UNIVERSITATEA POLITEHNICA BUCUREŞTI
FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

PROIECT DE DIPLOMĂ

Platforma Android pentru discuții pro și contra.
Portarea proiectului pe sistemul de operare Android.

Coordonatori științifici:

Prof. dr. ing. Adina Magda Florea
As. dr. ing. Andrei Olaru

Absolvent:

Iulia Moscalenco

BUCUREŞTI
2012

Abstract

Inteligența Ambientală (AmI) este viziunea unui mediu electronic omniprezent în care tehnologia este integrată în obiectele de zi cu zi, cu scopul de a face interacțiunea utilizatorilor cu mediul înconjurător mai simplă și mai intuitivă. Scopul studiului care a stat la baza elaborării acestei lucrări este de a extinde funcționalitatea proiectului tATAmI prin implementarea unor componente care permit agenților software să-și continue existența pe un dispozitiv cu sistem de operare Android. tATAmI oferă dezvoltarea sistemelor multi-agent dependente de context pentru Inteligența Ambientală, care, datorită contribuției aduse, permite execuția agenților pe dispozitivele mobile.

CUPRINS

Listă Figuri:

Notații și abrevieri

AmI – Inteligența Ambientală
SMA – Sisteme Multi-Agent
FIPA – Foundation for Inteligent Phisical Agents
JADE – Java Agent Development Framework
ACL – Agent Communication Language
CLAIM – Computational Language for Autonomous,Intelligent and Mobile agents
S-CLAIM – Smart Computational Language for Autonomous,Intelligent and Mobile agents
AM – Agent Mobil
C/S – Client Server
POA– Programarea Orientată Agent

CAPITOLUL 1. Introducere

1.1 Contextul proiectului

Dezvoltarea rapidă a calculatoarelor din ultimii ani a permis cercetătorilor în domeniu să rezolve probleme dificile, apropiate de complexitatea problemelor soluționate de om. La ora actuală oamenii sunt înconjurați de tehnologia care încearcă sa crească calitatea vieții lor de zi cu zi[1]. Totuși, există situații în care tehnologia este greu de manevrat sau oamenii au o dificultate de a o folosi. Acestea au dus la apariția unui nou domeniu al științei calculatoarelor: Inteligența Artificiala. Un sistem artificial poate modela fin intuiția noastră despre comportamentul inteligent, conducând la o noua perspectivă asupra reprezentării cunoștințelor și rezolvării problemelor. Agenții software folosesc aceste elemente de Inteligență Artificială pentru așa îndeplini scopurile.

1.2 Scopuri și obiective

Scopul studiului care a stat la baza elaborării acestei lucrări este de a extinde funcționalitatea sistemului multi-agent tATAmI prin implementarea unor componente care permit agenților software să-și continue existența pe un dispozitiv cu sistem de operare Android. Această lucrare își propune să abstractizeze viziunea asupra unui PDA ca un mediu software pentru dezvoltarea agenților.

Pentru realizarea proiectului au fost definite câteva obiective de bază:

- extinderea funcționalității unui model de sistem multi-agent pentru inteligenta ambientală care dispune de dependință de context;
- dezvoltarea unui scenariu care să cuprindă agenți pe PC și pe sistemul de operare Android;
- utilizarea unor instrumente de evaluare, vizualizare și logarea ale comportamentelor agenților;
- utilizarea limbajului S-CLAIM;

1.3 Structura lucrării

Lucrarea de față a fost structurată pe șapte capitole și se încheie cu o bibliografie.

Capitolul 1 este dedicat introducerii, prezentând pe scurt contextul lucrării împreună cu obiectivele care au stat la baza elaborării acestuia. În final, este prezentat pe scurt fiecare capitol în parte.

În capitolul 2 am prezentat **Aspectele teoretice** și anume: Agentul ca elementul de bază al acestei lucrări, Inteligența Ambientală(AmI) ca o viziune asupra tehnologiei, prezentă ori de câte ori avem nevoie de ea, și, nu în ultimul rând, limbajul S-CLAIM care permite agenților software o structurare ierarhică[2]. Tot aici expun o dezbatere între Client-Server și Agenții Mobili. Dezvoltarea conceptului de agent software are șanse și conduce la nașterea unei noi direcții de programare și anume programarea orientate agent, cu un impact similar programării orientate obiect, dar la un nivel superior. Într-o astfel de paradigmă, obiectele vor fi înlocuite cu agenți, entități dotate cu capacități de percepție, comunicare raționament și decizie. Astfel, un sistem multi-agent poate avea la bază un scenariu care descrie comportamentul agenților(Secțiunile 2.1 și 2.2)

În construirea unei platforma de comunicare între agenți, un aspect important îl reprezintă și **Tehnologiile de programare folosite** (Capitolul 3). Având în vedere elementele menționate în capitolul anterior, alegerea unor tehnologii de programare depind de scopul utilizării acestora. Unul din aceste tehnologii îl reprezintă framework-ul JADE (Java Agent Development Framework), implementat în limbajul de programare Java. JADE este un framework pentru dezvoltarea sistemelor multi-agent. Scopul său este de a simplifica dezvoltarea, asigurând în același timp respectarea standardelor printr-un set corespunzător de servicii de sistem. Se va menționa și despre sistemul de operare Android. Acesta este implementat în Java și este open source, disponibil pentru oricine dorește să-l utilizeze.

Arhitectura proiectului va fi relatată în capitolul următor: **Arhitectura și componentele sistemului.** Se aduc la cunoștință cele doua proiecte implicate: tATAmI-PC și : tATAmI-Android. tATAmI-PC este un model de sistem multi-agent implementat în Java care folosește limbajul orientat agent S-CLAIM (o versiune simplificată și îmbunătățită a limbajului CLAIM), dar și instrumente pentru centralizarea vizualizării și urmăririi sistemului de agenți. Tot aici se va exemplifica structura și comportamentul agentului software. Sistemul multi-agent se va baza pe un scenariu detaliat în acest capitol.

Capitolul următor **Platforma pentru discuții pro și contra** este dedicat prezentării conținutului proiectului. Aici se specifică problemele întâlnite împreună cu soluțiile acestora. Se prezintă detalii de implementare a unor elemente importante din cadrul acestui proiect.

Un scurt manual de utilizare a proiectului se găsește în capitolul 6: **Instalarea și configurarea aplicației.** Obiectivul acestui ghid de utilizare este de a furniza instrucțiuni referitoare la modul de instalare și utilizare a proiectului.

În ultimul capitol **Concluzii** s-a subliniat aspectele principale discutate în această lucrare, reliefând cu această ocazie cele mai importante idei la care s-a ajuns în urma elaborării proiectului.

CAPITOLUL 2. Aspecte teoretice

2.1 Agenți Prezentare generală

Ce sunt agenții, de ce se numesc ei așa și de ce a fost nevoie introducerea/folosirea acestui termen? Noțiunea de agent poate avea diverse interpretări în funcție de domeniul particular în care este utilizat. Există agenți în științele cognitive, economice, sociale, în biologie și, recent, în știința calculatoarelor. Din punct de vedere software, un agent este un program care se comportă analog cu un agent uman, de exemplu un agent al unei agenții de voiaj sau un agent de asigurări. În esență, acest termen se referă la un program care este capabil să comunice în mod inteligent cu alți agenți software sau persoane.

O descriere relevantă a noțiunii de agent poate fi dată prin enumerarea caracteristicilor de bază pe care acesta trebuie să le dețină:

- *Autonom*: capacitatea de a acționa autonom într-o anumită măsură, de exemplu monitorizarea evenimentelor și a schimbărilor în mediul lor.
- *Reactiv*: abilitatea de a reacționa și de a evalua evenimentele venite din exterior adaptându-și comportamentul și luând decizii adecvate pentru a îndeplini sarcini care îi ajuta să-și atingă obiectivele.
- *Comunicare și Co-operare*: abilitatea de a se comporta social, de a interacționa și de a comunica cu alți agenți (într-un sistem Multi-Agent (MAS)), adică schimbul de informații, de a primi instrucțiuni și de a răspunde atunci când aceștia îi ajută să să-și îndeplinească obiectivele proprii.
- *Negocierea*: capacitatea de a efectua convorbiri organizate pentru a atinge un grad de cooperare cu alți agenți
- *Învățarea*: capacitatea de a-și îmbunătăți performanța în timp, atunci când interacționează cu mediul în care sunt încorporați
- *Adaptabilitate*: posibilitatea de a învăța și de a-și mări succesul datorită experienței
- *Personalitate*: capacitatea de a manifesta atributele unui caracter “credibil”, cum ar fi emoțiile
- *Mobilitate*: posibilitatea de a migra într-un mod auto-dirijat de la o platforma gazdă la alta.

Pentru anumiți cercetători, în special cei din domeniul inteligenței artificiale, termenul de agent conține, mai ales, caracteristici cum ar fi: noțiuni mentale (păreri, dorințe, obligații, opțiuni), raționalitate, adaptabilitate și învățare. Totuși elementul de bază al unui agent rămâne inteligenta[3]. În lucrarea de față agenții nu vor îngloba toate aceste caracteristici, bazându-se doar pe o parte din ele.

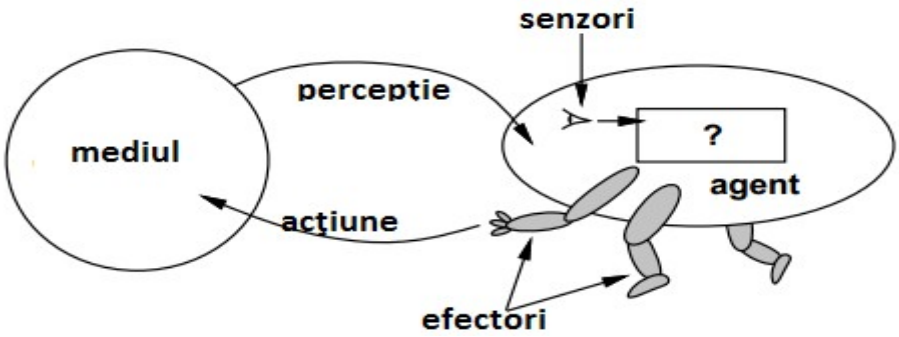

Fig.2.1 Interacțiunea agentului cu mediu.[4]

Dezvoltarea unui agent presupune găsirea unui comportament care va maximiza succesul acestuia. Depinde foarte mult ce își dorește programatorul să facă. Inteligența sa are ca consecință obținerea rezultatului dorit. De multe ori scopul depinde de timpul acordat acestuia. Nu întotdeauna soluțiile se pot găsi folosind metoda greedy, adică urmărirea subsoluţiilor maxime locale. Astfel timpul devine un element necesar pentru un agent.

În imaginea de mai sus este prezentat modul cum un agent software interacționează cu mediul înconjurător. Mediul poate fi unul nesigur, impunându-le agenților software să prezinte grad de dificultate. În aceste condiții agenții sunt obligați să dispună de strategii sau de experiențe rezultate în urma învățării. Proiectul tATAmI+Android nu va prezenta un astfel de mediu, dificultatea acestuia va reveni procesului de migrare.

2.1.1 Tipuri de probleme soluționate de către agenți

Agenții software inteligenți au o variațiune de utilizări. Din moment ce acești agenți software efectuează sarcini specificate de către utilizator, aceste programe sunt numite de obicei “bots” (roboți), o denumire care îi identifică rapid ca agenți inteligenți on-line, mai degrabă roboți în funcțiune Conform [5] agenții software inteligenți sunt utilizați în următoarele funcțiuni:

- ***Aplicații personale***
  - Agenți pentru căutare de informații (numiți și “infobots”): cei slabi sunt de genul “du-te adu”,cei mai puternici sunt capabili de a învăța
    - Software de căutare – “biblioteci electronice” care caută informații și le prioritizează
    - Instrumente independente de căutare – software pentru “detectare de date”
    - Instrumente pentru identificarea semnificației datelor
  - “Shopping bots”
    - Cumpărături generale on-line (exemplu : pentru căutarea lucrurilor noi)
    - Cumpărături bazate pe căutări (identificarea nevoilor utilizatorului și căutarea elementelor potrivite)

- Asistent personal
  - Adaptarea căutărilor în funcție de preferințele utilizatorului
  - Păstrarea calendarului personal
  - Notificarea utilizatorului sau modificarea intereselor

- ***Aplicații Business***
  - Vânzări și marketing
    - Direcționarea clienților pe piață
    - Oferte de vânzări ale catalogului
    - Ajutarea clienților (furnizarea de informații despre bunuri și servicii)
  - Banking
    - Servicii de creditare pentru clienți(căutarea celei mai bune rate de dobândă)
    - Transfer de bani electronic(automatizează protocoale)
    - Tendințele industriei de urmărire
    - Aplicații de procesare (credite)
  - Telecomunicații
    - "Smartphones" (operarea sau coordonarea telefoanelor mobile)
    - Interfața om-calculator (poate folosi limbajul natural)
    - Monitorizare, rutare (anticiparea sau maximizarea capacității de utilizare)
  - Sisteme expert (aplicații specializate pe cunoaștere)
    - De uz medical(procesarea diagnosticării, regimuri de tratament, înregistrarea pacienților)
    - Aplicarea legii (identificarea suspectului, analiza cazurilor, informații de urmărire)
    - Servicii de traducere (de conversie a textului scris într-o alta limbă)
    - Servicii de "troubleshooting"(solicitări de reparații, ajută serviciile de birou)
  - Managementul sistemului de rețea
    - Diagnosticarea calculelor, configurare, întreținere, reparații
    - Sisteme de diagnosticare a rețelei
    - Program de instalare în rețea
    - Mentenanța rețelei
    - Filtrare și sortarea de e-mail
    - Monitorizarea și protecția programelor

Aplicațiile bazate pe Agenți includ domeniile de probleme care necesită inteligenta umană pentru operarea în mod autonom. O listă mică de cerințe care pot fi solicitate de către astfel de aplicații care cuprinde una dintre cele mai dificile domenii de cercetare urmărite în Inteligența Artificiala (AI), până în prezent :

- Logica de interferențială și deducție
- Domeniul contextual de cunoaștere
- Pattern Recognition.
- Învățare și adaptabilitate.

O data ce capabilitățile inteligenței umane vor fi automatizate, funcționarea autonomă va permite aplicațiilor bazate pe agenți software de a analiza volume mari de date pe care oamenii nu sunt în stare să le proceseze. În plus, agenții software pot analiza, evalua, planifica și reacționa la evenimente venite din mediului cu o super-viteză.

2.1.2 Diferența între Client-Server și Agenții Mobili

În secțiunea anterioară am prezentat o bună parte a utilizării agenților software . Se poate observa ca un agent software nu este un program care rulează stand-alone. Colectarea de informații, monitorizarea rețelei sunt doar câteva funcțiuni care necesita mobilitate. După cum s-a precizat și la începutul acestui capitol, agenții software pot dispune de mobilitate. Pentru proiectul tATAmI-Android, această caracteristică este destul de importantă și prezintă un element de bază al acestei cercetări.

Un agent mobil(AM) dispune de doua componente: codul programului și starea de execuție a programului. Inițial, un AM se află pe un calculator numit mașină de bază după care este expediat pentru execuție pe un calculator la distanță numit gazdă sau platforma AM. Atunci când un AM este expediat, întregul cod al Agentului și starea acestuia sunt transferate pe mașină gazdă. Comunicarea între AM pe gazde diferite este destul de dificilă. Un AM migrează dea lungul funcționalității sale de la o gazdă la alta, datorită căruia este dificil să se stabilească unde se afla la un moment dat un anumit agent software.

Gazda trebuie să îi agenților software un mediu adecvat pentru a se putea executa. Cât timp informațiile necesare despre starea agentului software sunt transferate la mașină de bază, acesta poate să își reia execuția codului de la momentul în care s-a oprit la gazda anterioară, nu neapărat să ia de la început. Acest lucru continuă până când agentul ajunge la mașină de bază după terminarea execuției tascurilor. În continuare se va detalia un scenariu de bază specific fiecărui AM.

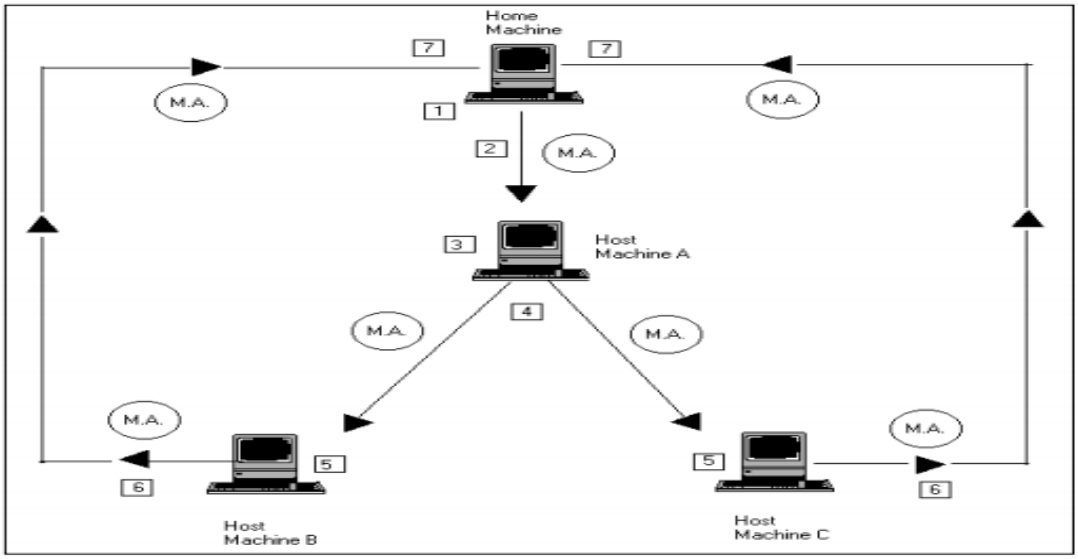

Fig.2.2 Ciclul de viață a unui Agent Mobil[15].

Ciclul de viată a unui AM conform figurii 2:

1. Agent mobil este creat pe mașină de bază.
2. AM este trimis la mașină gazdă A pentru execuție
3. Execuția agentului pe mașină gazdă A.
4. După execuție agentul este clonat pentru a crea două copii. O copie este expediată pe mașină gazdă B și cealaltă este trimisă pe mașină gazdă C.
5. Copiile clonate se execută pe gazdele respective.
6. După execuție, mașinile gazdă B și C, trimit AM înapoi la mașină de bază.
7. Mașina de bază retrage AM împreună cu datele lor și le analizează. Agenții sunt eliminați

Din acestea se observă că un AM trece prin următoarele evenimente în decursul vieții sale:

- **Crearea** : un agent se naște și se inițializează starea sa.
- **Expedierea** : un agent călătorește
- **Clonarea** : se nasc doi agenți, iar starea curentă a agentului original se se dublează și se asignează fiecărui agent în parte.
- **Dezactivarea** : un agent își oprește execuția și este pus în așteptare, se salvează starea curentă.
- **Activarea** : se trezește agentul dezactivat reluându-și starea salvată.
- **Retragerea** : un agent este expediat la mașină de bază după finalizarea execuției sale.
- **Eliminarea** : un agent își termină execuția și starea lui este pierdută.

Agenții sunt, de obicei, conduși de scopuri și obiective, prezentând “domenii” de cunoaștere De multe ori, agenții diferă mult unul de altul prin baza de cunoștințe de care dispun și rolul lor. Aceste două caracteristici diferențiază modelul Agenților Mobili de modelul Client-Server (C/S).

AM și C/S sunt folosite pentru dezvoltarea aplicațiilor distribuite. În continuare se vor prezenta diferențele de bază între cele două modele.

În paradigma C/S proprietarii de resurse (Serverele) sunt fizic depărtate de clienții lor (Utilizatorii). Comunicarea între cele două părți are loc printr-o rețea de calculatoare, având la mijloc următoarele mecanisme: apelul de procedură la distantă (RPC), schimb de mesaje, socketi, etc. În această paradigmă fiabilitatea legăturii de comunicare și sincronizarea apelurilor la distantă sunt cerințe importante. AM, prin mutarea de cod mai aproape de date, reduce latența tascurilor individuale, evitând transmiterea datelor intermediare prin rețea, continuându-și execuția chiar și în absența conexiunii la rețea și terminând mult mai repede în comparație cu soluția tradițională C/S[6]. Utilizarea unui sistem descentralizat folosind AM distribuie traficul de date în rețeaua locală, relaxând serverul central. Mai mult decât atât, interacțiunea între agent și resursă (după mutare) se efectuează pe aceeași mașină fără transmitere de mesaje peste rețea, astfel paradigma AM este indicată pentru anumite tipuri de aplicații distribuite în timp real. În figura 2.3 este prezentată această diferență.

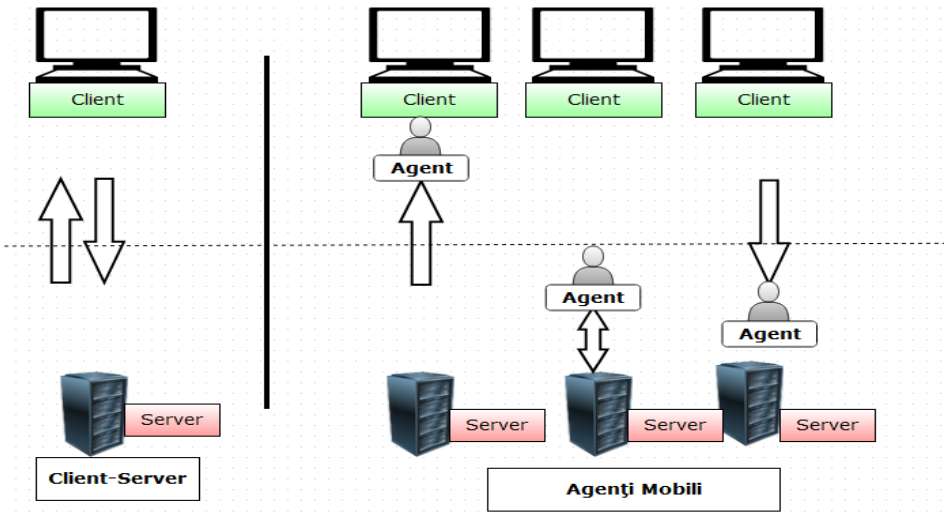

Fig. 2.3 Sincronizarea apelurilor la distantă în cele două paradigme.

O altă caracteristică care produce diferențe între cele două paradigme este gradul de dificultate al aplicației Procesul de mutare a agentului trebuie să fie eficient, pentru a compensa acel transfer, pe când în C/S nu se pune această problemă. În conformitate cu cerințele de dezvoltare a aplicației, este nevoie ca agentul să fie mic, permițând transferul său rapid peste rețea. Agentul poate, de asemenea, să fie capabil de a se executa pe o mașină cu restricții de memorie și prelucrare, de exemplu computerele mobile și calculatoarele portabile.

Principalul dezavantaj al AM este riscul de securitate implicat în utilizarea acestora. În primul rând, un AM de malware poate deteriora gazda, de exemplu un virus poate fi deghizat ca un AM cauzând pagube pe mașini gazdă. Pe de altă parte, o gazdă rău intenționată poate să manipuleze funcționarea AM într-un mod neconvenabil. Desigur acest risc este mult mai dificil de implementat.

Alegerea uneia dintre cele două paradigme depinde de aplicație și de nevoile programatorului. În această lucrare am încercat să prezint utilitatea Agenților Mobili evidențiindu-mi motivul pentru care am ales sa fac cercetare în acest domeniu.

2.2 Sisteme Multi-Agent

În secțiunea anterioară a fost prezentată problematica Agenților Mobili. Ce ar fi dacă am grupa AM într-un sistem care să prezinte caracteristici de comunicare?

Un sistem Multi-Agent (MAS) este format din mai multi agenți independenți, care interacționează într-un anumit domeniu. Fiecare Agent este un producător de decizii, care se află într-un mediu și acționează autonom, se bazează pe observațiile sale și pe domeniul său de cunoștințele pentru a-și atinge un anumit scop. Un design MAS poate fi benefic în multe domenii ale IA, în special atunci când un sistem este compus din mai multe entități care sunt distribuite spațial sau funcțional Ca exemplu ar fi roboții mobili (robotii de explorare spațială) sau rețele de senzori (radare meteorologice de urmărire)[7]. Colaborarea permite diferitor Agenți de a lucra mult mai

eficient și de a-și finaliza activitățile pe care nu sunt în măsură să le realizeze în mod individual. Chiar în domenii în care Agenții pot fi controlați centralizat, un MAS poate îmbunătăți performanța, fiabilitatea și scalabilitatea acționând în paralel. Agenții din MAS pot avea scopuri diferite, chiar contradictorii. În lucrarea de față agenții software prezintă scopuri diferite, în funcție de grupurile în care se află.

Într-un cadru cooperativ, fiecare agent software selectează acțiunile în mod individual, însă acestea contribuie la rezultatul comun. Coordonarea este, prin urmare, un aspect important în astfel de sisteme. Scopul coordonării este de a se asigura că deciziile individuale a agenților software sunt optime pentru întregul grup. Acest lucru este extrem de dificil mai ales atunci când agenții software operează într-un nivel mare de incertitudine. De exemplu, în domeniul fotbalului, fiecare robot operează autonom, dar fac parte dintr-o echipă și trebuie să coopereze cu alți membrii ai echipei pentru a câștigă Agenții din astfel de domenii ar putea avea nevoie să execute o secvență lungă de acțiuni pentru a se atinge obiectivele grupului [8].

2.3 Inteligența Ambientală

Progresul actual în sistemele informatice se îndreaptă spre epoca calcului omniprezent, când aproape în fiecare obiect din mediul nostru va găzdui capacități de calcul și de comunicare. Cu toate acestea, numărul și varietatea dispozitivelor inteligente ridică întrebări cu privire la timpul și atenția acordată din partea utilizatorilor și pledează insistent pentru noi dezvoltări care vizează "ameliorarea sarcinii de interacțiune cognitivă". Inteligența Ambientală (AmI), transformă un sistem de rețea cu dispozitive inteligente și de senzori într-un mediu care acționează ca o interfață între utilizatori la nivel mondial.

O cerință importantă pentru mediile AmI este adaptarea automată a comportamentului sistemelor pentru activitățile utilizatorilor. AmI încearcă să favorizeze comunicarea între obiecte pentru a evita calcul inutil. Proiectarea unor astfel de sisteme necesită abstractizarea între funcţionalităţile de bază oferite de către sistemele omniprezente și nevoile utilizatorilor. În secțiunea următoare se va prezenta problema proiectării sistemelor adaptabile și reconfigurabile pentru AmI. AmI și calculul omniprezent folosesc abordarea orientată pe servicii, spre exemplu SOA (Service Oriented Arhitecture). Astfel, fiecare dispozitiv furnizează servicii pe care clienții îi pot folosi. Această abordare oferă facilitatea de a construi sisteme pe dispozitive heterogene și servicii în medii deschise.

2.3.1 Caracteristicile Sistemelor Multi-Agent pentru AmI

Agenții sistemelor Multi-Agent au nevoie de interacțiune și cooperare pentru realizarea sarcinilor globale. Una dintre principalele proprietăți ale SMA este că se bazează mai degrabă pe distribuția algoritmilor de cooperare decât pe procesele de centralizare. Paradigma Sistemelor Multi-Agent se potrivește mai bine cerințelor mediilor AmI din mai multe perspective. SMA oferă control descentralizat bazat pe entități distribuite autonome. Un sistem de control centralizat are tendința să devină complex din cauza faptului că gama posibilelor servicii și situații se extinde, în timp ce

modelul Multi-Agent apare ca un mod natural de proiectare a sistemelor cu adevărat scalabil și robust.

Într-un SMA, entitățile autonome cu capabilități limitate cooperează în scopul realizării sarcinilor complexe. Coordonarea rapidă și modalități flexibile de organizare permit grupurilor de Agenți de a crea și reconfigura aplicații dinamice care depind de condiția curentă. Astfel de modele sunt bine adaptate la schimbările unui mediu AmI deschis.

Un SMA este o societate de agenți cu capacități și roluri. O astfel de reprezentare abstractă a unui sistem, similară cu societatea umană, reprezintă un interes sporit atunci când se dorește interacțiunea cu utilizatorii. În ciuda acestor caracteristici atractive, adoptarea SMA în calculul omniprezent și în domeniile AmI este încă limitată. La momentul de față , gestionarea mobilității utilizatorilor în calculul omniprezent, se focalizează mai mult pe utilizarea Agenților Mobili, lăsând deoparte SMA. Totuși proiectarea infrastructurii de servicii SMA peste Arhitectura Orientată pe Servicii AmI este o bună alegere. Abordarea SOA oferă o imagine realistă a mediilor omniprezente unde serviciile sunt furnizate de către producători diferiți, oferind o construcție rapidă și eficientă aplicațiilor

2.4 S-CLAIM

Ca platformă pentru aplicații AmI, proiectul tATAmI are la bază framework-ul JADE și folosește o versiune mai clară și mai simplificată a limbajului CLAIM(Computational Language for Autonomous,Intelligent and Mobile agents), denumit S-CLAIM (Smart-CLAIM), care este focalizat mai mult pe caracteristicile agenților decât folosit ca un limbaj de programare cu funcții complete. Pe lângă avantajul de a fi orientat-obiect, limbajul CLAIM este inspirat din programarea ambientală. Acesta folosește în spate limbajul Java, având acces la orice resursă Java necesară. Agenții implementați în CLAIM sunt executați folosind platforma Sympa, care coordonează ciclul de viață al agenților, dar și mobilitatea lor[2].

Un avantaj al acestuia este faptul ca oferă mobilitate puternică agenților: atunci când se mută pe o altă mașină, execuția lor continuă fără a pierde cunoștințe, mesaje sau capacități Mobilitatea agenților necesită o caracteristică importantă și anume locația sau contextul. CLAIM oferă o structurare arhitecturală a agenților software. Astfel, dezvoltarea agenților devine mult mai ușoară și mai rapidă. CLAIM folosește “Administration System” pentru a retine informații despre agenți, iar pentru securitate - validarea autorităților agenților, lăsându-i pe programatori să decidă dacă agenții au nevoie de resurse securizate sau nu. Toate acestea permit crearea unei platforme destul de puternice pentru dezvoltarea aplicațiilor care folosesc agenți mobili.

În dezvoltarea unui SMA pe arhitecturi diferite ca Android și PC, CLAIM este o soluție destul de bună. Pe lângă faptul ca oferă o transparență la nivel de rețea, permite și reprezentarea arhitecturală a întregului sistem. Proiectul tATAmi folosește toate aceste caracteristici, care vor fi detaliate în scenariul de bază.

Semantica limbajului S-CLAIM este strâns bazată pe semantica CoCoMo, dezvoltată de Abdelkader Behdenna. S-CLIAM este o variantă simplificată a limbajului CLAIM, care reduce lista de primitive la numai una, caracteristică interacțiunii și administrării agentului. Primitivele specificate de către S-CLAIM conform [2] sunt:

- **comunicarea** :
  - *send* – trimite un mesaj altui agent, răspunde unui mesaj sau accesează un serviciu web;
  - *receive* – primește un mesaj de la alt agent sau primește o invocare ca serviciu web;
- **mobilitatea** :
  - *în* – devine copilul altui agent; dacă agentul nu este obligat de a rămâne în containerul său, agentul se va muta împreună cu ierarhia sa;
  - *out* – iese din contextul agentului părinte;
- **administrarea agentului** :
  - *open* – ordonează dizolvarea unui agent copil, recuperând toate capacitățile și cunoștințele de agentul de copil;
  - *acid* – se dizolvă de la agentul părinte , părintele recuperând toate capacitățile și cunoștințele de agentului;
  - *new* – *creează* un nou agent care devine copilul agentului care îl creează;
- **administrarea cunoștințelor** :
  - *addK* – adaugă un element de cunoaștere în baza de cunoștințe;
  - *removeK* – elimină un element de cunoaștere;
  - *forAllK* – iterează peste toate cunoștințele care se potrivesc cu un anumit model, fiecare iterație execută un set de declarații, folosind elementul de cunoaștere selectat;
- **controlul primitivelor** :
  - *condition* – *condiționează* activitatea unui comportament folosind operații logice sau rezultatul boolean al unei operații;
  - *if* – condiționează execuția unui bloc de declarații;
  - *wait* – întrerupe comportamentul unui agent pentru o anumită perioadă de timp;

Semantica S-CLAIM folosește doar primitive pentru administrarea agenților software, administrarea cunoștințelor și a comunicării între agenți și o parte pentru controlul primitivelor. Acesta nu se bazează pe folosirea algoritmilor pentru agenți software, ci folosește funcții, operații aritmetice sau logice, etc. oferind utilizarea limbajului Java.

Sintaxa S-CLAIM folsește paranteze pentru a face codul mai clar și mai simplu. Variabile sunt precedate de semnul întrebării. O implementare obișnuită a agentului conține și numele definit al acestuia, parametrii și lista de comportamente. Există trei tipuri de comportamente:

- **iniţial** – executat la crearea agenților;
- **reactiv** – executat ca consecință a primirii de mesaje și îndeplinirea unor condiții;

- **proactiv** – comportament orientat pe scop care se execută fără a fi implicate evenimente externe;

Exemplu [2] :

```
(agent NumeleClasei ?destination
        (behavior
                (initial sender
                        (send ?destination (struct message salut)))))
```

Din exemplu se observă ca S-CLAIM folosește structura ca formă unică de structurare a datelor. O structură conține o listă de membri și se găsește între paranteze. Se întâlnesc două tipuri de structuri: message și knowledge. Este important faptul că S-CLAIM folosește câte un standard pentru fiecare primitivă, spre exemplu send și read sunt precedate de un set de parametrii :

- (send [destinatar] (struct message [conținut..]))
- (send ?părinte (struct message managesCourse this ?courseName))
- (receive [conținut..]) – toate simbolurile conținute se vor lega la variabilele din mesaj; trebuie să fie în partea de activare a unui comportament reactiv
- (receive managesCourse ?agentName ?courseName)

Scopul S-CLAIM nu este de a reinventa un nou limbaj de programare cu noi funcționalități mai bune decât cele existente, ci ține să folosească elementele necesare construirii unui agent și menținerii activității sale într-un mod mai simplu folosind doar acele primitive care sunt strâns legate de structura agentului. Având în vedere acestea, ne dăm seama ca un agent nu este foarte performant dacă nu suportă algoritmi sau chiar funcții aritmetice. De aceea S-CLAIM suportă clase Java care pot fi folosit la fel ca și primitivele. Spre exemplu dacă este nevoie de o afișare la consolă, se va crea o funcție Java *sysout* care va avea toate funcţionalităţile necesare. O funcție va conține un vector de parametrii și va returna o variabilă booleană . În S-CLAIM se va folosi: (sysout ?mesaj).

CAPITOLUL 3. Tehnologii folosite

Agenții inteligenți sunt considerați o abordare promițătoare pentru construirea unor sisteme software complexe, deoarece paradigma agent permite modelarea aplicațiilor într-un mod natural, asemănător cu modul în care oamenii percep domeniul problemei. Cu toate acestea, dezvoltarea aplicațiilor orientate agent nu se practică destul de des. Acest domeniu necesită multe cunoștințe cum ar fi: ingineria sistemelor distribuite, infrastructura comunicării și arhitectura agentului software. Un motiv ar fi că nu există un consens general pentru paradigmele de modelare a sistemelor multi-agent, chiar dacă s-au dezvoltat o mulțime de tehnologii. Un alt argument ar fi faptul că alegerea platformei este importantă pentru succesul unui proiect software, dar este dificil de găsit o platformă potrivită, având în vedere numărul mare de platforme disponibile. În acest capitol se va prezenta argumentele alegerii platformei JADE pentru dezvoltarea sistemului multi-agent. De remarcat faptul că specificațiile FIPA reprezintă, la momentul de față, un standard în tehnologia agenților software.

3.1 Programarea Orientată-Agent

Programarea orientată-agent este o paradigmă nouă în programare care aduce concepte legate de teoria Inteligenței Artificiale în domeniul integrării sistemelor distribuite. POA modelează o aplicație ca o colecție de componente numite agenți software caracterizați de autonomie, reactivitate și abilitate de comunicare[9]. Modelul architectural a unei aplicații orientate agent este punct la punct. Astfel, orice agent este capabil să inițieze o conversație cu oricare alt agent sau să interogheze orice agent la orice moment.

Tehnologia bazată pe agenți este în dezvoltare de o perioadă de timp și încă nu este folosită la maxim. Sistemele multi-agent sunt folosite într-o mare varietate de aplicații, de la sisteme relativ mici, pentru uz personal, la sisteme complexe, pentru aplicații industriale. Exemple de domenii industriale în care se folosesc sistemele multi-agent includ procese de control, de diagnosticare a sistemelor, producție, rețelistică, etc.

Când se adoptă abordarea orientată-agent pentru a rezolva o problemă, în prima etapă se rezolvă o serie de aspecte dependente de domeniu, cum ar fi modul de a permite agenților să comunice. În ultima vreme s-a adoptat politica de creare a sistemelor multi-agent peste un Middleware orientat-obiect care dispune de infrastructură pentru independența domeniului, permițând dezvoltatorilor să se concentreze asupra logicii sistemului.

În continuare se va descrie JADE (Java Agent DEvelopment Framework), probabil cel mai răspândit Middleware orientat-Agent de astăzi.

3.2 Noțiuni introductive despre JADE

JADE este un sistem Middleware complet distribuit cu o infrastructură flexibilă care permite ușor extinderea cu module Add-on framework-ul permite dezvoltarea completă a aplicațiilor bazate pe agenți prin intermediul unui mediu runtime, implementând caracteristicile ciclului de viată al agenților, logica de bază a agenților și o suită de unelte grafice. JADE este scris complet în Java, acesta beneficiază de un set mare de caracteristici lingvistice și, prin urmare, oferă un set bogat de abstractizări de programare permițând dezvoltarea sistemelor multi-agent fără a cunoaște o vastă teorie în domeniul agenților Inițial, JADE a fost dezvoltat de Departamentul de Cercetare și Dezvoltare al Telecom Italia s.p.a, dar acum este un proiect distribuit ca open source sub licență LGPL[14].

Pentru implementarea abstractizării agentului software, JADE folosește următoarele modele:

- Un agent este autonom și proactiv. Un agent nu poate să ofere referință la obiectul său altor agenți Acesta trebuie să se execute pe un nou fir de execuție, necesar controlului ciclului său de viată și să decidă ce și când să se execute.
- Un agent poate refuza mesaje. Se folosește comunicarea asincronă. Identificarea agenților implicați în comunicare are loc printr-un identificator unic. Nu este nevoie de o locație sau referință la alt agent pentru a-i trimite mesaje.
- Sistemul este Punct-la-Punct, fiecare agent este identificat printr-un nume global și unic (AgentIdentifier sau AID definit de FIPA). Este posibil ca agenții să se alăture sau să părăsească gazda la orice moment de timp și poate să descopere alți agenți prin intermediul serviciilor white-page și yellow-page(disponibile în JADE datorită agenților DF așa cum se specifică în FIPA).

Pe baza acestor alegeri, JADE a fost realizat pentru a oferi programatorilor funcționalități de bază și implementarea mai ușoară a sistemelor multi-agent. Aceste funcționalități sunt:

- *Un sistem complet distribuit*. Agenții rulează pe fire de execuție diferite și chiar pe mașini diferite, iar comunicarea cu alți agenți software este transparentă. Platforma folosește AID pentru fiecare agent software.
- *Deplină conformitate cu specificațiile FIPA***.** Platforma suportă toate elementele specificate în FIPA.
- *Transportul asincron al mesajelor prin intermediul unui API care oferă transparență locației*
- *Implementarea serviciilor white pages și yellow pages.*
- *O administrare eficientă și simplă a ciclului de viată* al *agenților* software. Când **agenții** sunt creați li se asignează un identificator global unic și o adresă de transport care este folosită pentru înregistrarea acestora la serviciile white-pages ale platformei. Oferă și elemente grafice pentru administrarea ciclului de viată pe mașină locală sau la distantă

- *Suport pentru mobilitatea agenților* Codul și starea agentului pot migra între procese sau mașini Migrarea agenților este transparentă, agentul poate să își continue execuția chiar și în timpul migrării.
- *Un set de instrumente grafice* care ajută la debugging și la monitorizare. Aceste elemente sunt destul de complicate într-un sistem distribuit ca JADE.
- *Suport pentru ontologie și conținutul limbajului.*
- *Integrarea cu tehnologia web* folosind jsp, servleţi, apleți și servicii web.

3.3 Crearea Sistemelor Multi-Agent folosind JADE

JADE beneficiază de specificațiile FIPA pentru sisteme multi-agent, oferind o implementare punct-la-punt a sistemelor distribuite și se bazează pe o comunicare slab-cuplată folosind mesaje asincrone ACL. FIPA (Foundation for Inteligent Phisical Agents) produce standarde pentru interacțiunea agenților software heterogeni. Este un standard care definește arhitectura abstractă a agenților pentru utilizarea globală a acestora.

Conform [10] FIPA are la bază următoarele principii:

1. Tehnologia Agenţilor Software furnizează o paradigmă nouă;
2. Unele tehnologii bazate pe agenți au ajuns la un grad considerabil de maturitate;
3. Folosirea tehnologiei agenților software solicită standardizare;
4. Standardizarea tehnologiei generică este posibilă;
5. Standardizarea mecanismelor de infrastructură nu este prima îngrijorare, în comparație cu cerințele limbajului și interoperabilitatea deschisă.

3.3.1 Platforma Agenților

Platforma JADE implementează specificațiile FIPA:

- furnizează infrastructura fizică în care agenții software pot fi utilizați;
  - mașini, sisteme de operare, suportul software pentru agenți;
  - AMS, DFS, MTS;
  - Nu implică configurarea fizică;
- AMS
  - Managementul AID și al descrierii de transport;
  - White-pages;
- DF
  - Yellow-Pages;
- MTS
  - O metodă de comunicare între agenții pe diferite platforme.

Figura următoare ilustrează elementele unei astfel de platforme.

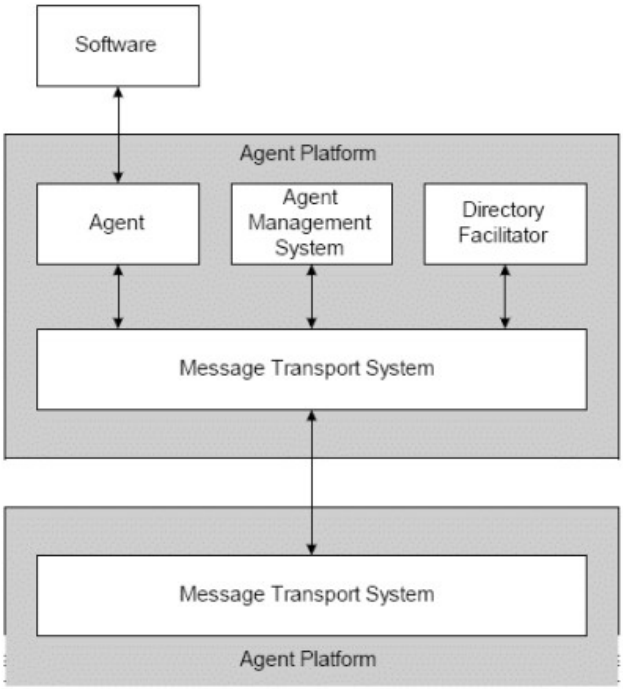

Fig.3.1 Structura Platformei conform specificațiilor FIPA[11].

3.3.2 Comunicarea Agenților

În secțiunile anterioare s-a precizat rolul unui sistem multi-agent în aplicațiile distribuite. Un element important pentru funcționarea acestor sisteme îl reprezintă comunicarea. Comunicarea poate fi de mai multe feluri: comunicarea între agenți și utilizatori, comunicarea cu resursele sistemului și comunicarea între agenți pentru cooperare, colaborare, negociere ş.a. În această secțiune se va prezenta limbajul de comunicare între agenți.

Un element destul de important de care dispun agenții dezvoltați în JADE este abilitatea de comunicare. Paradigma de comunicare adoptată este schimbul asincron de mesaje. Fiecare agent dispune de o coadă de mesaje unde se pot posta mesajele trimise de către alți agenți software. Odată cu apariția unui mesaj, agentul este înștiințat de aceasta. Programatorul are posibilitatea de a preciza care mesaje se vor procesa.

Limbajul ACL adoptat de FIPA este cel mai utilizat la momentul actual. Schimbul de mesaje în JADE are un format specificat de către limbajul ACL. Acest format cuprinde următoarele elemente:

- Transmițătorul – sender
- O listă de primitori – receivers
- Intenția de comunicare – transmițătorul intenționează să transmită un mesaj
- Contextul – informațiile cuprinse în mesaj
- Limbajul contextului – sintaxa folosită pentru a exprima conținutul
- Ontologia – vocabularul de simboluri folosit în context cu specificațiile aferente

- Unele câmpuri sunt folosite pentru a controla mai multe convorbiri concurente și pentru a preciza un interval de timp pentru primirea mesajelor de genul id-ul conversației

Un mesaj în JADE este implementat ca un obiect din clasa jade.lang.acl.ACLMessage care oferă metode de get și set pentru primirea și transmiterea mesajelor.

Transmiterea unui mesaj este destul de simplă. Se instanţiază un obiect de tipul jade.lang.acl.ACLMessage, se setează limbajul, ontologia, contextual și, nu în ultimul rând, identificatorul destinatarului și se apelează funcția send(). După cum s-a menționat, JADE introduce automat mesajul în coada destinatarului. Un agent poate alege un mesaj prin folosirea metodei receive(). Această metodă returnează primul mesaj din coadă sau null dacă coada este goală.

3.3.3 Serviciile Yellow-Pages

Serviciile "Yellow-Pages" permit agenților să publice descrierile mai multor servicii pe care aceștia le dețin pentru ca alți agenți să le poată descoperi și folosi. Un agent este capabil să înregistreze un serviciu sau să folosească unul deja existent. Înregistrarea, de înregistrarea, modificarea și căutarea pot fi efectuate în orice moment de timp din ciclul de viată al agentului.

Serviciile JADE yellow-pages respectă specificațiile FIPA Agent Management, furnizate de către un agent specializat numit DF(Directory Facilitator). Chiar și platforma FIPA trebuie să găzduiască un DF prestabilit.

Agentul DF interacționează cu restul agenților prin schimb de mesaje ACL, care au un limbaj și o ontologie proprii definite în FIPA. JADE oferă o versiune mai simplificată prin intermediul clasei jade.domain.DFService cu ajutorul căruia se pot publica și extrage servicii. Dacă un agent vrea să publice unul sau mai multe servicii trebuie să furnizeze DF-ului AID-ul său, o listă de servicii și, opțional, limbajul și ontologia pe care agentul o va folosi pentru a interacționa cu serviciile. Fiecare serviciu va avea un nume, tip, limbajul și ontologia necesară pentru a utiliza serviciu și o mulțime de proprietăți specifice serviciilor sub formă de cheie-valoare. JADE DF oferă, de asemenea, un mecanisme de subscriere care permit agenților software de a fi notificați cât mai curând când alți agenți înregistrează sau de înregistrează un serviciu. Toate aceste mecanisme impun inițierea unui protocol FIPA cu DF.

DF necesită și un grad de securitate. Acesta poate să restricționeze accesul la informația din directorul curent și să verifice toate permisiunile de acces pentru agenții care încearcă să-l informeze despre schimbările de stare a acestora.

3.4 JADE-LEAP, Agenți pe dispozitive mobile

Astăzi este posibil rularea agenților JADE pe dispozitivele mobile MIDP datorită extensiei LEAP dezvoltată în 2002 de către Motorola, Broadcom Eireann, Siemens AG, Telecom Italia și

Universitatea din Parma. Datorită extensiei LEAP, este posibilă execuția agenților JADE pe Microsoft .NET, dar și pe sistemul de operare Android.

JADE-LEAP dispune de metode diferite care corespund cu trei medii de bază Java (ediții, configurări și pofile) găsite pe dispozitive:

- J2SE: pentru a executa JADE-LEAP pe PC-uri și servere într-o rețea fixă pe care rulează o versiune de JDK1.4 sau mai mare.
- pJava: pentru a executa JADE-LEAP pe dispozitivele portabile care suportă J2ME CDC sau Java Personal
- MIDP : pentru a executa JADE-LEAP pe dispozitivele portabile care suportă MIDP1.0

Aceste trei versiuni de JADE-LEAP furnizează același subset de API pentru dezvoltare, oferind astfel un strat omogen peste o diversitate de dispozitive și rețele Doar câteva caracteristici care sunt disponibile în JADE-LEAP pentru J2SE și pJava nu sunt disponibile și pentru MIDP.

LEAP oferă o nouă modalitate de execuție – divizată; un utilizator nu își creează un obiect de tip container, ci un nivel mic numit *front-end*. Front-end oferă agenților aceleași caracteristici ca și un container normal, dar implementează direct doar o mică parte din ele, în timp ce celelalte caracteristici se deleagă pe un proces la distantă numit *back-end*. Front-end și back-end comunică printr-o conexiune specială.

Modul de execuție divizat este potrivit pentru dispozitivele cu constrângeri de resurse datorită următoarelor motive:

- Front-end rulează pe dispozitive și este mult mai simplu decât un container complet.
- Comunicările cu containerul principal necesită alăturarea la platformă, iar back-end nu se efectuează pe o conexiune wireless.
- Utilizarea conexiunii wireless este optimizată.
- Atât front-end cât și back-end încorporează mecanisme de stocare și eliminare pentru a face pierderea conexiunii transparentă față de aplicație
- Dacă conexiunea între front-end și back-end cade, mesajele transmise peste acea conexiune se salvează în cozi, iar când conexiunea se restabilește, se reia transmiterea lor.
- IP-ul dispozitivului mobil nu este văzut de către alte containere din platformă, deoarece acesta interacționează cu back-end. Acesta se poate schimba chiar și fără nici un impact asupra aplicației

Un alt Add-on important oferit de către JADE care oferă suport pentru LEAP este JADE-ANDROID, care va fi subiectul discuției în următoarea secțiune. Acesta este un nou framework care permite rularea agenților JADE pe dispozitive cu sistem de operare Android. Această combinație are o caracteristică importantă și anume faptul că ambele folosesc limbajul Java.

3.5 Sistemul de operare Android

Android este o platformă software și un sistem de operare bazat pe nucleul Linux pentru dispozitivele mobile ca SmartPhones sau tablete. Acesta este dezvoltat de către Open Handset Alliance, condus de Google.

Android are o largă comunitate de dezvoltatori care extind funcţionalităţile acestuia. Aplicațiile suportate de acesta pot fi dezvoltate doar într-o versiune specială de Java. S-a creat un site general Google Play special pentru instalarea aplicațiilor direct pe telefon.

Începând cu 21 octombrie 2008, sistemul de operare Android devine Open Source. Google a deschis întregul cod sursă (inclusiv suportul pentru rețea și telefonie), care anterior erau indisponibile, să-l distribuie sub licență Apache. Sub licență Apache dezvoltatorii sunt liberi să adauge extensii proprii, fără a le face disponibile comunității open source.

SDK-ul Android include un set complet de instrumente de dezvoltare: un program de depanare, biblioteci, un emulator, documentație, monstre de cod și tutoriale. Platformele de dezvoltare sprijinite în prezent includ calculatoare bazate pe x86, pe care rulează Linux, Mac OS X 10.4.8, Windows XP sau Vista.

Android a avut un mare succes pe piață, a reușit să combine cele două funcţionaliăţi: telefon și calculator într-un singur dispozitiv. Viziunea asupra unui telefon s-a schimbat odată cu apariția telefoanelor “smart”. Utilizatorii folosesc telefoanele nu doar pentru conversații, dar și pentru jocuri, GPS, internet ş.a. Astfel, telefoanele au devenit mult mai performante, ajungând la un nivel apropiat de un calculator PC. Datorită acestui fapt, aplicațiile pentru telefoane mobile partajează din ce în ce mai multe caracteristici cu aplicațiile desktop și în cele din urmă extind aceste caracteristici cu capacități de conștientizare referitoare la context, reactivitate, uzabilitate, și așa mai departe, toate aspectele care sunt importante în contextul Inteligenței Ambientale. Componentele sistemului de operare Android devin posibile de folosit pentru dezvoltarea aplicațiilor bazate pe agenți

Un telefon mobil cu sistem de operare Android poate fi văzut ca un agent software. Acest model de agent are o abordare mai mult practică, decât teoretică deoarece folosește abstractizare care face posibilă implementarea acestora pe diferite sisteme. Apariția Androidului ca un sistem de operare deschis bazat pe Linux a creat noi așteptări pentru implementarea agenților. Agenții pot să ruleze pe platforme hardware diferite; o abordare utilă în Ubiquitous Computing pentru a obține agenți inteligenți încorporați în mediu. Această viziune poate considera sistemul de operare Androidul ca un mediu real inteligent.

CAPITOLUL 4. Arhitectura și componentele sistemului

Acest capitol va cuprinde detalii despre arhitectura proiectului "Platforma pentru discuții pro și contra". În dezvoltarea unui sistem multi-agent este destul de importantă structura, deoarece aceasta trebuie să permită o interacțiune cât mai simplă între elementele sale. În funcție de utilizarea sistemului multi agent, acesta trebuie să respecte anumite standarde. Proiectul nu are ca scop crearea unei noi platforme pentru sisteme multi agent, ci de a extinde platforma tATAmi să permită execuția acestor sisteme distribuite pe dispozitive diferite legate prin rețea.

4.1 Proiecte implicate

Aplicația este construită peste proiectul tATAmI, care implementează scenarii simple pentru sisteme multi-agent orientate-AmI, în care agenții software sunt asignați diferitor elemente – locuri, dispozitive, servicii, utilizatori și cu o relație ierarhică între agenți Proiectul tATAmi-Android a fost implementat în colaborare cu Miruna Popescu, sub supravegherea as. dr. ing. Andrei Olaru, unul din dezvoltatorii proiectului tATAmI.

Pentru folosirea unei structurări ierarhice, tATAmI are la bază limbajul orientat agent S-CLAIM, despre care s-a discutat în capitolele anterioare. Agenții S-CLAIM sunt caracterizați de mobilitate și, în special, de mobilitatea ierarhică. Când un agent este plasat într-o ierarhie și pleacă pe o altă mașină, automat întregul sub-arbore de agenți pleacă cu el [12].

Agenții software sunt caracterizați de un părinte în ierarhie, de cunoștințe – reprezentate sub formă de predicate de ordin I, de scopuri, mesaje pe care agenții le pot primi, capabilități, procesul pe care îl execută, și, nu în ultimul rând, agenții copii. Capabilitățile sunt activate când se primesc mesaje sau în unele condiții critice care pot apărea

În viziunea proiectului tATAmI există două tipuri de structuri:

- structura *fizică*, fiecare agent se execută pe o mașină separată;
- structura *logică*, unde agenții fac parte dintr-o ierarhie logică, care poate să se găsească pe mai multe mașini

tATAmI este structurat în jurul unei idei critice: maparea diferitor contexte peste diferite părți a arhitecturii logice a agenților formate din relații părinte/copii în S-CLAIM. Agenții software, cât timp reprezintă dispozitive sau locații, pot să reprezinte și contexte, permițând dezvoltatorului să descrie o ierarhie de contexte.

Un exemplu de scenariu care aduce în prim plan această caracteristică este: când un utilizator se mută într-o cameră agentul său devine în ierarhie copilul agentului care administrează

camera. Copilul agentului definit de utilizator, dispozitiv sau serviciu, sunt la fel în același context. Când utilizatorul se mută într-o altă cameră, agentul său își schimbă părintele și automat cu el se mută și copii săi. Unele dispozitive nu sunt în stare să se mute împreună cu utilizatorul așa că ei vor constata că noul context este incompatibil cu proprietățile lor, mutându-se înapoi la copii agentului care administrează camera.

Contextul nu reprezintă doar locația sau arborele ierarhic oferit de CLAIM, acesta cuprinde și contextul computațional Când un utilizator folosește un serviciu se creează un agent care oferă acest serviciu și devine copilul agentului utilizatorului. Această structurare permite ca copii să găsească ușor serviciile oferite de către agenții părinți sau reciproc.

O altă caracteristică importantă care definește contextul o reprezintă preferințele utilizatorului. Utilizatorul poate să introducă preferințe în capabilitățile dispozitivelor necesare. CLAIM nu oferă aceste aspecte, preferințele pot să reducă considerabil procesul de calcul într-o căutare a dispozitivelor apropiate.

tATAmI suportă o varietate de scenarii, unul dintre care reprezintă subiectul discuției și evidențiază mutarea agenților pe dispozitive mobile. Proiectul tATAmI-Android va furniza agenților caracteristicile grafice pentru agenți și va permite existenta mutării agentului pe sistemul de operare Android cu toate funcţionalităţile predefinite înainte de transfer. Execuția agenților pe sistemul de operare Android, este posibilă datorită implementării portabilității proiectului tATAmI. Proiectul tATAmI este structurat pe componente bine definite care oferă extinderea acestuia. Astfel, componentele care au fost modificate sau extinse țin de identificare platformei, definirea unei componente de logare, schimbare de platformă și, nu în ultimul rând, interfață grafică. În secțiunea ce urmează se va prezenta componentele de bază împreună cu funcţionalităţile lor.

4.2 Componentele de bază

Arhitectura proiectului general este bazată pe componente și straturi după cum urmează:

- Componenta *Core*, conține clasele care definesc agentul software, organizate în următoarele straturi :
    - Comunicarea agenților, mobilitatea și administrarea, toate agestea utilizează elementele oferite de către JADE;
    - Logarea și vizualizarea – agenții sunt monitorizați permanent, transmițând date despre activitățile lor pentru o centralizare a acestora. Fiecare agent dispune și de un element propriu pentru afișarea activităților sale;
    - Mobilitatea ierarhică a agenților – protocoale și comportamente a agenților care permit acestora să se mute automat cu părintii lor;
    - Accesarea serviciilor web – expunerea agenților ca servicii web și permiterea acestora să acceseze serviciile web;
    - Implementarea și execuția S-CLAIM – un parser pentru fișierele de descriere a agenților S-CLAIM și componente care transformă definirile în comportamente;

- Cunoștințele – o componentă interschimbabilă care permite accesul cunoștințelor printr-un set de funcții;
- Contextul și cunoștințele – utilizarea contextului pentru rezolvarea problemelor specifice și pentru schimbul relevant de informații

- Componenta *Simulation* care servește la execuția repetabilă a scenariilor:
  - Folosește fișierele XML pentru definirea datelor de intrare;
  - Lansează agenții corespunzători scenariilor pe containere și mașini specifice;
  - Trimite mesajele evenimente "externe" echivalent cu percepțiile agenților software într-un mediu real;
- Componenta *Vizualization* care asigură vizualizarea centralizată a activităților agenților software:
  - Primește mesajele de log și evenimentele de mobilitate de la agenți;
  - Afișează toate mesajele log într-un mod centralizat;
  - Afișează structura sistemului(topologia), arătând relațiile între agenți;
  - Furnizează componente pentru elementele grafice ale agentului pe mașină care se execută;

Aceste componente asigură funcționalitatea platformei, permițând extinderea cât mai simplă a acesteia. S-a ales distribuirea componentelor în două proiecte datorită faptului că elementele de Android folosesc o versiune special de Java. Componentele grafice sunt total diferite între PC și Android.

O viziune a componentelor precizate mai sus se găsește în figura 4.1. Cele trei elemente din figură sunt:

- Simulation – tot ce ține de scenariu, definire,etc;
- Core – tot ce ține de de prezentarea agentului ca un sistem autonom;
- Visualization – administrează elementele grafice.

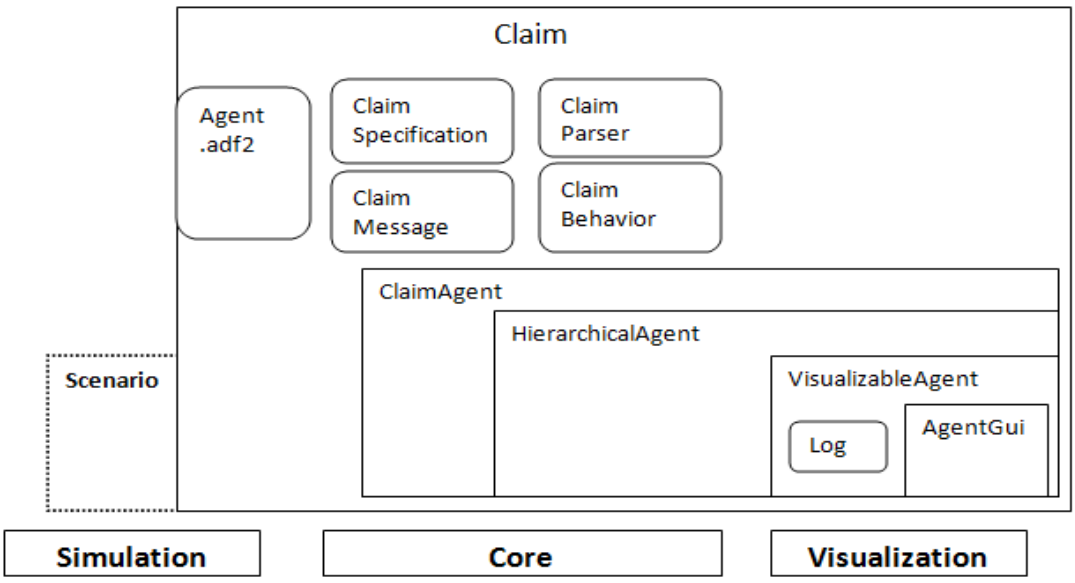

Fig.4.1 O reprezentare a componentelor proiectului tATAmI.

4.3 Structura Agentului

După cu s-a precizat și anterior, agenții software sunt de tip S-CLAIM, adică există o structură ierarhică a acestora care se poate modifica în timp. Acest lucru nu este ușor de implementat, de aceea s-a recurs la organizarea agenților pe nivele. Structurarea pe nivele permite modularizarea sistemului, separarea conceptelor, dar și ușurarea depanării sistemului. Extinderea platformei pentru migrarea agenților software pe sistemul de operare Android a fost posibilă datorită acestei caracteristici .

Agenții software ai platformei tATAmI sunt constituiți din cinci nivele de componente, care sunt denumite corespunzător cu clasele Java care le implementează:

- JADE *GuiAgent* – agenții software de bază care pot avea o interfață grafică. Acest nivel coordonează transmiterea mesajelor și mobilitatea la nivelul platformei JADE și reprezintă entitatea de bază a întregului proiect.
- *VisualizableAgent* – tratează vizualizarea agentului software în două moduri:
  - oferă nivelelor superioare obiectele de log obținute în urma monitorizării datelor agentului și raportează mesajele de log împreună cu schimbările produse în structura ierarhică a acestuia agentului de Visualization;
  - administrează ferestrele agentului și le integrează în schema ecranului;
- *WSAgent (web service agent)* – oferă funcționalități care sunt utilizate de către agentul S-CLAIM pentru a expune capabilitățile sale ca servicii web prin intermediul WSIG JADE Add-on, dar și de a accesa SOAP sau serviciile web;
- *HierarhichicalAgent* – administrează schimbările ierarhice, expunând obiectele pentru relații ierarhice între agenții software. Implementează comportamente pentru a instrui copiii agentului să urmărească mișcările acestuia, dar și să primească comenzi de la părintele său. Se pot stabili setări agentului care să-i permită de a sta fixat de containerul său, el nu va urmări părintele său care se mută;
- *Agentul S-CLAIM* – agentul care se execută folosind descrierile furnizate de către S-CLAIM; conține un set de elemente care identifică comportamentul și descriu capabilitățile agentului; acest nivel accesează componentele Bazelor de Cunoaștere pentru adăugare, ștergere sau pentru interogarea cunoștințelor

Pentru execuția agenților pe sistemul de operare Android a fost nevoie de definirea unei interfețe grafice pentru aceștia. Această interfață se va integra în componenta GuiAgent. Celelalte componente sunt nevoite să identifice platforma pe care se execută agentul (Android sau PC) și să facă schimbările necesare atunci când agentul software se mută de pe o platformă pe alta. tATAmI-Android nu va folosi servicii web, adică WSAgent.

O altă componentă de care VisualizableAgent trebuie să țină cont atunci când un agent își schimbă platforma este componenta care se ocupa de mesajele de log. JADE-Android folosește logging de la jade.util.Logger, iar agenții care se execută pe PC folosesc Apache log4j.

Platforma tATAmI inițial a fost construită pentru testarea și simularea sistemelor mult-agent pentru Inteligența Ambientală. Mai exact, pentru implementarea topologiei agenților și a comportamentelor lor.

Elementul important care îl oferă această platformă reprezintă mutarea ierarhică a agenților Nivelul HierarchicalAgent din tATAmI se ocupă de mutarea agenților și a copiilor lor. Mutarea ierarhică înseamnă ca un agent își păstrează contextul când se mută și se păstrează în contextul părintelui când părintele se mută.

În scenariul de bază vor exista trei tipuri de agenți:

- GroupCoordonatorAgent – se ocupă de centralizarea opiniilor și de administrarea grupului;
- PDAAgent – Agentul care se va executa pe sistemul de operare Android;
- EmissaryAgent – Agentul intermediar care va prelua mesajele de la foștii părinți PDAAgent și le vor transmite actualilor părinți GroupCoordonatorAgent;

Pentru ca agenții software să suporte cerințele necesare s-au definit, în fișiere adf2, câte o structură particulară pentru fiecare agent. Agentul PDAAgent are următoarea structură definită în S-CLAIM:

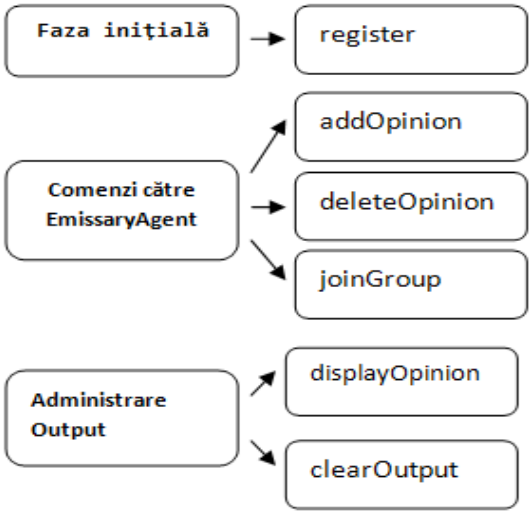

Fig.4.2 Structura agentului PDAAgent.

În *register* PDAAgent primește un mesaj de la EmissaryAgent prin care îi comunică de existenta sa. PDAAgent îl adaugă la baza de cunoștințe Prin *joinGroup* PDAAgent trimite emisarului să se deînregistreze de la grupul curent, dacă există, și să se alăture coordonatorului grupului dorit. Odată ce s-a adăugat la un grup, PDAAgent comunică cu EmissaryAgent pe care îl cunoaște pentru adăugarea, ștergerea sau afișarea opiniei (*addOpinion*, *deleteOpinion*, *displazOpinion*).

4.4 Scenariul

În scopul testării platformei tATAmI se folosesc scenarii. Pentru a demonstra portarea proiectului pe sistemul de operare Android am folosit un scenariu special și anume:

Într-o sală de prezentări se dorește ca la finalul fiecărei prezentări să se dea un feedback al lucrării. Datorită tehnologiei de astăzi, majoritatea dispunem de telefoane mobile cu sistem de operare Android. Acest lucru îl vom folosi pentru asignarea a câte unui agent pe fiecare telefon. Să presupunem că înainte de intrare în sală, participanții și-au instalat aplicația necesară pe telefon. Se vor forma două grupuri de discuții: grupul de discuții *pro* și grupul de discuții *contra*. Fiecare telefon va dispune de un agent PDA. Pentru a trimite o părere, un utilizator, mai întâi, se va alătura unui grup de discuții, în funcție de părerea pe care vrea să o expună. La finalul prezentării se va vedea pe un ecran părerile persoanelor din sală.

Pentru implementarea scenariului s-au folosit următoarele elemente:

- Doi agenți care administrează câte un grup, numiți Group_Master1, respectiv Group_Master2;
- Doi agenți care vor rula pe PC și vor trimite păreri despre discurs;
- Patru agenți pe câte un două emulatoare de Android, câte doi agenți pe emulator: PDA_Master și Emissary(Emissary nu va avea interfață grafică și va fi copilul agentului PDA_Master de pe dispozitivul respectiv);
- Patru agenți pe două telefoane mobile cu sistem de operare Android, vor fi organizați la fel ca pe emulator;
- Un agent care retine toate părerile și le va afișa pe ecranele respective.

Agenții PDA_Master se vor crea pe PC apoi se vor muta împreună cu câte un emisar pe sistemul de operare Android sau emulator. PDA_Master va comunica cu Group_Master, după alăturare, prin intermediul emisarului. Emisarul se va muta înapoi pe PC, va aștepta mesaje de la PDA_Master și le va trimite către Group_Master în care se află.

Acest scenariu va prezenta și o structurare ierarhică. Inițial se creează doi Group_Master împreună cu câte un container. Se creează câte un container corespunzător dispozitivului cu un agent PDA_Master și unul Emissary. Emissary va fi copilul PDA_Master-ului, după care se va muta automat pe PC devenind copilul Group_Master-ului. Astfel, agentul de pe sistemul de operare Android va putea să trimită mesajele către agentul care coordonează grupul de discuții

În secțiunea anterioară s-a discutat despre interacțiunea între cele trei tipuri de agenți care participă la un grup(Fig.4.2).

În figura următoare este prezentat modelul unui grup. Prin linie întreruptă se evidențiază starea precedentă a agentului Emissary.

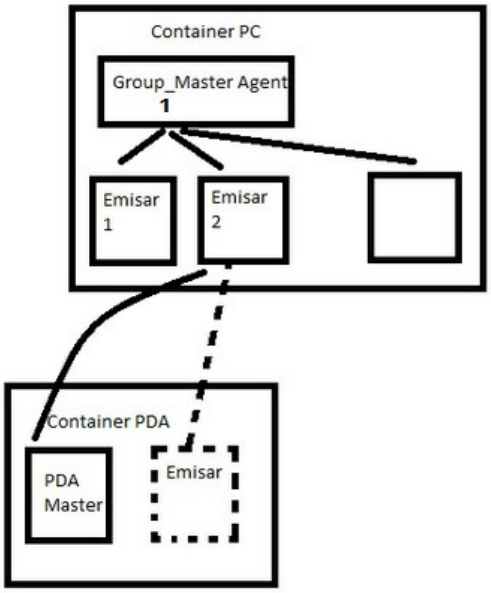

Fig 4.3. Relația între cele trei tipuri de agenți
Prin linie punctată se specifică starea de tranziție a agentului Emissary.

CAPITOLUL 5. Platforma pentru discuții pro și contra

Acest capitol va cuprinde detalii de implementare a proiectului, probleme întâlnite și soluțiile acestora. Organizarea pe componente a proiectului tATAmI a fost de mare ajutor, deoarece nu a fost nevoie restructurarea acestuia. În secțiunile următoare se vor prezenta detalii de portare a proiectului tATAmI pe sistemul de operare Android.

5.1 Elementele folosite în integrarea celor două proiecte

După cum am precizat în capitolul 4, aplicația conține două proiecte Java: tATAmI-PC și tATAmi-Android. Împărțirea în două proiecte s-a datorat sistemului de operare Android care suportă o variant specială a limbajului Java. tATAmI-PC conține toate clasele necesare parsării scenariului în S-CLAIM și componentele care se ocupă de generarea și simularea sistemului multi-agent. tATAmI-Android conține elementele legate de interfață și logging pentru agenții care rulează pe această platformă. Pentru integrarea celor două proiecte s-au folosit elemente de Java Reflection și încărcarea dinamică a claselor.

Platforma trebuie să permit agentului software să își continue execuția după migrarea de pe PC pe sistemul de operare Android. Agenții software ai acestei aplicații dispun de o interfață grafică. Atunci când un agent se mută de pe PC pe sistemul de operare Android, platforma detectează acest lucru și îi oferă agentului o nouă interfață grafică.

Pentru linkarea celor două proiecte, tATAmi-Android şi tATAmI-PC, am folosit o funcționalitate oferită de către mediul de dezvoltare Eclipse şi anume Build Path. Era nevoie ca tATAmi-Android să poată vedea clasele din tATAmI-PC. S-a folosit această funcționalitate pentru a nu introduce mari modificări pentru dezvoltările ulterioare. Soluţia de a introduce o arhivă .jar cu întregul proiect tATAmI-PC pare neadecvată, deoarece va extinde dimensiunea proiectului şi va trebui schimbată arhiva cu fiecare modificare în proiectul tATAmI-PC.

Un alt element care a folosit în integrarea celor două proiecte îl reprezintă programarea pe componente. Pentru componentele de log și grafică s-au stabilit interfețe. Aceste interfețe sunt: Logger și AgentGui. Mesajele de log sunt afișate nu numai la consolă, dar şi în interfața grafică. Acesta este unul dintre motive pentru care se dorește câte o clasa care se va ocupa de mesajele de log pentru un anumit sistem de operare.

VisualizableAgent folosește o clasă statică PlatformUtils, care detectează platforma. Singura soluţie pe care am considerat-o adecvată pentru detectarea platformei a fost interogarea sistemului pe ce maşină virtuală rulează:

```
if (System.getProperty("java.vm.name").equals("Dalvik")) {
            return Platform.ANDROID;
      }
      return Platform.PC;
```

Maşina virtuală Java „Java HotSpot(TM) Client VM” este diferită de cea pe care rulează aplicațiile Android. Fișierele compilate de Java sunt transformate în fișiere .dex (Dalvik Executable) recunoscute de către Dalvik.

Nu numai VisualizableAgent folosește această clasă statică pentru detectarea platformei , dar și AgentGui. După cum am precizat mai sus AgentGui este o interfață care este implementă de către clasele care definesc interfețele grafice. Aceasta are o funcție *setGuiClass* care reține ce clasă este folosită de către agent pentru interfața grafică. Această funcție este apelată de către VisualizableAgent de fiecare dată când resetează elementele de grafică.

Comportamentul agentului software deține funcționalități care să permită transmiterea sau recepția de mesaje. Funcțiile de handleInput şi handleOutput accesează connectInput şi doOutput ale obietului AgentGui din VisualizableAgent. A fost nevoie ca în ClaimAgent să se suprascrie funcţia de resetVisualization pentru a fi posibilă accesarea elementelor din cadrul AgentGui-ului după ce agentul se mută.

5.2 Mediul grafic al Agenților

Un agent software din tATAmI are următoarele nivele:

- ClaimAgent – ClaimDefinition, folosește specificațiile S-CLAIM;
- HierarchicalAgent – administrează relațiile ierarhice;
- VisualizableAgent – administrează elementele grafice ale agenților și mesajele de log;
- Jade.AgentGui – interfață care este implementă de către clasele care definesc interfețele grafice ale agenților;

Nivelele anterioare sunt prezentate în mod descrescător, ClaimAgent, cel mai de sus, este clasa de bază care definește agentul software în scenario.xml. ClaimBehavior este clasa care extinde Behavior și definește comportamentul agenților Acesta are rolul de administrarea cunoștințelor, a mesajelor, etc.

Elementele de grafică ale agentului software depind de sistemul de operare pe care se execută. Grafica pe PC diferă mult de cea de pe dispozitivele mobile cu sistem de operare Android. Un element de grafică pe Android este de tipul android.View. Un View ocupă o suprafață dreptunghiulară pe ecran și este responsabil pentru desenare și tratarea evenimentelor. Pe PC interfață grafică este de tip swing și conține elemente java.awt.Component.

Diferența între Swing și Activity:

Swing	Activity
Clasa extinde JFrame	Clasa extinde Activity
O clasă JFrame se instanţiază	O clasă Activity nu se instanţiază. Se definește un fișier AndroidManifest.xml în care se specifică clasele Activity
Componentele se instanţiază	Componentele nu se instanţiază
Se folosesc elemente de tip Container pentru organizarea componentelor grafice	Pe Android se definește un fișier xml care va conține așezarea pe ecran a componentelor, împreună cu tipul lor, dimensiunea și id-ul unic
Componentele pot fi accesate prin referințe la instanțele lor	Componentele într-un Activity se pot accesa folosind id-ul unic definit în fișierul xml
O clasă JFrame poate instanța o nouă clasă JFrame care va depinde de prima	Un Activity folosește elemente de Intent pentru a crea un nou Activity și de a comunica cu acesta

Tabel 1. Diferența între Swing și Activity.

După cum se vede în tabelul1 diferențele între cele două sunt mari. Sistemul de operare Android prezintă elemente de grafică care nu încarcă procesorul foarte mult. Pentru ca proiectul tATAmI să suporte ambele tipuri de GUI sau folosit componente cât mai generale care comunică cu elementele grafice indiferent de tip.

Elementele de bază ale unei aplicații pe sistemul de operare Android sunt următoarele:

- Activity – prezintă interfață grafică cu utilizatorul;
- Service – nu deține de interfață vizuală, dar de obicei rulează în background pentru o perioada nedefinită de timp;
- Broadcast Receiver – o componentă care se ocupă de mesajele difuzate;
- Content Provider – face ca un set de aplicații să fie disponibile pentru alte aplicații;

O aplicație pe sistemul de operare Android va conține doar elemente de tipul celor de mai sus. Majoritatea aplicațiilor pe telefoane sunt de tipul Activity, adică prezintă mai mult elemente de grafică. Aceasta a contribuit la organizarea elementelor grafice pe stări, care să cuprindă întreaga arie de funcționalitate a acestora. Stările unui Activity sunt prezentate în figura 5.1.

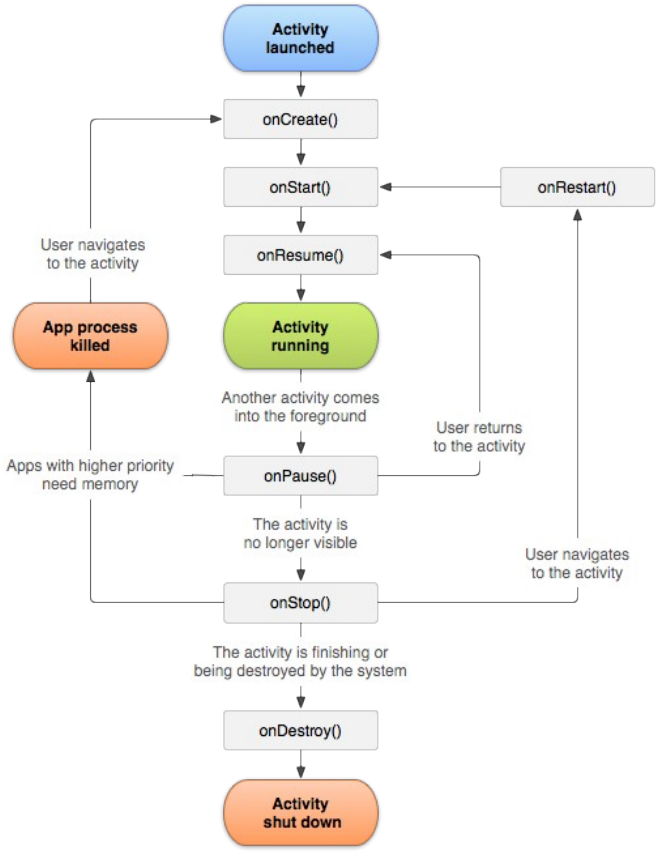

Fig.5.1 Stările unui Activity.
Prin pătrate se reprezintă metodele de tip callback care sunt implementate de către programatori. Elementele ovale reprezintă stările majore ale unui Activity[13].

O problemă majoră a fost faptul ca un Activity nu se poate porni decât dintr-un alt Activity. Pentru ca un agent care se mută pe Android să poată avea o interfață grafică, s-au folosit elemente de Java Reflection. Atunci când telefonul se conectează la platformă, se pornește un Activity care va crea interfață grafică atunci când se mută agentul. În figura următoare este prezentată metoda care pornește un nou Activity din ManagerActivity.

```
public static void newActivity(){

    Intent intent = new Intent();
    intent.setClass(context, AgentGuiActivity.class);
    intent.addFlags(Intent.FLAG_ACTIVITY_SINGLE_TOP);

    context.startActivity(intent);
}
```

Fig5.2. Metoda care pornește un nou Activity din ManagerActivity.

Iar în VisualizableAgent, în metoda de *resetVisualization,* se folosește următoarea secvență de cod:

```
Class<?> clasmain = null;
clasmain =  cl.loadClass(managerActiviy);

if(clasmain != null)
{
    Method meth = clasmain.getDeclaredMethod(neActivityMethod);
    meth.invoke(null);
}
```

Fig.5.3 Pornirea unui nou Activity din VisualizableAgent.

5.3 Exemplificări. Comportamentul Agenților

Comportamentul agentului software a fost proiectat astfel încât să fie flexibil și adaptiv la context. El trebuie sa se poată executa pe orice mașină, indiferent de proprietățile ei. Comportamentul general al agentului este asemănător cu cel al agenților cognitivi: în ciclul de execuție, agentul procesează mesajele care vin, formează noi cunoștințe, iar apoi își formează un scop și metode pentru a-l obține Agentul software acționează pentru a-și atinge scopul în funcție de cunoștințele sale.

La începutul fiecărui ciclu de viată, agentul verifică mesajele primite și adaugă fapte în baza de cunoștințe, dacă acestea sunt noi. Agentul deduce faptul că transmițătorul cunoaște faptele. Aceste deduceri contribuie la informarea agentului software despre vecinii săi.

În etapa următoare agentul își formează o listă de scopuri. Exista două tipuri de scopuri pe care agentul le poate avea: *informarea* altor agenți software despre un anumit lucru sau *eliberarea* unor capacități de stocare. Fiecare scop are asignat câte un grad de importanță, scopul cel mai important devine intenție *Eliberarea* este considerată un scop deoarece se dorește ca agentul să conțină o reprezentare uniformă a acțiunilor pe care agentul le poate executa. În scenariul prezentat mai sus, agenții vor avea doar scopuri de tip informare.

După cum s-a precizat anterior, comportamentul agenților software este definit într-un fișier .adf2. Cele trei tipuri de agenți vor avea doar comportamente reactive și anume:

- PDAAgent:
    - register – primește un mesaj de la EmissaryAgent și îl adaugă în baza de cunoștințe;
    - joinGroup – utilizatorul va trimite o comandă agentului pentru a se înregistra la un grup. PDAAgent trimite agentul EmissaryAgent pe containerul grupului respectiv
    - addOpinion – agentul va trimite o opinie sub formă de mesaj agentului EmissaryAgent;
    - deleteOpinion – agentul va trimite un mesaj emisarului său pentru ca acesta să șteargă o opinie.
    - clearOutput – va șterge toate opiniile;
    - displayOpinion – afișarea opiniei pe un ecran al agentului.

- EmissaryAgent:
  - joinGroup – va primi un mesaj de la PDAAgent pentru a se înregistra la un anumit grup;
  - leaveGroup – va primi de la PDAAgent un mesaj pentru a se de înregistra de la un grup;
  - addOpinion – va primi un mesaj de tip opinie de la PDAAgent și-l va trimite agentului părinte;
  - deleteOpinion – va primi un mesaj de ștergere a unei opinii de la PDAAgent și va trimite agentului părinte această înștiințare;
  - sendAllOpinions – va primi o comandă de a transmite toate opiniile acumulate. De obicei acest comportament are loc atunci când se mută într-un nou grup;
  - refresh – primește o comandă de a reîncărca opiniile;
  - displayOpinion – primește un mesaj de la PDAAgent pentru afișarea opiniei. Agentul trimite acest mesaj părintelui său.
- GroupCoordonatorAgent
  - register – primește un mesaj de la un EmissaryAgent și îl adaugă ca copil;
  - unregister – primește un mesaj de la un EmissaryAgent și îl șterge din lista de copii;
  - refreshOpinions – primește un mesaj de la un EmissaryAgent pentru a reîncărca opiniile pe ecran;
  - receiveOpinion – primește un mesaj de tip opinie de la un EmissaryAgent;

În S-CLAIM un comportament se definește prin cuvântul: *behavior*. În continuare este prezentat comportamentul agentului Coordonator, descris mai sus:

```
(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)
                 .....
     (reactive receiveOpinion
         (receive opinion ?name ?opinionId ?opinionTag ?opinionMsg)…)))
```

CAPITOLUL 6. Instalarea și configurarea aplicației

În continuare se va specifica elementele necesare pentru instalarea și rularea aplicației Proiectul a fost implementat în limbajul de programare Java pentru PC și Android. În secțiunea următoare se va specifica ce trebuie instalat pe un PC cu sistem de operare Windows7 pentru a rula aplicația.

6.1 Instalare.

Aplicația a fost implementată și testată pe un sistem de operare Windows7. Pentru rularea aplicației pe sistemele de operare Linux este necesar instalarea pachetelor suportate de către acestea.

Cerinţe:

- Java SDK, versiunea 1.6 sau mai mare;
- Android SDK versiunea 2.3.3;
- Variabile de mediu: JAVA_HOME, adb;
- AVD manager pentru eclipse;

Rularea:

- Pornirea emulatorului de Android din AVD Manager;
- Rularea scriptului tATAmI-Android/run.bat;
- Rularea Boot.java ca Java Application;
- Rularea tATAmI-Android ca AndroidAplication:
    - IP-ul mașinii pe care rulează platforma JADE;

O imagine a containerului JADE și cu agenții de start este prezentată în figura 6.1. Inițial se pornește platforma JADE împreună cu agenții Visualizer și Simulator cu interfețele grafice aferente.

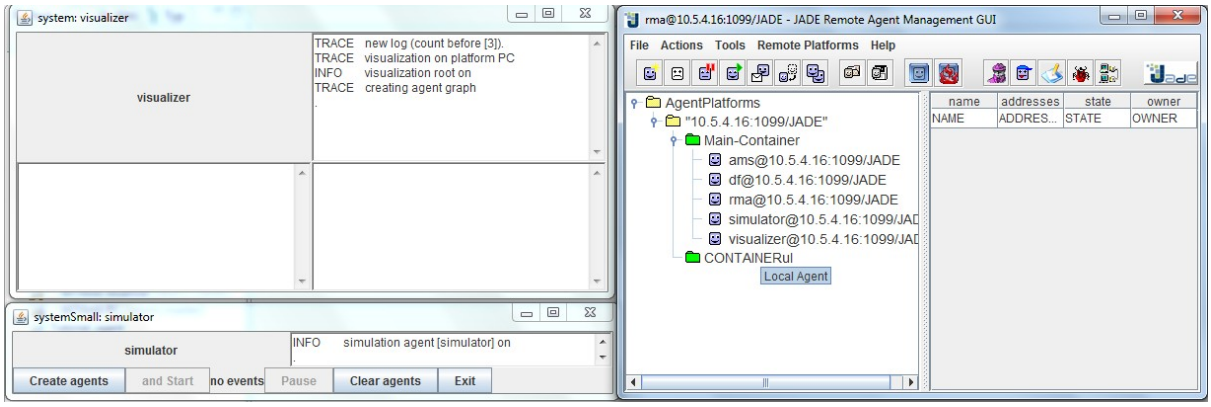

Fig. 6.1 Pornirea platformei.

6.2 Cum se folosește aplicația?

După ce a fost pornită platforma, prin rularea proiectului tATAmI-PC, se rulează proiectul tATAmI-Android. Această ordine de rulare este importantă pentru cazul în care aplicația de pe Android va crea un container pe platforma JADE. După rularea proiectului tATAmI-Android se va vedea pe emulator interfața de start. Se va introduce IP-ul mașinii pe care rulează tATAmI-PC, apoi "use". Se va observa pe platforma JADE ca s-a creat un container. Rezultatul rulării aplicației trebuie să arate ca în următoarele figuri pentru un scenariu cu doi agenți, unul pe Android și unul pe PC.

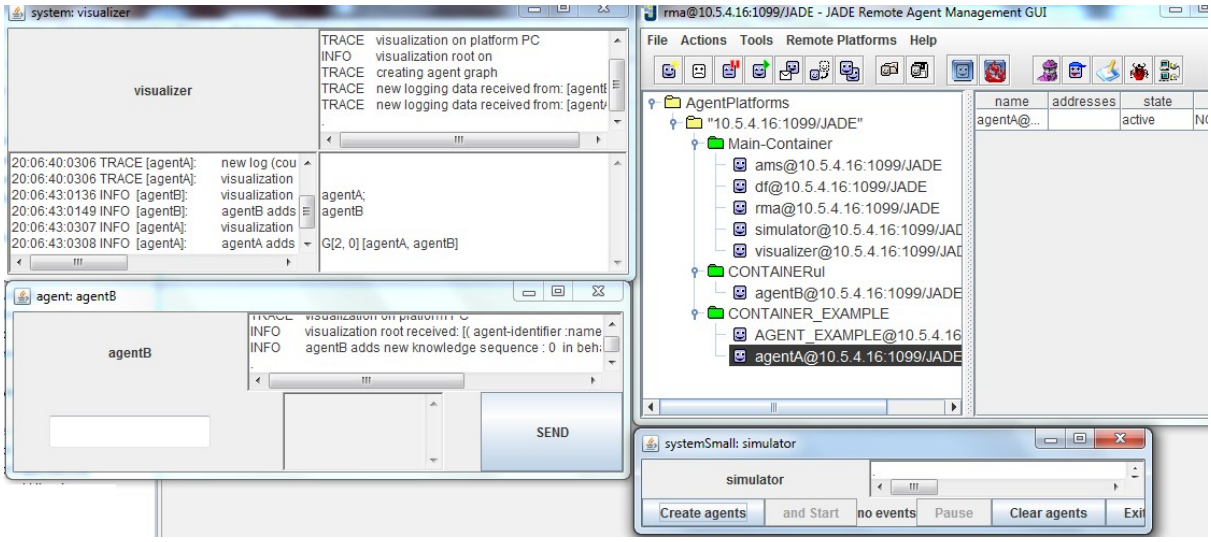

Fig.6.2 Execuția proiectului.

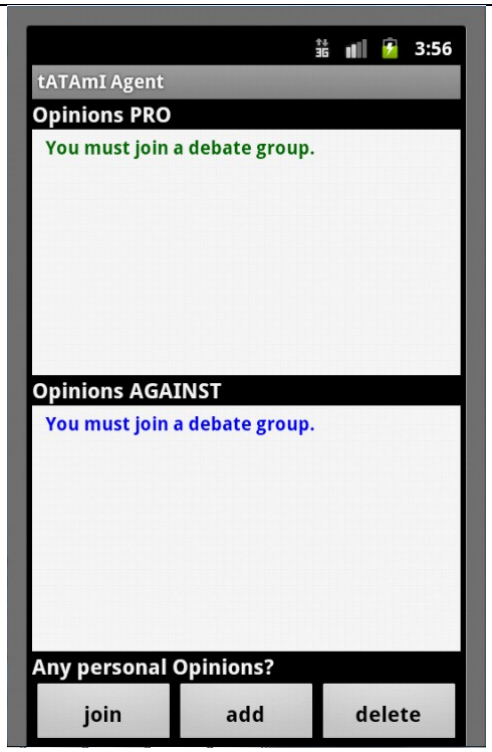

Fig 6.3 Interfața grafică a agentului pe Android

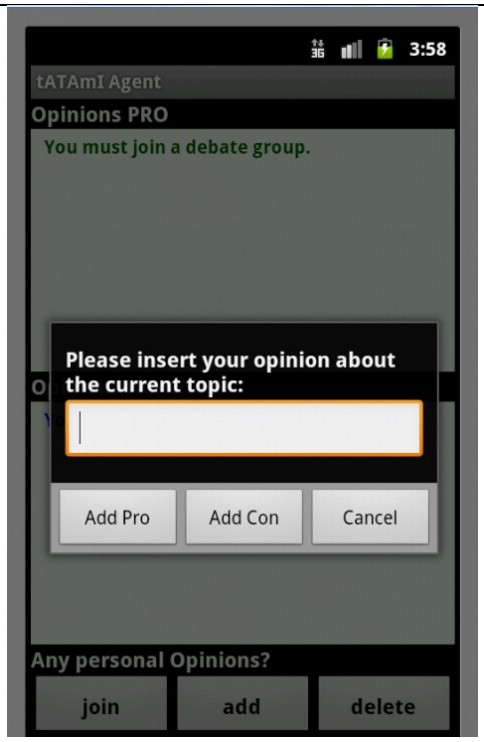

Fig 6.4 Adăugarea unei opinii.

Figura 6.3 reprezintă interfața grafică pe Android a agentului, iar în Fig 6.4 este reprezentat modul de adăugare a unei opinii de către utilizator.

6.3 Utilizarea emulatorului

În acest capitol am prezentat rularea aplicației pe un emulator. Pentru rularea aplicației pe telefon este necesară doar instalarea proiectului și folosirea lui după cum s-a prezentat mai sus. Rularea scriptului tATAmI-Android/run.bat este necesară pe emulator, deoarece acesta face un port forward pentru ruterul virtual al emulatorului pentru a fi posibilă transmiterea de mesaje între agenți.

CAPITOLUL 7. Concluzii

tATAmI este o platformă MAS pentru AmI, deschisă pentru îmbunătățiri, este bine structurată și se bazează pe componente cu un grad ridicat de generalitate.

Modelul sistemelor multi-agent dependente de context pentru AmI propus de către dezvoltatorii proiectului tATAmI este bazat pe trei elemente importante: comportamentul agenților, topologia sistemului și reprezentarea contextului.

Portarea platformei tATAmI pe sistemul de operare Android a fost realizată cu succes. Datorită contribuţiei aduse am reuşit să descoperim buguri în platformă care au fost corectate.

Dezvoltări ulterioare

Această lucrare este doar o extindere a platformei tATAmI. Multe căi pentru dezvoltarea conceptelor introduse în această platformă rămân deschise. Implementările care au fost realizate pot fi îmbunătățite și extinse, se pot crea noi scenarii pentru explorarea execuției agenților software pe Android.

Scenariul prezentat în această lucrare este unul simplu, folosit în demonstrarea posibilității agenților software de a se executa pe Android. Pe viitor se pot dezvolta scenarii mult mai complexe şi cu funţionalităti mult mai extinse.

Pe viitor se pot dezvolta noi funcţionalităţi care vor permite implementarea automată a interfețelor grafice pe sistemul de operare Android și PC pe baza unor descrieri.

BIBLIOGRAFIE

[1] Dante I.T, Ajith A., Juan M.C., Richardo S.A., Agents and ambient intelligence: case studies, 2009
[2] Andrei Olaru, A Context-Aware Multi-Agent System for AmI Environments, Teza de doctorat.
[3] Artificial Intelligence: A Modern Approach by Stuart Russell and Peter Norvig, c 1995 Prentice-Hall, Inc.
[4] Roman T., MACS 404 Lecture Notes Artificial Intelligence.
[5] Lynda A., Holly S., Inteligents Agents in Computer and Network Management
http://teachnet.edb.utexas.edu/~Lynda_abbot/webpage.html#intagt1
[6] R. S. Silva Filho. The Mobile Agents Paradigm. ICS221 - Software Engineering. Final Paper (Winter 2001).
[7] Feng W., Xiaoping C., Onlineplanning for multi-agent systems with bounded communication, Artificial Intelligence, Volum 175, Pagini 441-790 (Februarie 2011)
[8] Verónica V., Javier C., José M.M., An Ambient Inteligent Platform based on Multi-Agent System, XIII Workshop of Phisical Agents, 2012
[9] Agent-Oriented programming. Yoav Shoham.
[10] Siteul official la FIPA (http://www.fipa.org/, Accesat : Iunie 2012)
[11] Tomáš Poch, FIPA specification and JADE(Mai 2005)
[12] Andrei Olaru, Adina Magda Florea, Context-aware agents for developing AmI applications, Journal of Control Engineering and Applied Informatics CEAI, Volum 13, nr. 4 (Decembrie 2011)
[13] Android Development Page(http://developer.android.com/reference/android/app/Activity.html, Accesat: Iunie 2012)
[14] Fabio B., Giovanni C., Dominic G., Developing Multi-Agent System with JADE, 2007
[15] Parineeth M.R, Mobile Agents. Intelligent Assistants on Internet(Iulie 2002)