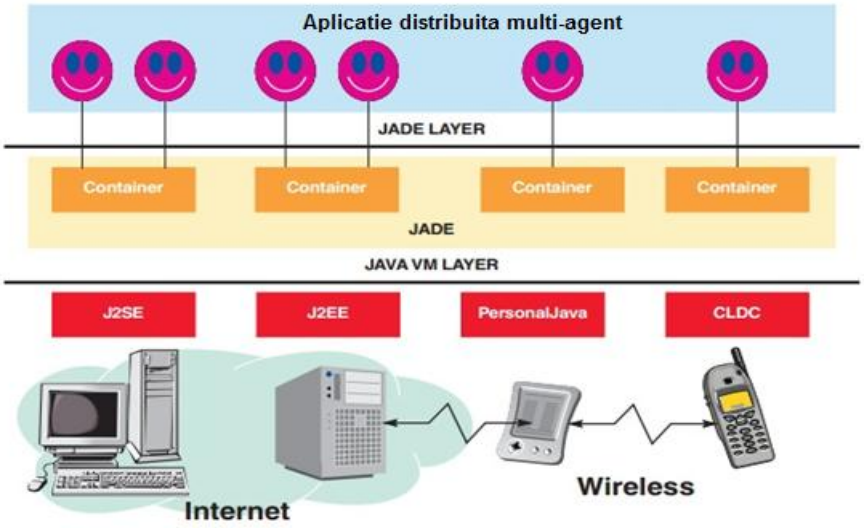

Figura 2. Arhitectura JADE [6]

Alte caracteristici ale platformei JADE sunt:

- un Agent poate rula numai într-un Container (fie el principal, MainContainer, sau nu);
- un Container poate să conţină nici un Agent, un Agent sau mai mulţi agenţi;
- un Container poate aparţine unei singure Platforme;
- containere ale aceleiaşi Platforme pot fi găzduite pe host-uri diferite, realizând astfel o Platformă distribuită;
- fiecare Platformă trebuie să conţină exact un container principal (MainContainer);
- Main Containerul este primul care porneşte pe Platformă. Restul containerelor se înregistrează la acesta când pornesc;
- Main Container include doi agenţi speciali: AMS si DF;
- AMS reprezintă autoritatea în platformă şi este singurul agent capabil de a efectua acţiuni de management, cum ar fi pornirea şi uciderea de agenţi sau închiderea platformei (agenţii normali pot solicita astfel de acţiuni la AMS);
- DF oferă serviciul *yellow pages* care ţine evidenţa serviciilor agenţilor conectaţi la Platformă;
- dacă există deja un Main Container pe Platformă şi se crează încă unul, se va creea o nouă Platformă;
- un host poate să gazduiască unul sau mai multe Containere ale aceleiaşi Platforme;
- un host poate să gazduiască una sau mai multe Platforme.

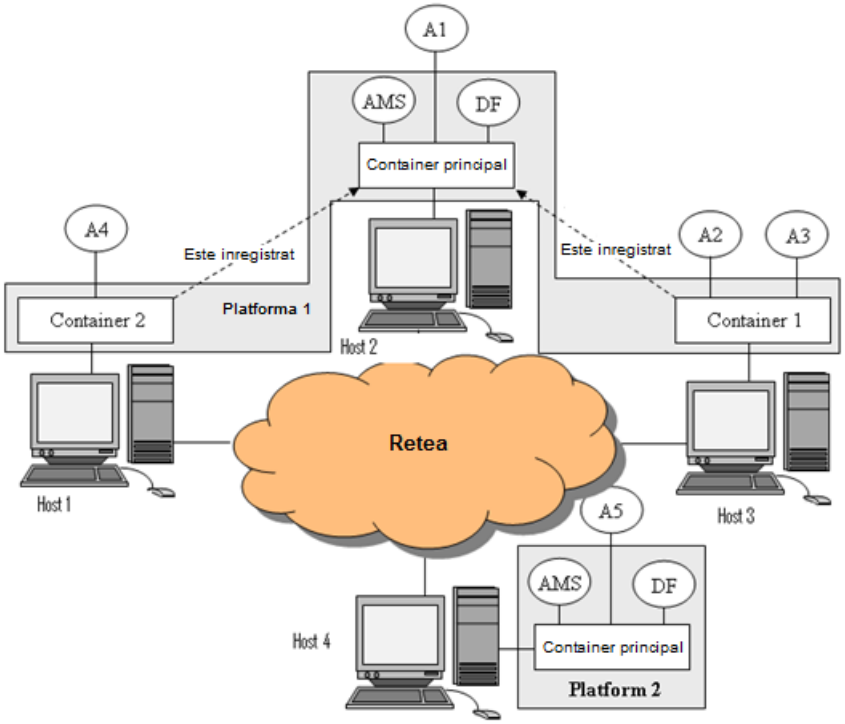

Figura 3. Arhitectura JADE [6]

3.1.2 Modelul funcţional

JADE oferă serviciile de bază, necesare unei aplicaţii distribuite peste o configuraţie peer-to-peer, dar şi pentru o aplicaţie specifică unui dispozitiv mobil. JADE permite fiecărui agent să descopere dinamic alţi agenţi şi să comunice cu aceştia conform paradigmei peer-to peer. Din punctul de vedere al aplicaţiei, fiecare agent este identificabil printr-un nume unic şi oferă un set de servicii. Poate să-şi înregistreze serviciile oferite dar şi să găsească alţi agenţi în funcţie de tipul de servicii pe care aceştia le pun la dispozitie. Un agent poate face singur modificări asupra ciclului său de viată.[6]

Comunicarea între agenţi se face prin schimburi asincrone de mesaje. Nu este necesar ca aceştia să deţină informaţii adiţionale unul despre celălalt, ci doar numele agentului căruia îi este destinat mesajul. Trimiterea mesajului se realizează printr-o simplă comandă în care este specificat numele destinatarului. Ca o consecinţa a acestui fapt, putem observa că nu există o dependenţă temporală în comunicarea dintre cei doi agenţi. În momentul trimiterii mesajului este posibil ca destinatarul să nu fie disponibil pentru a primii mesajul sau chiar, să nu existe la acel moment de timp. Trimiterea unui mesaj se poate realiza şi fără ca expeditorul să cunoască numele explicit al destinatarului, ci este de ajuns doar o proprietate sau un domeniu de interes al acestuia. De exemplu un agent poate trimite mesaje tuturor agenţilor interesaţi de un anumit domeniu. Se observă că platforma, prin serviciile oferite, face ca referinţele efective ale agenţilor să fie invizibile.[10]

Fiecare agent deţine o căsuţă poştală unde sunt păstrate mesajele primite. În figură este descris procesul de trimitere a unui mesaj. Agentul expeditor pregăteşte mesajul şi antetul acestuia cu informaţii specifice şi îl trimite prin intermediul platformei JADE. Aceasta este responsabilă cu identificarea agentului destinatar şi a adresei acestuia. Mesajul este pus în căsuţa poştală a agentului destinatar, iar acesta este notificat de primirea sa. Agentul destinatar va decide singur daca va citi mesajul din coada de aşteptare, când îl va citi şi cum va reacţiona în continuare.[10]

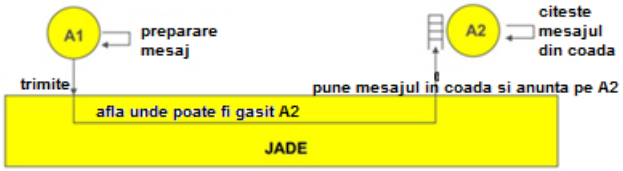

Figura 4. Modelul de comunicaţie inter-agent[10]

JADE oferă metode de securitate precum autentificarea şi verificarea drepturilor agenţilor, pentru aplicaţiile ce necesită asemenea mecanisme. Totodată se pot impune reguli şi drepturi asupra transmisiei şi recepţionării de mesaje. Conţinutul mesajului este încapsulat într-un „plic” care conţine informaţiile necesare nivelului transport. Conţinutul fiind astfel separat de antet, poate fi criptat ca măsură de securitate. Structura unui mesaj este conformă cu limbajul ACL (Agent Communication Language) definit de FIPA şi cuprinde informaţii referitoare la contextul în care s-a trimis mesajul, perioada de timp în care să se aştepte răspuns dar şi informaţii referitoare la transmisiile paralele de mesaje.

Pentru a facilita realizarea unor conversaţii de complexitate ridicată între agenţi, JADE oferă ca suport o serie de şabloane ce pot fi implementate în cazul unor comunicaţii cu scop specific (negocieri, licitaţii delegări de activităţi). Aceste şabloane se ocupă în special de problemele de sincronizare, timeout-uri, restricţii. Astfel programatorul va evita erorile ce pot apărea datorită logicii scenariului. JADE oferă deasemenea sprijin în manipularea conţinutului mesajelor, dar şi în schimbarea rapidă a conţinutului, permitând conversia în mod automat de la un mesaj primit, la unul ce trebuie trimis. Există integrat şi un set de instrumente ce permit programatorului să creeze grafic o ontologie. (O ontologie reprezintă informaţia ca un set de concepte dintr-un anumit domeniu, precum şi relaţiile dintre aceste concepte).[6]

Task-urile agenţilor sunt implementate sub formă de comportamente. Fiecărui agent i se poate asigna unul sau mai multe comportamente. JADE oferă suport pentru paralelizarea comportamentelor agenţilor, mărind astfel scalabilitatea şi adaptându-se mediilor de dezvoltare ce dispun de resurse limitate.

Exista 3 şabloane de comportamente:

- One shot, definit printr-un set de acţiuni finite ca timp. Agentul îl execută o singură dată, după care, comportamentul se încheie şi este scos din lista activă;
- Cyclic, definit printr-un set de acţiuni ce se execută ciclic la infinit;
- Complex, introduce o stare curentă a agentului, în funcţie de care acesta execută acţiuni diferite ce sunt definite în comportament. Se încheie doar dacă este satisfacută o condiţie de stare.

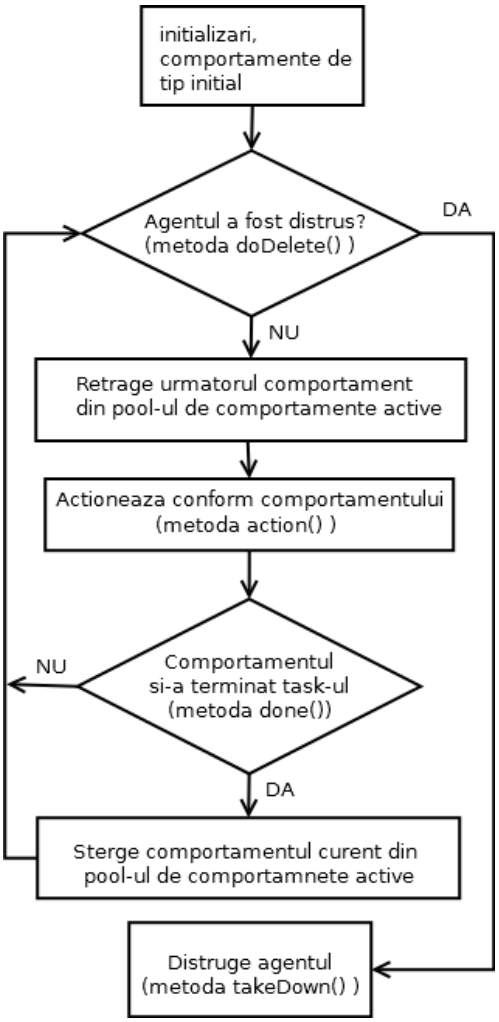

Figura 5. Ciclul de viată al unui agent JADE[10]

JADE implementează şi un mecanism de mobilitate al codului şi al stării de execuţie al unui agent. Astfel, un agent poate să se oprească din acţiunea pe care o face pe un anumit dispozitiv, să migreze pe un alt dispozitiv (fără a fi necesar ca pe acel dispozitiv să existe codul sursă al agentului în cauză) şi să-şi reia execuţia din punctul în care a fost întreruptă. Această funcţionalitate permite distribuirea puterii de calcul, prin mutarea agenţilor pe maşini mai puţin încărcate, fără a avea consecinţe nedorite asupra aplicaţiei sau a rezultatului obţinut.

JADE include mai multe servicii, printre care: un serviciu de nume al agenţilor care asigură că fiecare agent are un nume unic în cadrul platformei şi un serviciu „yellow pages”, unde agenţii îşi publică descrierea şi serviciile oferite, pentru a putea fi descoperiţi de către ceilalţi agenţi. Aceste servicii pot fi distribuite pe mai multe dispozitive din cadrul platformei.[9]

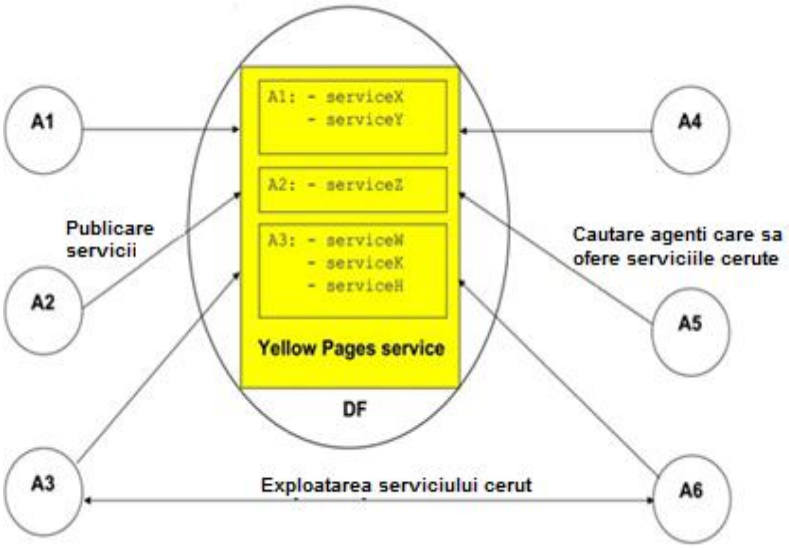

Figura 6. Serviciul yellow pages[9]

JADE oferă tool-uri grafice pentru debugging, management, control si monitorizare:

- RMA (Remote Monitoring Agent)
  - monitorizează şi controlează platforma împreună cu containerele sale
  - controlează de la distanţă ciclul de viaţă al agenţilor prin următoarele operaţii: creare, suspendare, repornire, distrugere, migrare, clonare;
  - poate compune şi trimite mesaje aleatoare unui agent;
  - porneşte restul instrumentelor grafice;
- Dummy Agent
  - compune şi trimite mesaje aleatoare altor agenţi;
  - încarcă/salvează stiva de mesaje din/în fişier;
- Sniffer Agent
  - interceptează mesajele schimbate între alţi doi agenţi;
  - arată fluxul de mesaje schimbate şi conţinutul acestora;
- Introspector Agent
  - monitorizează starea internă a unui agent;
  - oferă informaţii despre mesajele trimise, primite sau în aşteptare ale unui agent;
  - oferă informaţii despre comportamentele planificate pentru un agent;
  - oferă posibilitatea de debugging a execuţiei unui agent;
- Log Manager Agent
  - oferă posibilitatea de manipulare a logurilor în timp real;
- DF (Directory Facilitator)
  - permite interacţiunea cu serviciul *yellow pages* prin următoarele operaţii: căutare agenţi, înregistrare şi dezinregistrare agenţi, modificarea descrierilor agenţilor.

3.2 Sistemul de operare Android

Pentru a evidenţia aspecte fundamentale ale Inteligenţei Ambientale, cum ar fi acelea de flexibilitate şi omniprezenţă, am ales extinderea platformei tATAmI pe dispozitive mobile.
Acestea ocupă astăzi un rol esenţial în vieţile noastre. Conform estimărilor Cisco, anul acesta, numărul de dispozitive mobile cu acces la Internet (telefoane, laptopuri şi tablete în principiu) va depăşi ca număr populaţia la nivel global. S-a estimat că până în 2016 vor fi peste 10 miliarde de gadgeturi cu acces la Internet.[3]

Pentru a face posibilă crearea de agenţi pe platforma tATAmI, care să funcţioneze şi pedispozitive mobile, s-a ales integrarea în platformă a tehnologiei Android. Aceasta este cea mai răspândită tehnologie ce se regăseşte la momentul actual pe dispozitivele mobile. Reprezintă o platformă proiectată special pentru utilizarea sa pe telefoane şi tablete, şi cuprinde: un sistem de operare bazat în întregime pe Linux şi JAVA.

Android a avut întâietate în faţa celorlalte tehnologii de pe piaţa, precum Windows Phone de la Microsoft, sau iOS de la Apple, acestea din urmă fiind construite pe sisteme de operare proprietare, care restricţionează comunicarea între aplicaţiile software şi sistemul nativ al telefonului, restricţionând dezvoltarea de către terţi a aplicaţiilor ce ţintesc platformele lor.

Tehnologia Android fost dezvoltată de către alianţa OHA (Open Handset Alliance) – un grup format la momentul actual din 84 de companii: operatori de telefonie mobilă, producători de dispozitive mobile, producători de dispozitive semiconductoare, companii software şi companii de retail. Alianţa a fost fondată în anul 2007 şi în fruntea ei se află compania Google, care de altfel, este prima care a achiziţionat software-ul original de dezvoltare în anul 2005. Aceasta a lansat apoi codul Android ca open-source, sub licenţa Apache. În momentul de fată, proiectul AOSP (Android Open Source Project), condus de Google este responsabil de mentenanţa şi dezvoltarea continuă a produsului.[11] „Scopul proiectului Open Source Android este de a crea un produs de succes, care îmbunătăţeşte experienţa mobilă pentru utilizatori” ["Philosophy and Goals | Android Open Source". Sursa: android.com. Accesat la data 2012-02-20].

Sistemul Android este compus dintr-un nucleu (Kernel) de Linux şi o colecţie de biblioteci de funcţii scrise în limbajele C si C++. Funcţiile sunt expuse printr-un framework ce asigură servicii pentru gestiunea şi rularea de aplicaţii Java.

Structura sistemului de operare Android, descrisă pe niveluri este următoarea:

- **Kernel-ul Linux** al sistemului de operare oferă o interfaţă abstractizată de legătură între nivelurile superioare şi componentele hardware ale telefonului. El realizează gestiunea proceselor şi a memoriei;
- **biblioteci de metode scrise in C si C++** ce oferă funcţionalităţi de bază, precum funcţii grafice 2D şi 3D, funcţii audio, suport pentru baze de date, funcţii de securizare a comunicaţiei pe Internet şi altele;
- **modulul Run-Time**, modul ce dă posibilitatea rulării în paralel a proceselor din spatele aplicaţiilor Java;
- **framework-ul pentru dezvoltarea de aplicaţii** pune la dispoziţie clase necesare dezvoltării de aplicaţii Java;
- **nivelul aplicaţie** reprezintă totalitatea aplicaţiilor ce rulează la un moment dat deasupra sistemului de operare.

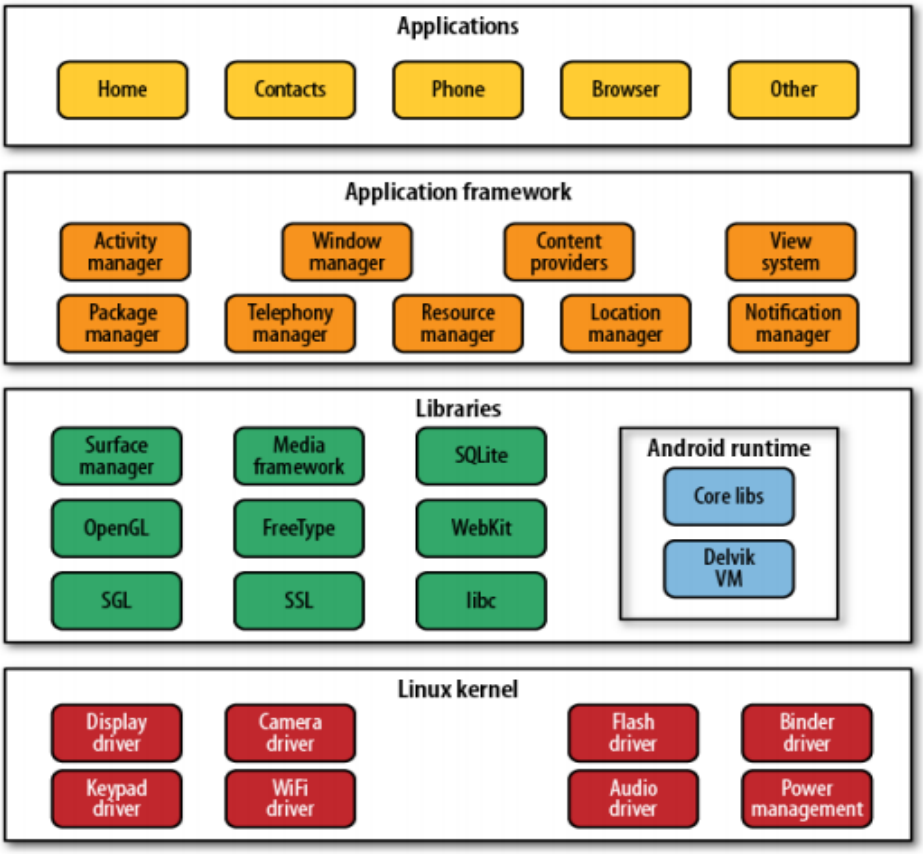

Figura 7. Niveluri software ale sistemului de operare pentru dispozitive mobile Android [11]

Un aspect deosebit de important este securitatea oferită de sistemul Android aplicaţiilor. Acestea se execută având fiecare o identitate proprie definită prin două valori “Linux User ID” şi “Group ID”. Aplicaţiile sunt astfel izolate între ele şi izolate de sistem. Fără acordul utilizatorului nici o aplicaţie nu poate să afecteze alte aplicaţii sau sistemul de operare. Accesul la resurse comune sau informaţii personale (mesaje, persoane de contact, email-uri, etc.) este declarat sub forma unei liste cu permisiuni ce trebuie aprobată de utilizator în momentul instalării aplicaţiei.

Un alt element cheie în gestiunea aplicaţiilor Android este Maşina Virtuală Dalvik (VMD), care face parte din modulul Run-Time. Ea substituie tradiţionala Maşină Virtuală Java (JVM), fiind o variantă adaptată rulării mai multor instanţe pe acelaşi device, urmărind utilizarea cît mai eficientă a resurselor limitate de memorie şi procesor pe care le pun la dispoziţie dispozitivele mobile.[11]

Programele pentru Android:

- sunt scrise în limbajul de programare Java;
- sunt compilate într-un limbaj intermediar denumit „bytecode”, în fişiere de tip „class”;
- fişierele „class”, compatibile cu Maşina Virtuală Java sunt apoi transformate în fişiere cu extensia „dex”, compatibile cu Maşina Virtuală Dalvik. Un fişier „dex”poate încorpora mai multe clase, pentru a se economisi memorie. „Bytecode-ul”este transformat într-un set alternativ de instrucţiuni;
- urmează instalarea fişierului „dex” pe un device ce rulează Android;
- etapa finală este execuţia aplicaţiei, DalvikVM realizând acest lucru cu ajutorul unui compilator de tip “just-in-time”.

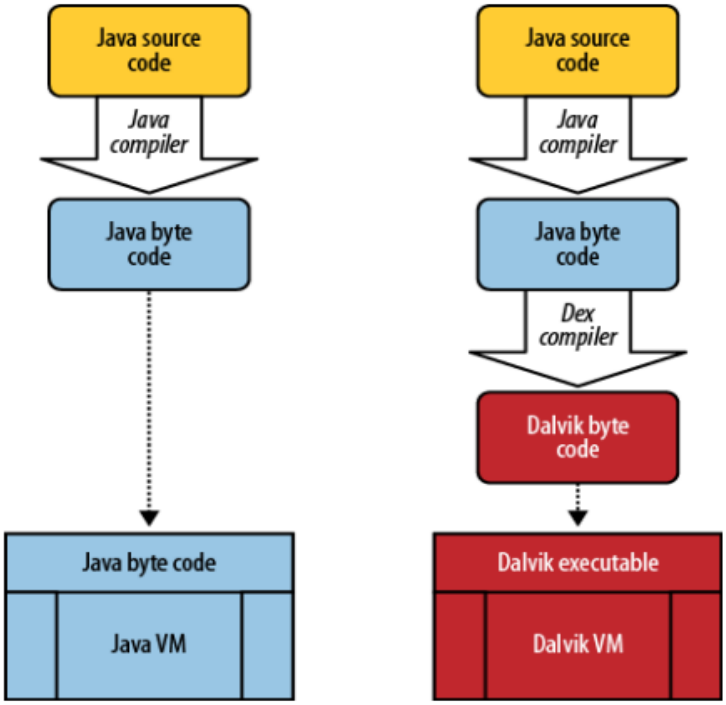

Figura 8. Java versus Dalvik [*11*]

3.3 Platforma JADE si Android – modulul LEAP

Având în vedere creşterea nevoii de aplicaţii pentru mediile mobile (telefoane inteligente, tablete, PDA-uri) şi necesitatea de integrare a acestora cu reţelele de calculatoare fixe, platforma JADE a trebuit regândita pentru a se plia pe sistemul Andoid.

Diferenţele principale ce intervin atunci când discutăm despre JADE şi platformele mobile sunt următoarele:

- mediul JADE complet are o dimensiune (Mega Bytes) care depăşeşte posibilităţile limitate ale dispozitivelor mobile;
- mediul JADE complet necesită versiunea Java 5 sau versiuni ulterioare;
- reţelele mobile au intârzieri mari, dimensiuni reduse ale lăţimii de bandă, conexiuni mai puţin stabile la Internet.

Modulul LEAP(Lightweight Extensible Agent Platform) al platformei JADE rezolvă aceste probleme şi asigură mediul propice de creare şi gestiune a agenţilor. Acesta înlocuieşte anumite parţi ale nucleului JADE, putând fi utilizat pe o multitudine de dispozitive mobile.
Există două moduri în care JADE LEAP poate rula:

- modul normal sau „Stand-Alone”, în care un Container complet este rezident pe gazda pe care platforma JADE este activă;
- modul împărţit sau „Split”, în care Containerul este distribuit într-un bloc simplificat „Front End” ce rulează pe dispozitivul gazdă şi un bloc „Back End” ce rulează pe un server; cele două parţi ale Containerului sunt legate printr-o conexiune permanentă.

Unul dintre atuurile platformei este faptul că programatorul sau dezvoltatorul de aplicaţii nu tratează diferit agenţii care rulează în modul normal „Stand-Alone” de cei care rulează în modul împărţit „Split”, fie ei pe server, fie pe dispozitivele mobile.[12]

Figura 9. Rularea agenţilor în mod normal sau împărţit între un device mobil şi un server[12]

Aplicaţia pe care am realizat-o foloseşte modul normal de rulare a agenţilor, containerele fiind integral pe dispozitivele mobile.

Capitolul 4. Arhitectura sistemului

4.1 Descrierea arhitecturii şi a funcţionalităţii proiectului

Proiectul tATAmI este un model de sistem multi-agent pentru inteligenţa ambientală ce poate fi văzut ca un middleware pentru dezvoltarea ulterioară a unei game mai largi de aplicaţii şi servicii inteligente. O caracteristică importantă a sistemului este genericitatea pe care o oferă aplicaţiilor care se doresc a fi implementate. Acest lucru a fost posibil de realizat prin abordarea doar a nivelului aplicaţie, din suita de nivele ce sunt implicate într-un mediu inteligent. [2]

Platforma tATAmI este un sistem distribuit ce prezintă fiabilitate în cazul în care un dispozitiv este înlăturat din sistem sau devine temporar indisponibil. Agenţii utilizaţi prezintă mecanisme de auto-organizare, nu este nevoie de un control global centralizat al acestora ci de o interacţiune la nivel local. Este posibilă astfel coordonarea unui numar mare de agenţi. Aceştia prezintă caracteristici cognitive şi au un comportament flexibil în funcţie de capacitaţile dispozitivului. [2]

O trasătură de bază a sistemului este dependenţa de context, agenţii accesează şi schimbă în mod natural informaţii de context. Agenţii sunt structuraţi ierarhic prin relaţii de tipul tata-fiu. În contextul unui agent este inclusă şi această proprietate de a fi fiul unui alt agent. Dacă părintele este definit, fiul se va deplasa odata cu acesta, păstrându-şi astfel o parte din context. Părintele se va deplasa cu toată subierarhia sa de agenţi. Această funcţionalitate se numeşte mobilitate ierarhică.[2]

Platforma tATAmI dispune de mecanisme de simulare a scenariilor de AmI şi de exemple de scenarii ce au fost implementate pentru testarea sistemului şi pentru a accentua cerinţele unui mediu de inteligenţă ambiantală la scară reală. Un astfel de scenariu am implementat şi noi pentru a testa funcţionalitatea proiectului. Scenariul şi modul său de implementare sunt descrise în Capitolul 5 al acestei lucrări. O funcţionalitate esenţială a platformei este uşurinţa de testare a unui scenariu odată ce acesta a fost definit. Este posibilă testarea unui scenariu în mod repetat fără a schimba parametrii de testare ceea ce implică obţinerea aceloraşi rezultate, deci reproducerea completă a scenariului. Parametrii se configurează într-un fişier XML şi se referă la platforma creată, containere, agenţi, cunoştiinţele lor iniţiale şi evenimente generate. Platforma foloseşte framework-ul pentru dezvoltare de agenţi JADE şi utilizează S-CLAIM ca limbaj de programare orientat-agent.[2]

Pentru a evidentia aplicabilitatea proiectului tatami pe dispozitivele mobile am ales adaptarea si implementarea acestuia pe platforma Andoid.

Am structurat astfel proiectul în două module principale:

1. tATAmI-PC, destinat rulării pe un calculator personal. Acest modul conţine clasele de bază (core) ce gestionează agenţii şi scenariile şi interpretează limbajul specific S-CLAIM. Tot aici este implementată şi partea grafică a aplicaţiei PC, sistemul de jurnalizare (logging) a evenimentelor definitorii în ciclul de viată al agenţilor. Aplicaţia PC realizează totodată şi iniţializarea platformei, containerelor şi a agenţilor.
2. tATAmI-Android, proiect destinat rulării pe un dispozitiv mobil, cuprinde clase specifice platformei, containerelor şi agenţilor în mediul Android, interfeţele grafice de gestiune a acestora.

Această abordare a fost aleasă deoarece programarea de aplicaţii ce rulează pe dispozitive mobile necesită utilizarea de tool-uri şi instrumente specifice. Bibliotecile de funcţii şi clase sunt diferite în cazul dezvoltării unui proiect Android. Datorită nivelului hardware diferit, ce impune

folosirea limitată a resurselor de memorie şi procesare, aplicaţiile rulează peste o masină virtuală (Dalvik), diferită faă de cea pentru aplicaţiile simple Java. Totodată separarea celor doua module este corectă din punct de vedere logic şi conceptual.

4.2 Arhitectura proiectului tATAmI-PC

Proiectul tATAmI-PC adoptă o arhitectură modulară pentru a facilita extinderea şi adăugarea de funcţionalităţi noi.

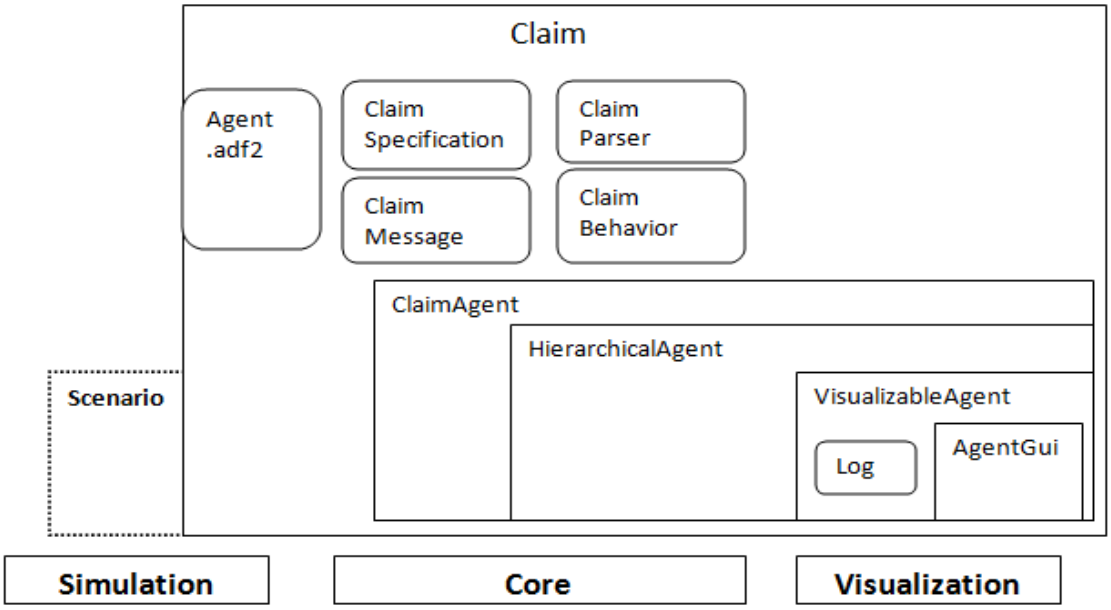

Figura 10. Arhitectura proiectului tATAmI-PC

- Modulul **Simulation** – este responsabil de executarea scenariilor:
  - parseaza fişierele XML de configurare a scenariilor. Acestea cuprind parametrii de execuţie a platformei, informaţii referitoare la containere şi informaţii despre agenţi;
  - execută agenţii pe containerele şi maşinile specificate;
  - transmite mesaje eveniment „externe” echivalente cu percepţiile agenţilor într-un mediu real;
- Modulul **Core** – cuprinde clasele specifice agenţilor, structurate pe mai multe niveluri de funcţionalitate:
  - comunicare, mobilitate şi gestionare de agenţi – sunt utilizaţi agenţi JADE;
  - transabilitate şi vizualizare – agenţii trebuie să transmită date despre activitatea lor, precum şi despre vecinii lor către o entitate centralizată. Fiecare agent va avea o interfată grafică (o fereastră) pe ecranul maşinii gazdă;
  - mobilitate ierarhică pentru agenţi – protocoale şi comportamente care să permită agenţilor să se deplaseze în mod automat împreună cu părinţii lor;
  - acces la servicii web – funcţionalitate pentru a expune agenţii ca servicii web şi pentru a permite agenţilor accesul la servicii web;
  - interpretarea şi execuţia S-CLAIM – un parser pentru fişierele S-CLAIM de descriere a agenţilor şi a componentelor care transformă definiţia în comportament real;
  - baza de cunostinţe – o componentă interschimbabilă care permite accesul la cunoştiinţe prin intermediul unui set standard de funcţii;

  - dependenţa de context – stabilirea contextului şi schimbul de informaţii relevante pentru context.
- Modulul **Visualisation** – asigură o vizualizare centralizată a activitaţii agenţilor:
  - primeşte rapoarte de jurnal de evenimente (log-uri) şi evenimentele de mobilitate de la agenţi;
  - afişeaza toate jurnalele într-un mod centralizat, cronologic;
  - afişează topologia sistemului evidenţiind relaţiile dintre agenţi;
  - oferă componente pentru plasarea automată a ferestrelor agent pe ecranul maşinii pe care se execută.

O componentă importantă a proiectului este JADE – Agent Development Framework. Ea permite executarea distribuită pe mai multe computere şi oferă funcţionalitaţi de mobilitate, comunicare şi management al agenţilor. Agenţii se pot muta de pe un container pe altul, chiar dacă acesta este gazduit pe o maşină diferită, se pot alătura sau pot părăsii oricând platforma fără a influenţa starea sistemului. Un add-on al acestei componete, JADE-LEAP, este folosit similar în proiectul tATAmI-Android. Acesta permite executarea containerelor şi agenţilor pe un dispozitiv mobil ce foloseşte Android şi oferă posibilitatea de deplasare a agenţilor spre şi înspre smartphone, văzând astfel dispozitivul ca o staţie de lucru. Aceste funcţionalitaţi fac invizibile, pentru nivelul platformei, diferenţele hardware dintre dispozitive.

Partea de jurnalizare a activitaţii agenţilor foloseşte un modul pentru Apache, denumit log4j care permite configurarea rapidă a unui jurnal. Acest wrapper trimite informaţiile sub forma unui mesaj JADE la agentul de vizualizare. Acesta le afişează într-o fereastră la consolă.

Partea de web service a fost implementată cu ajutorul add-on-urilor JADE WSIG si WSDC. Acestea integrează primitive pentru accesul la servicii web atât SOAP cât şi REST, oferind agenţilor posibilitatea de a interopera cu alte componente şi infrastructuri.

Agenţii sunt organizaţi pe mai multe niveluri. Acest lucru este util pentru modularizarea proiectului, pentru uşurarea depanării şi a adăugării de funcţionalitaţi.

JADE GuiAgent – gestionează deplasarea agenţilor şi comunicarea la nivelul platformei JADE. Acest nivel reprezintă baza întregii structuri de implementare a agenţilor.

VisualizableAgent – se ocupă de vizualizarea agentului din două puncte de vedere: oferă nivelurilor superioare obiectul *log,* care adună datele de jurnalizare ale agentului şi este trimis apoi agentului de vizualizare; pe de altă parte gestionează fereastra agentului şi o plasează pe ecran.

HierarhicalAgent – gestionează mobilitatea ierarhică a agenţilor; menţine relaţiile dintre agenţi şi implementează comportamentul de urmărire a agentului părinte. Instruieşte agentul fiu să se deplaseze odată cu părintele său şi să accepte ordine de deplasare de la părintele său. Agentul poate fi legat de containerul în care se află, caz în care nu se va mai deplasa odată cu părintele.

Agentul S-CLAIM – conţine tabele de simboluri ale agentului şi un set de descrieri de comportament care reprezintă capacitaţile agentului. Acest nivel accesează baza de cunoştinţe a agentului pentru adăugarea, eliminarea sau interogarea de cunoştinţe.[2]

4.3 Structura proiectului tATAmI-Android

Pentru a putea fi compilat, proiectul Android are o structură specifică de fișiere.

- Fișierul AndroidManifest.xml – este un fișier de control ce descrie natura aplicaţiei şi componentele sale. Aici am declarat folosirea serviciului de execuţie JADE, permisiunea conectării la internet şi versiunea minimă de API necesară rulării. Totodată, în proiect sunt folosite toate Activity-urile iar cel principal se va crea în momentul lansării în executie a aplicaţiei.
- Folderul src – conţine toate fişierele cu cod sursa .java. În pachetul gui am introdus toate fișierele Activity. Acestea gestionează o parte din interfaţa grafică.
- Folderul res/layout – cuprinde resursele folosite în proiect. Aici am introdus fișiere xml care definesc structura grafică a interfeţei agentului PDA. Aceste fișiere sunt compilate şi transformate în elemente de tip view a căror referintă se află în clasa R;
- Folderul res/values – cuprinde fișierul string.xml cu valori ale proprietăţilor declarate în fișierele din layout dar şi cu valori de constante folosite în clasele Activity;
- Folderul gen – cuprinde clase Java generate la compilare;
- Folderul libs - cuprinde bibliotecile necesare în format .jar.

În urma compilării este creat un pachet .apk care reprezintă aplicaţia Android. Pentru rulare este nevoie de execuţia acestui pachet. El este o arhivă zip ce conţine executabilul Dalvik.dex, resurse (imagini, fișiere xml) şi fișierul manifest.xml.

Aplicaţiile Android nu au în componenţă o funcţie main(). Ele sunt compuse din diferite componente ce există ca nişte entităti separate:

- Activitaţi (Activities);
- Servicii (Services);
- Furnizori de Conţinut (ContentProviders);
- Receptori de Difuzare (BroadcastReceivers).

O activitate reprezintă un ecran cu o singură interfaţă pentru utilizator. Dacă o altă activitate este pornită, sistemul o va afişa utilizatorului, iar prima va fi oprită şi introdusă într-o stivă de activităţi. În momentul în care activitatea curentă este oprită de către utilizator, ea va fi distrusă şi urmatoarea activitate din stivă va fi repornită şi afişată pe ecran. De fiecare dată când o activitate trece în altă stare, se apelează metode de callback, care oferă dezvoltatorului posibilitatea de a face operaţii specifice schimbării de stare. Aceste operaţii se referă în special la eliberarea resurselor folosite.[13]
Figura 11 descrie ciclul de viată al unei activitati şi ordinea în care fucţiile de callback sunt apelate.

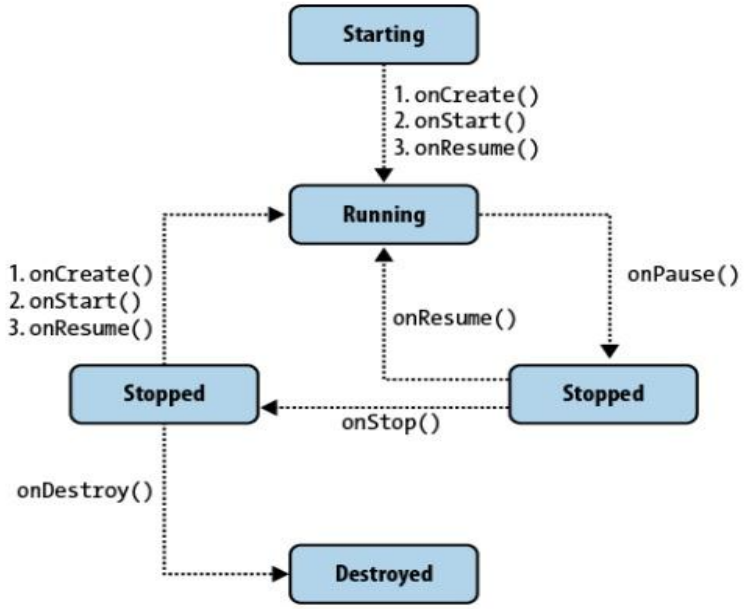

Figura 11. Ciclul de viata al activitati[13]

4.4 Modul de interconectare a celor doua proiecte

Proiectul *tATAmI-PC* este cel care porneşte Platforma JADE şi crează Main Container-ul. Aici sunt creaţi toţi agenţii ce se vor regăsii pe platformă. În funcţie de containerul specificat în fişierul scenario.xml, fiecare agent se mută, dupa ce este creat, pe dispozitivul unde se găseşte container-ul său. Ca urmare a acestui fapt, agenţii ce vor rula pe dispozitive mobile, vor trebui să se mute de pe PC pe PDA şi să-şi atribuie interfaţa grafică corespunzătoare, din proiectul *tATAmI-Android*. Această migrare presupune instantţerea în cadrul proiectului *tATAmI-Android* a agenţilor definiţi în proiectul *tATAmI-PC*. Pentru a satisface acest lucru s-au folosit elemente de Java Reflection şi încărcare dinamică a claselor (Java permite ca în timpul execuţiei unui program să se poată adăuga funcţionalităţi noi în mod dinamic, prin încărcarea claselor necesare la un moment dat).

Pentru a evita redundanţa codului şi a oferii o soluţie stabilă pe viitor, ușor de utilizat, am ales să creem o conexiune între cele doua proiecte, folosind instrumente puse la dispoziţie de mediul de dezvoltare Eclipse. Am inclus în proiectul *tATAmI-Android* o referinţă către proiectul *tATAmI-PC* şi am inclus proiectul *tATAmI-PC* în „Build Path”-ul proiectului *tATAmI-Android*. Astfel compilarea proiectului *tATAmI-Android* va include şi fișierele proiectului *tATAmI-PC*.

Capitolul 5. Detalii de implementare

5.1 Descrierea scenariului implementat

Pentru a testa şi exemplifica integrarea tehnologiei Android în platforma tATAmI am ales să implementăm următorul scenariu. Mai multe grupuri de persoane doresc să dezbată anumite subiecte de discuţie. Fiecare participant are la dispoziţie un smartphone pe care este instalată aplicaţia noastră. El se poate alătura unui grup de discuţii la care să participe. Fiecare participant are posibilitatea de a expune o opinie care să fie pro sau contra subiectului dezbatut. Fiecare grup are la dispoziţie un PC pe care sunt centralizate opiniile de la toate persoanele din grupul respectiv. Participanţii işi vor trimite opinia către PC, utilizând telefonul mobil, iar aceasta va fi afişată pe ecranul PC-ului. Afişarea tuturor opiniilor se va face în două ferestre diferite, una pentru opinii pro şi una pentru opinii contra şi se va specifica numele persoanei care a trimis-o. Participanţii vor putea vedea centralizarea opiniilor şi pe ecranul dispozitivului mobil pe care îl deţin. Aceştia pot retrage o opinie în cazul în care ea nu mai este validă, caz în care listele cu opinii se vor actualiza. În cazul în care o persoană doreşte să părăsească grupul de discuţii şi să se alăture altui grup, toate opiniile pe care şi le-a exprimat şi care se regăseau pe ecranul PC-ului, dar şi pe telefoanele celorlalte persoane, vor migra către noul grup. Participanţii, dacă nu dispun de un telefon mobil cu Android, pot folosi aplicaţia şi de pe PC.

5.2 Modul de implementare a scenariului

Pentru a implementa scenariul propus am folosit platforma JADE astfel:

- există o singură platformă care cuprinde toate grupurile de discuţii;
- vor exista două grupuri de discuţii, fiecare pe câte un PC. Pe fiecare dintre PC-uri va exista un Container specific grupului de discuţii aferent (*DebateGroupIdContainer*). Pe unul dintre cele doua PC-uri va exista şi Main Container-ul necesar coordonării platformei;
- vor exista cinci participanţi, dintre care doi vor folosi smartphone-uri Samsung Galaxy S Plus , doi vor folosi emulatoare pentru a simula mediul unui dispozitiv mobil şi unul va folosi un PC.
- fiecare participant când se va conecta la platformă va crea pe dispozitivul său un Container (*PDAContainerId*);
- există trei tipuri de agenţi ce vor fi gazduiţi în containere: *GroupCoordonatorAgent, PDAAgent, EmissaryAgent;*
- agenţii *GroupCoordonatorAgent* vor aparţine containerului *DebateGroupIdContainer* şi nu vor părăsi aceste containere. Ei vor fi responsabili cu coordonarea grupului de discuţii aferent şi cu centralizarea opiniilor participanţilor;
- agenţii *PDAAgent* vor aparţine containerului *PDAContainerId* de pe dispozitivele mobile şi nu vor părăsi aceste containere. Ei vor fi responsabili de preluarea opiniilor şi a cererilor de la participanţi şi totodată de afişarea pe ecranul personal a opiniilor tuturor participanţilor înscrişi în grup;
- agenţii *EmissaryAgent* se vor naşte ca fii ai agenţilor *PDAAgent,* deci iniţial vor aparţine containerelor *PDAContainerId*. În mometul în care participantul se va înscrie într-un grup de discuţii, agentul îşi va trimite emisarul către agentul coordonator al grupului. Deci agentul *EmissaryAgent* va deveni fiu al lui *GroupCoordonatorAgent* şi se va muta pe containerul *DebateGroupIdContainer;*
- agentul *EmissaryAgent* are rolul unui intermediar. El va memora în baza sa de cunoştinţe agentul *PDAAgent* de la care a plecat pentru a păstra permanent legătura cu acesta. Toate mesajele care se vor schimba între agenţii *PDAAgent* şi *GroupCoordonatorAgent* vor fi

intermediate de către el. Va prelua cererile de la agentul *PDAAgent* şi le va transmite către *GroupCoordonatorAgent* şi totodată va primi de la *GroupCoordonatorAgent* actualizări în ceea ce priveşte lista de opinii pe care le va trimite către *PDAAgent*;

- în cazul în care participantul doreşte să părăsească grupul şi să se alăture unui alt grup de discuţii, agentul *PDAAgent* îl va anunţa pe emisar de noul grup în care doreşte să îl trimită, iar acesta va deveni fiul noului coordonator şi se va muta în containerul aferent noului grup;
- agentul *EmissaryAgent* va păstra toate opiniile utilizatorului de care aparţine şi va fi responsabil de mutarea lor, în cazul în care utilizatorul se va alătura altui grup de discuţii.

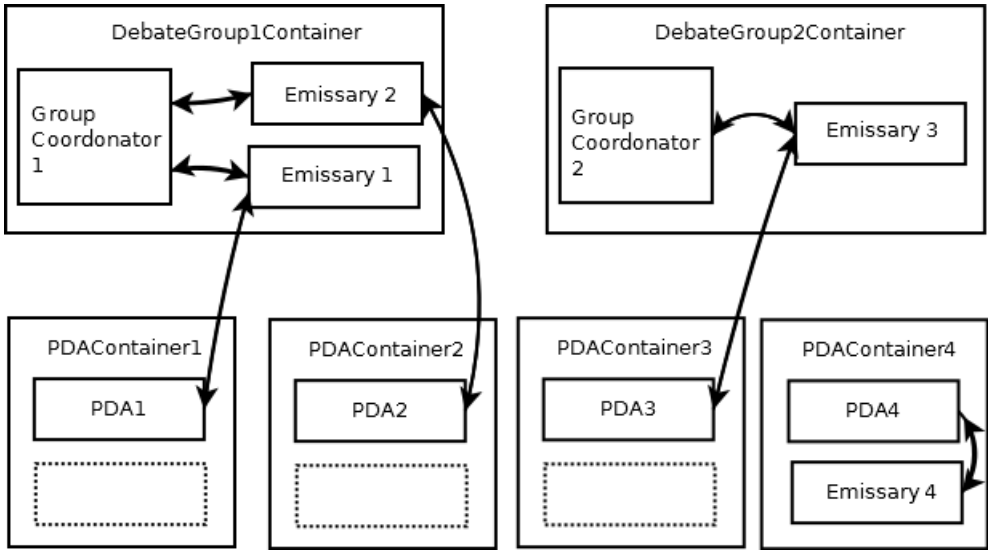

Figura 12. Stare a sistemului multi-agent în scenariului implementat.

Figura 12 reprezintă o posibilă stare în care se poate afla sistemul în decursul scenariului descris. Există cele două containere corespunzătoare grupurilor de discuţii şi patru participanţi la grupuri. Primii doi participă la primul grup de discuţii, al treilea participă la al doilea grup de discuţii, iar cel de-al patrulea încă nu s-a alăturat nici unui grup. Săgeţile reprezintă schimburile de mesaje ce se pot realiza. Se observă că emisarii care s-au înregistrat la un grup de discuţii sunt intermediari între agenţii PDA şi coordonatorii de grup.

Implementarea efectivă a scenariului a fost făcuta folosind un mecanism al proiectului tATAmI bazat pe procesarea de fişiere XML. Acesta are ca scop implementarea rapidă de scenarii sub forma de XML în care să fie specificaţi doar parametrii necesari pentru crearea platformei, a agenţilor şi a containerelor. Trebuie să existe un singur astfel de fişier per maşină. Fişierul pentru maşina care va găzdui Main Container-ul va conţine informaţii despre toţi agenţii care se vor crea. Celelalte fişiere XML vor conţine informaţii doar despre conectarea la platforma deja existentă şi despre containerele locale. Există mai multe scheme ce pot fi folosite pentru parsarea fișierelor XML. Schema ce se doreşte a fi implementată se specifică în fişier.

Pentru a specifica dacă maşina curentă va fi cea care porneşte platforma şi crează Main Container se foloseşte parametrul *isMain* din elementul *<scen:jadeConfig/>*. Tot aici se specifică parametrii necesari conectării la platformă în cazul în care aceasta este pornită: adresa şi portul maşinii unde se află Main Container şi ID-ul platformei (*Ipaddress, port, platformID*). Elementul *<scen:adfPath>* arată calea în proiect unde se află fişierele .adf2 de definire a agenţilor. Elementul *<scen:agentPackage>* arată calea în proiect unde se găsesc restul fişierelor ce sunt necesare pentru funcţionarea agenţilor: fişierele de GUI şi cele cu funcţii Java adiţionale.

```
<scen:jadeConfig isMain="true"/>
```

```
<scen:adfPath>scenario/examples/debateScenario-android</scen:adfPath>
<scen:agentPackage>agent_packages.example.debate</scen:agentPackage>
```

Elementul *<scen:initial>* cuprinde infomaţiile necesare creerii de containere şi agenţi. Un Container se defineşte prin elementul *<scen:container>*. Parametrul *name* specifică numele său. Dacă se doreşte crearea de agenţi în Main Container, containerul va fi definit aici şi va avea parametrul *name* identic cu cel al Main Container-ului. Parametrul *create* specifică dacă se crează sau nu containerul la parsarea fişierului. În cazul în care este *false*, containerul nu va fi creat iar agenţii definiţi vor migra către maşina care va crea containerul cu numele specificat. Un agent se defineşte prin elementul *<scen:agent>*. Acesta are o listă de parametrii definiţi prin nume, parametru şi valoare. Parametrul *class* specifică numele fişierului unde este definit agentul, *java-code* specifică numele fişierului unde se găsesc funcţiile Java folosite adiţional codului S-CLAIM, *GUI* specifică numele fişierului unde este definită interfaţa grafică. Aici se definesc şi informaţiile iniţiale din baza de cunoştinţe a agentului.

```
<scen:initial>
	<scen:container name="DebateGroup1Container">
		<scen:agent>
			<scen:parameter name="loader" value="adf2" />
			<scen:parameter name="class" value="GroupCoordonatorAgent" />
			<scen:parameter name="name" value="GroupCoordonator1" />
			<scen:parameter name="java-code" value="DebateFunctions" />
			<scen:parameter name="GUI" value="GroupCoordonatorAgentGui" />
		</scen:agent>
	</scen:container>
</scen:initial>
```

5.3 Implementarea agenţilor în limbajul S-CLAIM

Comportamentul agenţilor a fost implementat în limbajul S-CLAIM descris în Capitolul 2.3 folosind fişiere .adf2. Agentul *EmissaryAgent* are definit şi un comportament iniţial care presupune păstrarea în baza de cunoştinţe a agentului *PDAAgent,* de care aparţine, notificarea acestuia printr-un mesaj că este emisarul său şi iniţializarea unui număr de secvenţă ce va fi folosit pentru numerotarea listei de opinii pe care urmează să o păstreze.

```
(initial startclock
	(addK (struct knowledge sequence 1))
	(send ?parent (struct message register ?name))
	(addK (struct knowledge masterPDAAgent ?parent)))
```

Agentul va păstra informaţia cu numele emisarului său. Pe tot parcursul existenţei agenţilor, această legătură dintre *PDAAgent* şi emisarul său nu va fi modificată, acesta fiind singurul agent cu care *PDAAgent* va comunica.

```
(reactive register
	(receive register ?emissary)
	(addK (struct knowledge emissary ?emissary)))
```

PDAAgent poate primi din partea utilizatorului 3 tipuri de cereri: *joinGroup, addOpinion, deleteOpinion* pe care le va trimite mai departe către emisar.

```
(reactive joinGroup
	(input join tip ?groupCoordonatorAgent)
	(forAllK (struct knowledge emissary ?emissary)
		(send ?emysarry (struct message leaveGroup))
		(send ?emysarry (struct message join ?groupCoordonatorAgent))))

(reactive addOpinion
	(input add tip ?tag ?opinion)
	(forAllK (struct knowledge emissary ?emissary)
		(send ?emysarry (struct message add ?tag ?opinion))))
```

În cazul cererii *joinGroup*, *PDAAgent* va cere emisarului mai întâi să părăsească grupul curent în care se află înregistrat, ca urmare acesta îi va trimite părintelui său un mesaj de tipul *unregister.* Apoi se va muta în containerul grupului cerut ca fiu al lui *groupCoordonatorAgent,* îl va notifica pe agentul coodonator şi îi va trimite acestuia lista cu toate opiniile sale.

```
(reactive leaveGroup
	(receive leaveGroup)
	(send ?parent (struct message unregister ?name)))

(reactive joinGroup
	(receive join ?groupCoordonatorAgent)
	(in ?groupCoordonatorAgent)
	(send ?parent (struct message register ?name))
	(send this (struct message sendAllOpinions)))


(reactive sendAllOpinions
	(receive sendAllOpinions)
	(forAllK (struct knowledge agentOpinion ?opinionId ?opinionTag ?opinionMsg)
		(send ?parent (struct message opinion ?name ?opinionId ?opinionTag ?
		opinionMsg))))
```

În cazul cererii *addOpinion*, emisarul va incrementa numărul de secvecţă păstrat în baza de cunoştinţe, va memora opinia primită împreună cu noul număr de secvenţă şi va trimite opinia către părintele său, coodonatorul grupului. În cazul cererii *deleteOpinion* va şterge din baza de cunoştinţe opinia şi va cere părintelui să facă *refresh* la ecranele de afişare. Ca urmare a acestui lucru, părintele le va cere tuturor emisarilor înregistraţi la el să îi retrimită listele complete cu opinii.

```
(reactive addOpinion
	(receive add ?opinionTag ?newOpinion)
	(removeK (struct knowledge sequence ?opinionId))
	(addK (struct knowledge agentOpinion ?opinionId ?opinionTag ?newOpinion))
	(increment ?opinionId ?newOpinionId)
	(addK (struct knowledge sequence ?newOpinionId))
	(send ?parent (struct message opinion ?name ?opinionId ?opinionTag ?
	NewOpinion)))

(reactive deleteOpinion
	(receive delete ?opinionId)
	(removeK (struct knowledge agentOpinion ?opinionId ?tag ?opinion))
	(send ?parent (struct message refresh)))
```

Când groupCoordonatorAgent primeşte o opinie de la un emisar al său, aceasta va avea ataşat un tag care să specifice dacă opinia este pro sau contra subiectului. În funcţie de acest tag o va afişa pe unul dintre cele doua ecrane. Apoi va trimite opinia către toţi emisarii înregistraţi. Aceştia o vor trimite către agentul PDAAgent de care sunt legaţi.

```
(reactive receiveOpinion
	(receive opinion ?name ?opinionId ?opinionTag ?opinionMsg)
	(assembleOutput ?name ?opinionId ?opinionMsg ?output)
	(if (equalString ?opinionTag pro)
		then (output proOpinion ?output))
	(if (equalString ?opinionTag con)
		then (output conOpinion ?output))
	(forAllK (struct knowledge children ?child)
		(send ?child (struct message displayOpinion ?output ?opinionTag))))
```

Când *groupCoordonatorAgent* primeşte un mesaj *refresh* de la un emisar al său, şterge cele două ecrane şi le cere tuturor emisarilor înregistraţi să retrimită listele complete cu opinii şi să notifice şi agenţii *PDAAgent* pentru a reface afişajul.

```
(reactive refreshOpinions
	(receive refresh)
	(output clear)
	(forAllK (struct knowledge children ?child)
		(send ?child (struct message sendAllOpinions))
		(send ?child (struct message refresh))))
```

5.4 Conectarea aplicaţiei Android la platforma tATAmI

Pentru ca aplicaţia să se conecteze la platforma JADE iniţiată din proiectul tATAmI pe PC, este nevoie ca utilizatorul să cunoască adresa IP a staţiei şi portul pe care platforma ascultă cereri. Portul folosit implicit în cazul framework-ului JADE este *1099*. Aceste informaţii, împreună cu un nickname utilizat în cadrul aplicaţiei, se vor introduce de către utilizator în fereastra de început. Înainte de rulare, trebuie asigurat faptul că între dispozitive există conectivitate şi că acestea sunt configurate astfel încat JADE să poată deschide o conexiune TCP pe portul 1099. Am efectuat teste doar în situaţia în care PC-urile şi dispozitivele mobile se află interconectate în aceeaşi reţea. În conformitate cu arhictectura sa, Android oferă un serviciu peste JADE. Biblioteca jadeAndroid.jar include două servicii: clasele jade.android.RuntimeService şi jade.android.MicroRuntimeService care servesc la crearea de containere pe dispozitivul mobil, în modul *stand-alone* şi respectiv *split.*[15] Aceste servicii trebuie declarate în *fișierul manifest.xml*.

```
<service android:name="jade.android.RuntimeService" />
```

Pentru a avea acces dintr-un Activity la o instantă JADE prin care să se efectueze operaţii specifice, trebuie întâi să se realizeze conexiunea la unul dintre aceste servicii. Pentru a porni un Container se foloseşte funcţia oferită de framework *createAgentContainer*.

```
agentContainer = runtime.createAgentContainer(profile);
```

Parametrul *profile* reprezintă un obiect specific ce cuprinde toţi paramerii necesari conectării la platformă şi creerii unui container: adresa şi portul platformei, adresa şi portul local, numele containerului, dacă este sau nu Main Container.

```
final Properties prop = new Properties();
prop.setProperty(Profile.MAIN_HOST, host);
prop.setProperty(Profile.MAIN_PORT, port);
prop.setProperty(Profile.MAIN, Boolean.FALSE.toString());
prop.setProperty(Profile.JVM, Profile.ANDROID);
prop.setProperty(Profile.LOCAL_PORT, "1099");

if (AndroidHelper.isEmulator()) {
      prop.setProperty(Profile.LOCAL_HOST, AndroidHelper.LOOPBACK);
} else {
      prop.setProperty(Profile.LOCAL_HOST, AndroidHelper.getLocalIPAddress());
}

final Profile profile = new ProfileImpl(prop);
rofile.setParameter(Profile.CONTAINER_NAME,
                        this.getString(R.string.container_name));
```

5.5 Diferenţa dintre emulator şi un dispozitiv mobil real

Platforma Android pune la dispoziţie un emulator AVD (Android Virtual Device) ce are ca rol înlocuirea unui dispozitiv real. El oferă aceleaşi funcţionalitaţi, imitând componentele software şi hardware ale acestuia. Diferenţa dintre ele este că nu poate realiza un apel telefonic real. El permite configurarea, în funcţie de preferinţele utilizatorului, a unui profil hardware, prin activarea diferitelor componente (cameră, tipul tastaturii, capacitatea de memorie folosită). Permite alegerea versiunii de SDK folosite şi a interfeţei grafice, aspect ce influenţează caracteristicile şi dimensiunea ecranului. Acest emulator permite dezvoltarea şi testarea aplicaţiilor în lipsa unui dispozitiv real.[14]

Atunci când este folosit un emulator în locul unui dispozitiv, trebuie avute în vedere câteva aspecte. AVD-ul instalat trebuie să folosească minim versiunea API level 10 (Android 2.3.3) pentru care a fost concepută aplicaţia. Această versiune este specificată în fişierul manifest.xml prin parametrul minSdkVersion (*<uses-sdk android:minSdkVersion="10" />*). În etapa de conectare la plaftforma JADE, adresa locală ce va fi setată şi trimisă platformei va fi adresa de loopback.

Fiecare instanţă a emulatorului rulează în spatele unui router virtual, a unui serviciu de firewall, care îl izolează de interfeţele de reţea ale maşinii gazdă şi de internet. Un dispozitiv emulat nu poate vedea computerul care îl găzduieşte sau alte instanţe de emulatoare prin reţea. În schimb vede că este conectat prin Ethernet la un router. Acest router, are în folosinţă spaţiul de adrese de reţea 10.0.2/24. Fiecare instanţa are în spate propriul router virtual, ca urmare instanţe ce rulează în paralel vor asigna aceleaşi adrese şi nu va exista conectivitate între ele.[14]

Următoarele adrese sunt predefinite:

10.0.2.1 – adresa routerului (gateway)
10.0.2.2 – adresa de loopbak (alias al adresei 127.0.0.1 de pe maşina gazdă)
10.0.2.3 – prima adresă DNS
10.0.2.4 / 10.0.2.5 / 10.0.2.6 – adrese DNS opţionale
10.0.2.15 – interfaţa de reţea a emulatorului
127.0.0.1 – interfaţa de loopbak a emulatorului

Pentru ca Platforma JADE să poată comunica cu aplicaţia ce rulează pe emulator, a fost necesară translatarea de porturi. Aceasta s-a realizat prin două metode:

- cu ajutorul tool-ului abd (Android Debug Bridge) prin comanda: *adb forward tcp:1099 tcp:1099;*
- prin conectarea telnet la consola emulatorului: *telnet localhost 5554* si folosirea comenzii redir: *redir add tcp:1099:1099*.

Ambele comenzi realizează o redirecţionare a conexiunilor ce sosesc la adresa 127.0.0.1 pe portul TPC 1009 către adresa 10.0.2.2 pe portul TPC 1009.

5.6 Interfața grafică a aplicației Android

Interfaţa grafică a aplicaţiei este definită prin fişierul AgentGuiActivity şi printr-o serie de fişiere .xml. AgentGuiActivity este responsabil de interacţiunea cu utilizatorul, de încărcarea interfeţei grafice pe ecranul dispozitivului şi de comunicaţia cu agentul al cărui GUI îl reprezintă. AgentGuiActivity extinde clasa Activity şi implementează AgentGui.

Toate componentele grafice ale interfeţei sunt declarate în fişiere .xml ce se găsesc în directorul res/layout. Acestea pot fi create şi în interiorul Activity-ului, dar am ales prima variantă, deoarece diferenţiază aspectul grafic de comportamentul şi controlul elementelor. Descrierea grafică din fişierele .xml este separată de restul codului, ceea ce oferă posibilitatea de a face modificările vizibile fără a recompila codul sursă. Tot odată Android oferă un tool de vizualizare a aspectului grafic definit în fișierul .xml.

Pentru fereastra principală a aplicaţiei am folosit un layout linear cu orientare verticală ce cuprinde două elemente de tipul *TextView*, două elemente *ScrollView* în interiorul cărora se află elemente de tipul *TextView*, şi 3 elemente de tipul *Button*. Cele trei butoane sunt incluse într-un sub-layout orientat orizontal.

Figura 13. Interfaţă grafică a agentului PDA

La compilare fiecare layout xml este transformat într-o resursă de tip view. Acestea se încarcă în metoda onCreate() din activity-ul curent prin metoda *setContentView*. Metoda onCreate este apelata de către framework-ul Android, prima oară cînd Activity-ul este creat şi afisat utilizatorului. Primul Activity ce este afişat pe ecran la pornirea aplicaţiei este specificat în fişierul manifest.xml.

```
<activity android:name=".ManagerActivity" android:label="@string/app_name" >
   <intent-filter>
      <action android:name="android.intent.action.MAIN" />
      <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

Fiecare componentă grafică definită în fişierul xml poate avea ca atribut un *id* unic prin care să fie identificată în cadrul view-ului. Acest id este definit în lista de atribute sub forma unui *string*, dar la compilare, în fişierul generat *R.java,* id-ul este referenţiat printr-un *integer*.

```
android:id="@+id/pro_opinions"
public static final int pro_opinions=0x7f060007;
```

Simbolul „@” indică parser-ului de XML că nu trebuie să expandeze restul cuvântului, iar simbolul „+” indică faptul că o nouă resursă trebuie creată şi salvată în fişierul *R.java*. Pentru a crea o instantă în Activity a obiectului grafic definit în xml se foloseşte funcţia *findViewById (R.id.id_componeta)*.

Android permite definirea parametrilor atributelor separat de fişierul în care este descrisă structura grafică, pentru o mai uşoară dezvoltare. Valorile text ce sunt afişate pe ecran au fost definite în fişierul res/values/string.xml printr-o asociere nume-valoare.

În momentul în care utilizatorul vrea să facă o operaţie şi foloseşte unul dintre cele trei butoane, o fereastră de tipul *AlertDialog* se deschide peste fereastra principală. Această funcţionalitate este realizată cu ajutorul metodei *AlertDialog.Builder(context),* ce creză în contextul actual o fereastră de dialog şi atribuie obiectul *view* corespunzător. Acest obiect este instanţiat prin intermediul unui obiect de tipul *LayoutInflater*.

```
LayoutInflater li = LayoutInflater.from(context);
View joinView = li.inflate(R.layout.join, null);
AlertDialog.Builder alertDialogBuilder = new AlertDialog.Builder(context);
alertDialogBuilder.setView(joinView);
```

Figura 14. Interfaţă grafică a agentului PDA – introducerea unei opinii

Toate componentele grafice ce au atribuite metode de input sau output trebuie să comunice cu agenţii pentru a face schimb de informaţii. Fiecare componentă ce trebuie să transmită informaţii agentului PDA are asociat un obiect de tipul *InputListener*. Aceasta reprezintă legatura dintre agent şi interfaţa sa grafică. *InputListener* este asociat interfeţei prin metoda *connectInput*, de către instanţa *ClaimBehavior.* Transmiterea de informaţii de la agent către interfaţă se realizează prin funcţia *doOutput,* asemănător, tot prin intermediul lui *ClaimBehavior.*

5.7 Interfața grafică a agenților de pe PC

Pachetul *visualization* defineşte o interfaţa grafică standard a unui agent PCDefaultAgentGui. Aceasta implementează AgentGui. AgentGui este interfaţa pe care ClaimBehavior o vede pentru toţi agenţii, astfel caracteristica unui agent de a fi plasat pe PC sau smartphone devine invizibilă pentru comportamentul agentului. Pentru a adăuga funcţionalităţi acestei interfeţe grafice, clasa GroupCoordonatorAgentGui extinde clasa PCDefaultAgentGui.

Partea de conexiune cu agentul se realizează asemănător cu cazul agenţilor de pe dispozitivele mobile, prin instanţa de ClaimBehavior. Acesta are o referinţă către gui-ul fiecărui agent. Obiectul ce defineşte gui-ul unui agent este setat de către VisualizabeAgent prin funcţiile setup() şi resetVisualization(), în momentul creerii agentului şi în momentul în care acesta migrează către alt dispozitiv.

În Figura 15 este prezentată interfaţa grafică a agentului Group Coordonator. Acesta are rolul de a afişa pe ecranul computerului gazdă toate opiniile participanţilor la dezbatere. Ecranul este împărţit în două componente de afişare, în funcţie de tipul opiniei: pro sau contra subiectului. Interfaţa cuprinde şi componeta de afişare a log-urilor stării agentului, moştenită din clasa PCDefaultAgentGui.

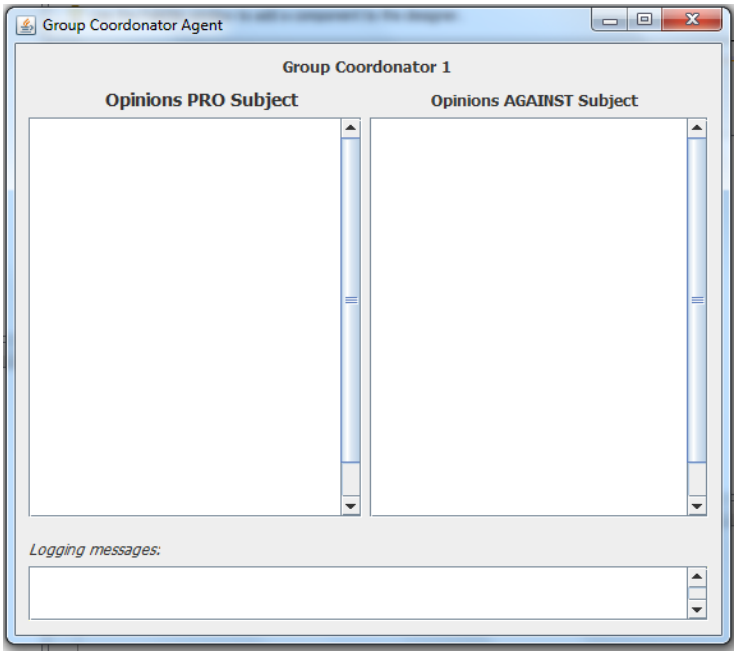

Figura 15. Interfata grafica a agentului Group Coordonator

Pentru realizarea interfeţei grafice am folosit elemente din pachetul javax.swing. Intreaga fereastră reprezintă un Jframe. Aceasta cuprinde trei elemente de tipul JscrollPane, în cadrul cărora se găseşte câte un element de tipul Jpanel. În cazul interfeţei pentru agentul PDA am introdus elemente de tipul Jbutton pentru cele trei butoane şi elemente de tipul JtextField pentru a introduce text de la tastatură.

În figura 16 este prezentată Interfaţa Grafică a agentului PDA în cazul în care utilizatorul nu dispune de un dispozitiv mobil şi doreşte să participe la discuţie folosind un PC. Aceasta oferă aceleaşi funcţionalitati ca şi în cazul interfeţei de la smartphone. Utilizatorul poate vedea pe ecran toate opiniile pro si contra subiectului dezbătut, se poate alătura unui grup de discuţii, îşi poate expune opinia în legătură cu subiectul discuţiei şi îşi poate retrage o opinie în cazul în care aceasta nu mai este validă.

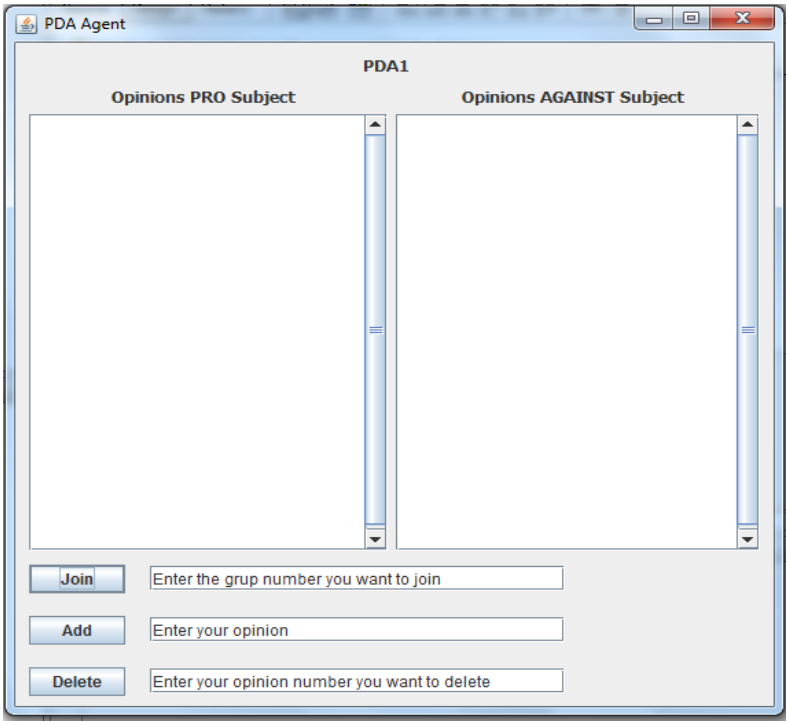

Figura 16. Interfața grafică a agentului PDA pe PC

Capitolul 6. Descrierea aplicaţiei şi instrucțiuni de utilizare

Înainte de a rula aplicaţia trebuie avute în vedere o serie de aspecte. Trebuie să existe conectivitate între computerele pe care se vor centraliza opiniile utilizatorilor şi dispozitivele cu ajutorul cărora se vor introduce aceste opinii în sistem. Trebuie ca participanţii la discuţie să cunoască adresa IP a maşinii gazdă pe care se găseşte platforma JADE cu container-ul principal. Înainte de a porni platforma tATAmI trebuie verificat fişierul BootSetings în care este specificat scenariul pe care platforma îl va simula (parametrul *scenarioFileName*). Trebuie rulat iniţial proiectul tATAmI-PC care porneşte platforma JADE şi construieşte Main Container-ul. În urma rulării se vor observa pe ecran interfeţele GUI a trei agenţi: RMA, simulator şi visualizer. Agentul RMA oferă instrumente de monitorizare şi controlare a platformei JADE şi a containere-lor, agentul simulator oferă funcţii de control asupra scenariului simulat, iar agentul visualizer oferă o jurnalizare centralizată a evenimentelor petrecute în sistem.

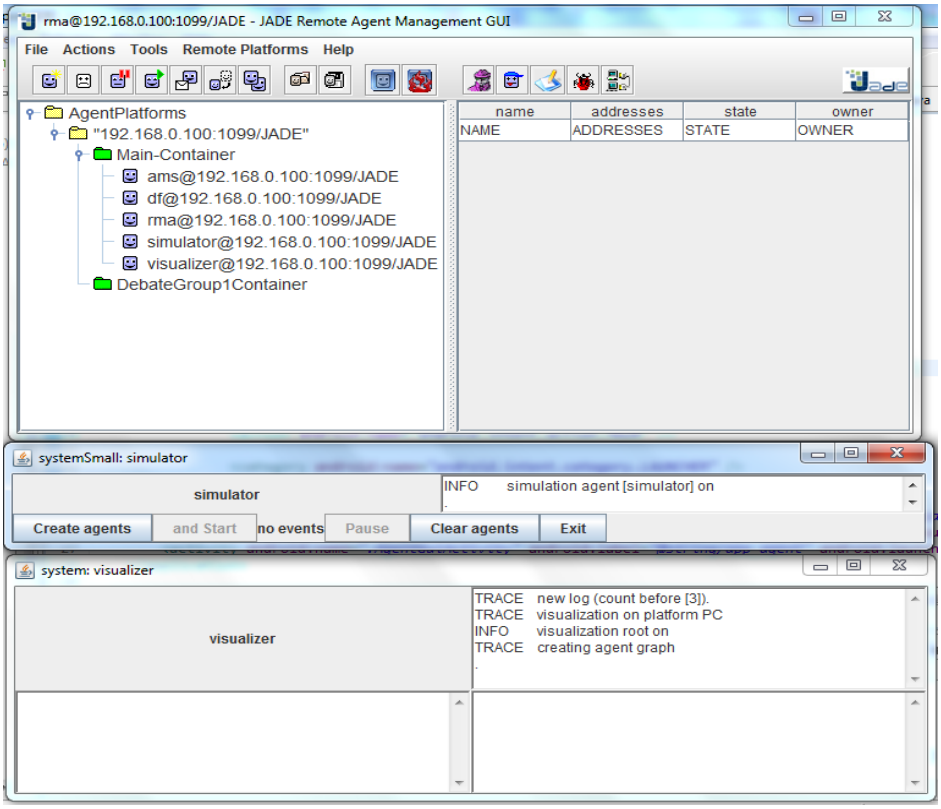

Figura 17. Interfețele grafice ale agenților RMA, simulator si visualizer

După ce platforma şi containerele au fost lansate, se poate rula proiectul/aplicaţia tATAmI-Android. Primul Activity afişat va cere utilizatorului să introducă adresa IP a platformei, portul pe care aceasta aşteaptă conexiuni şi un nickname pe care îl va folosi în cadrul dezbaterii. Utilizatorul are la dispoziţie două butoane: use (pentru a se conecta la platforma de discuţii) şi cancel (pentru a parasi aplicaţia). După ce toţi participanţii s-au conectat la platforma de discuţii, pentru a se crea agenţii, este necesară apasărea butonului Create agents oferit de agentul simulator.

Agenţii se vor naşte pe masină cu Main Container apoi vor migra în containerele aferente, corespunzător scenariului. Astfel agenţii GroupCoordonator se vor muta pe PC-urile ce gestionează grupurile de discuţii, iar agenţii PDA şi Emisarry vor migra pe dispozitivele mobile. Interfeţele grafice ale agenţilor se vor poziţiona automat pe ecran. Agenţii se vor putea observa în fereastra RMA în

containerele corespunzatoare.

Pe ecranul dispozitivelor mobile va apărea interfaţa prin care utilizatorul are acces la grupurile de discuţii. Aceasta pune la dispoziţia participantului trei butoane: join (prin care se poate înscrie într-un grup de discuţii; pentru acest lucru trebuie să introducă id-ul grupului), add (prin care îşi poate face publică o opinie pro sau contra, referitoare la subiectul dezbătut), delete (prin care îşi poate retrage o opinie, în cazul în care aceasta nu mai este validă, sau pur şi simplu participantul s-a răzgândit; pentru a retrage o opinie, este necesar să se introducă id-ul opiniei). Interfaţa va permite utilizatorului să vadă toate opiniile participanţilor la grupul curent de discuţii, organizate pe două ecrane, în funcţie de caracterul opiniei (pro sau contra).

Dacă un participant doreşte să părăsească grupul de discuţii şi să se alăture altui grup, va trebuii să folosească butonul *join*. Toate opiniile lui vor dispareă de pe ecranul PC-ului centralizator, dar şi de pe ecranele dispozitivelor mobile ale participanţilor şi se vor muta pe ecranele participanţilor din noul grup de discuţii. Totodată, pe ecranul dispozitivului său vor apărea toate opiniile din noul grup de discuţii, ce au fost enumerate anterior sosirii lui.

Pentru a instala aplicaţia Android pe dispozitivul mobil este nevoie de conectarea acestuia la computer. Pentru a rula aplicaţia pe emulatorul AVD, este necesar ca acesta sa fie pornit în prealabil, iar versiuna sa să fie minim API level 10 (Android 2.3.3) pentru care a fost concepută aplicaţia. Înainte de lansarea în execuţie trebuie să se verifice faptul că portul 1099 a fost redirecționat de la adresa 127.0.0.1 către adresa 10.0.2.2. Comenzile prin care acest lucru este posibil sunt descrise în Capitolul 5.5.

Capitolul 7. Concluzii

Principala contribuție pe care acest proiect o aduce este portarea platformei tATAmI pentru aplicații de Inteligență ambientală pe tehnologia Android. Această portare face posibilă dezvoltarea, testarea și simularea ulterioară în cadrul platformei tATAmI a scenariilor ce implică agenți software inteligenți găzduiți pe dispozitive mobile ce execută Android. Sistemul este acum compatibil cu versiuni de Andoid 2.3.3 sau mai recente.

Pentru a valida și testa funcționalitățile aduse, am implementat aplicația pentru discuții pro și contra. Această aplicație constă într-un scenariu ce implică existența a trei tipuri de agenți S-CLAIM. Comportamentul lor testează funcționalități ale platformei tATAmI, precum comunicarea între agenți, structurarea acestora în ierarhii, mobilitatea ierahică și mobilitatea între mașinile gazdă conectate prin rețea. Principala informație schimbată între agenți o reprezintă opiniile participanților la dezbatere. Cantitatea cea mai mare de date se schimbă între agenții emisari şi coordonatorul grupului de care aparțin. Caracteristica de mobilitate a agenților, mai exact faptul că agentul emisar își poate schimba părintele și se poate muta de pe smartphone pe computerul unde se află coordonatorul, contribuie la creșterea performanței aplicației.

În decursul dezvoltării proiectului am semnalat echipei de dezvoltare a platformei tATAmI o serie de mici neconcordanțe de implementare, bug-uri și totodată nevoia de îmbunătățire a unor funcționalități precum includerea în limbajul S-CLAIM a unor primitive de input și output prin care agenții să comunice cu utilizatorul. Am ajutat astfel la dezvoltarea întregului proiect tATAmI.

În viitor există multe perspective în care platforma poate fi imbunatățită. Un prim pas spre dezvoltarea ei îl constituie crearea de scenarii complexe și variate, prin care să fie puse în evidență toate caracteristicile unui agent. Acest lucru va determina cercetarea modului optim în care un astfel de scenariu trebuie simulat și testat. Totodata este necesară dezvoltarea comportamentului pro-activ al unui agent S-CLAIM și îmbogățirea limbajului S-CLAIM prin adăugarea și testarea de primitive. Pentru ușurința proiectării în întregime a unui agent, este util un mecanism de creare a interfeței grafice a unui agent, care să transparentizeze mediul în care agentul se execută. Mai concret, proiectantul să nu fie nevoit să țină cont de diferențele dintre dezvoltarea unei interfete grafice în mediul Android și cel Java.

B I B L I O G R A F I E

[1] Institutul Național de studii și cercetari pentru Comunicații http://www.inscc.ro/mediu-ambiental-inteligent/ ultima accesare 06.2012

[2] A. Olaru , *A Context-Aware Multi-Agent System for AmI Environments*, PhD Thesis, University "Politehnica" of Bucharest, Universite Pierre et Marie Curie (Paris 6), 2011

[3] Comunicat de presa Cisco: http://www.cisco.com/web/RO/news/2012/120531.html Bucureşti– 31 mai, 2012

[4] A. Leca, F. Leon - *Modelarea si analiza sistemelor multi-agent* Laborator http://florinleon.byethost24.com/lab_masma.htm

[5] G. Weiss *Multiagent Systems A Modern Approach to Distributed Artificial Intelligence*, 1999 Massachusetts Institute of Technology

[6] F. Bellifemine, G. Caire, A. Poggi, G. Rimassa, *JADE A White Paper* , Volume 3, September 2003

[7] Siteul official al FIPA (http://www.fipa.org/, ultima accesare: Iunie 2012)

[8] M. Wooldridge, *An introduction to multiagent systems*, John Wiley Ed., 2002. ISBN 0-471-49691-X.

[9] F. Bellifemine, *TILAB, JADE Tutorial for beginners*, The Hague, 12/10/04

[10] G. Caire, JADE Board Technical Leader, *JADE: the new kernel and last developments*, Pisa 2004

[11] M. Gargenta, *Learning Android*, Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

[12] A.Moreno, A. Valls, A. Viejo, *Using JADE-LEAP to implement agents in mobile devices*, Grup de Sistemes Multi-Agent (GruSMA) Departament d'Enginyeria Informàtica i Matemàtiques Escola Tècnica Superior d'Enginyeria (ETSE) Universitat Rovira i Virgili (URV)

[13] S. Davidson Pfalzer, *Hello, Android Introducing Google's Mobile Development Platform*, 3rd Edition, Ed Burnette, July 2010

[14] Android Developer http://developer.android.com , ultima accesare 07.2012

[15] G. Caire, G. Iavarone, M. Izzo, K. Heffner, *JADE Tutorual: JADE programming for Android*, last update: 15 November 2011. JADE 4.1.1

ANEXA 1

scenario.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<scen:scenario
    xmlns:pr="http://www.example.org/parameterSchema"
    xmlns:kb="http://www.example.org/kbSchema"
        xmlns:scen="http://www.example.org/scenarioSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.example.org/scenarioSchema
        ../../../config/scenarioSchema2.xsd ">

        <scen:jadeConfig isMain="true" />
        <scen:adfPath>scenario/examples/debateScenario-android</scen:adfPath>
        <scen:agentPackage>agent_packages.example.debate</scen:agentPackage>

        <scen:initial>
                <scen:container name="DebateGroup1Container">
                        <scen:agent>
                            <scen:parameter name="loader" value="adf2" />
                        <scen:parameter name="class" value="GroupCoordonatorAgent" />
                                <scen:parameter name="name" value="GroupCoordonator1" />
                                <scen:parameter name="java-code" value="DebateFunctions" />
                                <scen:parameter name="GUI" value="GroupCoordonatorAgentGui" />

                        </scen:agent>
                </scen:container>

                <scen:container name="DebateGroup2Container">
                        <scen:agent>
                            <scen:parameter name="loader" value="adf2" />
                                <scen:parameter name="class" value = "GroupCoordonatorAgent" />
                                <scen:parameter name="name" value="GroupCoordonator2" />
                                <scen:parameter name="java-code" value="DebateFunctions" />
                                <scen:parameter name="GUI"value = "GroupCoordonatorAgentGui" />
                        </scen:agent>
                </scen:container>

                <scen:container name="PDAContainer1" create="false">
                        <scen:agent>
                            <scen:parameter name="loader" value="adf2" />
                                <scen:parameter name="class" value="PDAAgent" />
                                <scen:parameter name="name" value="PDA1" />
                                <scen:parameter name="java-code" value="DebateFunctions" />
                        </scen:agent>
                        <scen:agent>
                            <scen:parameter name="loader" value="adf2" />
                                <scen:parameter name="class" value="EmissaryAgent" />
                                <scen:parameter name="name" value="Emissary1" />
                                <scen:parameter name="parent" value="PDA1" />
                                <scen:parameter name="java-code" value="DebateFunctions" />
                        </scen:agent>
                </scen:container>

                <scen:container name="PDAContainer2" create="false">
                        <scen:agent>
                            <scen:parameter name="loader" value="adf2" />
                                <scen:parameter name="class" value="PDAAgent" />
                                <scen:parameter name="name" value="PDA2" />
                                <scen:parameter name="java-code" value="DebateFunctions" />
                        </scen:agent>
                        <scen:agent>
                            <scen:parameter name="loader" value="adf2" />
                                <scen:parameter name="class" value="EmissaryAgent" />
                                <scen:parameter name="name" value="Emissary2" />
                                <scen:parameter name="parent" value="PDA2" />
                                <scen:parameter name="java-code" value="DebateFunctions" />
                        </scen:agent>
                </scen:container>

                <scen:container name="EmulatorContainer3" create="false">
                        <scen:agent>
                            <scen:parameter name="loader" value="adf2" />
                                <scen:parameter name="class" value="PDAAgent" />
                                <scen:parameter name="name" value="PDA3" />
                                <scen:parameter name="java-code" value="DebateFunctions" />
                        </scen:agent>
                        <scen:agent>
                            <scen:parameter name="loader" value="adf2" />
                                <scen:parameter name="class" value="EmissaryAgent" />
                                <scen:parameter name="name" value="Emissary3" />
```

```
                        <scen:parameter name="parent" value="PDA3" />
                        <scen:parameter name="java-code" value="DebateFunctions" />
                </scen:agent>
            </scen:container>

            <scen:container name="EmulatorContainer4" create="false">
                <scen:agent>
                    <scen:parameter name="loader" value="adf2" />
                        <scen:parameter name="class" value="PDAAgent" />
                        <scen:parameter name="name" value="PDA4" />
                        <scen:parameter name="java-code" value="DebateFunctions" />
                </scen:agent>
                <scen:agent>
                    <scen:parameter name="loader" value="adf2" />
                        <scen:parameter name="class" value="EmissaryAgent" />
                        <scen:parameter name="name" value="Emissary4" />
                        <scen:parameter name="parent" value="PDA4" />
                        <scen:parameter name="java-code" value="DebateFunctions" />
                </scen:agent>
            </scen:container>

    </scen:initial>
</scen:scenario>
```

ANEXA 2

PDAAgent.adf2, EmissaryAgent.adf2, GroupCoordonatorAgent.adf2

```
(agent PDAAgent
	(behavior

		(reactive register
			(receive register ?emissary)
			(addK (struct knowledge emissary ?emissary)))

		(reactive clearOutput
			(receive clear)
			(output clear))

		(reactive joinGroup
			(input join tip ?groupCoordonatorAgent)
			(forAllK (struct knowledge emissary ?emissary)
				(send ?emysarry (struct message leaveGroup))
				(send ?emysarry (struct message join ?groupCoordonatorAgent))))

		(reactive addOpinion
			(input add tip ?tag ?opinion)
			(forAllK (struct knowledge emissary ?emissary)
				(send ?emysarry (struct message add ?tag ?opinion))))

		(reactive deleteOpinion
			(input delete tip ?id)
			(forAllK (struct knowledge emissary ?emissary)
				(send ?emysarry (struct message delete ?id))))

		(reactive displayOpinion
			(receive displayOpinion ?opinion ?opinionTag)
			(if (equalString ?opinionTag pro)
				then (output proOpinion ?opinion))
			(if (equalString ?opinionTag con)
				then (output conOpinion ?opinion)))
	)
)

(agent EmissaryAgent ?parent ?name
	(behavior
		(initial startclock
			(addK (struct knowledge sequence 1))
			(send ?parent (struct message register ?name))
			(addK (struct knowledge masterPDAAgent ?parent)))

		(reactive joinGroup
			(receive join ?groupCoordonatorAgent)
			(in ?groupCoordonatorAgent)
			(send ?parent (struct message register ?name))
			(send this (struct message sendAllOpinions)))

		(reactive leaveGroup
			(receive leaveGroup)
			(send ?parent (struct message unregister ?name)))

		(reactive addOpinion
			(receive add ?opinionTag ?newOpinion)
			(removeK (struct knowledge sequence ?opinionId))
		       (addK (struct knowledge agentOpinion ?opinionId ?opinionTag ?newOpinion))
			(increment ?opinionId ?newOpinionId)
			(addK (struct knowledge sequence ?newOpinionId))
			(send ?parent (struct message opinion ?name ?opinionId ?
                                        opinionTag ?newOpinion)))

		(reactive deleteOpinion
			(receive delete ?opinionId)
			(removeK (struct knowledge agentOpinion ?opinionId ?tag ?opinion))
			(send ?parent (struct message refresh)))

		(reactive sendAllOpinions
			(receive sendAllOpinions)
			(forAllK (struct knowledge agentOpinion ?opinionId ?opinionTag ?
                                                                  opinionMsg)
				(send ?parent (struct message opinion ?name ?opinionId
                                                      ?opinionTag ?opinionMsg))))

		(reactive refresh
			(receive refresh)
			(readK (struct knowledge masterPDAAgent ?masterPDAAgent))
			(send ?masterPDAAgent (struct message clear)))
```

```
                (reactive displayOpinion
                        (receive displayOpinion ?opinion ?opinionTag)
                        (readK (struct knowledge masterPDAAgent ?masterPDAAgent))
                        (send ?masterPDAAgent (struct message displayOpinion ?opinion ?
                                                                                opinionTag)))
        )
)

(agent GroupCoordonatorAgent
        (behavior
                (reactive register
                        (receive register ?child)
                        (addK (struct knowledge children ?child)))

                (reactive unregister
                        (receive unregister ?child)
                        (removeK (struct knowledge children ?child))
                        (send this (struct message refresh)))

                (reactive refreshOpinions
                        (receive refresh)
                        (output clear)
                        (forAllK (struct knowledge children ?child)
                                (send ?child (struct message sendAllOpinions))
                                (send ?child (struct message refresh))))

                (reactive receiveOpinion
                        (receive opinion ?name ?opinionId ?opinionTag ?opinionMsg)
                        (assembleOutput ?name ?opinionId ?opinionMsg ?output)
                        (if (equalString ?opinionTag pro)
                                then (output proOpinion ?output))
                        (if (equalString ?opinionTag con)
                                then (output conOpinion ?output))
                        (forAllK (struct knowledge children ?child)
                                (send ?child (struct message displayOpinion ?output ?
                                                                        opinionTag))))
        )
)
```

ANEXA 3

AgentGuiActivity.java

```
package org.gui;

import java.util.HashMap;
import java.util.Hashtable;
import java.util.Map;
import java.util.Vector;

import core.interfaces.AgentGui;
import core.interfaces.AgentGui.InputListener;
import android.app.Activity;
import android.app.AlertDialog;
import android.content.Context;
import android.content.DialogInterface;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import android.view.View.OnClickListener;

public class AgentGuiActivity extends Activity implements AgentGui{

	private String nickname;
	public static AgentGui agentGuiContext;
	protected Map<String, View> components      = null;
	protected Map<String, InputListener> inputConnections = null;
	private final Context context = this;

	InputListener  joinListener   = null;
	InputListener  addListener    = null;
	InputListener  deleteListener = null;

	private final static String add_tag_pro = "pro";
	private final static String add_tag_con = "con";
	private final static String group_coord_agent_name= "GroupCoordonator";

	enum DebateGroupComponents {
		JOIN, ADD, DELETE, CLEAR, PROOPINION, CONOPINION;
	}

	@Override
	public void onCreate(Bundle savedInstanceState) {

		components = new Hashtable<String, View>();
		inputConnections = new HashMap<String, AgentGui.InputListener>();
		super.onCreate(savedInstanceState);

		Bundle extras = getIntent().getExtras();
		if (extras != null) {
			nickname = extras.getString("nickname");
		}

		setContentView(R.layout.debate);

		Button button_join = (Button) findViewById(R.id.button_join);
		Button button_add = (Button) findViewById(R.id.button_add);
		Button button_delete = (Button) findViewById(R.id.button_delete);
		TextView pro_output = (TextView) findViewById(R.id.pro_opinions);
		TextView con_output = (TextView) findViewById(R.id.con_opinions);

		pro_output.setText("You must join a debate group.");
		con_output.setText("You must join a debate group.");

		components.put(DebateGroupComponents.JOIN.toString(), button_join);
		components.put(DebateGroupComponents.ADD.toString(), button_add);
		components.put(DebateGroupComponents.DELETE.toString(), button_delete);
		components.put(DebateGroupComponents.PROOPINION.toString(), pro_output);
		components.put(DebateGroupComponents.CONOPINION.toString(), con_output);

		button_join.setOnClickListener(buttonJoinListener);
		button_add.setOnClickListener(buttonAddListener);
		button_delete.setOnClickListener(buttonDeleteListener);

		Register.addAgent(this);
		agentGuiContext = this;

	}
```

```
	public  static AgentGui getContext(){

		return agentGuiContext;
	}

	@Override
	protected void onDestroy() {
		super.onDestroy();
	}

	private OnClickListener buttonJoinListener = new OnClickListener() {
		public void onClick(View v) {

			LayoutInflater li = LayoutInflater.from(context);
			View joinView = li.inflate(R.layout.join, null);

			AlertDialog.Builder alertDialogBuilder = new
AlertDialog.Builder(context);
			alertDialogBuilder.setView(joinView);

			final EditText userInput =
			(EditText) joinView.findViewById(R.id.editTextDialogUserInput);

			// set dialog message
			alertDialogBuilder
				.setCancelable(false)
				.setPositiveButton("Join", new DialogInterface.OnClickListener()
{
				    public void onClick(DialogInterface dialog,int id) {

				        if(joinListener != null){

							Vector<Object> args = new
Vector<Object>(1);
							args.add(group_coord_agent_name +
userInput.getText().toString());

	joinListener.receiveInput(DebateGroupComponents.JOIN.toString().toLowerCase(), args);

	((EditText)(components.get(DebateGroupComponents.JOIN.toString()))).setText("");
				        }
					else
							System.out.println("nobody to receive the
input");
							// FIXME else, a log should pick up an error
				    }
				})
				.setNegativeButton("Cancel", new
DialogInterface.OnClickListener() {
				    public void onClick(DialogInterface dialog,int id) {
				        dialog.cancel();
				    }
				});

			// create alert dialog
			AlertDialog alertDialog = alertDialogBuilder.create();
			alertDialog.show();
		}
	};

	private OnClickListener buttonAddListener = new OnClickListener() {
		public void onClick(View v) {

			LayoutInflater li = LayoutInflater.from(context);
			View joinView = li.inflate(R.layout.add, null);

			AlertDialog.Builder alertDialogBuilder = new
AlertDialog.Builder(context);
			alertDialogBuilder.setView(joinView);

			final EditText userInput =
			(EditText) joinView.findViewById(R.id.editTextDialogUserInput);

			// set dialog message
			alertDialogBuilder
				.setCancelable(false)
				.setPositiveButton("Add Pro", new
DialogInterface.OnClickListener() {
				    public void onClick(DialogInterface dialog,int id) {

				        if(addListener != null){

							Vector<Object> args = new
Vector<Object>(2);
							args.add(add_tag_pro);
							args.add(userInput.getText().toString());
```

```
        addListener.receiveInput(DebateGroupComponents.ADD.toString().toLowerCase(), args);

        ((EditText)(components.get(DebateGroupComponents.ADD.toString()))).setText("");
                                        }
                                                else
                                                        System.out.println("nobody to receive the
input");
                                                // FIXME else, a log should pick up an error
                                        }
                                })
                                .setNeutralButton("Add Con", new
DialogInterface.OnClickListener() {
                                        public void onClick(DialogInterface dialog,int id) {

                                            if(addListener != null){

                                                        Vector<Object> args = new
Vector<Object>(2);
                                                        args.add(add_tag_con);
                                                        args.add(userInput.getText().toString());

        addListener.receiveInput(DebateGroupComponents.ADD.toString().toLowerCase(), args);

        ((EditText)(components.get(DebateGroupComponents.ADD.toString()))).setText("");
                                        }
                                                else
                                                        System.out.println("nobody to receive the
input");
                                                // FIXME else, a log should pick up an error
                                        }
                                })
                                .setNegativeButton("Cancel", new
DialogInterface.OnClickListener() {
                                        public void onClick(DialogInterface dialog,int id) {
                                            dialog.cancel();
                                        }
                                });

                        // create alert dialog
                        AlertDialog alertDialog = alertDialogBuilder.create();
                        alertDialog.show();
                }
        };

        private OnClickListener buttonDeleteListener = new OnClickListener() {
                public void onClick(View v) {

                        LayoutInflater li = LayoutInflater.from(context);
                        View joinView = li.inflate(R.layout.delete, null);

                        AlertDialog.Builder alertDialogBuilder = new
AlertDialog.Builder(context);
                        alertDialogBuilder.setView(joinView);

                        final EditText userInput = (EditText)
joinView.findViewById(R.id.editTextDialogUserInput);

                        // set dialog message
                        alertDialogBuilder
                                .setCancelable(false)
                                .setPositiveButton("Delete", new
DialogInterface.OnClickListener() {
                                        public void onClick(DialogInterface dialog,int id) {

                                            if(deleteListener != null){

                                                        Vector<Object> args = new
Vector<Object>(1);
                                                        args.add(userInput.getText().toString());

        deleteListener.receiveInput(DebateGroupComponents.DELETE.toString().toLowerCase(),
args);

        ((EditText)(components.get(DebateGroupComponents.DELETE.toString()))).setText("");
                                        }
                                                else
                                                        System.out.println("nobody to receive the
input");
                                                // FIXME else, a log should pick up an error
                                        }
                                })
                                .setNegativeButton("Cancel", new
DialogInterface.OnClickListener() {
                                        public void onClick(DialogInterface dialog,int id) {
                                            dialog.cancel();
                                        }
                                });
```

```
                        // create alert dialog
                        AlertDialog alertDialog = alertDialogBuilder.create();
                        alertDialog.show();
                }
        };

        @Override
        public void doOutput(String compName, Vector<Object> args) {

                final String componentName = compName;
                final Vector<Object> arguments = args;
                System.out.println("in DoOutput " +componentName +" "+arguments);

                this.runOnUiThread(
                  new Runnable(){
                      public void run(){;


        if(componentName.compareToIgnoreCase(DebateGroupComponents.CLEAR.toString()) == 0){

                                                        View component;
                                                        component=
components.get(DebateGroupComponents.PROOPINION.toString());
                                                        ((TextView)component).setText("");
                                                        component=
components.get(DebateGroupComponents.CONOPINION.toString());
                                                        ((TextView)component).setText("");
                                                }
                                                else
if((componentName.compareToIgnoreCase(DebateGroupComponents.PROOPINION.toString()) == 0) ||

        (componentName.compareToIgnoreCase(DebateGroupComponents.CONOPINION.toString()) ==
0)){

                                                        if(!components.containsKey(componentName))
                                                        {
                                                                System.err.println("component [" +
componentName + "] not found."); // FIXME: get a log from somewhere
                                                                return;
                                                        }

                                                        View component =
components.get(componentName);
                                                        if(component instanceof TextView)
                                                        {
                                                                if(arguments.size() > 0)
                                                                {
                                                                        TextView ta =
(TextView)component;

        ta.append((String)arguments.get(0));

        ta.append(System.getProperty("line.separator"));
                                                                }
                                                        }
                                                }

                    }
                 });
        }

        @Override
        public void connectInput(String componentName, InputListener listener) {

                System.out.println("in conectInput "+componentName);

                if(componentName.equals(DebateGroupComponents.JOIN.toString().toLowerCase()))
                {
                        joinListener = listener;
                        System.out.println("...done");
                }
                else
if(componentName.equals(DebateGroupComponents.ADD.toString().toLowerCase()))
                {
                        addListener = listener;
                        System.out.println("...done");
                }
                else
if(componentName.equals(DebateGroupComponents.DELETE.toString().toLowerCase()))
                {
                        deleteListener = listener;
                        System.out.println("...done");
                }
                else
                {
                        if(!components.containsKey(componentName))
```

```
                        {
                                System.err.println("component [" + componentName + "] not
found."); // FIXME: get a log from somewhere
                                return;
                        }
                        inputConnections.put(componentName, listener);
                }
        }

        @Override
        public void close() {
                finish();

        }

        @Override
        public Vector<Object> getinput(String componentName) {
                // TODO Auto-generated method stub
                return null;
        }
}
```

ANEXA 4

debate.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="vertical"
        android:layout_height="fill_parent"
        android:layout_width="fill_parent"
        android:weightSum="2">

        <TextView
        android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:text="@string/pro_title"
                android:textStyle="bold"
                android:textSize="16sp"
                android:textColor="#F8F8F8"/>

        <ScrollView
                android:layout_width="fill_parent"
                android:layout_height="0px"
                android:fillViewport="true"
                android:layout_weight="1"
                android:fadeScrollbars="false">

                <TextView
                        android:id="@+id/pro_opinions"
                        android:layout_width="fill_parent"
                        android:layout_height="wrap_content"
                        android:background="#F8F8F8"
                        android:paddingLeft="10dip"
                        android:paddingTop="3dip"
                        android:paddingRight="10dip"
                        android:paddingBottom="10dip"
                        android:textStyle="bold"
                        android:textColor="#006600"
                        android:text="@string/hello"/>
        </ScrollView>


        <TextView
                android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/con_title"
        android:textStyle="bold"
                android:textSize="16sp"
                android:textColor="#F8F8F8"/>

        <ScrollView
        android:layout_width="fill_parent"
                android:layout_height="0px"
        android:fillViewport="true"
        android:layout_weight="1"
                android:fadeScrollbars="false">

                <TextView
                        android:id="@+id/con_opinions"
                        android:layout_width="fill_parent"
                        android:layout_height="wrap_content"
                        android:background="#F8F8F8"
                        android:paddingLeft="10dip"
                        android:paddingTop="3dip"
                        android:paddingRight="10dip"
                        android:paddingBottom="10dip"
                        android:textStyle="bold"
                        android:textColor="#0000FF"
                        android:text="@string/hello"/>
        </ScrollView>

        <TextView
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:text="@string/f_title"
                android:textStyle="bold"
                android:textSize="16sp"
                android:textColor="#F8F8F8"/>

        <LinearLayout
                xmlns:android="http://schemas.android.com/apk/res/android"
                android:orientation="horizontal"
                android:layout_height="wrap_content"
```

```
                android:layout_width="fill_parent"
                android:weightSum="3">

                <Button
                        android:text="@string/button_join"
                        android:id="@+id/button_join"
                        android:layout_width="0px"
                        android:layout_height="wrap_content"
                        android:layout_weight="1"
                        android:textStyle="bold"
                        android:textSize="16sp"
                        android:textColor="#000000"
                        android:paddingTop="7dip"
                        android:paddingBottom="7dip">
                </Button>

                <Button
                        android:text="@string/button_add"
                        android:id="@+id/button_add"
                        android:layout_width="0px"
                        android:layout_height="wrap_content"
                        android:layout_weight="1"
                        android:textStyle="bold"
                        android:textSize="16sp"
                        android:textColor="#000000"
                        android:paddingTop="7dip"
                        android:paddingBottom="7dip">
                </Button>

                <Button
                        android:text="@string/button_delete"
                        android:id="@+id/button_delete"
                        android:layout_width="0px"
                        android:layout_height="wrap_content"
                        android:layout_weight="1"
                        android:textStyle="bold"
                        android:textSize="16sp"
                        android:textColor="#000000"
                        android:paddingTop="7dip"
                        android:paddingBottom="7dip">
                </Button>

        </LinearLayout>
</LinearLayout>
```