JADE LEAP

Avram Andrei
Bacanu Lorena
Iordache Ancuta

05.2010

Cuprins

0.1 Introducere

Platforma JADE (Java Agent DEvelopment Framework) dezvoltata de Telecom Italia Lab (TILAB) in conformitate cu specificatiile FIPA (Foundation for Intelligent Physical Agents) este un nivel intermediar (middleware) scris in intregime in tehnologie Java, care simplifica implementarea sistemelor multi-agent prin oferirea unui set de unelte grafice care sprijina fazele de depanare si punere in practica (deployment) ale proiectului. Agentiii pot fi distribuitii pe mai multe masini de calcul, care pot rula sisteme de operare diferite, intreaga configuratie de agenti putand fi controlata de la distanta dintr-o consola grafica.

LEAP(Light Extensible Agent Platform) este o extensie JADE capabila sa ruleze pe dispozitive wireless si PDA-uri ca telefoanele mobile si calculatoare PALM. Platforma de agenti JADE-LEAP se bazeaza pe Java si ofera suport pentru dezvoltarea rapida de aplicatii peer-to-peer.

Capitolul 1

JADE

JADE este un set de clase Java care permit dezvotarea de sisteme multiagent conform FIPA destul de usor. Acesta ofera un set de unelte grafice care faciliteaza sarcina complexa, de punere in aplicare un sistem multi-agent sistem.

Biblioteca Jade contine numeroase biblioteci Java, facilitand astfel implementarea functionarii si a interfetelor abstracte. Java a fost limbajul de programare ales, din cauza numeroaselor sale caracteristici, concepute in special pentru programarea orientata pe obiecte distribuite in medii eterogene; unele dintre aceste caracteristici sunt serializarea obiectelor, Reflection API si Remote Method Invocation (RMI).

JADE este compus din urmtoarele pachete principale:

- jade.core implementeaz kernel-ul sistemului. Acesta include clasa de agent care trebuie s fie extinsa de ctre programatorii aplicatiei. n plus, o ierarhie a clasei Behaviourl este continut in subpachetul

- jade.core.behaviours. Comportamentele implementeaza sarcinile, sau inteniile unui agent. Acestea sunt uniti logice active care pot fi compuse n diverse moduri pentru a realiza complexe modele de executie si care pot fi executate simultan. Programatorii aplicatiei definesc actiunile agentului implementand comportamente si cai de executie ale agentului ce le leaga.

- Jade.lang.acl este furnizat pentru a procesa Limbajul de comunicare al Agentilor (ACL) in conformitate cu specificatiile standard FIPA.

- Jade.content contine un set de clase pentru a sprijini ontologii definite de utilizator si limbile continutului. In special jade.content.lang.sl contine codecul SL, atat parser-ul si codificatorul.

- Jade.domain contine toate clasele Java care reprezinta entitatile de Gestionare a Agentilor, definite de standardul FIPA, in special AMS si agenti DF, ce furnizeaza ciclul de viata, serviciile de white and yellow pages. Subpachetul jade.domain.FIPAAgentManagement contine Ontologia FIPA de Gestionare a Agentilor si toate clasele ce reprezinta conceptele sale. Subpachetul jade.domain.JADEAgentManagement contine, extensiile JADE pentru Gestiunea Agentilor (de exemplu, pentru procesarea mesajelor, controlul ciclului de viata al agentilor, ..). Subpachetul jade.domain.introspection contine concepte utilizate pentru domeniul comunicarii dintre instrumentele JADE (de exemplu, Sniffer i Introspector) i kernel-ul Jade. Jade.domain.mobility contine toate conceptele utilizate pentru a comunica despre mobilitate. Jade.gui pachet contine un set de clase generice utile pentru a crea interfete grafice de afisare (GUI) pentru a afisa si modifica identificatorii agentilor, Descrierile agentilor, mesajele ACL.

- Jade.mtp contine o interfata Java ce trebuie implementata de fiecare protocol de comunicatie pentru a putea fi integrat in platforma JADE, si implementarea unui asemenea set de protocoale.

- jade.proto este pachetul care contine clasele ce modeleaza protocoalele standard de interactiune(de exemplu, FIPA-request, FIPA-query, FIPA-contract-net, FIPA-subscribe i altele definite de FIPA), si de asemenea si clase ce ajuta programatorii sa implementeze propriile protocoale.

- Pachetul FIPA contine modulul IDL definit de FIPA pentru transportul IIOP al mesajelor.

- pachetul jade.wrapper prevede "wrapper"-e a functionalitatii nivelului superior JADE care permite utilizarea JADE ca o biblioteca, in cazul in car aplicatii externe Java lanseaza agenti Jade si containere de agent.

JADE este furnizata impreuna cu unele instrumente care simplifica administrarea platformei si dezvoltarea aplicatiei. Fiecare instrument este continut intr-un sub-pachet din jade.tools. In prezent, urmtoarele instrumente sunt disponibile:

- Remote Management Agent, RMA pe scurt , actioneaza ca o consola grafica pentru platforma de gestionare i control. O prima instanta a RMA se poate fi initiata cu o optiune a liniei de comanda ("-gui"), dar apoi poate fi activata mai mult de o singura interfata grafica. Consola RMA este capabila de a porni alte instrumente JADE.

- Dummy Agent este o unealta de monitorizare si debugging, compusa dintr-o interfata grafica si un agent JADE la baza. Folosind GUI-ul se pot construi mesaje ACL si trimise la alti agenti, este de asemenea posibila afisarea listei de mesaje ACL trimise sau primite, completate cu informatii ale timplori de actiune pentru a permite inregistrarea comunicarii si repetarea.

- Agentul Sniffer este un agent ce poate intercepta mesajele ACL in timp ce sunt transmise si a le afisa graphic folosind o notatie similara diagramelor de secventa UML. Este folositor pt depanarea unui grup de agenti prin observarea modului de schimb al mesajelor ACL.

- Introspector este un agent ce permite monitorizarea ciclului de viata al unui agent, mesajele ACL comunicate si comportamentul acestuia in executie.

- DF GUI este o interfata grafica a utilizatorului care este folosita de DF-ul JADE.

- LogManagerAgent permite setarea in timpul executiei a informatiilor de logare ca nivelul de logare, pentru platforma JADE si clasele specifice aplicatiei ce folosesc Java Logging.

- SocketProxyAgent este un agent simplu, ce se comporta ca un pasaj intre platforma JADE si o conexiune normala TCPIP. Mesajele ACL comunicate prin serviciul de transport al JADE sunt convertite in stringuri ASCII si trimise prin conexiunea bazata pe socluri.Acest agent este folosit in controlul firewall-ului retelei si pentru a furniza interactiunea platformei cu applet-urile Java dintr-un browser.

1.1 Caracteristici JADE

- Platforma distrbuita de agent. Platforma de agenti poate fi impartita in mai multe gazde. Doar o singura aplicatie Java , si de altfel o singura Masina Virtuala Java, sunt executate pe fiecare gazda. Agentii sunt implementati ca fire de executie Java si sunt inclusi in containere pentru agenti ce furnizeaza suportul la executarea agentilor.

- Interfata Grafica pentru a controla diversi agenti si containere prin comunicarea la distanta(remote).

- Unelte de depanat pentru facilitarea implementarii de aplicatii multi agent JADE.

- Mobilitatea agentilor intre platforme, ce include transferul starii si a codului agentului.
- suport pentru executarea a mai multor activitati paralele si simultane ale agentilor prin modelul comportamentului.
- in conformitate cu specificatiile FIPA, include AMS(Agent Management System) si DF(Directory Facilitator). Aceste componente sunt activate la pornirea platformei.
- Pot fi pornite mai multe DF-uri la run time pentru a implementa aplicatii multi domain, unde un domeniu este un set de agenti ale caror servicii sunt publicate prin intermediul unui facilitator comun. Fiecare DF mosteneste GUI-ul si actiunile standar conforme FIPA(inregistrarea, dezinregistrarea, modificarea si cautarea descrierilor agentilor si capacitatea de asociere intr-o retea de df-uri.)
- Transport eficient al mesajelor ACL in interiorul aceleiasi platforme. Mesajele sunt transmise criptate ca obiecte Java si nu ca stringuri.
- Pachetul de protocoale de comunicare FIPA
- Inregistrare si Dezinregistrare a agentilor folosind AMS.
- Serviciul de generare GUID : la pornirea platformei fiecare agent obtine propriul GUID.
- suport pentru ontologii
- Interfata InProcess ce permite aplicatiilor externe de a lansa agenti autonomi.

1.2 Model arhitectural

JADE-LEAP, la fel ca JADE, are o arhitectura distribuita compusa din elemente numite containere ce pot rula pe diferite gazde si stoca agenti in ele. Agentii sunt componente active , fiecare are un nume unic si comunica bidirectional cu alti agenti, interschimband mesaje conform modelului de comunicare FIPA. Fiecare agent apartine unui container (care ii asigura timpul de executie) si doar un singur container are rolul principal(AMS, DF).Containerul principal poate fi replicat prin serviciile de replicare.

Agentii software nu comunica intre ei cu ajutorul metodelor ci prin mesaje de tip asincron. Toate mesajele ce sunt comunicate intre agenti trec printr-un

container primitor si unul transmitator(sender si receiver) inainte de a intra in lista de mesaje. In acest fel , cel ce trimite trebuie sa specifice doar numele primitorului, in timp ce maparea mesajului si a numelor sunt asigurate de container.

1.3 Clasa Agent

Un tip de agent este creat extinzand clasa jade.core.Agent si redefinind metoda setup().

```
import jade.core.Agent;

public class HelloWorldAgent extends Agent
{
       protected void setup()
       {
              System.out.println("Hello World. My name is "+this.getLocalName());
        }
}
```

Un agent poate fi pornit indirect prin clasa jade.Boot ce proceseaza parametrii de start. Descrierea unui agent e reprezentaat de 2 atribute: numele agentului si numele clasei pe care o reprezinta agentul respectiv.

Fiecare instanta a acestei clase este identificata de un AID (jade.core.AID), compus dintr-un nume unic si cateva adrese. Acest AID este obtinut prin apelul metodei getAID() din clasa Agent. Numele agentului are forma <local_name>@<platform_name >. Numele complet al unui agent trebuie sa fie unic global (GUID), numele initial al unei platforme este <mainhost>:<mainport>JADE, insa i se poate atribui un alt nume prin optiunea -name.

Intr-o singura platforma JADE se face referire la agenti doar prin numele lor. Cunoscand numele unui agent , AID-ul acestuia poate fi creat in felul urmator:

- AID id = new AID(localname, AID.ISLOCALNAME);
- AID id = new AID(name, AID.ISGUID);

Se pot transmite argumente agentilor prin metoda getArguments(). Daca un agent necesita informatii din linia de comanda, se poate extinde clasa precendenta in felul urmator:

```
import jade.core.Agent;
public class HelloWorldAgentWithParameters extends Agent {
    private String service;
    protected void setup()
   {
         Object[] args = getArguments();
         service = String.valueOf(args[0]);
```

```
        System.out.println("Hello World. My name is "+this.getLocalName()+
" and I provide "+service+"service.");
    }
}
```

Acest exemplu arata cum se proceseaza parametrii de pornire. In atributul privat service se stocheaza argumentul de pe prima pozitie din obiectul args a carui valoare este returnata de metoda getArguments().

Agentii folosesc ACL pentru comunicare, si fiecare mesaj e reprezentata de clasa jade.lang.acl.Message. La inceput , agentul SenderAgent trimite un mesaj agentului cu numele Ghita.ReceiverAgent asteapta mesajele si, in caz ca primeste vreun mesaj il tipareste in consola.Clasa SenderAgent este foarte simpla,creeaza o instanta a ACLMessage si instantiaza atributele receiver si content. La sfarsit este apelata metoda sendI(ACLMessage m).

```
import jade.core.AID;
import jade.core.Agent;
import jade.lang.acl.ACLMessage;
public class SenderAgent extends Agent
{
     protected void setup()
     {
          System.out.println("Hello. My name is "+this.getLocalName());
          sendMessage();
      }
     private void sendMessage()
    {
          AID r = new AID ("Ghita@"+getHap(), AID.ISGUID );
          ACLMessage aclMessage = new ACLMessage(ACLMessage.REQUEST);
          aclMessage.addReceiver(r);
          aclMessage.setContent("Hello! How are you?");
          this.send(aclMessage);
     }
}
```

Primirea mesajelor este putin mai complicata decat trimiterea acestora. Trebuie implementate comportamente responsabile cu procesarea mesajelor primite.Acest comportament trebuie adaugat in lista de comportamente in metpoda setup().

```
import jade.core.Agent;
public class ReceiverAgent extends Agent
{
     protected void setup()
    {
           System.out.println("Hello. My name is "+this.getLocalName());
          addBehaviour(new ResponderBehaviour(this));
     }
}
```

Un agent este distrus la apelul metodei doDelete(). Analog metodei setup( ) care este apelata la initializarea unui agent, exista si o metoda care va fi folosita la finalizarea executiei sale : takeDown().

1.4 Clasa Behaviour

Comportamentul unui agent consta in ceea ce el executa. Acesta este creat prin extinderea clasei jade.core.behaviours.Behaviour. Pentru a face un agent sa execute un task este suficienta crearea unei instante a clasei Behaviour si apelarea metodei addBehaviour() a clasei Agent.Fiecare subclasa Behaviour trebuie sa implementeze:

- public void action(): ceea ce trebuie executat
- public boolean done(): cand task-ul a luat sfarsit

Un agent poate executa mai multe sarcini in paralel, planificarea taskurilor nu este preemtiva, totul are loc intr-un singur fir de executie Java. Schimbul task-urilor are loc la apelarea metodei action() a task-ului planificat curent.

1.5 Inregistrarea serviciilor

Daca un agent furnizeaza servicii, acesta ar trebui sa le inregistreze la agentul DF. Alti agenti pot cauta aceste servicii si sa le foloseasca in rezolvarea problemelor lor. Inregistrarea serviciilor are loc la pornirea agentului, astfel ca metoda setup reprezinta cel mai bun loc pentru inregistrarea serviciilor.

```
import jade.core.Agent;
import jade.domain.FIPAException;
import jade.domain.DFService;
import jade.domain.FIPAAgentManagement.ServiceDescription;
import jade.domain.FIPAAgentManagement.DFAgentDescription;
public class ProviderAgent extends Agent
{
        private String service;
        protected void setup()
       {
               Object[] args = getArguments();
               service = String.valueOf(args[0]);
               System.out.println("Hello. My name is "+this.getLocalName()+
" and I provide "+service+" service.");
               registerService();
       }
       private void registerService()
      {
              DFAgentDescription dfd = new DFAgentDescription();
             dfd.setName(this.getAID());
             ServiceDescription sd = new ServiceDescription();
             sd.setType(service);
             sd.setName(service);
            dfd.addServices(sd);
            try
            {
                 DFService.register(this, dfd);
            }
            catch (FIPAException e)
```

```
            {
                  System.err.println(getLocalName() +" registration with DF unsucceeded. Reason: "
+ e.getMessage());
                  doDelete();
            }
      }
}
```

Metoda registerService() creaza o instanta a clasei DFAgentDescription in care stocheaza datele si o trimite agentului DF folosind metoda DFService.register(Agent a, DFAgentDescription d). Daca are loc vreo eroare inregistrarea va arunca o exceptie si agentul ca tipari mesajul de eroare in consola si se distruge apeland metoda doDelete().

1.6 Cautarea de servicii

Daca un agent si-a inregistrat serviciile, un alt agent le poate cauta, ca in exemplul urmator:

```
import jade.util.leap.List;
import jade.util.leap.Iterator;
import jade.core.Agent;
import jade.domain.FIPAException;
import jade.domain.DFService;
import jade.domain.FIPAAgentManagement.ServiceDescription;
import jade.domain.FIPAAgentManagement.DFAgentDescription;
import jade.domain.FIPAAgentManagement.SearchConstraints;
public class SearchAgent extends Agent
{
        protected void setup()
       {
              System.out.println("Hello. I am "+this.getLocalName()+".");
             this.searchAgents();
        }
        private void searchAgents()
       {
             DFAgentDescription dfd = new DFAgentDescription();
             SearchConstraints c = new SearchConstraints();
             doWait(2000);
             try
             {
                  DFAgentDescription[] result = DFService.search( this, dfd);
                  for( int i=0; i<result.length; i++)
                  {
                         String out = result[i].getName().getLocalName()+"@"+getHap()+" provide";
                         Iterator iter2 = result[i].getAllServices();
                        while (iter2.hasNext())
                        {
                              ServiceDescription sd = (ServiceDescription)iter2.next();
                              out += " "+sd.getName();
                        }
                        System.out.println( this.getLocalName()+": "+out);
                  }
            }
            catch (Exception fe)
           {
```

```
                    System.err.println(getLocalName() + " search with DF is not
succeeded because of " + fe.getMessage());
                    doDelete();
                }
        }
}
```

Metoda searchAgents() trimite cererea de cautare la DF. Daca nu sunt specificati ca parametri de cautare DF va returna toti agentii inregistrati.Rezultatul este tiparit in consola.

Capitolul 2

JADE LEAP

2.1 Cerintele agentior destinati dispozitivelor mici

Micile dispozitive portabile sunt extrem de diversificate variind de la cartele SIM la telefoane mobile, telefoane inteligente si PDA-uri. Chiar daca exista diferente sensibile intre ele, toate aceste tipuri de de dispozitiv au un anumite limite in raport cu PC-urile si serverele.

- Putere de procesare redusa adesea cu procesoare pe 16 biti si frecvente mai mici de 200MHz.
- Limitari mari de memorie : mai putin de 64kb la SIM-ur, de 19Mb la telefoane si mai putin d 64Mb pentru PDA-uri.
- Memorie permanenta limitata, fara sistem de fisiere valabila la telefoane.
- Durata de viata a bateriei limitata.
- Intreruperi de conexiuni datorita ariilor fara acoperire, spatii inchise si nevoia de a inchide dispozitivul pentru a economisi energie.
- Latenta mare a retelei si lungime de banda mica variind intre 9.6kbps(GSM) si 128kbps(GPRS).
- Ecrane de dimensiuni mici
- Mecanisme de intrare ca tastatura numerica.

Avand in vedere aceste limitari, agentii dezvotati pentru aceste dispozitive vor trebui sa fie construiti astfel incat sa ajunga la un echilibru intre memoria mica si algoritmii eficient.Aceasta problema depinde si de dipsozitivul ales si caracteristicile lui.

- Nu pot fi lansati agenti de o complexitate mare, cu o baza de date mare si algoritmi de deductie puternici pe astfel de dispozitive cat prea curand

Cu toate acestea, exista PDA-uri si telefoane inteligente care au resurse suficiente in ceea ce priveste memoria si puterea de procesare pentru lansarea agentilor cu un anumit grad de inteligenta si autonomie. Cu toate ca au dimensiuni si resurse reduse, agentii de pe dispozitivele portabile pot furniza o valoare aditionala mare atat timp cat sunt:

- autonomi in a executa task-uri lungi si complexe fara a fi nevoie de interventia utilizatorului
- comunicativi, mai ales tinand cont de resursele limitate de pe dispozitivele mici, ei nu pot rezolva probleme complexe singuri dar ei vor cauta alti agenti cu care sa coopereze pentru a gasi o solutie.

2.2 Extensia LEAP

Add-onul la JADE, numit LEAP inlocuieste unele parti a kernel-ului JADE, creand un mediu modificat numit JADE-LEAP, care permite punerea in aplicare a agentilor in dispozitive mobile cu resurse limitate. Acesta ofera trei moduri de lucru pentru a se adapteze la diferite circumstante.

- j2se:poate rula pe PC-ur servere din retele fixe ce suporta jdk1.2 sau superior
- Pjava: poate rula pe dispozitive ca PDA-urile care suporta Personal-Java
- MIDP (Micro Edition):poate rula pe dispozitive care suporta MIDP1.0 cum sunt majoritatea telefoanelor mobile compatibile cu Java.

Cele trei versiuni se bazeaza pe acelasi API dar cu mici diferente. Doar cateva caracteristici ce sunt continute in JADE-LEAP pentru j2se si pjava nu sunt compatibile cu JADE-LEAP pentru midp pentru ca sunt intrinsec legate de clasele Java care nu suporta MIDP.

Figure 2.1: JADE LEAP

Din punctul de vedere al programatorilor aplicatiilor si al utilizatorilor, JADE-LEAP pentr j2se este aproape identic cu JADE atat in privinta API-uui cat si a administrarii timpului de executie. Astfel programatorii isi pot lansa agentii JADE pe JADE-LEAP si invers fara a schimba o singura linie de cod. Urmatoarele probleme trebuie luate in considerare:

- la lansarea unui container divizat, un container j2se(posibil, dar nu neaparat containerul principal) trebuie sa fie deja activ pe gazda unde va fi creat BackEnd-ul.
- un container principal nu poate fi divizat.
- mobilitatea si clonarea agentilor nu sunt suportate intr-un container divizat.

2.3 Moduri de executie JADE-LEAP pe dispozitive mobile

Mediul de rulare a JADE-LEAP poate fi executat e dispozitive mobile in doua moduri:

- modul de executie independent (stand-alone) in care un intreg container este executat pe dispozitiv
- modul divizat (split) in care containerul este impartit in FrontEnd(ce ruleaza pe dispoziti) si BackEnd (ce ruleaza pe gazda j2se) , ambele legate printr-o conexiune permananta.Acest mod de executie este potrivit dispozitivelor wirelles si a celor cu resurse limitate deoarece:
  - FrontEndu-ul este mai mic decat un container intreg.

– faza de bootstrap este mai rapida

– mai putini octeti sunt transmisi prin reteaua wireless.

De vreme ce API-urile sunt identice programatorii nu trebuie sa-si faca griji de felul in care va rula agentul,fie pe un container independent sau pe FrontEndu-l unui container divizat. Este recomandata folosirea modului divizat de executie pe dispozitivele MIDP si a modului independent pe dispozitivele ce ruleaza PersonalJava. Atat BackEnd-ul cat si FrontEnd-ul nu contin un AMS sau un DF, functionalitatile acestora sunt furnizate de containerul principal al platformei. FrontEnd-ul este capabil sa detecteze intreruperea conexiunii cu Backend-ul, astfel el incearca sa se reconecteze o anumita perioada de timp. BackEnd-ul se comporta ca un dispecer pentru mesajele trimise de containerele din aceeasi platforma , ale caror destinatii sunt agentii din FrontEnd.

Exemplul urmator contine un container independent ce ruleaza pe un PDA cu Personal Java si un container divizat ce ruleaza pe un telefon MIDP
.

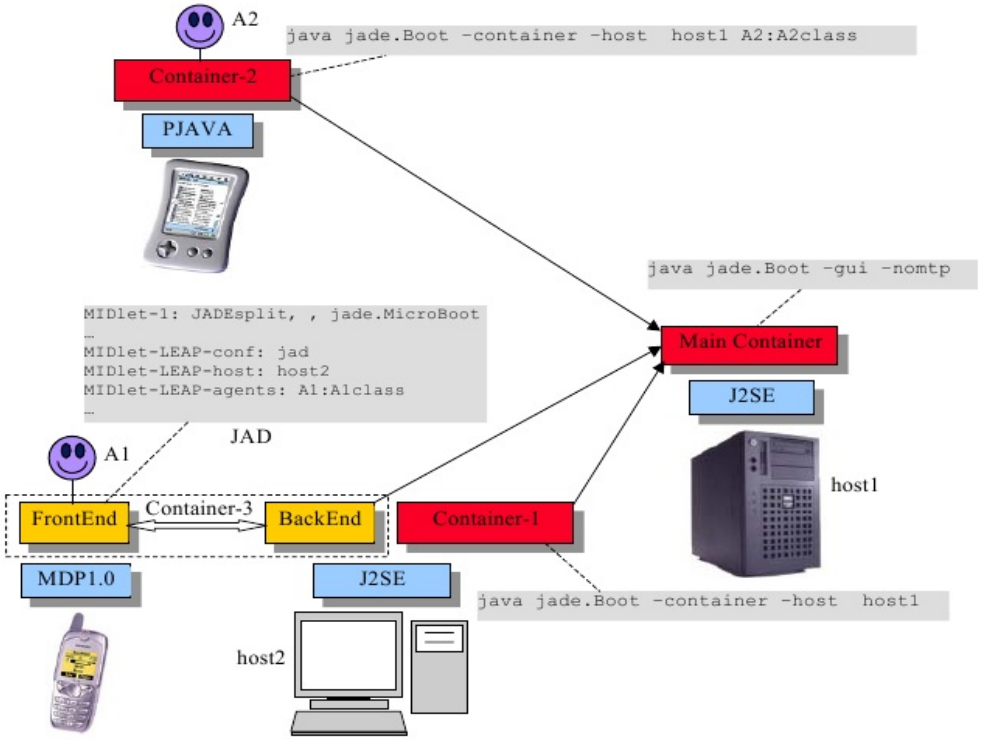

Figure 2.2: Moduri de executie

IMTP(LEAP internal message transport protocol) se bazeaza pe un protocol numit JICP (JADE intercontainer protocol). IMTP contine un dispecer de comenzi care e responsabil cu serializarea/ deserializarea comenzilor JADE, asignand protocolul potrivit (ICP) comenzilor serializate, si comenzilor de rutare primite de la ICP in containerul local. Selectarea unei comunicari depinde de locatia celui agentului ce trimite si a celui ce primeste.

- Comunicarea dintre agentii ce apartin de containere diferite ce apartin aceleiasi platforme folosesc IMTP.
- Daca agentii sunt in acelasi container, mesajul este trimis folosind evenimente. Mesajul nu este serializat, dar clonat si obiectul referinta este transmis agentului primitor.

Acest cadru de dezvoltare ofera multe posibilitati in implementarea de sisteme multiagent distribuite, in care agentii personali pot rula usor pe dispozitive mobile (PDA-uri si chiar telefoane mobile ce dispun de resurse suficiente) si care pot comunica wireless cu agenti care pot furniza servicii valabile intr-un oras(cinema, restaurante, atractii turistice, centre medicale, etc). Acest tip de cadru impreuna cu eforturile de standardizare in campul sistemelor multi agent depuse de organizatiile de genul FIPA, ne imping usor spre un viitor in care asistentii personali vor sti totul despre nevoile utilizatorului si vor oferi informatii personale si servicii utilizatorului si vor deveni in curand indispensabili.

Bibliografie

1. LEAP USER GUIDE, Giovanni Caire (TILAB ex CSELT)
2. Caire, G. LEAP 30 User Guide, TILAB
3. Foundation for Intelligent Physical Agents: http://www.fipa.org.
4. LEAP into Ad-Hoc Networks by Jamie Lawrence, Media Lab Europe
5. JADE-LEAP: http://sharon.cslet.it/project/jade.
6. Programming MAS with JADE, Giovanni Caire (Telecomm Italia LAB,JADE Board Technical Leader) WOA 2004 - Turin
7. Adorni, G., Bergenti, F., Poggi, A., and Rimassa, G. Enabling FIPA agents on small devices. Cooperative Information Agents, 2001 (Modena, Italy). 2001
8. Using JADE-LEAP to implement agents in mobile devices , Antonio Moreno, Ada Valls, Alexandre Viejo