PROJECTE FINAL DE CARRERA

Agents que proporcionen serveis de restauració als visitants de Tarragona dins el projecte *AgentCities*

Albert Gavarró Rodríguez, agr.si@alumne.etse.urv.es
Enginyeria Tècnica en Informàtica de Sistemes

Directors del projecte:
Aïda Valls
Antonio Moreno

Escola Tècnica Superior d'Enginyeria (ETSE)
Universitat Rovira i Virgili (URV)
http://www.etse.urv.es

Curs 2000-2001

Índex

1 Introducció

Imagini's per un moment que està de vacances, en una ciutat exòtica. La qüestió és que la calor apreta i, a aquestes alçades del dia, un ja comença a tenir gana. Ens hauríem d'haver quedat a l'hotel – pensa. De fet, sortir a dinar a fora si tenim en compte la desconeixença total de l'indret i les dificultats amb l'idioma, amb el que es descriuen innumerables plats amb ingredients del tot sospitosos, ho considera, ara, un suïcidi. La seva parella l'observa amb cara de "m'hauries d'haver fet cas a mí"...

Per sort, disposa d'un mòbil del tot *hi-tech*, que li ha costat més del que podia pagar, amb el seu assistent personal executant-se a dins (des de fa temps li diu Jefry per que li recorda els seus temps d'estudiant). A través de la seva interfície, i de forma ràpida i fàcil, ha sol·licitat que li informi dels restaurants que té més a la vora, del que fan pagar i sobretot el que li donaran per menjar. En pocs segons té aquesta informació a la seva pantalla. Ara que ho veu tot allí la situació no li sembla pas tan greu! Ans el contrari: li falta temps per reservar taula, ja, directament des del seu mòbil, en un restaurant on fan el plat preferit de la seva parella.

Aquesta situació, que podria semblar utòpica, no està gaire lluny de la realitat. I tot això gràcies a una nova tecnologia: la programació orientada a agents.

Dins del camp de la IA s'han desenvolupat recentment tot un conjunt de tècniques a mig camí entre la programació orientada a objectes i les tècniques d'enginyeria del coneixement que s'engloben sota el concepte de "programació orientada a agents".

Els agents es podrien considerar com objectes que tenen una autonomia pròpia que els permet prendre decisions i actuar segons les seves creences, desitjos o intencions.

Els sistemes d'agents o Sistemes MultiAgent (SMA) engloben a un conjunt d'agents que tenen objectius comuns. La seva habilitat per comunicar-se els permetrà dividir tasques complexes en components més petites que puguin ser abordades individualment.

Malauradament, com que es tracta d'una tecnologia encara incipient, està orfe d'estàndards que regulin des de l'arquitectura dels sistemes fins a les interaccions entre els agents, incloent llenguatges, protocols... Per resoldre aquestes mancances, es va crear la FIPA (*Foundation for Intelligent Phisical Agents*) [1] que està realitzant una tasca important en la definició d'estàndards destinats a unificar arquitectures, protocols i llenguatges per assegurar la introperablilitat.

Aquesta nova tecnologia ha suscitat prou interès com per que hagin aparegut innumerables projectes que pretenen posar en pràctica els postulats de la FIPA. D'entre ells destaca una iniciativa europea anomenada *AgentCities* [2] que pretén traslladar els serveis que habitualment ens ofereix una ciutat (sanitaris, de restauració...) a un SMA global i uniformement accessible.

Amb l'ànim de contribuir en aquest projecte, el GRUp de Sistemes MultiAgent (GRUSMA) [3] d'aquesta universitat ha volgut implementar un SMA que ofereixi serveis de restauració per la ciutat de Tarragona.

Per portar-lo a terme, s'ha utilitzat l'eina de desenvolupament de SMA JADE [4], recolzada per la FIPA i de la que en segueix totes les especificacions. JADE és de lliure distribució i es pot obtenir en codi obert. A més ens proporciona l'avantatge de ser multiplataforma per recolzar-se en codi Java.

Prestacions del sistema

- El sistema permetrà cercar i, eventualment, reservar taula en un restaurant.

- El sistema permetrà definir una sèrie de restriccions que s'aplicaran al procés de cerca d'un restaurant.

- El sistema ha de proporcionar informació imparcial.

El tercer punt potser necessita un aclariment. Quan dissenyem un sistema que modela l'accés a serveis relacionats amb la restauració, hem de tenir en compte que aquest món engloba molta gent de parés diferents. Mai arribarem a una decisió que satisfaci a totes les parts però hem de ser curosos a l'hora de decidir quina informació publicarà (i quina no) el nostre sistema.

Tipificació de l'usuari potencial

Podem diferenciar dos tipus d'usuaris potencials:

- L'usuari local, amb coneixements sobre l'indret i amb un marge de moviment en les reserves relativament gran. Les possibilitats de que torni al restaurant són notables.

- L'usuari forani, amb un desconeixement total sobre l'indret i amb un marge de maniobra en les reserves estret. Segurament no tornarà.

El primer tipus d'usuari utilitzarà l'aplicació per comoditat. Per tant, l'aplicació ha de ser tant còmode com una trucada per telèfon. El segon l'utilitzarà principalment perquè desconeix l'indret i necessita una guia de restaurants. Aquest usuari valorarà positivament la presència d'informació i la qualitat d'aquesta. A més a més, si la ciutat es gran, li pot interessar una búsqueda geogràfica que li proporcioni els restaurants més propers.

El mecanisme de cerca

La cerca del restaurant és un dels punts més importants del sistema. Definirem els criteris principals que suportarà el nostre mecanisme de cerca:

- **Localització**: Els restaurants es troben en algun lloc de la ciutat. Pot ser interessant gaudir de l'opció de poder restringir la cerca en una determinada zona de la ciutat, que vindrà determinada pel seu codi postal. Tanmateix aquest és un criteri arbitrari que s'ha aplicat a aquesta aplicació en particular. Poden haver-hi perfectament altres aproximacions en que la ciutat es divideixi seguint altres criteris.

- **Preu**: Per experiència és un concepte important en la tria d'un restaurant. Els preus inclouen els preus mitjos del menú i de la carta.

- **Especialitat**: Després de la cerca per preu, és un dels més importants. Per especialitat ens referim al conjunt tipus de cuina i especialitats. Ho fem així per que la barrera entre els dos conceptes és bastant difusa i es confon amb facilitat.

- Altres criteris que últimament comencen a tenir un pes important. Per exemple:

    - **Disponibilitat d'aparcament**

    - **Disponibilitat de menú nocturn**, molt important per alguns usuaris que pernocten a la ciutat i no sopen a l'hotel.

    - **Disponibilitat d'una zona reservada als fumadors**, bastant estès a la resta d'Europa però que aquí encara no ha arrelat

    - **L'acceptació d'una targeta de crèdit en particular**

    - **Capacitat**, un factor a tenir en compte si ens han refusat unes quantes peticions de reserva de taula o si volem gaudir d'una velada una mica més íntima.

Informació susceptible de ser obtinguda

Com hem comentat abans, s'ha intentat fer accessible una sèrie d'informació que considerem que no deixa en desavantatge a cap dels restaurants. També hem procurat que el cost de manteniment per part dels restaurants sigui mínim. En aquest sentit, per exemple, s'ha desestimat la inclusió del menú diari com a possible camp d'informació per la modificació diària que implica. En canvi s'ha inclòs la carta doncs té un cost de manteniment menor i no és tant susceptible a variacions.

La informació que es podrà obtenir de cada restaurant serà la següent:

- **Informació de contacte**: Això inclou el nom, l'adreça, el telèfon, el fax i un camp addicional on cada restaurant pot posar-hi altres mètodes de contacte alternatius: www, e-mail, etc.

- **Preus**: Inclou els preus mitjos del menú i de la carta. Òbviament és suporta la mancança d'un o, pel motiu que sigui, dels dos preus per inexistència del menú (i)/o la carta.

- **Especialitats**: Inclou també el tipus de cuina.

- **Capacitat**

- **Disponibilitat** de **menú nocturn**, d'**aparcament** i/o **zona de fumadors**.

- **Targetes admeses**

- **Carta**, ja sigui tota sencera o només una part.

Estructuració del document

El document consta de tres parts en les que s'estudiaran els agents i Sistemes MultiAgent (SMA), l'eina de desenvolupament JADE i el SMA implementat.

Al capítol 2 s'explicaran els conceptes d'agent i de Sistema MultiAgent, descriurem els diferents tipus d'agents i les seves característiques, veurem els diferents models de SMA i estudiarem el sistema de comunicació entre agents.

El capítol 3 es dedicarà exclusivament a introduir el lector a l'eina de desenvolupament de SMA JADE. Veurem com implementar un SMA sobre aquesta plataforma i farem una repassada a les eines d'administració i monitorització que ens ofereix.

Al capítol 4 es detallarà l'arquitectura del SMA implementat. El manual d'usuari el trobarem al capítol 5.

En el capítol 6 es realitza el joc de proves, analitzant-se els casos més representatius.

Els capítols 7 i 8 presenten les conclusions i valoracions que es poden despendre la realització d'aquest treball.

Finalment, en el capítol 9 es detallen les futures directives de treball.

2 Agents i Sistemes MultiAgent (SMA)

Un agent es la unitat bàsica d'un SMA i es pot considerar com una entitat autònoma que es capaç d'interactuar amb el seu entorn. El seu comportament deriva del conjunt d'objectius o motivacions que té.

2.1 Atributs dels agents

No hi ha un acord en la comunitat científica sobre quins són els atributs que ha de tenir forçosament un programa per a ser considerat un agent. Algunes de les propietats que se li demanen són:

- **Autonomia**: Un agent ha de poder operar de forma autònoma i sense intervenció externa. Això implica la presència d'accions que s'executen periòdicament o de forma espontània o el modelat d'aspectes com la iniciativa. Els agents també han de poder prendre decisions de forma autònoma que beneficiïn a l'usuari.

- **Sociabilitat i cooperació**: Els agents han de poder interactuar amb altres entitats a través d'algun tipus de llenguatge, amb l'objectiu de compartir coneixements, creences i directives de treball per tal de poder solucionar problemes que estan fora de les seves possibilitats individuals.

- **Reactivitat**: Els agents han de poder reaccionar a canvis en el seu entorn en un temps raonable.

- **Iniciativa**: Els agents no només han de tenir un comportament reactiu, sinó que han de han de ser incisius per poder assolir els seus objectius.

- **Mobilitat**: Els agents han de poder moure's a través del seu medi. Alhora han de poder traslladar instruccions per que s'executin remotament.

- **Continuïtat temporal**: Els agents s'executen constantment.

- **Veracitat**: Un agent no comunicarà mai informació falsa de forma premeditada.

- **Benevolència**: Un agent no pot tenir objectius conflictius que puguin perjudicar a tercers. Els agents faran fer sempre allò encomanat.

- **Racionalitat**: Un agent sempre actuarà per assolir l'objectiu que té marcat. No realitzarà una acció si les precondicions per que es pugui dur a terme no es compleixen.

2.2 Arquitectures d'agents

Es distingeixen tres models d'arquitectures utilitzades en la implementació d'agents:

- **Deliberatives**: Donada una representació del món real, es poden portar a terme accions intel·ligents. S'ateny al model BDI (*Believes, Desires and Intentions*).

- **Reactives**: Les accions es duen a terme com a resposta a estímuls externs. La combinació de diversos agents reactius pot simular un comportament intel·ligent.

- **Híbrides**: Aquest model barreja les diferents tècniques.

2.3 Tipus d'agents

Els agents es poden classificar segons la seva funcionalitat en set grans grups:

- **Agents d'informació o d'Internet**: Aquests agents tenen accés a grans fonts d'informació (Internet) i són capaços de manipular i resumir la informació que obtenen.

- **Agents d'interfície**: Col·laboren amb l'usuari per resoldre un problema. Normalment adquireixen coneixement monitoritzant les accions que realitza l'usuari.

- **Agents de col·laboració**: El seu fet diferencial es que es comuniquen i cooperen amb altres agents per resoldre un problema comú. Utilitzen tècniques d'IA distribuïda.

- **Agents mòbils**: La seva principal característica és la seva mobilitat, que els permet desplaçar-se a través d'una xarxa (p. ex. WWW). Per poder moure's necessiten interactuar amb els diferents *hosts*.

- **Agents reactius**: El comportament d'aquests agents es basa en respostes a estímuls de l'entorn. En conjunció amb altres agents reactius es poden modelar comportaments complexos.

- **Agents híbrids**: Són combinacions dels anteriors. Mesclen les parts més interessants de cada tipus.

- **Sistemes Heterogenis**: No són pròpiament un agent sinó un conjunt d'almenys dos agents de dues o més classes diferents.

2.4 Sistemes Multiagent

Un Sistema Multiagent és un conjunt d'agents que interactuen entre ells per tal d'assolir un objectiu comú que ells individualment no podrien assolir, ja sigui per que no tenen prou coneixements, per que no tenen accés a prou recursos o per que el volum de feixa és molt gran.

2.4.1 Per què un SMA?

Les avantatges d'una aproximació amb un SMA són clares:

- **Modularitat**: El problema es redueix en components més senzilles que poden ser resoltes individualment.

- **Eficiència**: El model distribuït adoptat per les tecnologies orientades a agents permet l'execució simultània i paral·lela dels agents.

- **Fiabilitat**: Que un element del sistema deixi de funcionar no implica que la resta també ho deixi de fer. La fiabilitat es reforça, a més, per la naturalesa distribuïda del sistema.

- **Flexibilitat**: L'entrada i sortida d'agents es pot negociar en temps real.

2.4.2 Model d'un SMA

Segons les especificacions de la FIPA, un SMA ha d'estar composat per (veure la Figura 1):

- L'***Agent Platform***: Proporciona la plataforma, la infrastructura bàsica per que es puguin crear o executar agents.

- **Agents**: Són la unitat bàsica del sistema. Es poden considerar com entitats autònomes que encapsulen una sèrie de serveis.

- L'agent **DF** o ***Directory Facilitator***: És un agent que proporciona un servei de *Pàgines Grogues* dins del sistema. Aquesta agent manté informació sobre els serveis que poden oferir els diferents agents que es troben al SMA. Aquesta informació no l'obté directament, doncs són els agents, de forma individual, els que es registren en aquest agent.

- L'agent **AMS** o ***Agent Management System***: És l'agent que controla l'accés i l'ús de la plataforma d'agents. Només pot haver-n'hi un per plataforma. Ofereix, a més a més, un servei de *Pàgines Blanques* doncs emmagatzema les adreces de tots els agents de la plataforma. Quan algun agent vol passar a formar part de la plataforma s'ha de registrar a l'AMS.

- L'**MTS o *Message Transport Service***: És el mètode de comunicació entre els agents de diferents plataformes.

- **Programari**: No forma part dels agents però que és accessible per aquests.

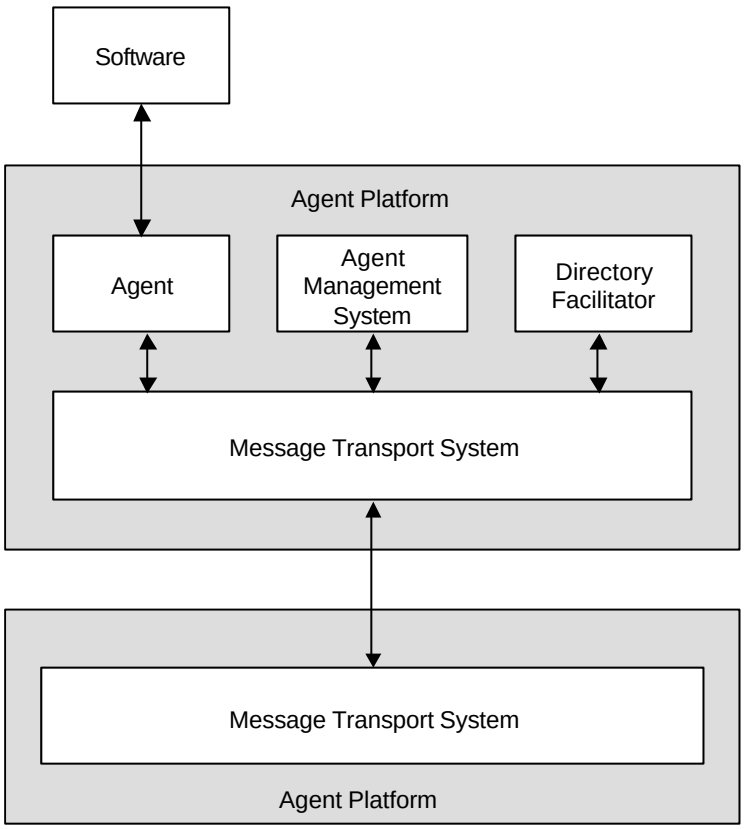


Figura 1: Estructura d'un SMA

2.4.3 Comunicació entre els agents

La capacitat comunicativa és una de les característiques més importants d'un agent. De fet, és imprescindible per poder tenir un SMA.

2.4.3.1 El llenguatge de comunicació dels agents: l'ACL

Hi ha diferents especificacions pel llenguatge de comunicació dels agents, com ara el KQML (*Knowledge Query Modeling Language*), però nosaltres ens centrarem en l'estàndard de la FIPA: l'ACL (*Agent Communication Language*).

L'element bàsic de qualsevol acte comunicatiu és el missatge. La FIPA determina quina forma ha de tenir un missatge i la seva utilització [5].

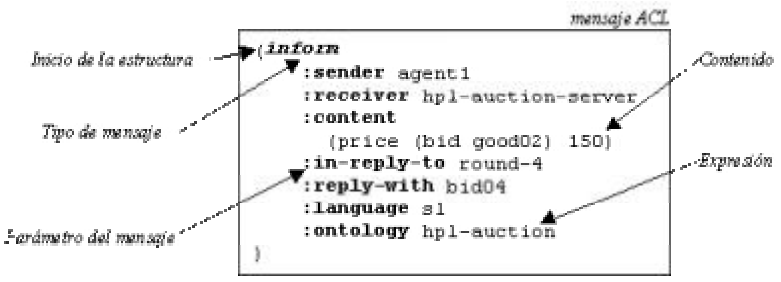


Figura 2: Format d'un missatge ACL

Tot missatge té una estructura prefixada i subjecte a les nostres necessitats (Figura 2). Els seus paràmetres tenen moltes similituds amb els missatges SMTP i es poden dividir en cinc categories segons la seva missió:

- **Definició del tipus de missatge**. Aquí hi tenim l'*slot performative* que indica el tipus d'acte de comunicació que volem representar amb el nostre missatge. Més endavant es veurà que les *performatives* estan fortament lligades al protocol.

- **Identificació dels agents participants**, amb els *slots sender, receiver* i *reply-to*.

- **Contingut del missatge**, amb un únic *slot*: *content*. El missatge pot ser escrit en qualssevol llenguatge, ja sigui Java, C++, Prolog, Lisp... però la FIPA, en l'esforç de cercar la compatibilitat entre implementacions, ha definit quatre tipus de continguts predefinits [6]:

    - **FIPA-CCL** (FIPA *Constraint Choice Language*): Permet l'especificació de predicats amb satisfacció de restriccions (CSP), suportant la representació de problemes, recol·lecció d'informació i tècniques de fusió i resolució de problemes.

    - **FIPA-SL** (FIPA *Semantic Language*) [7]: Permet la formació d'expressions lògiques, l'intercanvi òptim d'informació entre agents i l'expressió d'accions a realitzar. Serè el que nosaltres utilitzarem.

    - **FIPA-KIF** (FIPA *Knowledge Interchange Format*): Permet l'expressió d'objectes com a termes i preposicions com a sentències.

    - **FIPA-RDF** (FIPA *Resource Description Framework*): Permet la representació d'objectes i les interaccions i accions entre ells.

- **Descripció del contingut**: Aquest conjunt d'*slots* ens proporciona informació sobre el contingut del missatge, tal com el llenguatge en que està escrit, FIPA-SL, Java, Prolog, Lisp, etc., el tipus de codificació del missatge i, sobretot, la ontologia. Una ontologia és el conjunt de termes, predicats i accions lligades a un camp del coneixement. Necessitarem definir ontologies per que els agents puguin compartir coneixement sense

ambigüitats. Les ontologies es veuran amb més detall en els propers capítols. Aquest camp, doncs, determinarà unívocament la ontologia usada en el contingut del missatge.

- **Control de conversa**. Aquests slots permeten emmarcar un conjunt de missatges dins d'una conversa així com determinar el protocol que s'està fent servir. Tenim doncs el *conversation-id*, que identifica els missatges d'una mateixa conversa, el *reply-with*, que serveix per identificar els diferents passos d'una conversa, el *in-reply-to*, que relaciona un missatge de resposta amb el corresponent missatge de pregunta, el *reply-by*, que defineix una marca de caducitat pel missatge de resposta i el *protocol*, que identifica el protocol en que s'emmarca el missatge.

2.4.3.2 Els protocols de comunicació

Quan un agent vol establir comunicació amb un altre, s'inicia un acte comunicatiu entre els dos. El conjunt d'actes comunicatius entre dos agents està tipificat i respon a la naturalesa de les intencions que tingui l'agent *initiator* respecte a l'agent amb qui vol interactuar (agent participant o *responder*). Tenim un conjunt de protocols dissenyats per les interaccions més habituals, des d'una simple pregunta (*Query*) o una sol·licitud (*Request*) fins a una negociació (*Contract-net*). El protocol determina el flux de la conversa i el tipus de missatge que es pot rebre en cada pas. Vegem amb una mica més de detall els dos protocols utilitzats en aquest projecte:

- **FIPA-*Query*** [8]: A la Figura 3 podem veure el diagrama de flux del protocol. El FIPA-*Query* l'inicia un agent quan vol sol·licitar algun tipus d'informació a un altre. En el llenguatge humà correspondria a una simple pregunta a la que esperem una resposta. En un primer pas, l'agent *initiator* enviarà un missatge del tipus *query-if* o *query-ref* a l'agent participant. La diferència entre els dos tipus de missatge és que el *query-if* serveix per sol·licitar una resposta booleana mentre que el *query-ref* serveix per sol·licitar un objecte. Un cop rebut el missatge, en un segon pas, l'agent participant respondrà a la pregunta amb un missatge del tipus *inform*. Si l'agent participant no sap la resposta, pot respondre amb un missatge del tipus *failure*, i, si es nega a contestar-la (per exemple perquè l'agent *initiator* no està autoritzat a fer la pregunta), amb un *refuse*. En ambdós casos es pot detallar el motiu de la resposta. Si el participant no ha entès la pregunta (per exemple perquè no coneix la ontologia) respondrà amb un *not-understood*.

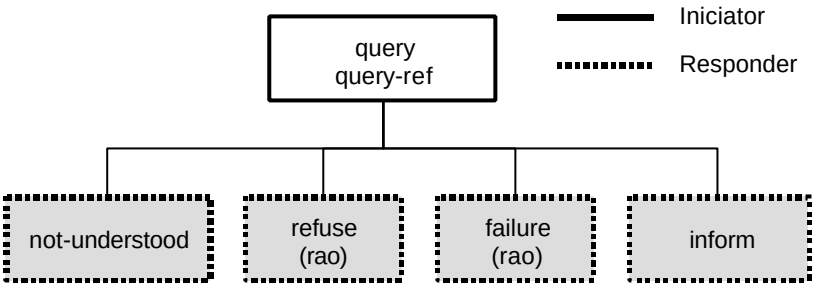


Figura 3: El protocol FIPA-*Query*

- **FIPA-*Request*** [9]: A la Figura 4 podem veure el seu diagrama de flux. El FIPA-*Request*, a diferència de l'anterior, consta de tres etapes i és utilitzat per un agent quan vol sol·licitar a un altre que li realitzi una acció. En un primer pas, l'agent *initiator* envia un missatge del tipus *request* a l'agent participant. Quan l'agent participant rep el missatge, en un segon pas, si vol o pot realitzar l'acció, contestarà amb un missatge *agree*. En aquest punt l'agent participant s'ha compromès a realitzar la tasca. Si no hagués pogut o no hagués volgut, pel motiu que sigui, podia haver contestat amb un *refuse* i si no hagués entès l'acció a realitzar, amb un *not-understood*. Un cop realitzada l'acció, en un tercer pas, s'ha d'informar a l'*initiator* sobre els resultats que aquesta ha produït. Per fer-ho tenim dos tipus de missatge, l'*inform* i l'*inform-done*. La diferència resideix en que l'*inform-done* ens comunica que l'acció s'ha realitzat mentre que *l'inform* pot retornar algun objecte com a resultat de l'acció. Si, per algun motiu l'agent participant no ha pogut completar la tasca, sempre pot sortir del pas enviant un *failure*, especificant, addicionalment, el motiu de la fallada.

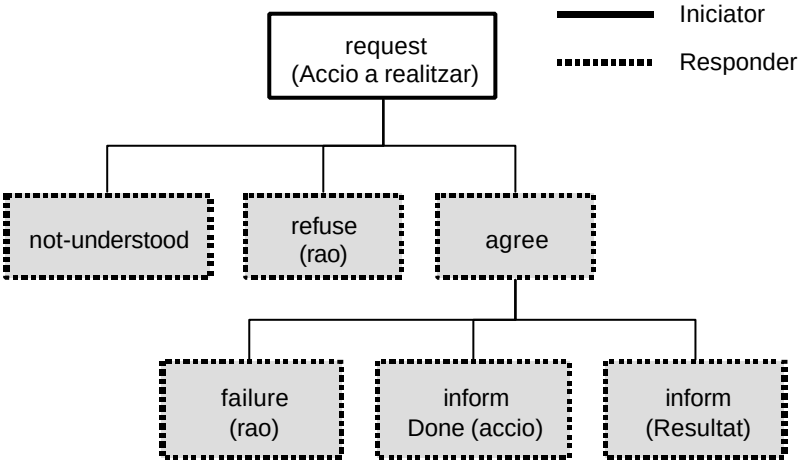


Figura 4: El protocol FIPA-*Request*

- **FIPA-*Contract Net*** [10]: Aquest protocol s'utilitza quan es vol realitzar una negociació entre dos o més agents. La figura 5 mostra el seu diagrama de flux. En un primer pas, l'agent *initiator* envia un missatge del tipus *cpp* (*Call For Proposal*) sol·licitant propostes per l'execució d'una acció. Els

agents participants, en un segon pas, li contestaran amb les corresponents propostes. Com sempre, si un agent participant es nega a contestar ho indicarà amb un missatge *failure* i si no ha entès el missatge, amb un *not-understood*. Després d'estudiar les propostes, l'agent *initiator*, en un tercer pas, enviarà un missatge *accept-proposal* a l'agent que cregui que li ha fet la millor oferta (potser cap). A la resta els hi enviarà un missatge *reject-proposal*. Ara, l'agent participant seleccionat ha d'executar l'acció. Quan acabi, en un últim pas, avisarà d'aquest fet a l'agent *initiator* per mitjà d'un *inform-done*. Si, per contra, ha tingut algun problema, ho pot indicar amb un *failure*. L'execució de l'acció pot ser avortada en qualssevol moment per l'agent *initiator* per mitjà d'un missatge *cancel*, especificant, addicionalment, el motiu.

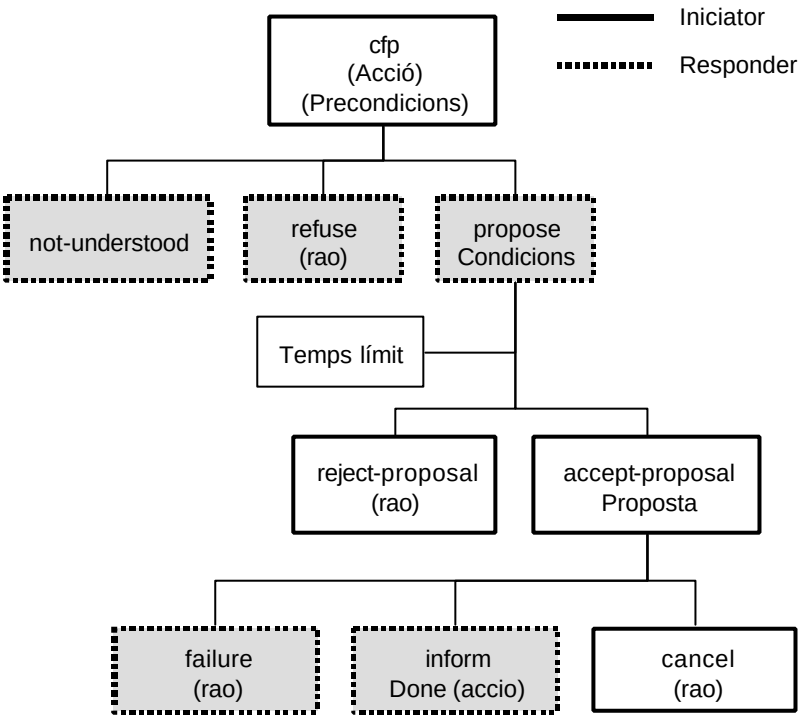


Figura 5: El protocol FIPA-*Contract Net*

3 JADE

Com que la tasca d'implementació d'una plataforma d'agents amb tots els serveis que proporciona és complexa, s'ha usat una eina externa de desenvolupament de SMA anomenada JADE (*Java Agent DEvelopment framework*).

D'entre les dotzenes d'entorns per a la creació de SMA disponibles a l'actualitat ens hem decantat pel JADE pels següents motius:

- La FIPA i el projecte *AgentCities* recolzen l'ús d'aquesta plataforma.
- Contínuament apareixen noves versions millorades.
- És un producte de lliure distribució.
- Està disponible en codi obert. Per tant el podrem modificar per adaptar-lo a les nostres necessitats.
- Ens ofereix un excel·lent suport a través dels manuals i de la llista de distribució activa.
- És FIPA *compliant*.

JADE és un conjunt de llibreries de lliure distribució escrites en JAVA que ens proporciona un entorn de desenvolupament de SMA [11]. Bàsicament proporciona la plataforma, amb els agents AMS i DF, un entorn segur que permet que els agents es connectin i es puguin comunicar i altres serveis de valor afegit com utilitats de monitorització i depuració. A més a més, JADE segueix les especificacions FIPA2000.

La versió utilitzada ha estat la 2.01 del 15/9/2000, l'última que es trobava disponible en el moment de començar aquest treball.

3.1 Els p*ackages* del JADE

- **jade.core**: És el nucli del sistema. Proporciona la classe **jade.core.Agent** que és la classe de la que derivaran els nostres agents. També conté la classe **jade.core.bevahiour** amb classes que implementen pautes de comportament pels agents.
- **jade.lang**: Conté les classes que permeten treballar amb missatges ACL i el llenguatge SL.

- **jade.domain**: Conté totes les classes que implementen la plataforma d'agents i els models de domini: les entitats per a la administració d'agents, llenguatges i ontologies (AMS, DF, ACC, etc.).
- **jade.proto**: Conté la implementació dels protocols de la FIPA.

- **jade.tools**: Aquí hi trobem un conjunt d'eines que ens facilitaran el desenvolupament del SMA i l'administració de la plataforma:

    - ***Remote Management Agent* (RMA)**: Mostra de forma gràfica l'estat de la plataforma (Figura 7) i ens permet realitzar tasques d'administració.

    - ***Dummy Agent***: És una eina de depuració que es materialitza en forma d'agent. Mitjançant la seva interfície (Figura 10) podem crear i enviar missatges ACL a altres agents.

    - ***Sniffer***: Visualitza una part, o tot, del tràfic de la plataforma (Figura 9).

    - ***Socket Proxy Agent***: És un agent que fa les tasques de *gateway* entre la plataforma i una connexió TCP/IP.

3.2 L'entorn del JADE

Pel JADE els agents resideixen en contenidors. Alhora, tots els contenidors estan inclosos dins d'un entorn predefinit anomenat plataforma (Figura 6).

Podem considerar els contenidors com un domini d'agents. Per fer-nos una idea, un contenidor podria ser una ciutat, que conté establiments (agents) que ens proporcionen els seus serveis.

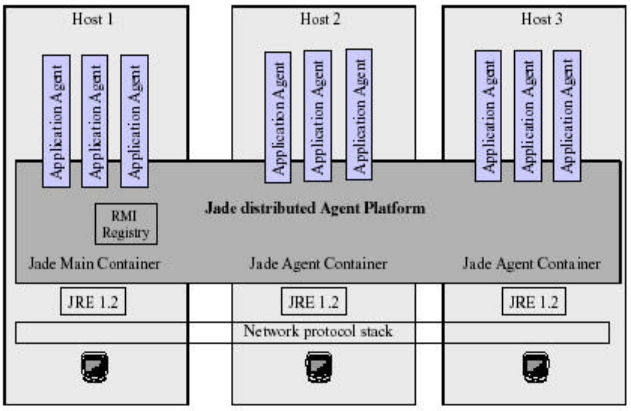


Figura 6: Arquitectura del JADE

Tal i com defineix la FIPA [12], en cada plataforma ha d'haver-hi els agents DF, que proporciona el servei de *Pàgines Grogues*, i AMS, *Pàgines Blanques*. A la Figura 7 es pot apreciar la jerarquia de contenidors del JADE: tenim la plataforma, a

l'arrel de l'arbre, anomenada JADE, i, penjant d'aquesta, un únic contenidor (*Main-Container*). Dintre d'aquest contenidor hi ha tots els agents del sistema, incloent l'AMS i el DF.

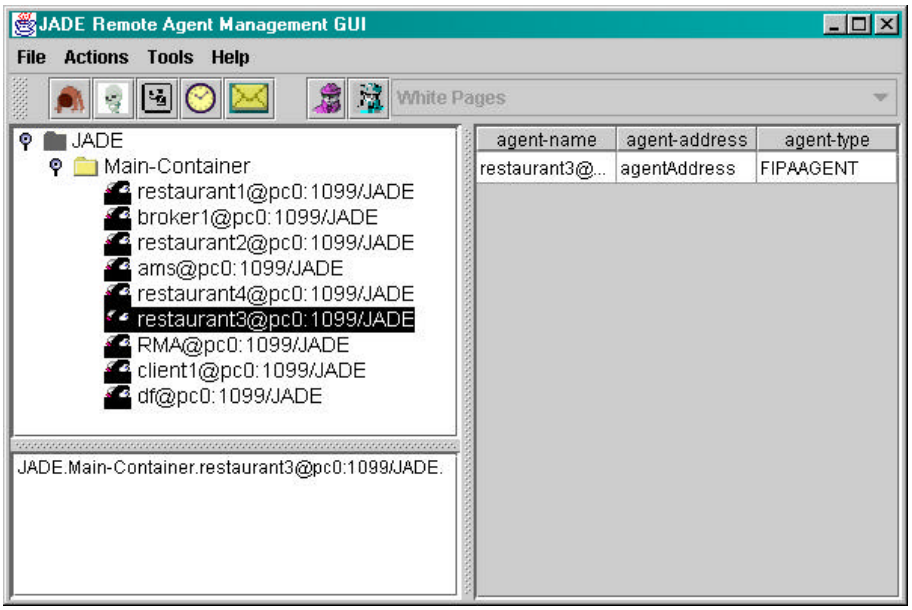


Figura 7: La interfície gràfica de l'RMA

3.3 Implementació d'un agent

El primer pas per poder crear un agent és derivar la classe abstracta jade.core.Agent. Obligatòriament haurem d'implementar els mètodes *setup()* i *takedown()* que s'invocaran durant els processos d'inicialització i finalització de l'agent.

Serà en el mètode *setup()* on es crearan els diferents fils d'execució mitjançant el que el JADE anomena *behaviours*.

Un *behaviour*, com el seu nom indica, modela un comportament, una forma d'executar el codi dintre de l'agent. Els diferents *behaviours* s'executen de forma pseudo-concurrent dirigits per un *scheduler* intern. Els *behaviours* es divideixen en simples i complexos, depenent de si poden o no tenir comportaments fills. Anem a veure'ls en detall.

3.3.1 Els comportaments simples

Com hem comentat abans són els comportaments que no poden tenir fills. Tots deriven de la classe abstracta jade.core.behaviours.Behaviour. Els mètodes a sobrecarregar són l'*action()*, que engloba el conjunt de sentències a executar, el *done()*, que retorna un booleà indicant si el comportament ha acabat o no, i el *reset()*, que reinicia el comportament. Els comportaments simples són tres:

- ***SimpleBehaviour***. Representa a un comportament atòmic que s'executa sense interrupcions. És una classe abstracta de la que en deriven:
    - ***OneShotBehaviour***: El comportament s'executa una sola vegada.

- ***CyclicBehaviour***: El comportament s'executa de forma cíclica.

- ***ReceiverBehaviour***: Engega un comportament que espera la recepció d'un missatge.
- ***SenderBehaviour***: Equivalent a l'anterior però des del punt de vista de l'enviament d'un missatge. Tot i estar implementat no té una utilitat pràctica doncs es pot fer el mateix amb el mètode *send()* de la classe jade.core.Agent.

3.1.2 Els comportaments complexes

Els comportaments complexes es diferencien dels simples en el fet de que poden tenir fills. És a dir, podem associar un o més *subbehaviours* a un comportament complex, que seran gestionats per un *scheduler* propi i independent del de l'agent.

Tots aquests comportaments deriven de la classe jade.core.bahaviours.ComplexBehaviour. Els mètodes a sobrecarregar són el *preAction()* i el *postAction()*. El primer mètode que es cridarà serà el *preAction()* i serà justament allí on afegirem els comportaments fill. Un cop finalitzada l'execució d'aquest mètode es llençaran els comportaments fill i, quan acabin tots, s'invocarà el mètode *postAction()*. La política d'execució dels fills serà la que diferenciarà els dos tipus de comportaments complexos:

- ***SequentialBehaviour***: Els fills s'executen en seqüència i en l'ordre en que s'han creat.

- ***NonDeterministicBehaviour***: La política d'*scheduling* és Round Robin. Es fixa una condició d'acabament per acabar el comportament pare.

3.4 Pas de missatges

En aquest aspecte el JADE també segueix les especificacions de la FIPA. Recordi's que el llenguatge de comunicació és l'ACL, que ha estat descrit a la secció 4.3 del capítol 2.

Per poder crear i manipular el contingut dels missatges, el JADE ens proporciona la classe jade.lang.acl.ACLMessage. Per cada *slot* del missatge tenim els corresponents mètodes *set* i *get* que ens permetran escriure o llegir el valor de l'*slot* respectivament. Tanmateix, pel camp *receiver* no tenim mètode *set*. Això és així per que el receptor pot ser més d'un agent. Per afegir un receptor farem servir el mètode *addReceiver()*.

Per enviar un missatge disposem del mètode *send()* mentre que per rebre'n un hem d'invocar el mètode *receive()*. Ambdós estan encapsulats dins de la classe jade.core.Agent.

3.5 Els protocols FIPA que el JADE implementa

A part de proporcionar-nos les classes necessàries per poder manipular missatges i modelar comportaments, el JADE també ens proporciona un conjunt de classes que, combinant comportaments, implementen els protocols FIPA-*Query*, FIPA-*Request* i FIPA-*Contract Net*, descrits a la secció 4.3.2 del capítol 2.

Aquestes classes les podem trobar al *package* jade.proto.

Cl remarcar que, per cada protocol tenim dues classes que corresponen als rols de l'agent *initiator* i de l'agent responder. Els noms de les classes seran del tipus *xxxInitiatorBehaviour* i *xxxResponderBehaviour*, on xxx dependrà del protocol.

Per utilitzar-les nosaltres només haurem de sobrecarregar els mètodes abstractes que representen cada pas del protocol. Així, per exemple, pel protocol FIPA-*Query* l'agent *initiator* tindrà els mètodes *handleInformMessages()* i *handleOtherMessages()*.

3.6 Suport pel llenguatge SL

Per assistir-nos en la creació de textos en llenguatge SL, el JADE posa a la nostra disposició tota una sèrie de classes que representen a les primitives del llenguatge. Mitjançant els mètodes *get* i *set* de cada un dels *slots* d'aquestes primitives podrem escriure o llegir qualssevol expressió. Així, per exemple, per construir el predicat (= a b) utilitzarem la classe *SL_Equals* que representa al rol "=". Per especificar cada un dels termes de l'expressió utilitzarem les següents sentències:

```
SL_Equals eq = new SL_Equals();
eq.set_0(a);
eq.set_1(b);
```

D'una forma semblant es faria per la resta de primitives. Aquestes classes les podem trobar al package jade.lang.sl.sl.

Quan tinguem construïda l'expressió utilitzarem el mètode *fillContent()* de la classe jade.core.Agent per traduir aquesta expressió a una cadena de text que s'escriurà en el camp de contingut del missatge que li passem com a argument.

3.7 Ontologies definides per l'usuari

El JADE també ens facilita la creació d'ontologies. A l'element bàsic d'una ontologia el JADE l'anomena *frame*. Cada *frame* el podem subdividir en *slots* que poden correspondre a tipus simples (booleans, *strings*...) o a altres *frames*. Un *slot* també pot ser un *set* o una *sequence* d'un tipus bàsic o d'un *frame*. La diferència entre aquests dos tipus és purament matemàtica i es basa en el fet que un *set* (conjunt) no té elements repetits mentre que una *sequence* (seqüència) sí. De forma addicional es podrà especificar un nom per cada *slot*.

Per cada element de la ontologia haurem d'implementar, a més, una classe Java que el representi. En aquesta classe, hi haurà d'haver un mètode *set* i *get* per cada *slot* que haguem definit. Si l'*slot* representa a un *set* o una *sequence*, els mètodes seran el l'*add* i el *getAll*.

Cal remarcar que els objectes de la ontologia es poden combinar amb els objectes d'SL.

3.8 Eines del JADE

El JADE ens ofereix un conjunt d'eines d'administració i monitorització destinades a ajudar-nos durant el procés de depuració del sistema.

Ja hem comentat abans que l'agent RMA és la cara visible del sistema. A partir de la seva interfície podrem accedir a totes les facilitats que ens proporciona el JADE. A la Figura 7 es mostra l'aspecte que ofereix. En ella podem veure un llistat dels agents que s'estan executant a la plataforma i una sèrie de botons que ens permetran:

- **Afegir un nou agent**. Haurem d'indicar-ne el nom, la classe que l'implementa i el contenidor al que volem confinar-lo.

- **Eliminar un agent**.

- **Suspendre l'execució d'un agent**.

- **Revifar un agent**. Per revifar-lo l'agent ha d'estar suspès.

- **Enviar un missatge a un agent**. Òbviament haurem d'indicar el nom del destinatari i haurem d'omplir els camps del missatge. La Figura 8 mostra l'aspecte de la finestra que ens ajudarà a escriure un missatge ACL.

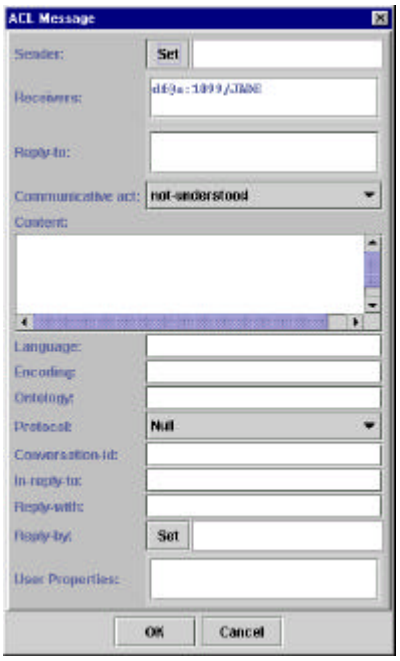


Figura 8: Escriptura d'un missatge

- **Arrancar l'*sniffer***: L'agent *sniffer* ens permetrà veure el trànsit de missatges dins de l'SMA. Els resultats es presentaran en forma de diagrama de flux on cada enviament es representarà com una fletxa que va de l'agent emissor a l'agent receptor. Si fem doble clic sobre qualssevol de les fletxes podrem veure fins a l'últim detall del missatge en una finestra com la de la Figura 8. Val a dir que el conjunt d'agents a monitoritzar és configurable des del menú *Actions* opció *Add/Remove Agent*. La Figura 9 mostra a l'agent *sniffer* en plena acció.

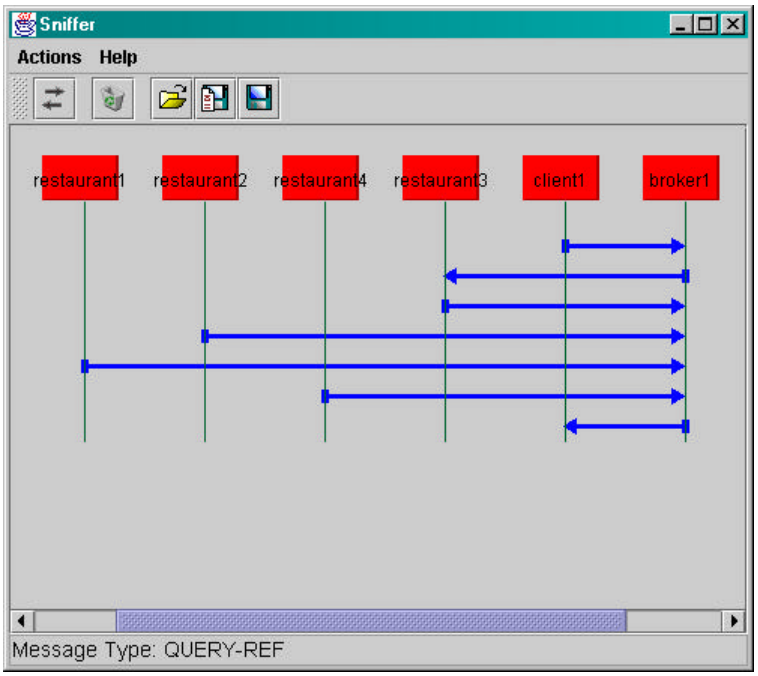


Figura 9: Agent s*niffer*

- **Arrancar el *DummyAgent***: L'agent *dummy* s'infiltra al SMA i ens permet enviar i rebre informació de la resta d'agents. És com qualsevol altre agent però està sota el nostre control. Té alguna opció curiosa com és la possibilitat de salvar un missatge al disc. La Figura 10 mostra l'aspecte que ofereix de la seva interfície.

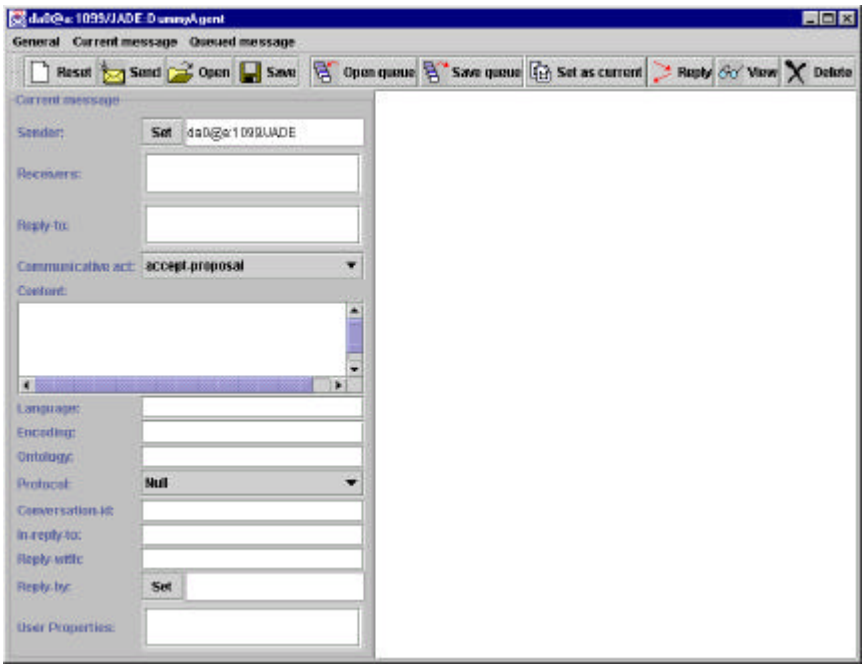


Figura 10: Agent *dummy*

Com a última opció destacable que ens ofereix la GUI de l'RMA, tenim la possibilitat de visualitzar la interfície del DF. Per fer-ho hem d'anar al menú *Tools* opció *Show the DF GUI*. En ella podrem veure el llistat d'agents que s'han registrat al DF i, per cada un d'ells, per quins serveis ho ha fet. De cada servei podem obtenir-ne el nom, el tipus, les propietats, etc. També ens ofereix la possibilitat d'afegir o eliminar entrades del registre. La Figura 11 mostra la GUI del DF.

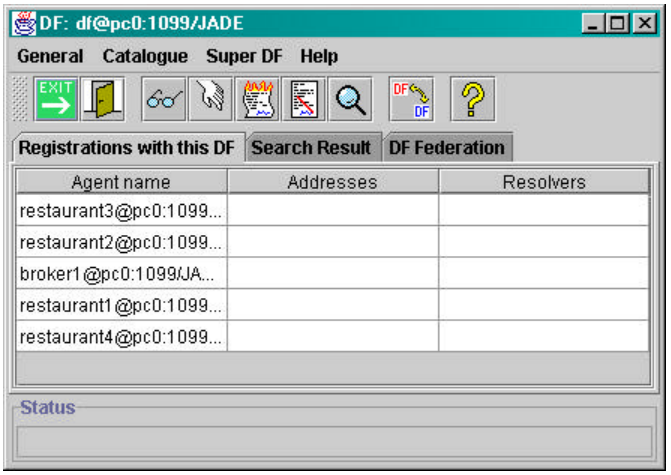


Figura 11: La GUI del DF

4 Arquitectura del SMA

4.1 Introducció

Primer de tot, anem a veure les tasques que ha de realitzar el sistema i a les entitats que involucra:

- El sistema ha de permetre cercar restaurants que compleixin un conjunt de restriccions especificades per l'usuari.

Tenim doncs una entitat usuari que, a través d'algun procediment, vol obtenir informació sobre restaurants. Aquesta informació podria estar, perfectament, en una Base de Dades d'un servidor. Els usuaris es connectarien al servidor per obtenir la informació desitjada.

- Reservar taula en un restaurant.

Com que cada restaurant pot tenir una política de reserves diferent, centralitzar la seva gestió no és la solució gaire adequada. A més tindríem problemes d'ineficiència (cal tenir en compte que els restaurants necessiten accedir contínuament a la informació de les seves reserves), fiabilitat (la caiguda del gestor implicaria la caiguda de tot el sistema) i de flexibilitat que té qualssevol aproximació centralitzada. En aquest cas, la millor solució seria optar per un disseny distribuït.

Tenim doncs tres tipus d'entitats ben diferenciades:

- Els **clients** o usuaris: Són els que utilitzen els serveis del sistema.

- **Entitats de gestió** o de suport del sistema: Són les entitats que proveeixen la infrastructura, que permet el funcionament del sistema, i la gestionen.

- Els **restaurants**: Són les entitats que ofereixen els seus serveis a través del sistema.

L'arquitectura del problema pot basar-se en molts models però nosaltres ens hem decantat per un SMA per les avantatges que ens ofereix:

- **Modularitat**: El problema de la gestió de cada entitat (a partir d'ara agent) es trasllada a ell mateix. El sistema només ha de facilitar l'entrada i sortida d'aquests agents.

- **Eficiència**: El model distribuït ens permet l'execució simultània i paral·lela dels agents.

- **Fiabilitat**: Que un element del sistema deixi de funcionar no implica que la resta també ho deixi de fer. La fiabilitat es reforça, a més, per la

naturalesa distribuïda del sistema: per exemple, si l'ordinador d'un restaurant cau, la resta del sistema continua funcionant. A més a més, els serveis crítics es poden replicar, aconseguint així redundància.

- **Flexibilitat**: L'entrada i sortida d'agents restaurant es pot negociar en temps real.

Abans d'estudiar l'estructura del SMA, composada bàsicament per agents client i agents restaurant, introduirem abans un tipus d'agent que farà de pont entre aquests dos: l'agent *broker* (Figura 12).

Quan un agent arriba a una plataforma, necessita conèixer la seva topologia per poder fer ús dels seus serveis. Aquí és quan entra en joc l'agent *broker*. Un agent *broker* representa i és coneixedor d'un domini d'agents, generant una jerarquia dins de la plataforma. Per exemple, si tenim en compte el conjunt de serveis que ens pot oferir una ciutat, veiem que els podem classificar en serveis de restauració, allotjament, etc., on cada conjunt o domini pot ser gestionat per un *broker*. Els agents *broker*, doncs, exploren contínuament el seu domini, mantenint informació sobre l'estat dels agents que en formen part i emmagatzemant algunes de les seves característiques, per tal de proporcionar, entre d'altres, un servei de pàgines grogues més proper a la realitat del domini. Per fer-nos una idea, les pàgines grogues ens informen sobre quins restaurants hi ha en una ciutat i com contactar amb ells però no ens diuen quins estan oberts aquest diumenge o quins fan colçotades. En canvi, un fulletó d'informació sobre restaurants, sí que ens pot indicar tot això. Aquesta és doncs la utilitat dels agents *broker* i la seva principal diferència sobre l'agent DF.

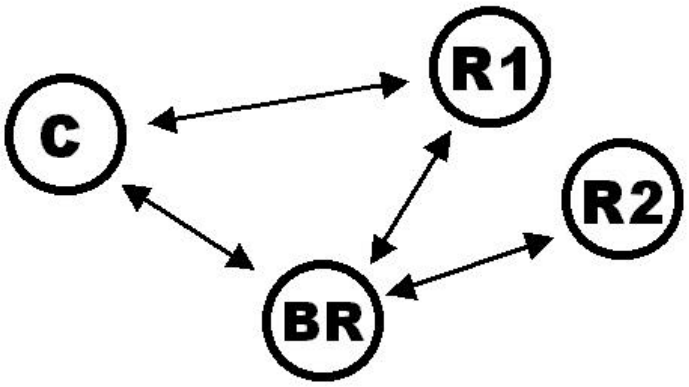


Figura 12: Interaccions entre els agents del sistema

4.2 Els agents i les seves tasques

 **Agents client:** La principal funció dels agents client és la de fer d'interfície entre el SMA i l'usuari. Les seves habilitats són:

- **Comunicar-se amb l'agent BR** (*Broker*) **per tal d'obtenir informació sobre els restaurants**, del domini del *broker*, que satisfacin unes determinades restriccions. Aquestes restriccions les imposarà l'agent client (en extensió l'usuari) d'entre un conjunt de restriccions preestablertes.

Aquestes restriccions poden fer referència al preu del menú, al tipus de cuina del restaurant, a la disponibilitat d'aparcament, etc.

- **Comunicar-se amb agents del tipus R (restaurant) per sol·licitar reserva de taula**. També pot comunicar-se amb ells per sol·licitar-los informació encara que, habitualment, ja ho fa a través del *broker*.

Agent broker: Aquest agent gestiona el domini de restauració del SMA**,** controlant els agents que estan presents, a cada moment, en aquest domini. Serà el primer agent que orientarà als agents client. La seva tasca serà doncs:

- **Retornar informació sobre els restaurants del seu domini**. Aquesta informació, sol·licitada pels agents restaurant, pot demanar-se exigint algunes restriccions. Per agilitar els processos de búsqueda, el *broker* mantindrà una *caché* amb algunes de les característiques més representatives dels restaurants. Veurem més endavant que el manteniment d'aquesta *caché* implicarà l'implementació d'alguns mecanismes d'actualització de la mateixa.

Agents restaurant: Representa a les entitats restaurant que estan presents al SMA. A part de fer d'interfície entre el restaurant real i el SMA**,** té altres tasques:

- **Gestionar el restaurant**. Això vol dir implementar tots els mecanismes que tinguin a veure amb la gestió interna del restaurant, ja sigui reserva de places, manteniment d'informació d'estat o planificació d'horaris i períodes vacacionals, de tal manera que tota aquesta informació sigui accessible des del SMA.

- **Retornar informació sobre el restaurant**. Aquesta informació pot ser sol·licitada sota algunes condicions.

- **Servir peticions de reserva de taula**. Les reserves es confirmaran o es refusaran depenent de si reserva taules per aquell dia i té places lliures.

4.3 Detalls del funcionament

El DF registra els serveis dels agents segons el nom i el tipus. Addicionalment, per cada servei podem especificar una sèrie de característiques relacionades. Per decisió de disseny, el tipus de servei ofert pels agents relacionats amb serveis de restauració s'identifica per la cadena *agent-cities.restaurants.services*. Per tant, els serveis proporcionats pel *broker* i pels agents restaurant, s'inclouran dins d'aquest grup. Quan un agent restaurant es dóna d'alta a la plataforma i es registra al DF, declara un únic servei de nom *restaurant*, amb dues característiques: *specialities*, que enumera les especialitats o tipus de cuina que ofereix i *domain* que l'emmarca dins d'un domini d'agents. El contingut de la cadena *domain* està fortament lligat amb la localització geogràfica del restaurant, que ens servirà alhora com a propietat d'agrupament. Així, tots els serveis de restaurant declaren el seu *domain* com

tarragona.<codi_postal>, on codi_postal ens proporcionarà una subdivisió geogràfica que ens servirà per classificar els serveis per zones.

Els agents *broker*, declaren també un únic servei: *broker*. Igual que els serveis de restaurant, defineixen la propietat *domain*, que serà igual a *tarragona*. En aquest punt es pot veure clarament la idea de jerarquia que ofereixen els *broker*: tenim un domini d'agents, *tarragona*, que ofereix serveis relacionats amb la restauració (*agent-cities.restaurant.services*). Quan un agent client vol accedir als serveis proporcionats pel domini, té una porta d'entrada clara: el servei *broker* del tipus *agent-cities.restaurants.services* del *domain tarragona*. Internament, el *broker* presenta subdominis (que podrien estar gestionats per altres *brokers*) que estan representats pel codi postal dins de la propietat *domain*.

Cal notar que l'agent client no publicita cap servei. De fet, no ofereix cap mena de servei, només els utilitza.

Passem a veure ara el funcionament del *broker*. Recordem que la seva tasca és tenir informació del seu domini per tal d'accelerar els processos de búsqueda de serveis. Durant la fase de disseny del SMA, es va detectar que les restriccions més habituals en una búsqueda de restaurants, són l'especialitat (o tipus de cuina), el preu del menú i la seva localització geogràfica (representada pel codi postal). Per tant, el *broker* hauria de conèixer aquestes propietats de cada un dels restaurants del seu domini, per què, quan es realitzi una consulta, pugui descartar la major part dels restaurants abans de preguntar-los per característiques menys habituals, com poden ser, per exemple, la disponibilitat d'aparcament o les targetes admeses.

En un primer pas, el *broker* obtindrà els restaurants del seu domini consultant al DF. Recordem que cada servei de restaurant té la propietat *domain* que ens ajudarà en aquesta tasca. Una vegada localitzats sap, per cada restaurant, les seves especialitats i el seu codi postal. Només li cal obtenir el preu del menú que no està publicitat al DF. La única forma: preguntant-ho de forma individual a cada restaurant. Una vegada obtinguda aquesta informació muntarà la seva *caché* i ja estarà preparat per atendre les consultes dels clients.

Quan un client generi una consulta, el *broker* comprovarà si hi ha alguna restricció sobre l'especialitat, el codi postal o el preu del menú. Si és així, seleccionarà els restaurants que satisfacin aquestes restriccions i els hi enviarà un nou missatge amb la resta de restriccions que no pot comprovar. Els restaurants, en una última fase, proporcionaran la informació sol·licitada si s'ajusten a aquestes.

La presència de la *caché* fa necessària la utilització d'una sèrie de mecanismes que mantinguin actualitzada la informació que conté. En aquest sentit, la *caché* s'actualitza un cop al dia, consultant el DF i preguntant de nou pels preus dels menús. Si, per qualsevol motiu, els preus dels menús necessiten ser actualitzats d'una forma més ràpida, el restaurant té la possibilitat de comunicar-li al *broker* que el preu del menú ha canviat i de proporcionar-li el nou valor.

4.4 La comunicació entre els agents

Per que els agents es puguin comunicar els uns amb els altres i puguin realitzar les accions descrites, necessiten posseir habilitats comunicatives, un llenguatge i un vocabulari comuns.

Les habilitats comunicatives es recolzen sobre el conjunt de protocols. Els protocols que nosaltres utilitzarem seran els especificats per la FIPA per als actes comunicatius més habituals (FIPA-*Query* per formular preguntes i FIPA-*Request* per sol·licitar l'execució d'accions). El llenguatge també vindrà determinat per la FIPA i serà el FIPA-SL. El vocabulari o ontologia l'haurem de definir nosaltres i haurà de ser adequat al domini de coneixement que volem representar.

4.4.1 L'ontologia del camp de la restauració

Com hem apuntat abans, per que els agents es puguin comunicar sense ambigüitats, necessiten compartir un mateix llenguatge i ontologia. La ontologia defineix el vocabulari, el conjunt de predicats que podem formar i les accions que, mitjançant un acte comunicatiu, es poden sol·licitar a un agent. En aquest apartat detallarem el conjunt d'objectes lingüístics que utilitzen els agents de l'SMA.

4.4.1.1 L'objecte restaurant

És la peça fonamental de l'ontologia i a la que aquesta deu el seu nom. Conté tota la informació que podrem sol·licitar d'un restaurant.

restaurant		
:contact-info	restaurant-cinfo	Informació de contacte
:prices	prices	Preus
:has_night_menu	Boolean	Indica que el restaurant serveix menús per sopar
:capacity	Long	Capacitat
:has_parking	Boolean	Indica que el restaurant disposa d'aparcament
:has_smokers_zone	Boolean	Indica que el restaurant disposa d'una zona per fumadors
:admitted_cards	Set of String	El conjunt de targetes que accepta el restaurant
:comments	String	Un camp de format lliure
:specialities	Set of String	El conjunt d'especialitats del restaurant
:name	String	L'AID (*Agent ID*) re l'agent que gestiona el restaurant
:carte	Carta	La carta

- ***restaurant-cinfo***: informació de contacte aplicable a qualssevol tipus d'establiment.

restaurant-cinfo		
:name	String	El nom de l'establiment
:postal_adress	postal-adress	L'adreça del restaurant
:phone	String	El número de telèfon
:fax	String	El número de fax
:out_of	String	Un camp de format lliure

- ***postal-adress***: l'adreça postal de qualssevol edifici.

postal-adress		
:country	String	El país
:zip_code	String	El codi postal
:city	String	La ciutat
:street	String	El carrer
:street_number	Long	El portal

- ***prices***: els preus del menú i de la carta

restaurant-cinfo		
:menu_price	Long	El preu del menú
:carte_price	Long	El preu de la carta

- **carta**: la descripció completa de la carta. Els plats de la carta es divideixen en quatre categories: entrants, primers, segons i postres.

carta		
:entrants	Set of plat	Els entrants
:primers	primers	Objecte que classifica els primers plats
:segons	segons	Objecte que classifica els segons plats
:postres	Set of plat	Les postres

- **primers**: classifica els primers plats en cinc categories: pastes, sopes, verdures, llegums i arròs. Aquestes subdivisions ens permetran obtenir aquestes parts de la carta de forma individual.

primers		
:pastes	Set of plat	Tots els plats de pasta
:sopes	Set of plat	Tots els plats de sopa
:llegums	Set of plat	Tots els plats de llegums
:verdures	Set of plat	Tots els plats de verdures
:arros	Set of plat	Tots els plats d'arròs

- **segons**: classifica els segons plats en cinc categories: ous, peixos, marisc, carns i aus. Aquestes subdivisions ens permetran obtenir aquestes parts de la carta de forma individual.

segons		
:ous	Set of plat	Tots els plats d'ous
:peixos	Set of plat	Tots els plats de peix
:marisc	Set of plat	Tots els plats de marisc
:carns	Set of plat	Tots els plats de carn
:aus	Set of plat	Tots els plats d'aus

- **plat**: és la peça mínima del menú. Per cada plat tenim el seu nom i el seu preu.

plat		
:nom	String	El nom del plat
:preu	Long	El seu preu

També necessitarem definir una enumeració de les especialitats i les targetes de crèdit amb les que esperem treballar. En aquest punt ens n'adonem que, si mai apareixen especialitats o targetes noves, serà necessària una ampliació de la ontologia. Aquest problema, però, és intrínsec als llenguatges i, de fet, és el mateix que succeeix amb les llengües naturals quan necessiten crear nous mots per definir termes fins llavors desconeguts.

4.4.1.2 Les accions

A vegades, però, amb els objectes no n'hi ha prou. Per fer algunes tasques tenim la necessitat d'expressar accions. En el nostre cas, un agent restaurant necessita sol·licitar d'alguna forma a l'agent *broker* que li canviï el preu del menú. També necessitem sol·licitar les reserves als restaurants. Haurem de definir, doncs, les accions:

- ***change-menu-price***: sol·licitada pels agents restaurant, demana al *broker* que li actualitzi el preu del menú a totes les seves taules internes. Recordem que aquesta informació es utilitzada pel *broker* per fer un primer filtrat durant el procés de cerca.

change-menu-price		
:new_price	Long	El nou preu del menú

- ***make-booking***: sol·licitada pels agents client, demana al restaurant que li realitzi una reserva per un dia i hora determinats per un cert número de persones. També li ha de passar un nom i un telèfon de contacte.

make-booking		
:people	Long	Número de persones
:when	timestamp	Dia i hora en que es vol fer la reserva
:who	user-cinfo	El sol·licitant

- ***timestamp***: una marca de temps de propòsit general.

timestamp		
:minute	Long	Minut
:hour	Long	Hora
:day	Long	Dia
:month	Long	Mes
:year	Long	Any

- ***user-cinfo***: informació que ens permet identificar un client o posar-nos en contacte amb ell.

user-cinfo		
:name	String	El nom amb el que el client fa la reserva
:phone	String	El seu número de telèfon

L'execució de les accions no sempre comportarà els resultats desitjats. L'ontologia ens ha de proporcionar una sèrie d'objectes mitjançant els quals puguem expressar condicions d'error. Val a dir que, per expressar l'èxit en l'execució d'una acció, el llenguatge SL ja ens proporciona la primitiva *done*. Només caldrà definir, doncs, els possibles objectes d'error.

Per a l'acció *change-menu-price* tenim els termes:

- ***not-in-cache***: Indica que l'agent sol·licitant no és troba a la *caché* i que, per tant, el seu preu del menú no es pot actualitzar.

- ***inconsistent-price***: Serveix per indicar que el preu del menú és un valor invàl·lid, com, per exemple, un valor negatiu.

- ***change-menu-price-error***: Assenyala qualssevol altra condició d'error succeïda durant l'execució de la tasca.

I per a l'acció *make-booking*:

- ***too-many-people***: Informa sobre la negativa, per part del restaurant, de reservar taula per un grup tan gran de persones.

- ***full***: Ens avisa que el restaurant aquell dia a aquella hora està ple.

- ***not-allowed***: El restaurant no reserva taula per aquell dia, ja sigui per que es un dia especial o perquè, simplement, fa festa.

- ***never-allowed***: El restaurant no reserva taules mai, almenys per via telemàtica.

- ***too-far***: La data sol·licitada excedeix els dies d'antelació que el restaurant considera màxims per poder realitzar la reserva. També serveix per indicar que la data ja ha passat.
- ***booking-error***: Com en l'acció *change-menu-price* ens reservem un objecte que servirà per indicar qualssevol altra condició d'error.

4.4.1.3 Els predicats

Per completar l'ontologia hem de tenir una sèrie de predicats que ens permetin definir relacions entre els objectes o expressar les seves propietats. Els dos predicats definits a la ontologia són:

- ***is-restaurant***: representa a un objecte *restaurant*, indicant-nos els *slots* que conté i les restriccions que compleix.

is-restaurant		
:restaurant	restaurant	Un objecte del tipus restaurant
:slots	Set of String	El nom dels *slots* que volem que té l'objecte restaurant
:restriccions	Set of any type	Les restriccions que compleix l'objecte restaurant. Pot incloure altres predicats.

- ***specialized***: aplicat a un objecte *restaurant* indica que està especialitzat en l'especialitat que es passa com a argument.

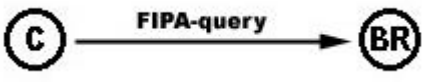
Specialized		
:speciality	Set of String	Nom de l'especialitat

4.4.2 Les tasques i els seus protocols

A continuació es descriuen les tasques que es realitzen dintre de l'SMA detallant els protocols involucrats en cada una.

4.4.2.1 La cerca dels restaurants

Quan un agent client arriba al sistema desconeix per complet els restaurants que es troben en el seu domini. Per poder fer la cerca l'agent client haurà de demanar ajuda a l'agent *broker*, coneixedor del domini, que serà el que li proporcionarà aquesta informació. El missatge serà un Query i el protocol un FIPA-Query:

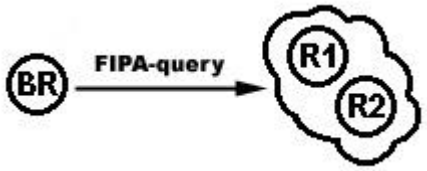


Suposem ara que el client engega un FIPA-query contra el *broker* sol·licitant-li informació sobre els restaurants que tenen, per exemple, aparcament i que la seva especialitat sigui *cuina_mexicana*. El missatge tindrà el següent aspecte:

```
((all ?x
      (is-restaurant ?x
            (set contact-info prices)     // slots a retornar
            (set                          // restriccions
                  (matches (restaurant :has_parking true))
                  (specialized (set cuina_mexicana))
            )
      )
))
```

Sol·licitem doncs tots (*all*, és una predicat llenguatge FIPA-SL i equival, aproximadament, a un "tots") els restaurants *(is-restaurant*, un dels predicats definits per la nostra ontologia que serveix per descriure a objectes del tipus restaurant, tant pels *slots* que ha de contenir com per les restriccions que ha d'acomplir) que tinguin aparcament (*matches*, és un altre predicat d'SL que retorna cert quan l'objecte del que estem parlant i el que li passem com argument segueixen el mateix patró) i estiguin especialitzat en cuina mexicana (*specialized*, un predicat de la nostra ontologia que serveix per imposar aquesta restricció sobre un objecte restaurant).

Després de seleccionar tots els restaurants que serveixin *cuina_mexicana*, el *broker*, els hi enviarà un missatge preguntant si disposen d'aparcament. Enviarà, doncs, un missatge query mitjançant el protocol FIPA-query:



El missatge serà el següent:

```
((iota ?x
      (is-restaurant ?x
            (set contact-info prices)
            (set
                  (matches (restaurant :has_parking true))
            )
      )
))
```

Podem veure que el *broker* ha eliminat la restricció imposada pel predicat *specialized* i ha canviat l'*all* per un *iota*. El predicat *iota* és també un predicat d'SL que equival, aproximadament, a un "algun".

Després de rebre les respostes, l'agent *broker* generarà un missatge de resposta pel client del tipus:

```
((=
      (all ?x ...)      // la pregunta inicial
```

```
        (restaurant-set (set
              // i un conjunt amb els objectes restaurant que
              // compleixen les restriccions
              (restaurant :contact-info (restaurant-cinfo ...)
              (restaurant :contact-info (restaurant-cinfo ...)
        ))
))
```

4.4.2.2 Reserva de taula en un restaurant

Ara, l'agent client vol fer una reserva al restaurant R1. Per fer-ho ha d'enviar un missatge Request enmarcat dins del protocol FIPA-request:

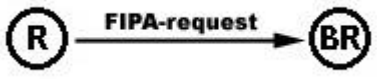
El missatge s'assemblarà al següent:

```
((action
      // AID de l'agent que sol·licita l'acció
      (agent-identifier :name client1@pc0:1099/JADE...)
      (make-booking
            5  // número de persones
            // el dia i l'hora
            (timestamp :minute 30 :hour 22 :day 29 :month 9
            :year 2001)
            // informació de contacte del sol·licitant
            (user-cinfo :name "Gavarró" :phone 677867656)
      )
))
```

L'acció *make-booking* i els objectes *timestamp* i *user-cinfo* són de la nostra ontologia i han estat detallats a la secció 4.1 d'aquest capítol. El predicat *action* és d'SL i descriu a una acció.

4.4.2.3 Canviar el preu del menú

Com que quan un restaurant vol canviar el preu del menú sol·licita la realització d'una acció (canviar el preu del menú), el missatge serà un Request i el protocol, el FIPA-request:



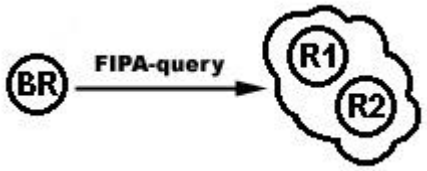
El missatge tindrà la forma:

```
((action
      (agent-identifier :name restaurant1@pc0:1099/JADE...)
      // el nou preu del menú
      (change-menu-price :price 1375)
))
```

Com abans tenim un predicat *action* i a dins la funció *make-booking* definida a la nostra ontologia.

4.4.2.4 Sol·licitud del preu del menú

Quan el *broker*, durant el procés d'actualització de la *caché*, necessita preguntar als restaurants el preu del menú, envia un Query emmarcat dins del protocol FIPA-query:



El missatge tindrà la forma:

```
((iota ?x
      (is-restaurant ?x
            // sol·licitem el preu sense cap mena de restricció
            (set prices)
      )
))
```

5 Manual de l'usuari

En aquest capítol es parlarà de com posar en marxa el sistema i s'explicaran les interfícies dels agents client i restaurant. El *broker* està mancat d'interfície però podem seguir el que va fent a través de la sortida estàndard.

5.1 Posada en marxa del sistema

Per arrancar la plataforma hem d'invocar al mètode *main()* de la classe jade.Boot que es troba a la llibreria Jade_sl.jar proporcionada per la distribució. Com a paràmetres li hem de passar, per cada agent que vulguem tenir al sistema, el nom que tindrà l'agent dins de la plataforma (nom local) i el nom de la classe Java que l'implementa. Per exemple, per arrancar quatre restaurants amb els noms restaurant1, restaurant2, restaurant3 i restaurant4, un *broker* i un client, executarem la següent sentència:

```
java -classpath .;jade_sl.jar jade.Boot
client1:restaurants.ClientAgent broker1:restaurants.BrokerAgent
restaurant1:restaurants.RestaurantAgent
restaurant2:restaurants.RestaurantAgent
restaurant3:restaurants.RestaurantAgent
restaurant4:restaurants.RestaurantAgent
```

Els agents restaurant han estat programats per simular un agent restaurant final. Per tant, per simplicitat i per evitar que les execucions de prova siguin molt costoses pel que fa a l'entrada de dades, s'ha optat per que els agents restaurant llegeixin les seves característiques des de disc. Per que funcionin correctament han de poder llegir el seu fitxer de característiques. El nom del fitxer serà igual al nom local de l'agent (el nom que li haguem donat dins de la plataforma) més la extensió .txt. El format d'aquests fitxers es detallarà a la secció 5.3.

Per que sigui operatiu, el sistema ha de contenir com a mínim un *broker*, un restaurant amb el que pugui interaccionar i un agent client amb el que poder fer les consultes.

Si no hi ha hagut cap problema, hauria d'aparèixer per pantalla una finestra per cada restaurant i cada agent client.

5.1 Agent client

La interfície de l'agent client serà la que veurà l'usuari final. A continuació descriurem com utilitzar-la per fer cerques i reserves. També veurem com configurar els criteris de cerca i com detallar la informació que volem obtenir de cada restaurant.

La Figura 13 mostra l'aspecte de la GUI de l'agent client. Tenim dues pestanyes, *Cerca* i *Reserva*, tenint seleccionada inicialment la primera.

Figura 13: La GUI de l'agent client

En aquesta podem distingir dues àrees: *Criteri de cerca* i *Informació a obtenir*. Si premem el botó *Canviar* de la primera, apareixerà una finestra com la de la Figura 14 que ens permetrà especificar les restriccions que han de complir els restaurants que busquem. Si, per contra, premem el botó *Canviar* de la segona, la finestra resultant (Figura 15) ens ajudarà a seleccionar la informació que volem obtenir de cada un dels restaurants. Un cop restringida la cerca i determinada la informació desitjada podem prèmer el botó *Buscar ara!* per iniciar la búsqueda. Quan aquesta finalitzi es mostrarà directament la pestanya *Reserva* amb els resultats de la cerca (Figura 16).

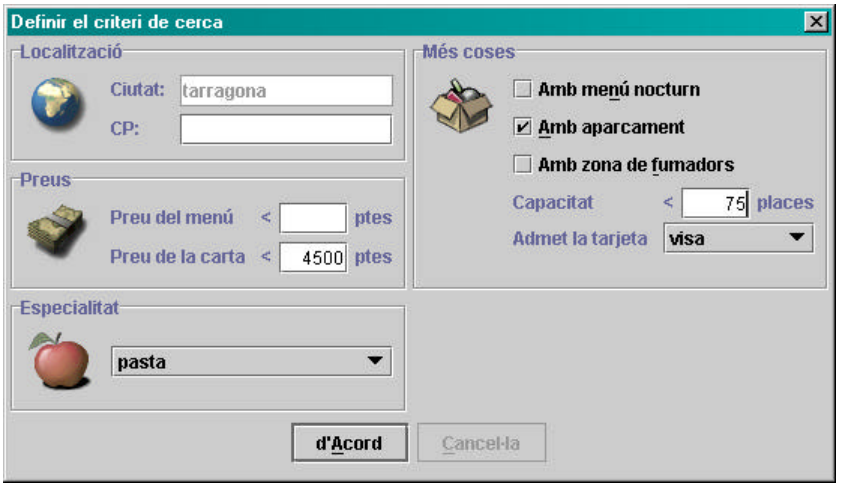


Figura 14: Finestra on podem especificar el criteri de cerca

Anem a estudiar ara la finestra on podem definir el criteri de cerca (Figura 14). A l'àrea *Localització* podem especificar restriccions sobre el codi postal (CP) sobre el que volem restringir la cerca. La ciutat no es pot canviar i, per defecte, és *tarragona*. Això és així perquè aquest projecte pretén, únicament, modelar un servei de restaurants per aquesta ciutat, tot i que, en un futur, es pretén eliminar aquesta restricció.

A l'àrea *Preus* podem fixar el preu màxim pel menú i per la carta. Tots els restaurants que tinguin un preu superior a un d'aquests dos valors serà descartat del conjunt resultant. Cal advertir que, per exemple, quan especifiquem un valor pel preu del menú, demanem implícitament que el restaurant serveixi menús. Per tant, els restaurants que no serveixin menús seran filtrats. El mateix s'aplica al preu de la carta.

A l'àrea *Especialitats*, podem restringir la cerca als restaurants que cuinin una certa especialitat. Si deixem el camp en blanc s'acceptarà qualssevol especialitat.

Finalment, a l'àrea *Més coses*, podem sol·licitar que s'excloguin els restaurants que no serveixin menú a la nit, que no tinguin aparcament o que no tinguin zona de fumadors. També podrem imposar un límit màxim sobre el número de places (descartant-se els restaurants de capacitat superior) o imposar l'acceptació d'una *targeta de crèdit* en particular.

Prement el botó *d'Acord* confirmarem les restriccions i tornarem a la finestra principal.

Un cop definit el criteri de cerca, podem determinar quina informació volem obtenir de cada un dels restaurants que rebem com a resultat. Ho farem mitjançant la finestra mostrada a la Figura 15, on podrem seleccionar individualment la informació que ens interessi. *Informació de contacte* i *Preus* no es poden deseleccionar perquè l'aplicació les utilitza internament. Si activem la casella *Extracte de la carta* s'activa un menú desplegable on podrem seleccionar la part de la carta que volem rebre. Prement el botó *Canviar* confirmarem la selecció i tornarem a la finestra principal.

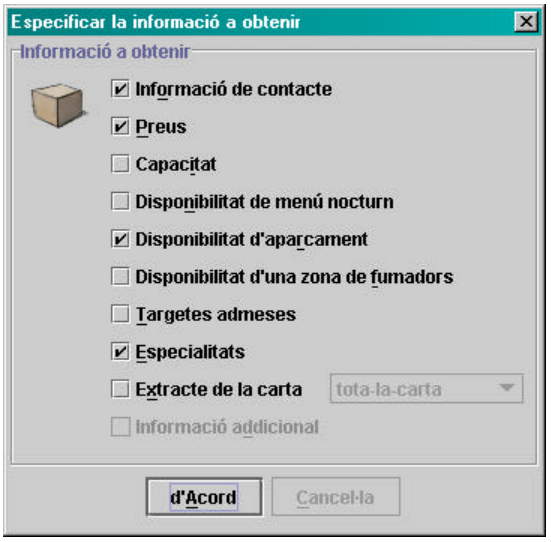


Figura 15: Finestra que ens permet determinar la informació a obtenir

Com ja hem apuntat abans, un cop finalitzat el procés de cerca es mostrarà automàticament la pestanya *Reserva* on una taula mostrarà un resum de la cerca amb el nom i els preus de cada restaurant coincident (Figura 16). Si volem veure la resta de la informació sol·licitada només hem de seleccionar el restaurant desitjat i prémer el botó *Informació detallada*. Apareixerà una finestra com la de la Figura 17. La informació que hi haurà en aquesta finestra dependrà de la que l'usuari hagi sol·licitat (Figura 15). Si el restaurant ens convenç podem fer una reserva prement sobre el botó *Reservar*.

Figura 16: Resultat de la cerca

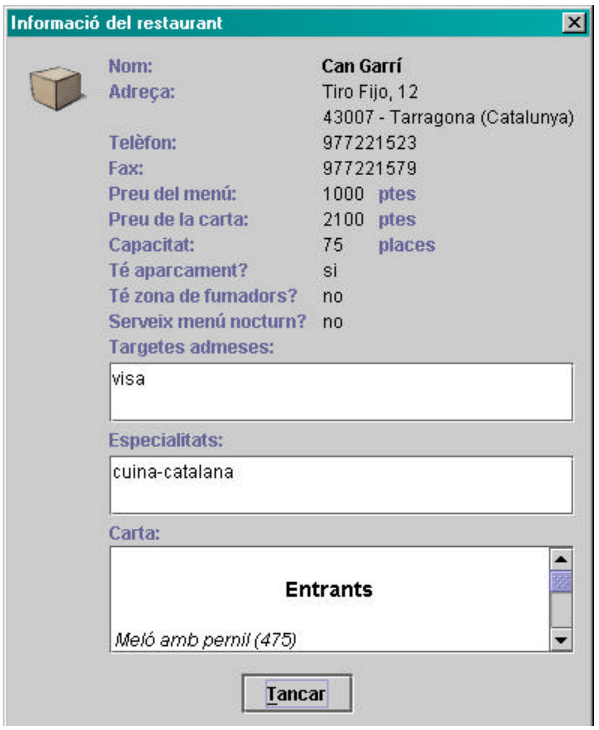


Figura 17: Informació detallada del restaurant

Automàticament apareixerà la finestra de la Figura 18 on haurem d'inserir el *número de persones* per les que volem fer la reserva, el *dia* i l'*hora* i un *nom* i un *telèfon* de contacte. Per enviar la sol·licitud de reserva al restaurant hem de prémer de nou sobre el botó *Reservar*.

Figura 18: Formulari de reserva

5.2 Agent restaurant

L'agent restaurant també posseeix una interfície. Cal recordar que els agents restaurant implementats només tenen una funció de simulació i que, per tant, qui interactuarà amb aquesta interfície serà el desenvolupador. Bàsicament té una funcionalitat informativa encara que també ens permet canviar el preu del menú a la *caché* del *broker*. La Figura 19 mostra el seu aspecte.

Figura 19: La GUI de l'agent restaurant

Els botons *Estat de les reserves* i *Informació* són purament informatius. El primer ens mostra una taula com la de la Figura 20 on es pot veure l'estat actual de les reserves, ordenades per dia. El segon ens informa de les característiques del restaurant mostrant-se uns finestra semblant a la de la Figura 17. La informació que apareixerà pot variar de restaurant a restaurant i dependrà de la informació proporcionada en el fitxer de configuració.

Figura 20: Taula on es mostra l'estat de les reserves del restaurant

Si el restaurant ha canviat el preu del menú ha d'informar d'aquest fet al *broker*. Per fer-ho hem de prémer sobre el botó *Canviar el preu del menú*. Apareixerà una finestra com la de la Figura 21 on haurem d'entrar el nou preu, podent usar la casella de selecció *No disponible* si el restaurant ha decidit no servir més menús. Quan premem sobre el botó *Canviar* s'enviarà el nou preu al *broker*.

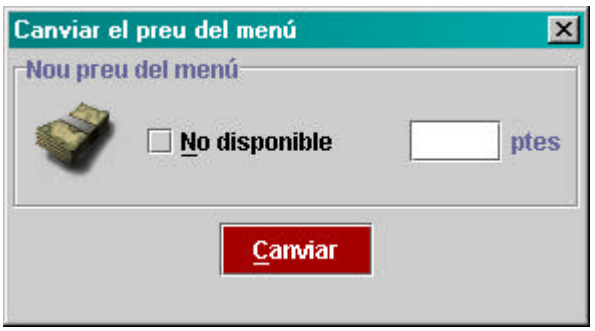


Figura 21: Finestra que ens permet canviar el preu del menú

5.2.1 El fitxer de configuració del restaurant

El fitxer de configuració detalla les característiques (el nom, els preus, la carta...) de cada restaurant. El fitxer és, bàsicament, la descripció en llenguatge FIPA-SL d'un objecte de tipus *restaurant* definit a la ontologia. Els *slots* d'aquest objecte han estat detallats al capítol 4 secció 5 d'aquest document.

A tall d'exemple inclourem aquí el fitxer de configuració d'un dels restaurants utilitzats en el joc de proves:

```
(restaurant
      :contact_info
            (restaurant-cinfo
                  :name "Can Garrí"
                  :postal_adress
                        (postal-adress
                              :country Catalunya
                              :zip_code "43007"
                              :city Tarragona
```

```
                        :street "Tiro Fijo"
                        :street_number 12
                  )
            :phone "977221523"
            :fax "977221579"
      )
:prices
      (prices
            :menu_price 1000
            :carte_price 2100
      )
:has_night_menu false
:capacity 75
:has_parking true
:has_smokers_zone false
:admitted_cards (set visa)
:specialities (set cuina-catalana)
:carte (carta
      :entrants (set
            (plat :nom "Meló amb pernil" :preu 475)
            (plat :nom "Amanida verda" :preu 450)
      )
      :primers (primers
            :pastes (set
                  (plat :nom "Tallarines a la crema" :preu 500)
                  (plat :nom "Amanida de pasta" :preu 475)
            )
            :sopes (set
                  (plat :nom "Sopa de peix" :preu 725)
                  (plat :nom "Sopa de crema de marisc a la
                  marinera" :preu 775)
            )
            :verdures (set
                  (plat :nom "Àpit amb nous" :preu 470)
                  (plat :nom "Curry de verdures" :preu 375)
            )
            :llegums (set
                  (plat :nom "Cigrons" :preu 575)
                  (plat :nom "Amanida de llenties i salmó
                  fumat" :preu 825)
            )
            :arros (set
                  (plat :nom "Paella valenciana" :preu 750)
                  (plat :nom "Arròs a la marinera" :preu 775)
            )
      )
      :segons (segons
            :ous (set
                  (plat :nom "Ous al plat" :preu 825)
                  (plat :nom "Ous guisats amb pèsols i patates"
                  :preu 925)
            )
            :peixos (set
                  (plat :nom "Lluç a la crema" :preu 1050)
                  (plat :nom "Llenguado amb bi blanc" :preu
                  1250)
            )
            :marisc (set
                  (plat :nom "Canapès de cranc" :preu 790)
                  (plat :nom "Menjar blanc de llagosta" :preu
                  1700)
```

```
                )
                :carns (set
                      (plat :nom "Fricandó" :preu 1250)
                      (plat :nom "Vedella farcida" :preu 1025)
                )
                :aus (set
                      (plat :nom "Ànec rostit amb olives" :preu
                      1900)
                      (plat :nom "Ànec amb taronja" :preu 1750)
                )
          )
          :postres (set
                (plat :nom "Panetó italià" :preu 575)
                (plat :nom "Maduixes amb nata" :preu 675)
                (plat :nom "Flam amb nata" :preu 525)
                (plat :nom Sorbet :preu 625)
          )
    )
)
```

6 Joc de proves

Per poder provar la funcionalitat del sistema hem creat quatre agents restaurant, totalment representatius per les seves especials característiques, i hem comprovat els procediments de cerca i reserva.

Detallem primer les característiques més representatives de cada restaurant:

Nom del restaurant	CP	Preu del menú	Preu de la carta	M	A	F	Capacitat	Tarjetes admeses
Can Garrí	43007	1000	2100		✓		75	Visa
La llar del lluç	43007	1000		✓	✓		35	Visa, American Express
El racó del caníbal	43007		1300				50	
Farts	43005	1200	2000			✓	120	Visa, Eurocard, 6000, 4B

M = Serveix menú per sopar
A = Disposa d'aparcament
F = Disposa d'una zona de fumadors

Farem una primera prova per la cerca. Buscarem els restaurants que el seu codi postal sigui el 43005. En teoria el *broker* hauria de filtrar aquesta petició i només generar un *query* pel restaurant "Farts". A la sortida estàndard podem comprovar que això ha estat així:

broker1@pc0:1099/JADE: matching Zip Code with 43005.
broker1@pc0:1099/JADE: sending query to [restaurant1@pc0:1099/JADE].
broker1@pc0:1099/JADE: received 1 Inform message(s).
broker1@pc0:1099/JADE: sending back to client1@pc0:1099/JADE:
(INFORM
:sender (agent-identifier :name broker1@pc0:1099/JADE)
:receiver (set (agent-identifier :name client1@pc0:1099/JADE))
:content ((= (all ?x (is-restaurant ?x (set name contact_info prices has_parki
ng specialities) (set (matches (restaurant :contact_info (restaurant-cinfo :pos
tal_adress (postal-adress :zip_code "43005"))))))) (restaurant-set (set (
restaurant :contact_info (restaurant-cinfo :name "Farts" :postal_adress (postal-
adress :country Catalunya :zip_code "43005" :city Tarragona :street "Rep·blica A
rgentina" :street_number 35) :phone "977227896" :fax "977227897") :prices (pri
ces :menu_price 1200 :carte_price 2000) :has_parking false :specialities (set c
uina-catalana bufet-lliure) :name restaurant1@pc0:1099/JADE)))))
:reply-with client1@pc0:1099/JADE1001619250090
:in-reply-to Queryclient11001619250030
:language FIPA-SL
:ontology agent-cities.restaurants.ontology
:protocol FIPA-Query
:conversation-id Queryclient11001619250030
)

Ara, li sol·licitarem els restaurants amb codi postal 43006. Si tot va bé, el *broker* ens respondrà immediatament informant-nos que no hi ha cap restaurant amb aquest codi postal. A la sortida estàndard podem veure que ha estat així:

```
broker1@pc0:1099/JADE: matching Zip Code with 43006.
broker1@pc0:1099/JADE: sending back to client1@pc0:1099/JADE:
(INFORM
 :sender  ( agent-identifier  :name broker1@pc0:1099/JADE)
 :receiver  (set ( agent-identifier  :name client1@pc0:1099/JADE) )
 :content  ((= (all ?x (is-restaurant ?x (set name contact_info prices has_parki
ng specialities ) (set (matches (restaurant :contact_info (restaurant-cinfo :pos
tal_adress (postal-adress :zip_code "43006" ) ) ) ) ) ) ) (restaurant-set (set )
 ) ) )
 :reply-with  client1@pc0:1099/JADE1001619521970
 :in-reply-to  Queryclient11001619521970
 :language  FIPA-SL
 :ontology  agent-cities.restaurants.ontology
 :protocol  FIPA-Query
 :conversation-id  Queryclient11001619521970
)
```

Per completar aquesta part del joc de proves cercarem els restaurants amb aparcament. Com que és una característica que no pot filtrar, el *broker* ha de generar quatre *querys*, un per cada restaurant. Comprovem-ho:

```
broker1@pc0:1099/JADE: sending query to [restaurant1@pc0:1099/JADE
restaurant2@pc0:1099/JADE restaurant3@pc0:1099/JADE
restaurant4@pc0:1099/JADE ].
broker1@pc0:1099/JADE: received 2 Inform message(s).
broker1@pc0:1099/JADE: sending back to client1@pc0:1099/JADE:
(INFORM
 :sender  ( agent-identifier  :name broker1@pc0:1099/JADE)
 :receiver  (set ( agent-identifier  :name client1@pc0:1099/JADE) )
 :content  ((= (all ?x (is-restaurant ?x (set name contact_info prices has_parki
ng specialities ) (set (matches (restaurant :has_parking true ) ) ) ) ) (restaur
ant-set (set (restaurant :contact_info (restaurant-cinfo :name "Can GarrÝ" :post
al_adress (postal-adress :country Catalunya :zip_code "43007" :city Tarragona :s
treet "Tiro Fijo" :street_number 12 ) :phone "977221523" :fax "977221579" ) :pri
ces (prices :menu_price 1000 :carte_price 2100 ) :has_parking true :specialities
 (set cuina-catalana ) :name restaurant2@pc0:1099/JADE ) (restaurant :contact_in
fo (restaurant-cinfo :name "La llar del lluþ" :postal_adress (postal-adress :cou
ntry Catalunya :zip_code "43007" :city Tarragona :street "Baixada de l'espantaso
gres" :street_number 2 ) :phone "977221523" ) :prices (prices :menu_price 1000 )
 :has_parking true :specialities (set peix cuina-marinera ) :name restaurant4@pc
0:1099/JADE ) ) ) ) )
 :reply-with  client1@pc0:1099/JADE1001620226060
 :in-reply-to  Queryclient11001620226060
 :language  FIPA-SL
 :ontology  agent-cities.restaurants.ontology
 :protocol  FIPA-Query
 :conversation-id  Queryclient11001620226060
)
```

Com a resultat, els restaurants esperats: “Can Garrí” i “La llar del lluç”.

Anem a provar ara el mecanisme de reserva.

Primer provarem de sol·licitar reserva a un restaurant per una data que ja ha passat. Per exemple, el 24 de setembre del 2001. El restaurant, com era d'esperar, ens respon que la data és incorrecta (*out-of-date*):

```
restaurant1@pc0:1099/JADE: Received REQUEST message from
client1@pc0:1099/JADE:
(REQUEST
 :sender  ( agent-identifier  :name client1@pc0:1099/JADE)
 :receiver  (set ( agent-identifier  :name restaurant1@pc0:1099/JADE) )
 :content  ((action (agent-identifier :name client1@pc0:1099/JADE :addresses (se
quence ) :resolvers (sequence ) ) (make-booking 5 (timestamp :minute 30 :hour 22
 :day 24 :month 9 :year 2001 ) (user-cinfo :name "Gavarr¾" :phone "677867656" )
) ) )
 :reply-with  Req1001620481740
 :language  FIPA-SL
 :ontology  agent-cities.restaurants.ontology
 :protocol  fipa-request
 :conversation-id  Req1001620481740
)
restaurant1@pc0:1099/JADE: Making booking for Gavarr¾ (677867656) on Mon Sep 24
22:30:00 GMT+02:00 2001: Out-of-date.
```

Ara, provarem de reservar més enllà de la data que el restaurant considera màxima per realitzar una reserva (8 dies a partir del dia actual). Reservem doncs pel 20 de novembre del 2001. El resultat és el mateix que abans:

```
restaurant2@pc0:1099/JADE: Received REQUEST message from
client1@pc0:1099/JADE:
(REQUEST
 :sender  ( agent-identifier  :name client1@pc0:1099/JADE)
 :receiver  (set ( agent-identifier  :name restaurant2@pc0:1099/JADE) )
 :content  ((action (agent-identifier :name client1@pc0:1099/JADE :addresses (se
quence ) :resolvers (sequence ) ) (make-booking 5 (timestamp :minute 30 :hour 13
 :day 20 :month 11 :year 2001 ) (user-cinfo :name "Gavarr¾" :phone "677867656" )
 ) ) )
 :reply-with  Req1001620809420
 :language  FIPA-SL
 :ontology  agent-cities.restaurants.ontology
 :protocol  fipa-request
 :conversation-id  Req1001620809420
)
restaurant2@pc0:1099/JADE: Making booking for Gavarr¾ (677867656) on Tue Nov 20
13:30:00 GMT+01:00 2001: Out-of-date.
```

Ara, provarem de reservar taula per 75 persones (el màxim és 25). El restaurant informa correctament del motiu del rebuig:

```
restaurant1@pc0:1099/JADE: Received REQUEST message from
client1@pc0:1099/JADE:
(REQUEST
 :sender  ( agent-identifier  :name client1@pc0:1099/JADE)
 :receiver  (set ( agent-identifier  :name restaurant1@pc0:1099/JADE) )
 :content  ((action (agent-identifier :name client1@pc0:1099/JADE :addresses (se
quence ) :resolvers (sequence ) ) (make-booking 75 (timestamp :minute 30 :hour 1
3 :day 29 :month 9 :year 2001 ) (user-cinfo :name "Gavarr¾" :phone "677867656" )
 ) ) )
 :reply-with  Req1001620970190
 :language  FIPA-SL
```

:ontology agent-cities.restaurants.ontology
:protocol fipa-request
:conversation-id Req1001620970190
)
restaurant1@pc0:1099/JADE: Making booking for Gavarr¾ (677867656) on Sat Sep 29 13:30:00 GMT+02:00 2001: Too many people.

Provarem de fer una reserva fora de l'horari d'obertura del restaurant, per exemple, per a les 6:30. El restaurant hauria de respondre amb un *not-open*:

restaurant3@pc0:1099/JADE: Received REQUEST message from client1@pc0:1099/JADE:
(REQUEST
:sender (agent-identifier :name client1@pc0:1099/JADE)
:receiver (set (agent-identifier :name restaurant3@pc0:1099/JADE))
:content ((action (agent-identifier :name client1@pc0:1099/JADE :addresses (se quence) :resolvers (sequence)) (make-booking 5 (timestamp :minute 30 :hour 6 :day 29 :month 9 :year 2001) (user-cinfo :name "Gavarr¾" :phone "677867656"))))
:reply-with Req1001621117060
:language FIPA-SL
:ontology agent-cities.restaurants.ontology
:protocol fipa-request
:conversation-id Req1001621117060
)
restaurant3@pc0:1099/JADE: Making booking for Gavarr¾ (677867656) on Sat Sep 29 06:30:00 GMT+02:00 2001: Not open.

Per acabar aquest joc de proves, demostrarem que el restaurant també és capaç de reservar-nos una taula:

restaurant1@pc0:1099/JADE: Received REQUEST message from client1@pc0:1099/JADE:
(REQUEST
:sender (agent-identifier :name client1@pc0:1099/JADE)
:receiver (set (agent-identifier :name restaurant1@pc0:1099/JADE))
:content ((action (agent-identifier :name client1@pc0:1099/JADE :addresses (se quence) :resolvers (sequence)) (make-booking 5 (timestamp :minute 30 :hour 22 :day 29 :month 9 :year 2001) (user-cinfo :name "Gavarr¾" :phone "677867656"))))
:reply-with Req1001621280740
:language FIPA-SL
:ontology agent-cities.restaurants.ontology
:protocol fipa-request
:conversation-id Req1001621280740
)
restaurant1@pc0:1099/JADE: Making booking for Gavarr¾ (677867656) on Sat Sep 29 22:30:00 GMT+02:00 2001: Ok.

7 Conclusions

La tecnologia d'agents, tot i ser encara incipient, té un gran futur. Justament és ara quan s'està començant a posar en pràctica els seus postulats i els resultats són realment esperançadors. Sobretot perquè s'ha demostrat, per exemple amb aquest treball, que són aplicables a problemes reals i que proporcionen solucions amb uns costos en desenvolupament bastant continguts.

L'esforç que està fent la FIPA en el disseny d'estàndards i el treball d'altres organitzacions que han fet possible que puguem utilitzar eines com el JADE són només el començament. La tecnologia d'agents rebrà un fort impuls quan s'adoptin estàndards que unifiquin les diferents aproximacions aparegudes fins al moment i serà llavors quan projectes com l'*Agent Cities* puguin portar-se a terme de forma global, no només en petites, i no tant petites, implementacions puntuals.

Tanmateix, les tecnologies d'agents demanen una evolució en el camp dels llenguatges com el que va passar amb la irrupció de les tecnologies orientades a objectes. De moment implementem sistemes multiagent mitjançant llenguatges de programació orientada a objectes amb el suport d'unes quantes llibreries que ens proporcionen la plataforma. Potser seria interessant el desenvolupament d'un nou llenguatge que tingués en compte les especials característiques d'aquest nou paradigma i s'hi adaptés.

8 Valoració personal

Aquest treball m'ha proporcionat la possibilitat d'entrar en contacte amb una tecnologia, la d'agents, que pràcticament desconeixia, i de posar en pràctica els coneixements adquirits a l'assignatura d'Intel·ligència Artificial.

Pel que fa a les eines utilitzades, JADE és bo però potser et limita massa a l'hora de dissenyar cada agent. El fet de tenir que cenyir-se als *behaviours* com a fils d'execució de l'agent, amb les corresponents polítiques de *preemtive multitasking*, implica un cost de desenvolupament addicional quan es vol treballar amb funcions bloquejants. A més, el codi no està preparat per suportar execucions multifil fet que t'obliga a modificar-lo o a crear una capa addicional de processament per cada agent per poder, per exemple, interactuar amb la GUI.

El present projecte encara està en fase embrionària però cal tenir en compte que ha nascut d'una necessitat que encara no ha estat coberta. Per tant, pensar que el que estàs desenvolupant té una utilitat pràctica i pot ser útil, t'anima continuar treballant.

9 Treball futur

El treball futur hauria de derivar cap a l'adaptació del codi existent per que suporti una cerca per ciutat. Aquest fet ens obligaria a plantejar-nos una jerarquia de *brokers* i a modificar l'agent client per que sigui capaç de canviar de *broker* quan sigui necessari. Establir una jerarquia de *brokers* implicarà, sens dubte, l'aparició d'actes comunicatius entre aquests.

Seria també interessant poder oferir una cerca per plat, tenint en compte que segurament hauríem de modificar la representació ontològica de la carta. El punt crucial seria definir una jerarquia prou representativa de tot el panorama gastronòmic mundial.

Finalment, i no menys important, caldria aprofundir en les necessitats dels potencials clients, tant els restaurants com els seus usuaris, i intentar plasmar aquestes necessitats en l'arquitectura del sistema.

Referències

[1] Foundation for Intelligent Physical Agents, web page: http://www.fipa.org

[2] AgentCities, web page: http://www.agentcities.org

[3] GruSMA, Grup de Sistemes Multi-Agent, DEIM, Universitat Rovira i Virgili, plana web: http://www.etse.urv.es/recerca/banzai/toni/MAS

[4] JADE, Java Agent DEvelopment Framework, web page: http://sharon.cselt.it/projects/jade

[5] Foundation for Intelligent Physical Agents (2000), *FIPA ACL Message Structure Specification*, FIPA

[6] Foundation for Intelligent Physical Agents (2000), *FIPA Content Language Specification*, FIPA

[7] Foundation for Intelligent Physical Agents (2000), *FIPA SL Content Language Specification*, FIPA

[8] Foundation for Intelligent Physical Agents (2000), *FIPA Query Interaction Protocol Specification*, FIPA

[9] Foundation for Intelligent Physical Agents (2000), *FIPA Request Interaction Protocol Specification*, FIPA

[10] Foundation for Intelligent Physical Agents (2000), *FIPA Contract Net Interaction Protocol Specification*, FIPA

[11] Fabio Bellifine; Giovanni Caire; Tiziana Trucco and Giovanni Rimassa (18-December-2000), *JADE Programmer's Guide*. CSELT S.p.A and University of Parma, http://sharon.cselt.it/projects/jade/

[12] Foundation for Intelligent Physical Agents (2000), *FIPA Agent Management Specification*, FIPA

[13] Isern D., Moreno A., Valls A. (2001), *Desenvolupament de sistemes Multi-Agent en JADE*, Report de Recerca DEIM-RR-01-003, Dept. Enginyeria Informàtica i Matemàtiques, Universitat Rovira i Virgili