PROJECTE DE FINAL DE CARRERA

Ninots, un entorn d'aprenentatge en matèria de sistemes multi-agent i mineria de dades

Albert Gavarró Rodríguez
albert.gavarro@estudiants.urv.cat
Enginyeria en Informàtica

Dirigit per la **Dra. Aida Valls Mateu**

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

Curs 2005-2006

Taula de continguts

Índex de figures

Índex de taules

Abreviacions

ACC	Agent Communication Channel
ACL	Agent Communication Language
AMS	Agent Management System
DF	Directory Facilitator
FIPA	Foundation for Intelligent Physical Agents
FIPA-CCL	FIPA Costraint Choice Language
FIPA-KIF	FIPS Knowledge Interchange Format
FIPA-RDF	FIPA Resource Description Framework
FIPA-SL	FIPA Semantic Language
GUI	Graphical User Interface
IA	Intel·ligència Artificial
IAII	Itel·ligència Artificial II
IEEE	Institute of Electrical and Electronics Engineers, Inc.
JADE	Java Agent DEvelopement framework
JDK	Java Development Kit
Jess	Java Expert System Shell
JVM	Java Virtual Machine
KQML	Knowledge Query Modeling Language
MTS	Message Transport System
RMA	Remote Management Agent
RMI	Remote Method Invocation
SBC	Sistemes Basats en el Coneixement
SE	Sistemes Experts
SMA	Sistema Multi-Agent
SMTP	Simple Mail Tranfer Protocol
UCI	University of California, Irvine

1 Introducció

Aquest document presenta Ninots, un sistema multi-agent (SMA) concebut com a suport per realitzar pràctiques en matèria de mineria de dades i agents intel·ligents.

En aquest capítol es veurà quins són els objectius d'aquest projecte, així com els requisits generals que es deriven d'aquests.

1.1 Objectius

Aquest projecte pretén dissenyar i construir un SMA que serveixi d'esquelet per poder desenvolupar una de les pràctiques de l'assignatura d'Intel·ligència Artificial II (IAII).

La idea és crear un entorn que permeti als alumnes realitzar una pràctica no gaire extensa que serveixi per entendre millor els temes "Descobriment de coneixement" i "Sistemes multi-agent". Per tant, es vol muntar un SMA on els alumnes puguin posar-hi agents que apliquin tècniques de descobriment de coneixement.

L'objectiu final és crear una eina que permeti entendre l'estructura i la dinàmica d'un SMA sense arribar-lo a implementar, alhora que es codifiquen i s'avaluen diferents algorismes de mineria de dades.

1.2 Esquema general

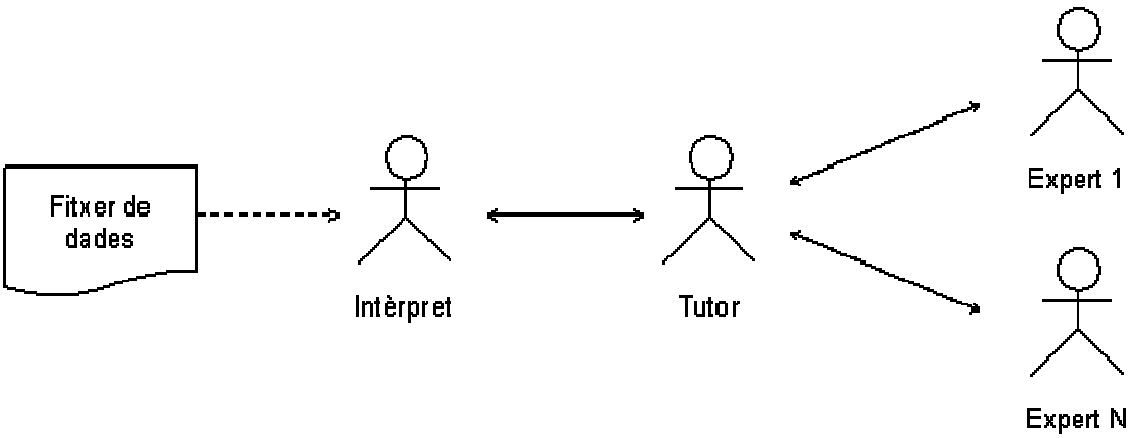


Figura 1: Esquema general

Esquemàticament, el sistema estarà compost per les següents entitats:

- Un o més agents intèrprets que s'encarregaran de llegir un conjunt de dades sobre un domini qualsevol.

- Un o més agents experts que, donat un conjunt de dades, l'hi apliqui tècniques de mineria de dades per descobrir-hi nou coneixement en forma de regles, que li permetran construir una base de regles.

- Un o més agents tutors que dirigeixin l'adquisició del coneixement dels agents experts, i als que els hi puguin fer preguntes sobre el coneixement que han adquirit.

D'aquesta manera els agents són prou generals com per ser utilitzats en diferents tipus de problemes en cada curs.

1.3 Requisits generals

A l'hora d'abordar el disseny del sistema, ens fixem una sèrie de requisits generals que aquest ha de complir. El sistema ha de permetre:

- Formar als alumnes en l'arquitectura i el funcionament d'un SMA, deslliurant-los de la seva implementació però sense comprometre l'aprenentatge en aquest camp.

- Als alumnes, crear agents experts de forma ràpida i fàcil que, per mitjà d'un algorisme de mineria de dades implementat per ells mateixos, obtinguin coneixement en forma de regles sobre un conjunt de dades determinat. Aquest coneixement s'haurà d'emmagatzemar a la base de regles de l'agent.

- Consultar el coneixement dels agents experts per mitjà de preguntes.

- Als alumnes, observar les diferències entre els diferents algorismes de mineria de dades que implementin.

- Al professor, avaluar la qualitat dels diferents algorismes de mineria de dades implementats pels alumnes.

- Accedir a diferents conjunts de dades de forma homogènia, encara que el seu format d'emmagatzematge sigui notablement diferent.

A banda de tot això, s'haurà de proporcionar documentació completa i detallada sobre el funcionament del sistema, així com de les llibreries de programació a les que tindran accés als alumnes per tal de generar agents experts. Cal remarcar que aquest punt és vital si tenim en compte que els alumnes, a priori, tindran pocs coneixements sobre implementació de SMA, ja que aquest no és l'objectiu de l'assignatura.

1.4 Representació del coneixement

Per tal que els agents puguin adquirir un coneixement sobre un domini determinat, cal que aquest coneixement es pugui expressar d'alguna manera. El mecanisme de representació del coneixement ha de ser, alhora, prou flexible com per poder representar els diferents dominis que s'utilitzaran en les diferents pràctiques.

Donat que ja existeixen formats de representació d'aquest tipus de dades, s'ha optat per adaptar un format ja conegut. Concretament, s'ha pres com a referència el format d'un dipòsit de dades molt utilitzat per la comunitat científica en Intel·ligència Artificial (IA)

per testejar mètodes de mineria de dades: el *Machine Learning Repository*, de la Universitat de Califòrnia (UCI)[1].

En aquests conjunts de dades la informació del domini es representa en la forma de parells atribut-valor. Cada domini disposa d'un conjunt d'atributs que el descriuen. Els objectes del domini prenen un valor per cadascun d'aquests atributs.

Els agents han de ser capaços d'interpretar quins són els atributs (o característiques) del domini amb el que treballaran així com els objectes (o instàncies) associats amb aquest domini, i els seus valors.

A l'Apèndix C es pot trobar un conjunt de dades d'exemple.

1.5 Aprenentatge, extracció del coneixement o mineria de dades

Quan es parla d'aprenentatge, extracció del coneixement o mineria de dades, es fa referència a una sèrie de tècniques que, donat un conjunt de dades, permeten subdividir-los en grups, basant-se en característiques que fan que el conjunt de les dades que integren cadascun d'aquests grups ens semblin pròximes o idèntiques.

Per exemple, si agafem un grup prou gran d'individus de la raça humana i estudiem algunes de les seves característiques físiques, ens adonem que podem classificar-los en tres grups ben diferenciats (subraces) segons la seva pigmentació i la seva pilositat: la subraça caucasoide o euròpida, de pigmentació més aviat clara i pilositat alta, la subraça nègrida o negroide, de pigmentació fosca i pilositat mitja, i la raça mongòlida o mongoloide, de pigmentació grogosa i pilositat molt baixa. Observant aquestes característiques podem construir, doncs, una regla que digui que "un individu que presenti pigmentació clara i pilositat alta serà caucasoide". Aquest serà el coneixement que haurem adquirit.

1.6 Sistemes experts i consulta del coneixement

Quan tenim un conjunt de regles sobre un determinat domini del coneixement, aquestes es poden usar per construir un sistema intel·ligent que usi aquest coneixement per fer inferència; És el que es coneix com Sistemes Basats en el Coneixement (SBC) o Sistemes Experts (SE).

En els sistemes experts, s'utilitza la base de regles per fer el procés invers al de la mineria de dades: donades unes característiques típiques d'un ens del domini, classificar aquest ens en alguna de les categories detectades durant el procés d'extracció del coneixement. Tornant a l'exemple de les subraces humanes, si de cop i volta ens presenten un individu pàl·lid que presenta un alt índex de pilositat, podrem inferir que es tracta d'un caucasoide, perquè així ens ho diuen les regles que hem après abans.

[1] http://www.ics.uci.edu/~mlearn/MLRepository.html

1.7 Estructura del document

Aquest document consta de tres grans blocs en els que es farà una breu introducció als sistemes multi-agent, es descriurà concisament el disseny i l'arquitectura del SMA implementat, i s'introduirà a l'ús de les eines JADE i Jess.

Al capítol 2 s'explicaran els conceptes d'agent i d'SMA, es descriuran les principals característiques dels agents, i es farà una breu exposició de l'estàndard FIPA pel que fa a la gestió i a la comunicació entre agents.

Al capítol 3 s'abordarà el disseny de Ninots, s'explicarà el model del coneixement utilitzat així com la interfície programable del sistema, i es descriuran les classes que aquest ofereix per accedir a les dades d'entrada i per generar regles.

Al capítol 4 es detallaran els agents que conformen del sistema i el paper que juga cadascun d'ells en el seu funcionament, els serveis que ofereixen i com interaccionen entre ells. També s'explicaran els protocols i l'ontologia que utilitzen per comunicar-se, i s'explicarà el funcionament del pont de traducció de regles entre Ninots i el Jess.

El manual de l'usuari, tant d'utilització com de programació, es troba al capítol 5.

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

Els capítols 7 i 8 estan reservats a les conclusions que es desprenen de la realització d'aquest treball, i a les línies de treball futur.

L'Apèndix A està dedicat, exclusivament, a introduir al lector a l'eina de desenvolupament de sistemes multi-agent JADE. Es veurà com implementar un SMA sobre aquesta plataforma i s'estudiaran les eines d'administració i monitorització que ofereix.

Finalment, a l'Apèndix B es fa una breu introducció a l'entorn de creació de sistemes experts Jess, incidint, sobretot, en la interfície de programació entre Java i Jess.

2 Agents i sistemes multi-agent

En aquest apartat es descriurà què és un agent i com aquests s'agrupen per formar sistemes multi-agent. A més a més, s'introduirà la figura de la FIPA com a entitat reguladora tant dels protocols que regeixen la comunicació entre els agents com de l'estructura mateixa dels sistemes multi-agent.

2.1 Agents: definició i propietats

En informàtica, un agent és una abstracció, un model mental que descriu un programari que actua per un usuari o un altre programa en una relació de representació. Aquesta "acció en representació de" implica l'autoritat de decidir quan (i si) l'acció és apropiada. La idea és que els agents no són estrictament invocats per una tasca sinó que s'activen ells mateixos.

El concepte d'agent proveeix un mètode apropiat i potent de descriure a una entitat de programari capaç d'actuar amb un cert grau d'autonomia amb l'objectiu d'acomplir tasques en representació del seu usuari. Contràriament als objectes de programari, que estan definits en termes dels seus mètodes i atributs, un agent està definit en termes del seu comportament.

Diversos autors han proposat diferents definicions sobre els agents, les quals inclouen, de forma comuna, els següents conceptes:

- **Persistència:** El codi de l'agent no s'executa sota demanda sinó que corre contínuament i decideix per ell mateix quan ha de realitzar una determinada activitat.

- **Autonomia:** Els agents tenen la capacitat de triar, de forma autònoma i sense intervenció humana, la tasca que han de realitzar per assolir el seu objectiu, i de prioritzar-les en cas de conflictes.

- **Sociabilitat:** Els agents són capaços d'establir una comunicació amb altres entitats de programari (siguin agents o no) a través de mecanismes de comunicació i coordinació. Eventualment, dos o més agents poden col·laborar en una mateixa tasca.

- **Reactivitat:** Els agents són capaços de percebre el context en el que s'executen i reaccionar als seus canvis de forma apropiada.

També presenten altres propietats, menys destacades:

- **Mobilitat:** Els agents poden moure's a altres entorns a través d'una xarxa.

- **Veracitat:** S'assumeix la veracitat de la informació proporcionada per l'agent. Un agent no comunicarà mai informació falsa de forma premeditada.

- **Benevolència:** S'assumeix la missió no conflictiva de l'agent. Un agent únicament farà allò pel que és requerit.

- **Racionalitat:** S'assumeix una conducta coherent de l'agent. Un agent sempre actuarà per assolir els seus objectius.

2.2 Què és i què no és un agent?

Per aclarir el concepte d'agent, el compararem amb altres conceptes relacionats.

Els agents no són programes. Franklin & Graesser (1996)[2] defineixen quatre nocions clau que distingeixen els agents dels programes comuns:

- La reactivitat.
- L'autonomia.
- El comportament orientat a la consecució d'objectius.
- La persistència.

Els agents no són objectes:

- Els agents són més autònoms que els objectes.
- Els agents tenen un comportament flexible: reactiu, proactiu i social.
- Els agents tenen un fil de control però en poden tenir més.

Els agents no són sistemes experts:

- Els sistemes experts no interactuen amb el seu entorn.
- Els sistemes experts no estan dissenyats per tenir un comportament actiu.
- Als sistemes experts no es contempla la proactivitat.

2.3 Sistemes multi-agent

Quan diversos agents actuen o interactuen formen un SMA. Generalment, els agents que els formen no disposen de totes les dades o de tots els mètodes disponibles per assolir els seus objectius, fet que els obliga a col·laborar amb la resta d'agents. En aquest tipus de sistemes, les dades estan descentralitzades i l'execució és asíncrona.

La utilització de sistemes multi-agent proporciona uns beneficis clars:

- **Modularitat:** La distribució en tasques redueix la complexitat i permet una programació més estructurada. El codi dels agents sol ser breu i molt específic.
- **Eficiència:** L'essència distribuïda dels sistemes multi-agent ens aporta paral·lelisme quan tenim els agents repartits en màquines diferents.

[2] http://www.msci.memphis.edu/~franklin/AgentProg.html

- **Fiabilitat:** El sistema funciona al marge dels agents. Si un agent cau, la resta d'agents i el sistema en si continuen funcionant. Es pot aconseguir una major tolerància a fallides replicant els agents, generant així una redundància de serveis.

- **Flexibilitat:** Es poden afegir i treure agents del sistema de forma dinàmica. El sistema no s'ha d'aturar per eliminar un agent o afegir-ne un de nou.

2.4 L'estàndard de la FIPA

La FIPA[3] és una organització d'estandardització de la IEEE Computer Society que promou la tecnologia basada en agents i la interoperabilitat dels seus estàndards amb altres tecnologies.

Les especificacions de la FIPA representen una col·lecció d'estàndards destinats a promoure la interoperabilitat entre agents heterogenis i els serveis que ofereixen.

2.4.1 Estructura d'un SMA

La FIPA estableix l'entorn en el qual un agent neix, (inter)actua i mor. Es tracta d'un model lògic que regeix la creació, el registre, la comunicació, l'establiment, la mobilitat i la destrucció d'agents.

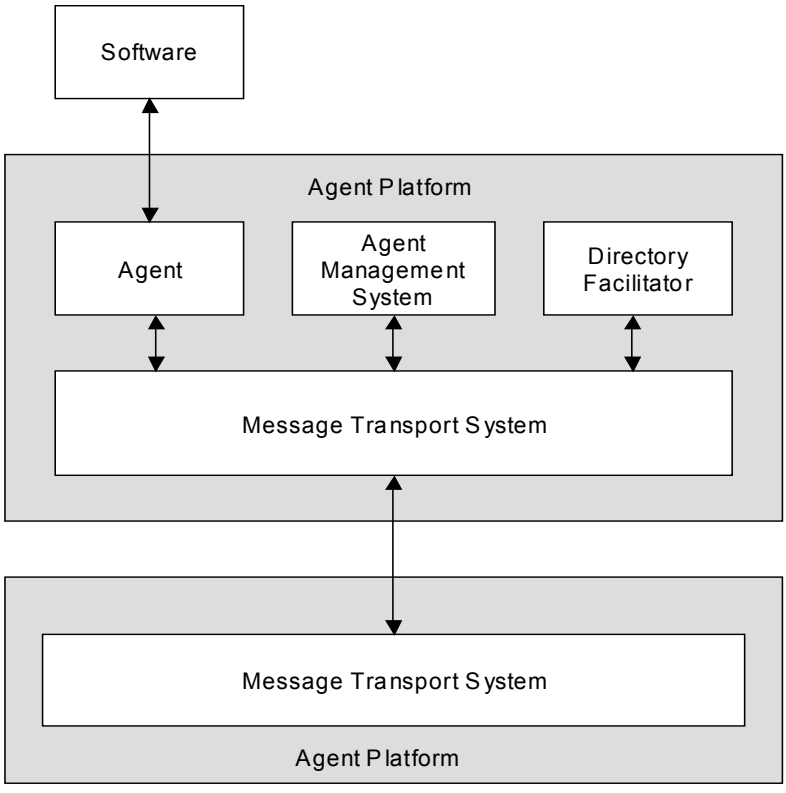


Figura 2: Estructura d'un SMA

[3]http://www.fipa.org/

Segons la FIPA, un SMA ha d'estar composat per:

- **L'*Agent Platform*:** Proporciona la plataforma, la infraestructura bàsica per que es puguin crear o executar els 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 l'agent que proporciona el servei de pàgines grogues dins del sistema. Aquest agent manté informació sobre els serveis que poden oferir els diferents agents que es troben a l'SMA. Aquesta informació no l'obté directament, doncs són els agents els qui, de forma individual, 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-ne un per plataforma. Ofereix, a més a més, un servei de pàgines blanques perquè 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 System*:** És la via de comunicació entre els agents del sistema, estiguin o no a la mateixa plataforma.

- **Programari:** És un conjunt d'eines accessibles pels agents però que no forma part d'aquests.

2.4.2 Comunicació entre els agents

La capacitat comunicativa és un tret fonamental dels sistemes multi-agent. L'acció d'estandardització de la FIPA també aborda aquest aspecte, imprescindible per poder tenir un SMA.

2.4.2.1 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*).

Els missatges ACL es basen en la teoria de la parla, que estipula que les comunicacions individuals entre agents poden reduir-se a un petit nombre d'idees o, més generalment, d'actes comunicatius. Aquests actes son les peces sobre les que es construeix la comunicació.

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

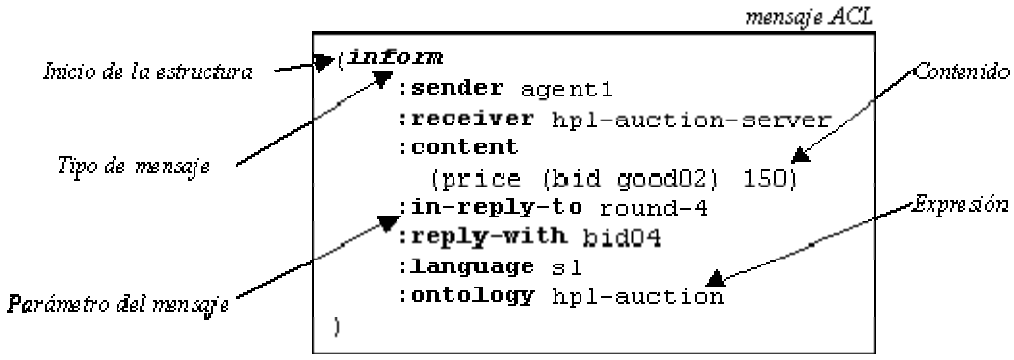


Figura 3: Format d'un missatge ACL

Tot missatge té una estructura prefixada i subjecte a les nostres necessitats (veure Figura 3). 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:** El paràmetre *performative* indica el tipus d'acte comunicatiu que volem representar amb el nostre missatge. Més endavant es veurà que el valor del paràmetre *performative* està fortament associat al protocol. Exemples de *performatives* són:
    - **Agree:** Utilitzada per confirmar a l'agent sol·licitant d'una acció que aquesta serà realitzada.
    - **Cancel:** Utilitzada quan l'agent sol·licitant d'una acció vol cancel·lar-la renunciant als resultats.
    - **Failure:** Utilitzada per l'actor d'una acció per notificar al sol·licitant una eventual fallida realitzant l'acció encomanada.
    - **Inform:** Utilitzada per l'actor d'una acció per notificar al sol·licitant els resultats de la realització de l'acció. Només s'utilitza quan l'acció s'ha completat correctament.
    - **Not-understood:** Utilitzada pel receptor d'un missatge per notificar a l'emissor de que el missatge no s'ha entès.
    - **Refuse:** Utilitzada per refusar la realització d'una acció, explicant sempre la raó de la negativa.
    - **Request:** Utilitzada per sol·licitar la realització d'una acció a un altre agent.
- **Identificació dels agents participants:** El paràmetre *sender* identifica l'emissor del missatge, el paràmetre *receiver*, el receptor, i el paràmetre *reply-to*, l'agent al que se li ha d'enviar la resposta.
- **Contingut del missatge:** El paràmetre *content* agrupa el conjunt d'idees o de fet que es volen comunicar. El contingut pròpiament dit del missatge pot estar escrit en qualssevol llenguatge, ja sigui Java, C++, Prolog, Lisp... però la FIPA, en un

esforç per cercar la compatibilitat entre les diferents implementacions, ha definit quatre tipus de llenguatges predefinits:

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

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

  - **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 de paràmetres proporciona informació sobre el contingut del missatge, tal com el llenguatge en el que està escrit (FIPA-SL, Java, Prolog, Lisp, etc), el tipus de codificació del missatge i, sobretot, la ontologia utilitzada. Una ontologia és un conjunt de termes, predicats i accions lligades a un camp del coneixement. Necessitarem definir ontologies perquè els agents puguin compartir coneixement sense ambigüitats. Les ontologies es veuran amb més detall en els propers capítols. Són paràmetres d'aquesta categoria: *language*, *encoding* i *ontology*.

- **Control de conversa:** Aquest conjunt de paràmetres 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, l'*in-reply-to*, que relaciona un missatge de resposta amb el missatge en el que es formulava la pregunta, el *reply-by*, que defineix un temps màxim en el que s'espera la resposta, i el *protocol*, que identifica el protocol en el que s'està enviant el missatge.

2.4.2.2 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 iniciador respecte a l'agent participant. La FIPA estableix una sèrie de protocols dissenyats per les interaccions més habituals, des d'una simple pregunta (*query*) o una sol·licitud (*request*), fins a una negociació completa (*contract-net*). El protocol determina el flux de la conversa i el tipus de missatges que es poden rebre a cada pas. Anem a veure amb detall el protocol més utilitzat a Ninots: el *FIPA-Request*.

A la Figura 4 podem veure el seu diagrama de flux. El *FIPA-Request* consta de tres etapes, tot i que, si l'actor proporciona una resposta ràpida i afirmativa, es pot abreujar en dues. Aquest protocol es utilitzat pels agents quan volen sol·licitar a un altre agent que els realitzi una acció.

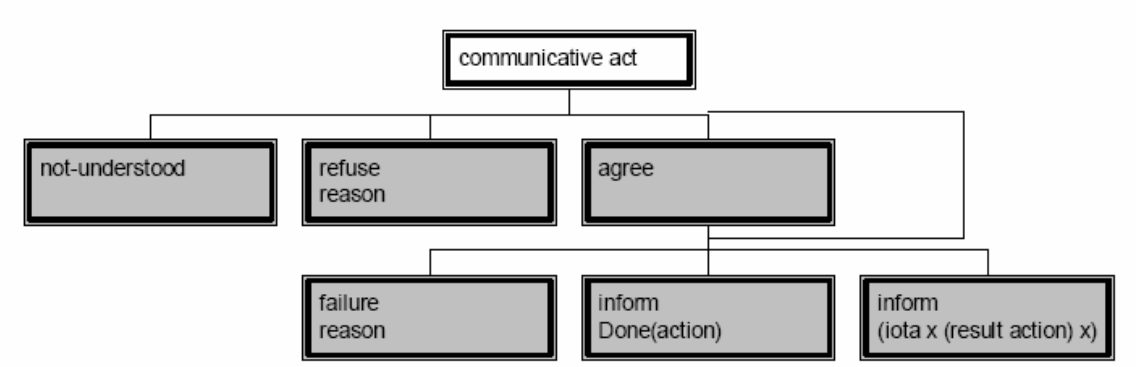


Figura 4: FIPA *Request*

En un primer pas, l'agent iniciador envia un missatge del tipus *request* (veure les *performatives*, 2.4.2.1) a l'agent participant. Quan l'agent participant rep el missatge, en un segon pas, i només si pot realitzar l'acció, pot respondre directament amb un *inform* o, si l'acció a realitzar és llarga, amb un *agree*. Si envia l'*inform*, l'acció ja s'ha realitzat i, per tant, el protocol acaba. En canvi, si respon amb un *agree*, l'acció encara s'ha de dur a terme i, per tant, l'agent participant, quan hagi acabat l'acció, haurà d'enviar un segon missatge amb el resultat de la mateixa. Si l'acció a realitzar fallés, l'agent participant respondria amb un *failure*.

Si l'agent participant, en rebre un *request*, no pogués realitzar la tasca sol·licitada, respondria amb un *refuse*. Si no hagués entès el missatge o la tasca a realitzar, ho faria amb un *not-understood*.

Finalment, cal remarcar que hi ha dues formes d'*inform*: l'*inform* pròpiament dit i la construcció *inform Done*. La diferència entre ambdós resideix en que l'*inform Done* ens comunica únicament que l'acció s'ha realitzat correctament, mentre que l'*inform* pot retornar informació sobre el resultat de l'acció.

A banda del *FIPA-Request*, la FIPA defineix aquests protocols:

- ***FIPA-Query*:** 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. És un protocol de dos passos: pregunta i resposta.

- ***FIPA-Contract-Net*:** Aquest protocol s'utilitza quan es vol realitzar una negociació entre dos o més agents. L'agent iniciador sol·licita propostes per realitzar una acció. És un protocol de quatre passos: sol·licitud, arribada de propostes, tria i notificació del resultat de l'acció.

- ***FIPA-Iterated-Contract-Net*:** És una variant de l'anterior que permet diverses rondes de negociació.

- ***FIPA-Brokering*:** Aquest protocol s'utilitza per gestionar una demanda cap a un conjunt d'agents, generalment desconeguts o poc coneguts per l'agent iniciador, que poden donar un servei. El destinatari del missatge és un agent *broker* que fa d'enllaç entre els agents que proporcionen el servei i l'agent iniciador. El nombre de passos d'aquest protocol depèn del protocol utilitzat entre el *broker* i els agents que proporcionen el servei.

- ***FIPA-Recruiting*:** És una variant de l'anterior. En aquest cas l'agent *broker* realitza operacions de reclutament sobre un conjunt d'agents (que coneix de privilegiada), que, a priori, l'agent iniciador no coneix o no en té prou informació. En aquest cas, el nombre de passos també és variable.

- ***FIPA-Subscribe*:** Permet a un agent A sol·licitar a un agent B que el notifiqui cada cop que es compleixi una determinada condició. És un protocol de tres passos: subscripció, acceptació i confirmació.

- ***FIPA-Propose*:** L'agent iniciador proposa als agents participants que farà les accions descrites a la proposta quan aquesta l'acceptin. Si els agents participants accepten, l'agent iniciador realitzarà l'acció proposada i en retornarà el resultat.

3 Disseny

3.1 Representació del coneixement

En aquest apartat es detalla com s'ha modelat el coneixement perquè pugui ser tractat pels agents. A banda de descriure'n el model general, s'aprofundeix en la representació física de les dades i en com els agents maneguen aquesta informació.

3.1.1 Model del coneixement

El coneixement, de forma genèrica, s'estructura en dominis. Un domini del coneixement de la humanitat podria ser, per exemple, l'àlgebra o la genètica.

Cada domini del coneixement té els seus atributs, i un conjunt d'objectes del domini sobre els que es construeix el model de funcionament del mateix. Així, per exemple, la Llei de la Gravitació Universal[4] és un model que es basa en observacions puntuals, com la caiguda d'una poma d'un arbre. La "caiguda d'una poma d'un arbre" seria un objecte del domini. El coneixement, la teoria deduïda a partir d'aquest i d'altres objectes.

Per poder treballar amb diferents problemes en cada curs, cal trobar un model de representació del coneixement precís, potent i flexible. Aquest model ha de permetre representar amb exactitud un domini del coneixement.

El model adoptat és el mateix que utilitzen sistemes experts com el CLIPS[5] o el Jess[6]: modelar el domini com una classe amb els seus atributs, i els objectes concrets com a instàncies d'aquesta classe. D'ara en endavant, a la classe l'anomenarem *plantilla*, als objectes del domini, *instàncies*, i als camps de la plantilla, i per extensió de les instàncies, *atributs*.

Els atributs són les propietats o característiques que observem en les instàncies. Per poder treballar amb un ampli ventall de problemes, s'ha imposat que el sistema admeti els següents tipus d'atributs:

- Numèric
- Categòric
- Booleà

Els atributs numèrics poden prendre valors dins d'un rang determinat. El rang vindrà determinat per un valor màxim i mínim reals. Els valors dels atributs numèrics també seran reals.

Els atributs categòrics o qualitatius poden prendre valors dins d'un conjunt de categories preestablertes. Així, per exemple, podem tenir l'atribut *color* que pot valer *verd*, *vermell* o *blau*. Aquests valors no tenen cap semàntica ni ordre associats.

[4] *Mathematical Principles of Natural Philosophy*. Sir Isaac Newton, 1687.
[5] http://www.ghg.net/clips/CLIPS.html
[6] http://herzberg.ca.sandia.gov/

Els atributs booleans són un cas d'atribut categòric. Un booleà pot valer *cert* o *fals*.

Independentment del seu tipus, un atribut pot prendre el valor *nil*. En aquest cas, l'atribut no tindrà cap valor. Els valors *nil* s'utilitzen per representar la manca d'informació (*missing values*).

Entre els atributs d'una plantilla pot haver-ne un d'especial al que anomenarem atribut classificador. L'atribut classificador conté informació sobre la classe a la que pertany la instància. Ara, quan parlem de classe, no parlem pas de la plantilla en si, sinó d'un subconjunt d'instàncies d'una mateixa plantilla amb característiques comunes. Així doncs, les instàncies més semblants entre si tindran un atribut classificador igual. Per exemple, dins del domini dels vehicles de dues rodes, les motocicletes podrien tenir un atribut classificador comú que és la presència del motor. Aquest atribut classificador les diferenciaria, per exemple, de les bicicletes, que no en tenen. Cal notar que l'atribut classificador sempre serà categòric.

L'atribut classificador juga un paper important en un tipus d'aprenentatge que s'anomena aprenentatge supervisat. Els algorismes que l'implementen generen regles a partir de les relacions que observen entre els valors de l'atribut classificador i els de la resta d'atributs, que permeten determinar, un cop finalitzat l'aprenentatge, la classe a la que pertany una instància sense consultar-ne el seu atribut classificador (bé perquè no existeix, bé perquè es vol comprovar la precisió del resultat).

3.1.2 Els fitxers de dades

Tant la plantilla com les instàncies necessiten emmagatzemar-se en un suport físic per ser distribuïdes o perpetuar. Durant la fase de disseny s'ha considerat oportú utilitzar un format de dades llegible i fàcilment modificable per l'usuari. Aquest format ha de permetre definir plantilles i instàncies amb totes les característiques i limitacions definides prèviament:

- Les plantilles podran tenir tres tipus d'atributs: numèrics, categòrics i booleans.
- Els atributs numèrics estan limitats a un rang acotat per un valor màxim i un valor mínim, ambdós reals.
- Els valors categòrics només poden prendre per valor un conjunt de categories definides a priori.
- Els atributs poden no tenir valor.
- Els atributs han de tenir un nom que els identifiqui.

Per reduir al màxim el nombre de definicions de plantilla i per poder tenir convenientment emmagatzemades en diferents fitxers a diferents instàncies d'una mateixa plantilla (per exemple, les instàncies d'aprenentatge i les instàncies de validació), s'ha optat per emplaçar la definició de la plantilla i la de les instàncies en dos fitxer separats. Així doncs, tindrem un fitxer que definirà la plantilla i els seus atributs, i un altre que definirà les diferents instàncies (amb els seus valors corresponents) de la plantilla.

3.1.2.1 El fitxer de plantilla

El fitxer de plantilla defineix una plantilla amb els seus atributs. El fitxer està dividit en línies, i a cada línia hi ha la definició d'un atribut. El format general per cada tipus d'atribut es mostra a continuació:

- Atributs numèrics

  `nom:N:valor_min,valor_max[:valor_nil]`

  on *nom* és el nom de l'atribut, *valor_min* i *valor_max*, el valor mínim i màxim real, respectivament, que pot prendre l'atribut, i *valor_nil*, que és opcional, el valor de l'atribut que es considerarà com a valor *nil*. Així, per exemple, la línia:

  `nota:N:0,10:sense-nota`

  defineix un atribut anomenat *nota*, de tipus numèric, que pot prendre qualsevol valor real entre 0 i 10, i que té com a valor *nil* la cadena "sense-nota".

- Atributs categòrics

  `nom:Q:cat1,cat2...catN[:valor_nil]`

  on *nom* és el nom de l'atribut, *cat1, cat2 ... catN*, els diferents valors que pot prendre l'atribut, i *valor_nil*, que és opcional, el valor de l'atribut que es considerarà com a valor *nil*. Així, per exemple, la línia:

  `qualificacio:Q:AP,NT,EX,MH`

  defineix un atribut anomenat *qualificacio*, de tipus categòric, que pot valer "AP", "NT", "EX" o "MH", sense cap valor *nil*.

- Atributs booleans

  `nom:B[:valor_nil]`

  on *nom* és el nom de l'atribut, i *valor_nil*, que és opcional, el valor de l'atribut que es considerarà com a valor *nil*. Així, per exemple, la línia:

  `aprovat:B:NA`

  defineix un atribut anomenat *aprovat*, de tipus booleà, i que té com a valor *nil* la cadena "NA" (No Aplicable).

- Atribut classificador

  Tot i que no és pròpiament un tipus d'atribut, doncs l'atribut classificador és sempre categòric i només pot haver-n'hi un, presenta una construcció especial:

  `nom:CLASSE:cat1,cat2...catN[:valor_nil]`

  Com es pot observar es declara de la mateixa forma que un atribut categòric, excepte per la paraula reservada "CLASSE".

Cal remarcar que el valor *nil* dels atributs pot ser qualsevol cadena, excepte un número. L'excepció és pels atributs numèrics, doncs accepten valors *nil* tant numèrics com alfabètics.

La notació BNF del format del fitxer de plantilla es mostra a la Figura 5:

```
<fitxer> ::= <linia> {<linia>} TT_EOF
<linia> ::= [<atribut>] TT_EOL
<atribut> ::= <nom>:<tipus>[:<rang>][:<nil>]
<nom> ::= TT_WORD
<tipus> ::= TT_WORD
<rang> ::= <rang-enumerat> | <rang-numeric>
<rang-enumerat> ::= <valor-enum> [, <rang-enumerat>]
<valor-enum> ::= TT_WORD
<rang-numeric> ::= <valor-min>, <valor-max>
<valor-min> ::= TT_NUMBER
<valor-max> ::= TT_NUMBER
<nil> ::= <valor>
<valor> ::= TT_NUMBER | TT_WORD
```

Figura 5: Gramàtica BNF del fitxer de plantilla

A l'Apèndix C es pot veure un exemple d'un fitxer de plantilla.

3.1.2.2 El fitxer d'instàncies

El fitxer d'instàncies defineix una sèrie d'instàncies amb els seus atributs. El fitxer d'instàncies està dividit per línies i, a cada línia es defineix una instància. El format general és el següent:

```
valor1 valor2 ... valorN
```

Com es pot observar, el format és molt més simple que el del fitxer de plantilla. Tan sols hi ha els valors de cada atribut de la instància separats per espais en blanc o tabuladors. Els atributs segueixen el mateix ordre que en el que han estat definits en el fitxer de plantilla.

Cal remarcar que si es vol deixar un atribut com a indefinit, cal introduir el seu valor *nil*. El valor *nil* cal que estigui declarat convenientment al fitxer de plantilla.

La notació BNF del fitxer d'instàncies es mostra a la Figura 6:

```
<fitxer> ::= <linia> {<linia>} TT_EOF
<linia> ::= [<valors>] TT_EOL
<valors> ::= {TT_WORD | TT_NUMBER}
```

Figura 6: Gramàtica BNF del fitxer d'instàncies

A l'Apèndix C es pot veure un exemple d'un fitxer d'instàncies.

3.1.3 La implementació, usant orientació a objectes

Les dades, un cop llegides dels fitxers de dades, s'han d'estructurar perquè puguin ser llegides còmodament per l'aplicació. En aquest sistema això és especialment crític,

doncs són els alumnes els que hauran de manipular aquestes estructures de dades. Per tant, han de disposar d'una interfície que reuneixi les següents característiques:

- **Clara:** els noms de les classes i dels mètodes ha de permetre identificar fàcilment el seu propòsit o el tipus de dades que encapsulen.
- **Còmoda:** el codi generat per accedir a les dades ha de ser el mínim possible.
- **Segura:** les dades no poden ser modificades per l'alumne. De la mateixa manera, un error de codificació per part de l'alumne no es pot traduir en una caiguda del sistema.

Seguint aquestes premisses s'han identificat les següent classes i mètodes. Les relacions entre les classes es mostren a la Figura 7.

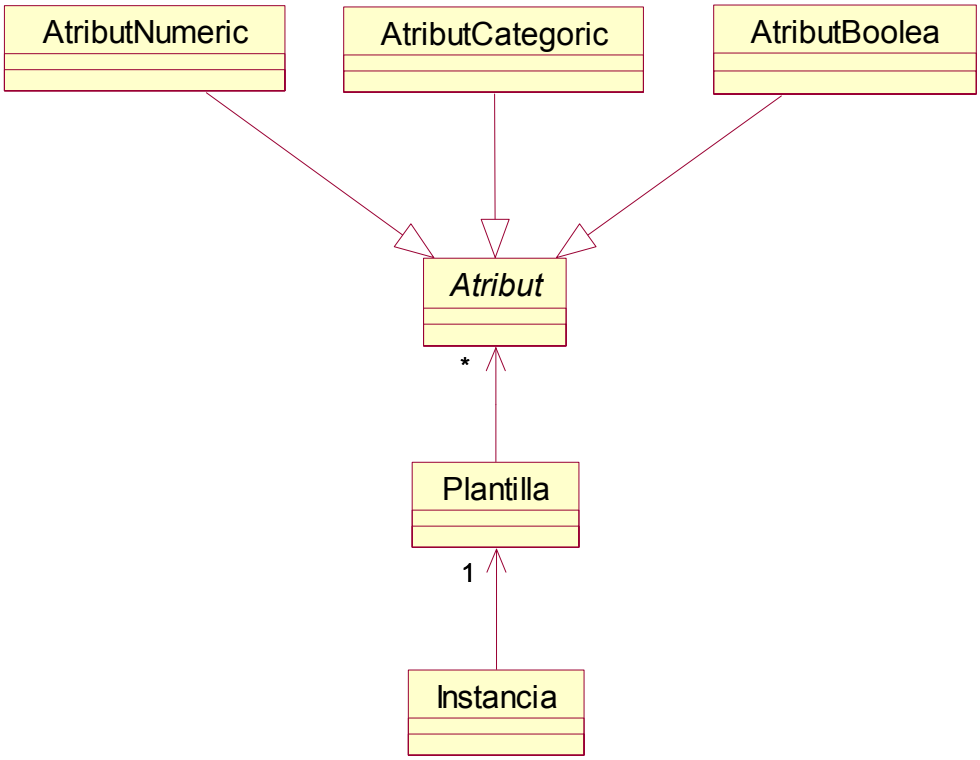
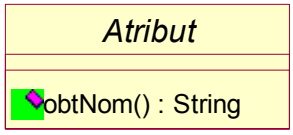


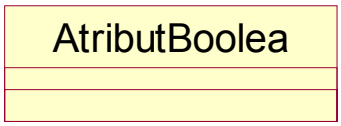
Figura 7: Relació entre les classes de dades

3.1.3.1 La classe *Atribut*



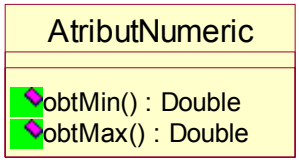
La classe *Atribut* és la classe mare de la que en deriven els tres tipus d'atributs definits pel sistema: els atributs numèrics, els categòrics i els booleans. Aquesta classe és abstracta i només implementa el mètode *obtNom()* que retorna el nom de l'atribut.

3.1.3.2 La classe *AtributBoolea*



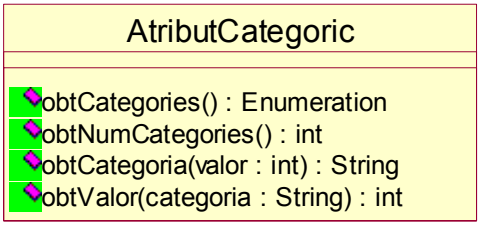
La classe *AtributBoolea* deriva directament de la classe *Atribut* sense implementar cap mètode nou. De fet, un atribut booleà només pot ser cert o fals.

3.1.3.3 La classe *AtributNumeric*



La classe *AtributNumeric* encapsula la definició d'un atribut numèric. Els atributs numèrics, com ja s'ha explicat al punt 3.1.1, només poden prendre valors dins d'un rang real prèviamnt definit. Els valors màxim i mínim d'aquest rang poden obtenir-se per mitjà de les funcions *obtMax()* i *obtMin()* respectivament.

3.1.3.4 La classe *AtributCategoric*



La classe *AtributCategoric* encapsula la definició d'un atribut categòric. Con ja s'ha explicat al punt 3.1.1, els *AtributsCategorics* es caracteritzen per poder prendre valors dins d'un conjunt de categories preestablertes. Aquestes categories es poden consultar mitjançant els mètodes *obtCategories()* i *obtNumCategories()*. El primer retorna una enumeració de cadenes de text amb el nom de les categories. El segon, el número de categories definides per l'atribut.

Per facilitar la tasca de l'alumne, s'han inclòs un parell de mètodes de conveniència per manegar les categories com si fossin enters. El mètode *obtValor()* retorna el valor numèric associat a la categoria passada com a paràmetre. El mètode *obtCategoria()* fa l'operació inversa: retorna la categoria equivalent al número passat com a paràmetre.

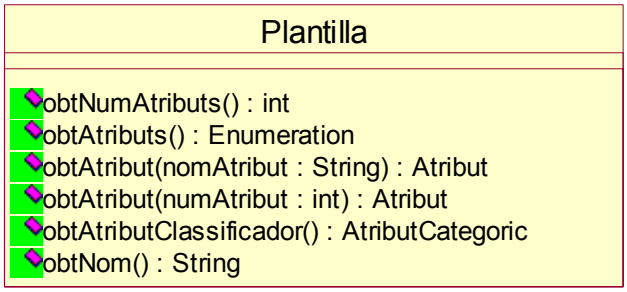
Cal imaginar-se el conjunt de categories com una seqüència de cadenes de text. La posició de cada cadena dins de la seqüència és el seu valor numèric equivalent. Per exemple, les categories de l'atribut:

```
qualificacio:Q:AP,NT,EX,MH
```

tenen els següents valors numèrics equivalents:

Categoria	Valor numèric
AP	0
NT	1
EX	2
MH	3

3.1.3.5 La classe *Plantilla*

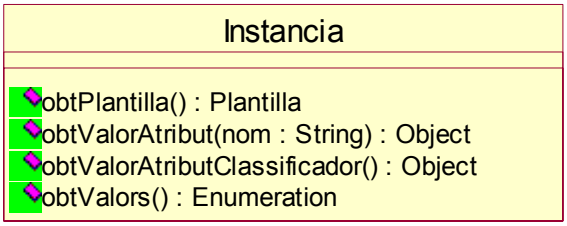


Com ja hem vist anteriorment, una plantilla es pot definir com una unitat que agrupa la declaració d'una sèrie d'atributs. En el disseny de la classe *Plantilla* s'ha mantingut aquesta filosofia, proporcionant-se una sèrie de mètodes que permeten accedir als objectes *Atribut* que formen part de la *plantilla*. Així doncs, la funció *obtAtributs()* retorna una enumeració d'objectes de tipus *Atribut* corresponents a tots els atributs que integren la plantilla. També es pot obtenir un atribut en particular per mitjà del mètode *obtAtribut(String)* o *obtAtribut(int)*. Aquest últim accedeix als atributs segons la seva posició dins la seqüència de declaració, d'una forma anàloga a com la classe *AtributCategoric* accedeix a les seves categories (veure 3.1.3.4).

Com ja s'ha comentat anteriorment, les plantilles poden tenir un atribut classificador. L'atribut classificador genera sub-classificacions dins de la plantilla que classifiquen les instàncies segons els valors dels seus atributs. L'atribut classificador es pot obtenir a través del mètode *obtAtributClassificador()*. Cal notar que l'atribut classificador és sempre de tipus categòric.

Finalment, els mètodes *obtNumAtributs()* i *obtNom()* retornen el número d'atributs que defineix la plantilla i el nom d'aquesta, respectivament. Generalment, el nom de la plantilla serà el nom del fitxer on està guardada.

3.1.3.6 La classe *Instancia*



Anomenarem *instància* d'una *plantilla* a la representació d'un objecte concret del domini. Per tant, consistirà en un conjunt de valors, un per atribut, que satisfan les restriccions imposades per la *plantilla*. Per tant, una instància està fortament lligada a

una plantilla a la que implementa o satisfà. Això es pot veure clarament en el disseny de la classe, sobretot per la presència del mètode *obtPlantilla()* que retorna la *plantilla* associada a la instància. Per altra banda, els mètodes *obtValorAtribut()* i *obtValorAtributClassificador()* retornen el valor de l'atribut passat com a paràmetre, i el valor de l'atribut classificador, respectivament. Si el que es desitja és obtenir tots els valors de la instància es pot aconseguir per mitjà d'una crida a la funció *obtValors()*. Aquest mètode retorna una enumeració d'objectes corresponents als valors dels *atributs*, ordenats segons l'ordre de definició dels atributs dins de la plantilla.

Cal remarcar que, típicament, la classe dels valors retornats per les funcions *obtValorAtribut()*, *obtValorAtributClassificador()* i *obtValors()*, dependrà de l'atribut que es vulgui llegir. A la Taula 1 es mostra la correspondència entre el tipus d'atribut i la classe de l'objecte retornat per aquestes funcions.

Tipus de l'atribut	Classe de l'objecte
Booleà	java.core.Boolean
Numèric	java.core.Integer
Categòric	java.core.String

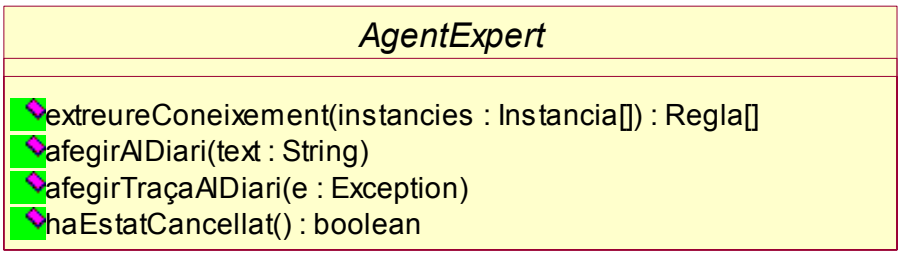
Taula 1: Correspondència entre els tipus d'atributs i les classes de Java

3.2 La classe *AgentExpert*

L'objectiu principal del sistema és facilitar als alumnes la creació d'agents experts que, mitjançant tècniques d'adquisició del coneixement, creïn una base de regles sobre la que se'n puguin fer consultes més tard. Com que la missió dels alumnes és únicament la implementació de l'algorisme de mineria de dades, cal proporcionar un esquelet d'agent expert que acompleixi les següent accions:

- Rebre plantilles i instàncies sobre les que s'aplicarà l'algorisme de mineria de dades. Aquestes són proporcionades pels agents tutors del sistema.
- Executar l'algorisme de mineria de dades implementat per l'alumne.
- Recol·lectar les regles que aquest generi i introduir-les a la base de regles.
- Respondre a les preguntes que li facin els agents tutors del sistema utilitzant les regles que ha après. Això implica engegar un motor d'inferència.

Com que els agents es materialitzen en classes, la solució adoptada ha estat crear la classe *AgentExpert* genèrica, abstracta, de la qual en derivaran els agents experts funcionals:



Els alumnes tan sols hauran de crear una nova classe que derivi de la classe *AgentExpert* i que implementi el mètode abstracte *extreureConeixement()*. Aquest mètode es cridat per l'agent quan aquest ha rebut un conjunt d'instàncies d'aprenentatge i en vol obtenir un coneixement en forma de regles. Com es pot observar, són precisament aquestes regles el valor de retorn de la funció. Les classes *Instancia* i *Regla* estan descrites als apartats 3.1.3.6 i 3.3.1 respectivament.

Les funcions *afegirAlDiari()* i *afegirTraçaAlDiari()* són un parell de mètodes de conveniència que es proporcionen als alumes perquè puguin escriure informació de sortida que més tard podran consultar. Aquesta informació es mostrarà en una consola de text en temps real.

La funció *haEstatCancellat()* permet consultar si l'usuari ha sol·licitat la cancel·lació del procés d'extracció del coneixement. Si la funció retorna cert, cal interrompre l'algorisme de mineria de dades i retornar immediatament.

A la Figura 8 es mostra l'esquelet típic d'un agent expert com el que implementaran els alumnes. Cal destacar la senzillesa del codi a generar així com les poques possibilitats d'error que, a priori, suggereix aquesta solució. Així, els alumnes es poden centrar exclusivament en la implementació de l'algorisme de mineria de dades sense haver de preocupar-se pels detalls d'implementació dels agents i dels mecanismes de comunicació entre ells, perquè no és l'objectiu de l'assignatura.

```
import net.urv.etse.ninots.agents.comu.*;
import net.urv.etse.ninots.agents.expert.regles.*;

public class ExempleAgentExpert extends AgentExpert {

    public Regla[] extreureConeixement(Instancia[] instancies) {
         // aquí hi va el codi de l'alumne
    }

}
```

Figura 8: Esquelet típic d'un agent expert

3.3 Generació de regles

L'algorisme de mineria de dades implementat pels alumnes ha de retornar una o més regles que sintetitzin el coneixement extret de l'estudi de les instàncies que se li han proporcionat. Els alumnes, doncs, han de tenir a l'abast una sèrie d'eines per generar aquestes regles de forma senzilla.

És en aquest punt on entra en joc la classe *Regla* i un conjunt reduït de classes que permeten construir l'expressió que la regeix. Una regla està composta per un conjunt de premisses i una conclusió. Les premisses defineixen les condicions necessàries per tal que s'activi la regla. La conclusió és el resultat que es deduir si les condicions són certes. En el nostre sistema només utilitzarem regles conjuntives. Això significa que les condicions de les premisses només es poden combinar utilitzant l'operador AND lògic. No s'ofereixen l'OR ni el NOT lògics. La raó d'aquesta decisió és que les regles conjuntives són la sortida habitual dels algorismes d'aprenentatge automàtic.

A la Figura 9 es mostra un diagrama de classes amb les classes que intervenen en la generació de regles.

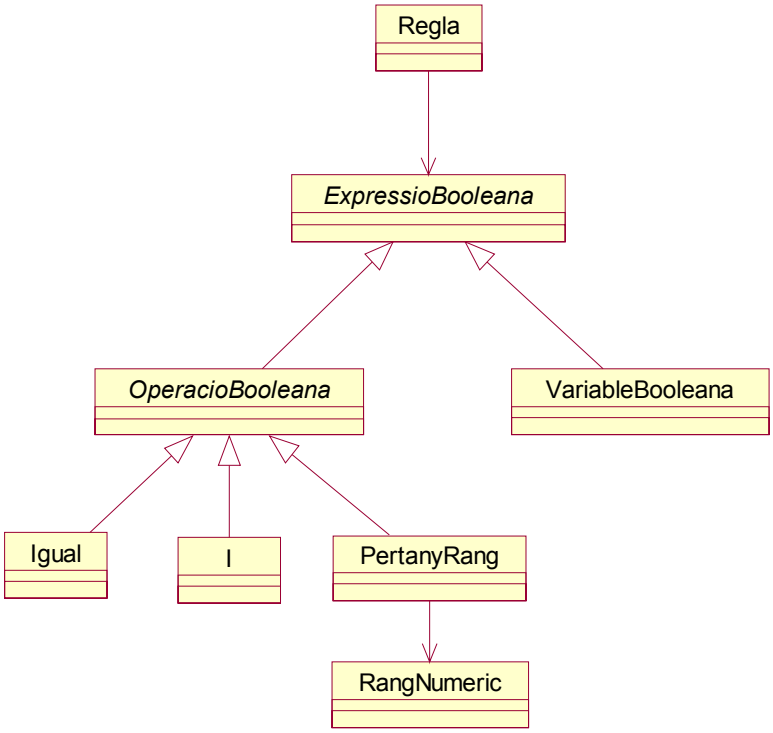
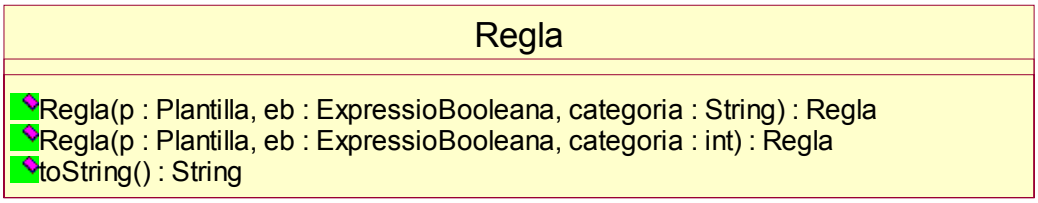


Figura 9: Classes relacionades amb la generació de regles

3.3.1 La classe *Regla*



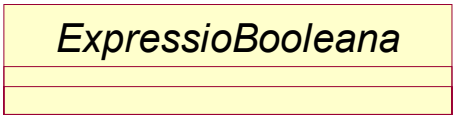
La classe *Regla* encapsula la definició d'una regla. Els seus mètodes constructors tenen tres paràmetres d'entrada: la plantilla a la que serà aplicable la regla, és a dir, la plantilla dels objectes que es pretén processar amb aquesta regla, un objecte de la classe *ExpressioBooleana* que defineix la condició les premisses de la regla (aquesta classe es descriu amb detall al punt 3.3.2), i la conclusió de la regla.

Cal remarcar que si la plantilla associada a la regla té l'atribut classificador definit, el valor passat com a categoria ha de ser vàlid per aquest atribut. Si l'atribut classificador està present, només es pot classificar una instància en alguna de les categories que aquest defineix.

Contràriament, si l'atribut classificador no està definit, es pot especificar qualsevol valor pel paràmetre categoria. Simplement, les categories s'aniran creant a mesura que es necessitin.

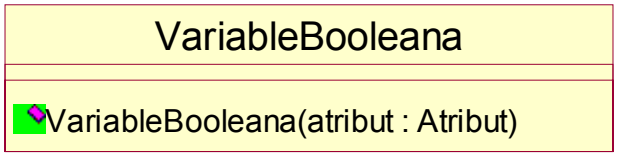
Addicionalment, la classe implementa el mètode *toString()* que permet obtenir una representació textual de la regla.

3.3.2 La classe *ExpressioBooleana*



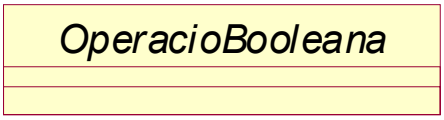
La classe *ExpressioBooleana* és abstracta i no implementa cap mètode. La seva missió és ser l'arrel de la jerarquia que permet definir una expressió que, en avaluar-la, genera un resultat booleà.

3.3.3 La classe *VariableBooleana*



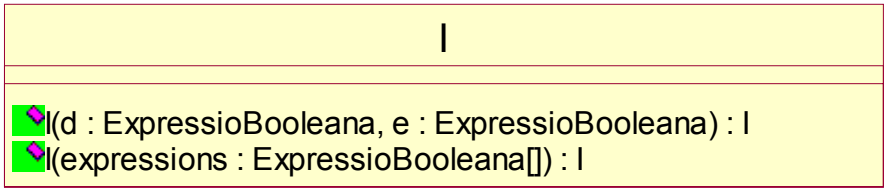
La classe *VariableBooleana* representa una expressió que avalua un atribut booleà. El seu constructor només accepta un paràmetre: l'atribut a avaluar. No cal dir que aquest ha de ser una instància de la classe *AtributBoolea* (veure 3.1.3.2). Quan s'avaluï, l'expressió retornarà el valor del l'atribut.

3.3.4 La classe *OperacióBooleana*



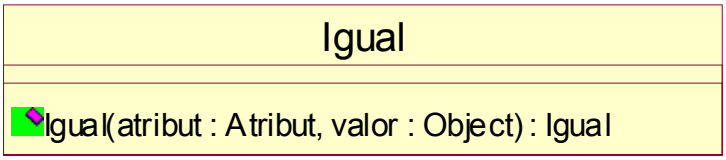
Es tracta altre cop d'una altra classe abstracta, mare de totes les operacions (o operadors) que generen un resultat booleà.

3.3.5 La classe *I*



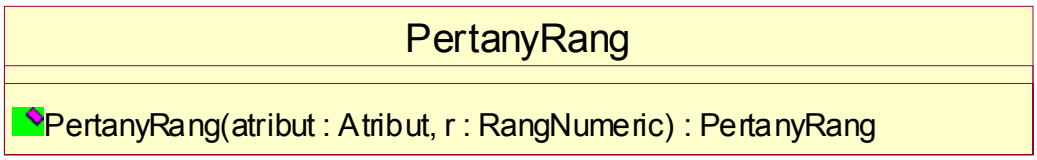
La classe *I* és la representació de l'operador AND N-ari. Té dos constructors: un per generar un AND binari, amb dos paràmetres de la classe *ExpressioBooleana* que representen les expressions esquerra i dreta a avaluar, i el que genera un AND N-ari, que rep com a paràmetre un vector d'objectes de la classe *ExpressioBooleana* que representen les diferents expressions a avaluar. És a dir, el primer constructor genera expressions del tipus *e AND d*, on *e* i *d* són expressions booleanes, i el segon, expressions del tipus *e1 AND e2 AND e3 ... AND eN*, on *e1, e2, e3 ... eN*, són expressions booleanes.

3.3.6 La classe *Igual*



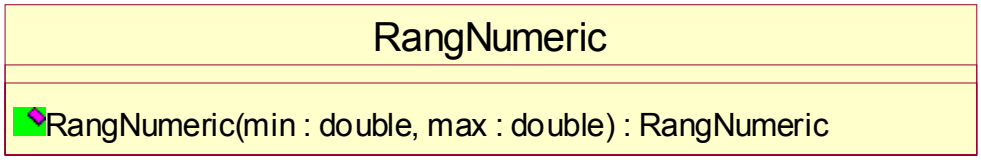
La classe *Igual* representa l'operador d'igualtat. Aquest operador retorna cert quan l'atribut passat com a paràmetre és igual al valor especificat, i fals altrament. Al constructor se li ha d'especificar l'atribut a avaluar i el valor amb el que s'ha d'igualar. No cal dir que la classe del valor ha de ser congruent amb el tipus de l'atribut. No es pot pretendre pas comparar un atribut numèric amb una categoria.

3.3.7 La classe *PertanyRang*



Aquesta classe representa a l'operador de pertinença ($\in$). Donat un rang numèric $[r_1..r_n)$ real, aquest operador retornarà cert quan el valor de l'atribut $\in$ $[r_1..r_n)$, o el que és el mateix, que $r_1 \leq$ valor $< r_n$. Altrament, retornarà fals.

El constructor, doncs, requereix un atribut, per descomptat de la classe *AtributNumeric* (veure 3.1.3.3), i un objecte de la classe *RangNumèric* que representa el rang en el que s'avaluarà el valor de l'atribut.



El constructor de la classe *RangNumeric* es trivial: només requereix que se li proporcionin el límits inferior i superior del rang que es vol representar.

3.3.8 Exemple de construcció d'una regla

Suposem un domini *Automòbils* amb tres atributs: *potencia*, numèric, *numero-de-portes*, també numèric, *canvi*, categòric amb dos possibles valors: *manual* i *automatic*, i *longitud*, numèric.

Suposem un domini *Automòbils* amb tres atributs: *posicio-del-motor*, categòric amb tres possibles valors: *davantera*, *central* i *darrere*, *numero-de-seients*, numèric, i *es-descapotable*, booleà.

Suposem que volem construir una regla que reciti "si un cotxe té un motor central, un o dos seients, i és descapotable és un *roadster*". El codi Java per construir la regla és el següent:

```
// Suposem que tenim les instàncies a avaluar en un vector
// anomenat "instancies".

Plantilla plantilla;
AtributCategoric posicioDelMotor;
AtributNumeric numeroDeSeients;
AtributBoolea esDescapotable;
Regla regla = null;

try {
      // llegeix la plantilla de les instàncies
      plantilla = instancies[0].obtPlantilla();

      // obté els atributs
      posicioDelMotor = (AtributCategoric)
            plantilla.obtAtribut("posicio-del-motor");
      numeroDeSeients = (AtributNumeric)
            plantilla.obtAtribut("numero-de-seients");
      esDescapotable = (AtributBoolea)
            plantilla.obtAtribut("es-descapotable");


      // crea la regla
      regla = new Regla(
            plantilla,
            new I(
                  new ExpressioBooleana[] {
                        new Igual(posicioDelMotor, "central"),
                        new PertanyRang(numeroDeSeients,
                              new RangNumeric(1, 3)),
                        new VariableBooleana(esDescapotable)
                  }),
            "roadster");
} catch (Exception e) {
}
```

3.4 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 sistemes multi-agent 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.

- Compleix amb les especificacions de la FIPA.

JADE és un conjunt de llibreries de lliure distribució escrites en Java que ens proporciona un entorn de desenvolupament de sistemes multi-agent. 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 3.2 del 2004/07/26, l'última que es trobava disponible en el moment de començar aquest treball.

A l'Apèndix A es pot trobar una introducció al JADE.

3.5 Jess

Com que la tasca d'implementació d'un sistema expert és complexa, s'ha utilitzat una llibreria externa de desenvolupament de sistemes experts anomenada Jess.

Ens hem decantat pel Jess pels següents motius:

- Està escrit en Java.

- El seu llenguatge permet interactuar amb els objectes de Java en temps d'execució.

- Proporciona un rendiment molt semblant a la d'altres llibreries compilades en codi natiu.

- Es distribueix amb el codi font.

- El seva llicència és gratuïta per finalitats acadèmiques i no comercials.

Jess és una llibreria de programació de sistemes experts escrita en Java. La llibreria en si és un intèrpret d'un llenguatge (el llenguatge del Jess) que serveix per construir sistemes experts basats en regles.

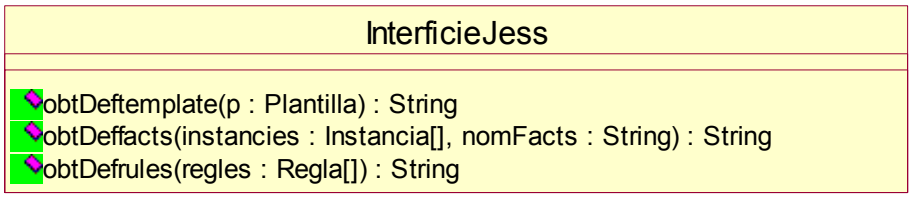
La versió utilitzada ha estat la 5.1 (24/4/2000).

A l'Apèndix B es pot trobar una introducció al Jess.

3.6 El pont Ninots-Jess

Les regles generades pels agents experts necessiten ser traduïdes al llenguatge del Jess perquè es puguin afegir a la base de regles. D'aquesta tasca se n'encarrega la classe *InterficieJess*. Aquesta classe fa d'interfície entre Ninots i Jess, proporcionant sèrie de

mètodes que amaguen les característiques del llenguatge del Jess a la resta de l'aplicació.



La classe proporciona tres mètodes:

- ***obtDeftemplate():*** retorna la construcció *deftemplate* del Jess equivalent a la plantilla passada com a paràmetre.
- ***obtDeffacts():*** retorna la construcció *deffacts* del Jess equivalent al vector d'instàncies passat com a paràmetre.
- ***obtDefrules():*** retorna les construccions *defrule* del Jess equivalents al vector de regles passat com a paràmetre.

De les construccions *deftemplate*, *deffacts* i *defrule* se'n parla més extensament a l'Apèndix B.

3.6.1 Traducció de plantilles

Una plantilla de Ninots es tradueix en una construcció *deftemplate* del Jess. Les construccions *deftemplate* generades per la classe *InterficieJess* tenen la següent estructura:

```
(deftemplate T
   (slot id (type INTEGER))
   (slot <nom1> (type <tipus1>)[(default <default1>)])
   ...
   (slot <nomN> (type <tipusN>)[(default <defaultN>)])
)
```

Aquests *deftemplates* tenen sempre un *slot* anomenat *id* de tipus *INTEGER* que permet numerar les instàncies, i un *slot* per cada atribut de la plantilla. Depenent del tipus d'atribut (numèric, categòric o booleà), el codi de cada *slot* variarà. A la Taula 2 es mostra el codi que genera cada tipus d'atribut. Cal remarcar que al *deftemplate*, que sempre és un de sol, se l'anomena *T*. Aquest nom és important perquè més endavant servirà per definir instàncies i regles.

Tipus d'atribut	Codi Jess
Numèric	`(slot (type FLOAT)(default NaN))`
Categòric	`(slot (type ATOM))`
Booleà	`(slot (type ATOM))`

Taula 2: Codi Jess generat segons el tipus d'atribut de la plantilla

3.6.2 Traducció d'instàncies

Un vector d'instàncies de Ninots es tradueix en una construcció *deffacts* del Jess. Les construccions *deffacts* generades per la classe *InterficieJess* tenen la següent estructura:

```
(deffacts nom-facts
      (T (id 1) (<atribut1-1> <valor1-1>)
                ...
                (<atribut1-N> <valor1-N>))
      ...
      (T (id M) (<atributM-1> <valorM-1>)
                ...
                (<atributM-N> <valorM-N>))

)
```

on *M* és l'índex de la instància dins del vector i <*atributX-Y*> i <*valorX-Y*>, el nom i el valor de l'atribut *Y*-èsim de la instància que es troba a la posició *X* del vector. Com es pot observar totes les instàncies es defineixen seguint el *deftemplate T*. Si algun atribut és *nil*, no s'afegeix a la llista d'atributs.

3.6.3 Traducció de regles

Una regla de Ninots es tradueix en una construcció *defrule* del Jess. Les construccions *defrule* generades per la classe *InterficieJess* tenen la següent estructura:

```
(defrule r<numero-de-regla>
       ?i<-(T (id ?id)(<condicio1>)... (<condicioN>))
       =>
       (classificar ?id "<categoria>")
       (retract ?i)
)
```

El funcionament d'una regla és senzill: quan hi ha una instància de tipus *T* que compleix la sèrie de restriccions imposades sobre els seus atributs <*condicio1*>... <*condicioN*>, llavors s'executa la funció *classificar*, que relaciona la instància (que té com a identificador *?id*) amb la categoria <*categoria*>, i, seguidament, s'elimina la instància perquè no es torni a activar la regla.

El codi que imposa restriccions sobre els atributs varia depenent del tipus d'atribut i dels operadors que s'hagin utilitzat a l'hora de definir la regla. A la Taula 3 es mostren els diferents operadors i el codi que generen.

Operador	Paràmetres	Codi Jess
VariableBooleana	<*nom*>, nom de la variable	`(<nom> true)`
Igual	<*nom*>, nom de la variable, i <*valor*>, valor amb el que es vol comparar	`(<nom> <valor>)`
PertanyRang	<*nom*>, nom de la variable, <*min*> valor mínim del rang, i <*max*>, valor màxim del rang	`(<nom> ?n&:(>= ?n <min>)&:(< ?n <max>))`

Taula 3: Codi Jess generat segons l'*ExpressioBooleana*

3.7 El model didàctic del SMA

En la seva forma més bàsica, un SMA està format per un agent *AMS* i un agent *DF* que ofereixen serveis a un conjunt d'agents que varia segons la funcionalitat del sistema que estiguem estudiant. Aquesta visió ideal d'un SMA pren una sèrie de matisos segons l'eina que estiguem utilitzant per construir-lo; Així, per exemple, el JADE treballa amb el conceptes de contenidor, que és la unitat mínima que pot contenir un agent, i de plataforma, que és un conjunt d'un o més contenidors i que sol ser una instància d'execució del motor del JADE.

Si bé conèixer aquestes peculiaritats és extremadament útil a l'hora d'abordar el disseny d'un SMA basant-se en una determinada llibreria, no és necessari quan es pretén introduir a l'alumne en l'estructura i la dinàmica d'un sistema d'aquesta mena. Així, per tant, el sistema que es mostrarà a l'alumne serà pla, sense plataformes ni contenidors, com un "lloc" on es troben tots els agents del sistema – inclosos l'*AMS* i el *DF* – i on interactuen entre ells. Fent un símil amb el funcionament d'un sistema operatiu, diríem que el "lloc" on es troben els agents és l'entorn d'execució de les aplicacions, i els agents, les aplicacions en si.

D'aquesta manera aconseguim simplificar la visió d'un SMA, aproximant-lo al model teòric estudiat a classe i deixant de banda les característiques pròpies de cada implementació, sense descuidar en cap moment les peculiaritats que diferencien aquest paradigma de la programació de la resta.

4 Arquitectura del SMA

En aquest capítol s'exposarà l'arquitectura completa del SMA subjacent a Ninots. S'enumeraran els agents que el conformen, el seu rol dins del sistema, i els serveis que ofereixen a la resta d'agents. Per acabar es descriurà concisament l'ontologia que utilitzen per comunicar-se entre ells.

4.1 Els agents i els seus rols

A banda dels agents intrínsecs (*DF* i *AMS*) a qualsevol SMA, Ninots està compost per quatre tipus d'agents ben diferenciats: els agents gestors, els agents intèrprets, els agents experts i els agents tutors. Cada tipus d'agent compleix un rol específic dins del sistema que associarem amb els requisits del sistema detectats prèviament.

4.1.1 Els agents intèrprets

Els agents intèrprets s'encarreguen de llegir les dades dels fitxers de dades (bàsicament, definicions de plantilles i conjunts d'instàncies) i de proveir-les a la resta dels agents en un format homogeni, independentment del format en que aquestes estiguin guardades als fitxers. En aquest sentit, aïlla a la resta d'agents del sistema de conèixer els diferents formats en que s'emmagatzemen les dades.

Tot i que només s'ha implementat un sol tipus d'agent intèrpret que reconeix el format de dades explicat a l'apartat 3.1.2, se'n podrien preparar d'altres que reconeguessin altres formats de dades.

4.1.2 Els agents experts

Els agents experts encapsulen els algorismes de mineria de dades i les bases de regles, assolint, per mitjà d'aquests, un coneixement sobre un domini representat per una determinada plantilla.

Com ja s'ha dit, aquest coneixement s'emmagatzema en forma de regles, i pot ser utilitzat pel sistema expert que aquests agents porten incorporat. Inicialment, la seva base regles està buida. A petició d'algun agent tutor s'inicia el procés d'extracció del coneixement, que és basa en el mètode que han d'implementar els estudiants. Aquest procés genera una sèrie de regles que s'emmagatzemen a la base de regles. Els agents experts també pot utilitzar les regles que tenen emmagatzemades per fer el pas invers: trobar la classe corresponent dels objectes que li enviï un agent tutor.

4.1.3 Els agents tutors

La missió dels agents tutors es dirigir els processos d'extracció del coneixement i de validació, i mostrar-ne els resultats. Els agents tutors, doncs, dirigeixen l'activitat del sistema: llegeixen les dades a través dels agents intèrprets, les combinen quan procedeixen de fonts diferents i les passen als agents experts.

Els agents tutors tenen una potent interfície que permet a l'usuari dirigir processos d'aprenentatge i de validació, i estudiar les regles generades per l'algorisme de mineria

de dades dels agents experts. Val a dir que els agents tutors no actuen proactivament. Tan sols són fan de pont entre l'usuari i el sistema.

4.1.4 Els agents gestors

Els agents gestors tenen una doble missió: per una banda, controlar l'entrada i la sortida d'agents (sobretot experts) del sistema, i, per l'altra, mostrar l'estructura del mateix que, per analogia, és la de qualsevol SMA que segueix l'estàndard definit per la FIPA. L'agent gestor ha estat dissenyat des del punt de vista pedagògic per mostrar el SMA des d'un punt de vista ideal i teòric, sense entrar en els detalls específics de la implementació (veure 3.7).

4.2 Els serveis que ofereixen els agents

Com hem vist a l'apartat anterior, cada agent té unes habilitats pròpies que el caracteritzen. Cada agent és una peça del trencaclosques que és el sistema, sense la qual aquest no es podria completar (no funcionaria).

Al conjunt (o un subconjunt) d'habilitats d'un agent, se l'anomena servei. Així doncs, tenim una sèrie d'agents que presten serveis a la resta d'agents o a un subconjunts d'aquests. Si fem un símil amb el món real, tenim per exemple un mecànic (l'agent) que ofereix els serveis de planxa i pintura, i electricitat als seus conciutadans (la resta dels agents que, alhora, poden oferir altres serveis).

Quan un agent vol fer publicitat sobre els serveis que ofereix, per tal que la resta dels agents el puguin localitzar, es registra a l'agent *DF*. Com hem vist al capítol 2, l'agent *DF* és una mena de servei de pàgines grogues. Quan un agent cerca un servei, el consulta al *DF* i aquest li retorna una llista dels agents que l'ofereixen.

Cal remarcar que tots els agents han de parlar un mateix idioma, i referir-se a un mateix servei amb un mateix nom. És evident que cal que existeixi un consens entre els agents a l'hora de referir-se als serveis, car és vital perquè es puguin comunicar entre ells.

En el cas de Ninots, els noms dels serveis que ofereix cada agent es mostren a la Taula 4. Cal adonar-se que cada agent ofereix un únic servei que engloba i descriu el conjunt d'habilitats que té.

Tipus de l'agent	Nom del servei
Intèrpret	interpret
Expert	expert
Tutor	tutor
Gestor	gestor

Taula 4: Els noms dels serveis

Addicionalment, els serveis es poden englobar dins d'un tipus determinat. Per exemple, tornant al símil del mecànic, el servei de planxa i pintura es podria incloure dins del tipus mecànica de l'automòbil. En aquest conjunt s'hi podrien englobar també els

serveis de restauració de cotxes antics, venda de recanvis, etc. En el cas de Ninots tots els serveis que ofereixen els agents són del tipus *net.urv.etse.ninots*[7].

4.3 Interaccions entre els agents

Com s'ha pogut veure fins ara, els agents interactuen entre ells per mitjà d'una sèrie d'actes comunicatius. Havent determinat la missió de cadascun dels agents (veure 4.1), podem determinar que es donen tres actes comunicatius fonamentals al sistema:

- **Lectura de dades:** Els agents tutors sol·liciten les dades d'un fitxer als agents intèrprets, que són els que en coneixen el format.
- **Aprenentatge o extracció del coneixement:** Els agents experts obtenen coneixement a través de conjunts d'instàncies que els hi proporcionen els agents tutors.
- **Validació del coneixement:** Els agents experts classifiquen una sèrie d'instàncies que els hi proporcionen els agents tutors.

Anem-los a veure en detall. A la Figura 10 es mostren els actes comunicatius que succeeixen durant els processos d'aprenentatge i de classificació.

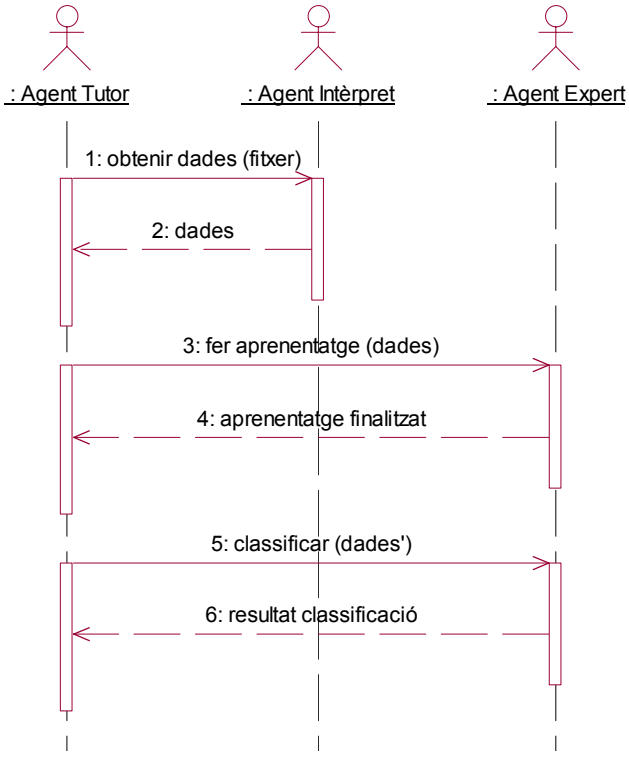


Figura 10: Els actes comunicatius dels agents

[7] Si bé el nom del tipus d'un servei pot ser completament arbitrari, cal trencar amb la tendència d'assignar un nom poc identificatiu al tipus o, pitjor encara, de no assignar-lo. Aquest greu error neix, segons el meu parer, de la tendència subconscient de veure el SMA com un bloc monolític, com una aplicació. Cal pensar que els agents són entitats independents que, individualment o col·lectivament, es poden moure a un altre SMA més gran, amb molta altres agents que també ofereixen serveis. En aquest escenari, un nom de tipus de servei massa comú i fàcilment repetible, a falta d'un consens, pot generar confusió.

En un primer pas (1), i responent a les demandes de l'usuari, l'agent tutor sol·licita un conjunt de dades a l'agent intèrpret. Aquestes dades són una plantilla i un conjunt d'instàncies. L'agent intèrpret obre els fitxers sol·licitats, interpreta les dades, i les passa al tutor (2). En aquest moment el tutor pot mesclar diferents fonts de dades a voluntat de l'usuari. Seleccionades les dades que formaran part de l'aprenentatge, es passen a un o més agents experts (3). Aquests les analitzen mitjançant l'algorisme d'extracció del coneixement implementat pels alumnes, i generen unes regles que representen el coneixement que n'han pogut extreure. Aquest coneixement, com ja s'ha dit anteriorment, s'emmagatzema a una base de regles. Completat aquest procés s'envia un missatge de confirmació a l'agent tutor que ha sol·licitat l'aprenentatge (4). En aquest punt, els agents expert tenen prou coneixement com per classificar noves instàncies, diferents a les utilitzades durant el procés d'extracció del coneixement. L'agent tutor, novament responent a les demandes de l'usuari, aplega un conjunt d'instàncies i les passa als agents experts, aquest cop, però, sol·licitant que les classifiquin (5). Els agents experts responen a la sol·licitud amb una classificació de les instàncies que se li han proporcionat (6). Aquesta classificació la fan basant-se en el coneixement que tenen emmagatzemat en forma de regles i que han adquirit durant el procés d'aprenentatge.

No cal dir que l'agent tutor pot retenir plantilles i conjunts d'instàncies per passar-les, posteriorment, als agents experts. Per tant, no sempre que es vulgui sol·licitar un aprenentatge o una classificació cal llegir les dades per mitjà d'un agent intèrpret. Cal no confondre's però: no cal que les llegeixi a cada aprenentatge o classificació però sí que cal que, en algun moment, les hagi sol·licitades a l'agent intèrpret. Simplement restaran emmagatzemades en memòria fins que l'usuari decideixi utilitzar-les. Cal recordar que l'agent tutor no pot interpretar els fitxers de dades sense ajuda d'un agent intèrpret.

A banda de la comunicació entre agents experts i tutors, destinada a treballar les habilitats de generació i consulta del coneixement del sistema, hi ha un altre acte comunicatiu que mereixen ser explicat amb detall. Es tracta d'una sol·licitud que els agents gestors poden enviar a qualsevol agent: la de "mostrar la GUI". Per defecte, tots els agents del sistema, excepte els experts, no mostren la seva GUI. Quan un agent rep la sol·licitud "mostrar la GUI", mostra la seva interfície d'usuari. A la Figura 11 es mostra un exemple d'aquest acte comunicatiu, amb destinatari un agent tutor.

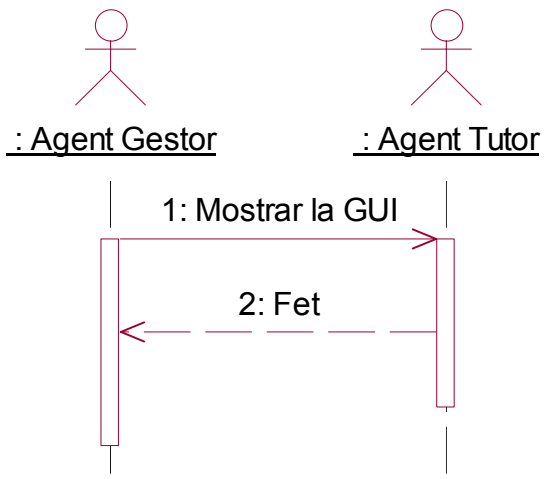


Figura 11: La sol·licitud de "mostrar la GUI"

4.4 L'ontologia de Ninots

Perquè es pugui establir un diàleg entre dos agents, cal que aquests tinguin un llenguatge i un vocabulari comuns. Si bé el llenguatge que utilitzen els agents ja ha estat definit per la FIPA (veure 2.4.2.1), no passa el mateix amb l'ontologia, que s'ha hagut de dissenyar tenint molt presents els actes comunicatius que es donen dins del sistema.

L'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 es detallarà el conjunt d'objectes lingüístics que utilitzen els agents del SMA.

4.4.1 Els conceptes atribut, plantilla i instància

Els conceptes són mots sobre els que construïm els predicats i les accions. Com succeeix en els llenguatges naturals, els conceptes fan referència a ens del món dels agents.

Els conceptes estan formats per un conjunt de camps de tipus primitius, o d'altres conceptes. Els camps poden ser obligatoris o optatius.

- ***atribut:*** Fa referència a un atribut d'una plantilla. Conté la característica comuna a tots els atributs: el nom. Cal no confondre'l amb el valor d'un atribut d'una instància. El concepte atribut és genèric i fa referència a tots els tipus d'atributs en general: categòrics, numèrics i booleans.

Camp	Tipus	Obligatorietat	Descripció
:nom	String	Obligatori	El nom de l'atribut.

Taula 5: El concepte atribut

- ***atribut-boolea:*** Un atribut booleà. Com ja hem vist anteriorment, un atribut booleà no té cap característica addicional a banda del nom.

- ***atribut-numeric:*** Un atribut numèric. Els atributs numèrics tenen un valor màxim i un valor mínim.

Camp	Tipus	Obligatorietat	Descripció
:min	Float	Obligatori	El valor mínim que pot prendre l'atribut.
:max	Float	Obligatori	El valor màxim que pot prendre l'atribut.

Taula 6: El concepte *atribut-numeric*

- ***atribut-categoric:*** Un atribut categòric. Els atributs categòrics tenen definides una sèrie de categories que limiten els valors que poden prendre.

Camp	Tipus	Obligatorietat	Descripció
:categories	String[]	Obligatori	Les categories que pot prendre per valor l'atribut.

Taula 7: El concepte *atribut-categoric*

- ***plantilla:*** Fa referència a una plantilla. Com ja hem vist anteriorment, una plantilla és un conjunt d'atributs, on pot haver-n'hi un que sigui l'atribut classificador.

Camp	Tipus	Obligatorietat	Descripció
:atribut-classificador	atribut-categoric	Optatiu	L'atribut classificador. Cal remarcar que es tracta d'un concepte del tipus 'atribut-categoric'.
:atributs	atribut[]	Obligatori	La resta d'atributs que formen part de la plantilla.

Taula 8: El concepte *plantilla*

- ***intancia:*** Fa referència a una instància d'una plantilla. Com ja s'ha explicat anteriorment, una instància és un conjunt de valors emparellats amb els atributs d'una plantilla.

Camp	Tipus	Obligatorietat	Descripció
:valors-atributs	valor-atribut[]	Obligatori	Un conjunt de parells valor-atribut.

Taula 9: El concepte *instancia*

El terme *valor-atribut*, representa un parella (atribut, valor):

Camp	Tipus	Obligatorietat	Descripció
:atribut	String	Obligatori	El nom de l'atribut.
:valor	String, Double o Boolean[8]	Obligatori	El valor de l'atribut.

Taula 10: El concepte *valor-atribut*

- ***dades:*** Engloba una plantilla i un conjunt d'instàncies.

Camp	Tipus	Obligatorietat	Descripció
:plantilla	plantilla	Obligatori	La plantilla.
:instancies	instancia[]	Obligatori	Les instàncies.

Taula 11: El concepte *dades*

- ***correspondencia***: Associa una instància amb una classe.

Camp	Tipus	Obligatorietat	Descripció
:num-instancia	Integer	Obligatori	El número d'instància.
:classe	String	Obligatori	La classe en la que ha estat classificada la instància.

Taula 12: El concepte *correspondencia*

- ***fixters-dades:*** Engloba els noms dels fitxers que contenen la plantilla i les instàncies.

[8] Segons el tipus de l'atribut.

Camp	Tipus	Obligatorietat	Descripció
:fitxer-plantilla	String	Obligatori	El nom del fitxer que conté la plantilla.
:fitxer-instancies	String	Obligatori	El nom del fitxer que conté les instàncies.

Taula 13: El concepte fitxers-dades

4.4.2 Accions i predicats

A banda dels conceptes, l'ontologia de Ninots també conté accions que un agent pot sol·licitar a un altre. Naturalment, no tots els agents seran capaços de realitzar totes les accions; Les accions que pot realitzar un agent estan estretament relacionades amb el tipus de servei que proporciona.

- ***extreure-coneixement:*** Aquesta acció la sol·liciten els agents tutors als agents experts. L'agent tutor proporciona un conjunt d'instàncies i una plantilla a l'agent expert, el qual, mitjançant un procés de mineria de dades, n'obtindrà un coneixement.

Sol·licitant	Tutor	Actor	Expert
Paràmetres			
Camp	Tipus	Obligatorietat	Descripció
:dades	dades	Obligatori	La plantilla i les instàncies de les que se'n vol extreure el coneixement.

Taula 14: L'acció *extreure-coneixement*

Si l'agent expert ha pogut realitzar l'acció, respondrà amb un predicat del tipus *aprenentatge-completat*. El predicat *aprenentatge-completat*, a banda d'indicar el final satisfactori de l'acció, proporciona un llistat de les regles que s'han generat.

Camp	Tipus	Obligatorietat	Descripció
:regles	String[]	Obligatori	La representació textual de les regles generades.

Taula 15: El predicat *aprenentatge-completat*

- ***classificar:*** Aquesta acció la sol·liciten els agents tutors als agents experts. L'agent tutor proporciona un conjunt d'instàncies a l'agent expert sobre les quals aquest haurà d'aplicar el coneixement que hagi extret prèviament arran d'una acció *extreure-coneixement*. Bàsicament, l'agent expert aprofita un saber previ per classificar les instàncies que se li han proporcionat. Com que les instàncies proporcionades es basen en la mateixa plantilla en la que rau el coneixement de l'agent expert, no cal que l'agent tutor proporcioni la plantilla associada a les instàncies.

Sol·licitant	Tutor	Actor	Expert
Paràmetres			
Camp	Tipus	Obligatorietat	Descripció
:instancies	instancia[]	Obligatori	Les instàncies a classificar.

Taula 16: L'acció classificar

Si l'agent *expert* ha pogut realitzar l'acció, respondrà amb un predicat del tipus *el-resultat-de-la-classificacio-es*. El predicat *el-resultat-de-la-classificacio-es* és una sèrie de termes *correspondencia* que associen un número d'instància (que ve determinat per l'ordre en que s'ha rebut) amb un valor de classe:

Camp	Tipus	Obligatorietat	Descripció
:classificacio	correspondencia[]	Obligatori	La sèrie de parells (número d'instància, classe).

Taula 17: El predicat *el-resultat-de-la-classificacio-es*

- ***interpretar:*** Aquesta acció la sol·liciten els agents tutors als agents intèrprets quan necessiten llegir un fitxer d'instàncies. L'agent tutor ha de proporcionar els noms del fitxers que contenen la plantilla i les instàncies.

Sol·licitant	Tutor	Actor	Intèrpret
Paràmetres			
Camp	Tipus	Obligatorietat	Descripció
:fitxers	fitxers-dades	Obligatori	Els noms dels fitxers que contenen la plantilla i les instàncies a llegir.

Taula 18: L'acció *interpretar*

Si l'agent intèrpret ha pogut llegir les dades dels fitxers, respondrà amb un predicat *conte*:

Camp	Tipus	Obligatorietat	Descripció
:fitxers-dades	fitxers-dades	Obligatori	Els noms dels fitxers que contenen la plantilla i les instàncies a llegir.
:dades	dades	Obligatori	Les dades llegides.

Taula 19: El predicat *conte*

Si, contràriament, s'ha produït un error durant la lectura de les dades, l'agent intèrpret respondrà amb un predicat *no-llegible* o *mal-formatat*, segons si l'error ha estat motivat per la inaccessibilitat del fitxer o perquè presenta errors de sintaxi, respectivament.

Camp	Tipus	Obligatorietat	Descripció
:fitxer	String	Obligatori	El nom del fitxer que no pot ser llegit.

Taula 20: El predicat *no-llegible*

Camp	Tipus	Obligatorietat	Descripció
:fitxer	String	Obligatori	El nom del fitxer que presenta errors de sintaxi.
:descripcio-errors	String	Obligatori	La descripció dels errors.

Taula 21: El predicat *mal-formatat*

- ***mostrar-gui:*** Aquesta acció la pot sol·licitar un agent gestor a qualsevol agent del sistema. L'agent destinatari de l'acció respondrà mostrant la seva GUI a l'usuari. Aquesta acció no té paràmetres.

5 Manual de l'usuari

5.1 Prerequisits

Ninots està programat en Java i la seva distribució binària necessita un JRE per funcionar. Si bé tan sols és necessari un JRE per posar en marxa Ninots, es necessita un JDK si se'n volen extreure totes les possibilitats. Sense el JDK no és possible, per exemple, compilar nous agents. Tanmateix, si no s'espera crear nous agents (o aquests s'obtenen de forma externa a Ninots) amb un JRE n'hi ha prou.

El JRE o el JDK estan disponibles a la pàgina web de Java[9] per una àmplia varietat de plataformes. La versió mínima requerida és la 1.4, tot i que es recomana instal·lar l'última disponible.

Per estar basat en la plataforma Java, Ninots és virtualment compatible amb totes les plataformes que aquest suporta. Tanmateix, només ha estat provat sota Windows XP.

5.2 Guia d'instal·lació

Ninots es distribueix en un únic fitxer *ninots.zip*. Aquest fitxer conté totes les llibreries i recursos que Ninots necessita per funcionar.

En primer lloc cal descromprimir el fitxer *ninots.zip* en algun directori de l'ordinador. Un cop desempaquetat, s'hauria de tenir un directori anomenat *ninots 0.8.0*. Dins d'aquest directori hi hauria d'haver els següents fitxers i directoris:

- ***ninots.jar:*** Aquest fitxer és l'executable de Ninots. Conté totes les llibreries i recursos que Ninots necessita per funcionar.
- ***ninots.bat:*** Seqüència que serveix per iniciar Ninots des d'un entorn Windows.
- ***ninots.sh:*** Seqüència que serveix per iniciar Ninots des d'un entorn Unix.
- ***docs/:*** Aquest directori conté diversos documents d'ajuda i referència.
- ***exemples /:*** Aquest directori conté uns quants agents i dades d'exemple.
- ***src/:*** Aquest directori conté el codi font de Ninots. Si es disposa de l'utilitat Ant[10] es pot compilar el codi font per obtenir el fitxer *ninots.jar*.

[9] http://java.sun.com

[10] L'Ant (http://ant.apache.org) és una eina que proporciona les mateixes funcionalitats que la utilitat *make* de GNU. L'Ant, a diferència del *make*, està basat en Java i funciona en totes les plataformes que aquest suporta.

5.3 Executar Ninots

Ninots es pot iniciar des de la línia de comandes o directament des de la interfície del sistema operatiu que s'estigui utilitzant si està correctament configurada per treballar amb aplicacions Java.

Per iniciar-lo des de la línia de comandes de Windows o des d'un terminal Unix, s'utilitza la comanda:

```
$ java -jar ninots.jar
```

al mateix directori a on es troben els fitxers de Ninots. Cal remarcar que el directori *bin* de la instal·lació del JRE s'ha de trobar a la variable *PATH* del sistema. Sinó, cal utilitzar el *path* absolut a l'hora de cridar a la comanda *java*.

Alternativament, es poden utilitzar les seqüències d'arrencada *ninots.bat* i *ninots.sh* depenent de si l'executem des d'un sistema Windows o Unix, respectivament.

En alguns sistemes n'hi ha prou en fer doble clic sobre el fitxer *ninots.jar*. Tanmateix, que aquest mètode funcioni depèn de cada sistema en particular i de com estigui configurat.

Un cop en marxa, Ninots mostra un parell de finestres: la barra d'eines del sistema (ubicada a la cantonada superior dreta de la pantalla), i la GUI de l'agent Gestor (Figura 12).

5.4 La barra d'eines del sistema

Ninots és en si mateix un sistema multi-agent que es controla globalment a través de la barra d'eines del sistema. Aquesta permet aturar i reiniciar el sistema, així com administrar les finestres dels agents que el formen.

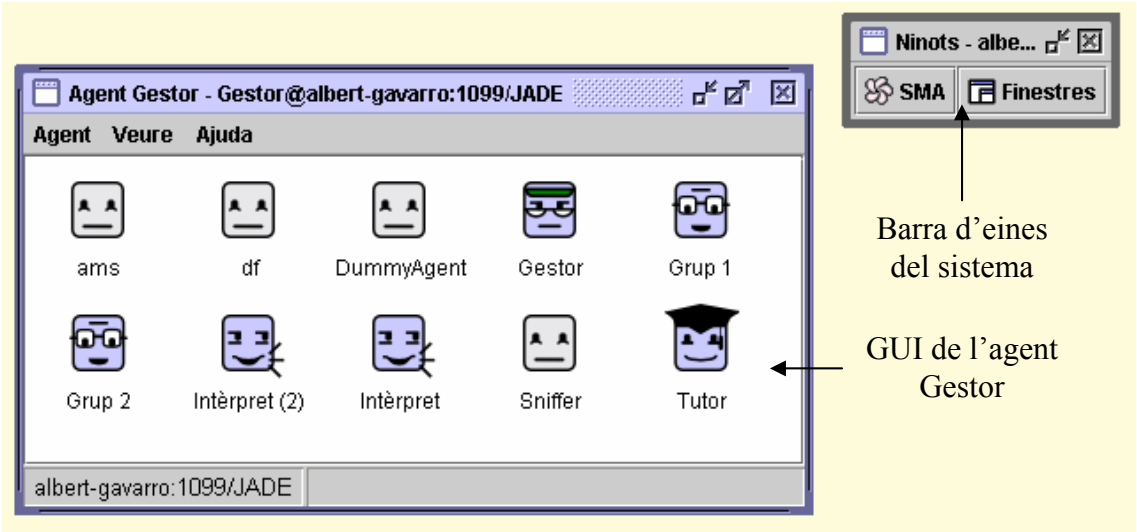


Figura 12: La interfície inicial de Ninots

El menú *SMA* proporciona les següents opcions:

- ***Apagar*:** Atura la plataforma d'agents i tots els agents que estan en execució. Finalitza l'execució de Ninots.

- ***Reiniciar*:** Atura la plataforma d'agents i tots els agents que estan en execució. Seguidament, reinicia la plataforma i posa en marxa els agents inicials del sistema.

El menú *Finestres* proporciona les següents opcions:

- ***Minimitzar totes les finestres*:** Minimitza totes les finestres dels agents del sistema, excepte les dels agents externs.

- ***Restaurar totes les finestres*:** Restaura totes les finestres dels agents del sistema, excepte les dels agents externs. Les finestres maximitzades i minimitzades tornen a la seu estat normal.

Cal remarcar que si es minimitza la barra d'eines del sistema, automàticament es minimitzen totes les finestres dels agents que s'estan executant.

Tancar la barra d'eines del sistema equival a la comanda *Aturar* del menú *SMA*.

5.5 L'agent Gestor

L'agent Gestor mostra els agents que s'estan executant al sistema i permet afegir nous agents o aturar els que ja es troben en execució. Per cada tipus d'agent es mostra una icona característica que permet distingir-los amb facilitat. La Taula 22 mostra els diferents tipus d'agents i la icona que els correspon.

Icona	Tipus d'agent
	Expert
	Extern
	Gestor
	Intèrpret
	Tutor

Taula 22: Les icones dels diferents tipus d'agents

El rol de cada agent dins del sistema es descriu a l'apartat 4.1. Tanmateix, els agents externs mereixen una explicació addicional.

Ninots està compost per quatre tipus d'agents: experts, gestors, intèrprets i tutors. Aquests agents s'encarreguen exclusivament de realitzar les tasques d'aprenentatge i de validació del coneixement per les que va ser dissenyat el sistema. Aquests agents són, doncs, específics de l'aplicació. Tanmateix, com ja s'ha vist a l'apartat 2.4.1, hi ha un

parell d'agents inherents a tot sistema multi-agent: el DF i l'AMS. Aquests agents, que compleixen una funció més genèrica, són representats com a agents externs. També es representen com a agents externs els agents *Sniffer* i *DummyAgent* provinents del JADE (veure A.8 i 5.5.7).

5.5.1 El menú *Veure*

De forma predeterminada, la GUI de l'agent Gestor mostra una distribució en icones dels agents, a més de mostrar-los tots, tant els agents propis de Ninots com la resta d'agents que es puguin estar executant al sistema.

La vista sobre el sistema multi-agent es pot variar a través del menú *Veure*. Aquest menú posa a disposició de l'usuari les següents opcions:

- ***Agents DF i AMS*:** Si està activada, es mostren els agents DF i AMS a la llista d'agents.
- ***Agents Gestors*:** Si està activada, es mostren els agents gestors a la llista d'agents.
- ***Llista*:** Mostra els agents disposats en una llista.
- ***Icones*:** Mostra els agents disposats en icones.

Pot ser interessant desactivar la visualització dels agents DF i AMS, i dels gestors perquè només es mostrin els agents que prenen part activa dels processos de validació i aprenentatge que succeeixen al sistema.

El menú *Veure* ofereix addicionalment l'opció de mostrar la barra d'eines de l'agent gestor. Per defecte, aquesta opció està desactivada.

5.5.2 Iniciar un agent expert

Un dels objectius de l'agent Gestor és el de permetre afegir i treure còmodament agents experts del sistema.

Els agents experts es poden afegir des d'un fitxer *.java* o des d'un fitxer *.class*. Si es fa des d'un fitxer *.java*, Ninots cridarà al compilador de Java per crear el fitxer *.class* corresponent. En aquest cas, el sistema proporcionarà les llibreries necessàries perquè el codi pugui ser compilat satisfactòriament.

Per afegir un agent expert al sistema:

1) Al menú *Agent*, seleccionar *Nou*, i, a continuació, *Expert*. Es mostrarà un quadre de diàleg com el de la Figura 13.
2) A la pestanya *Fitxer*, seleccionar un o més fitxers *.java* o *.class*. Addicionalment, es pot iniciar un agent ja compilat des de qualsevol lloc d'Internet. Tan sols cal introduir la seva URL a la pestanya *URL*.
3) Prémer *Continuar*.

4) Si es desitja que el sistema assigni els noms als agents de forma automàtica, prémer *Acabar*. Si, contràriament, es desitja assignar els noms manualment als agents, cal prémer sobre l'opció *Introduir els noms manualment* i introduir-los a la taula que hi ha a continuació. Un cop introduïts tots els noms, prémer *Acabar*.

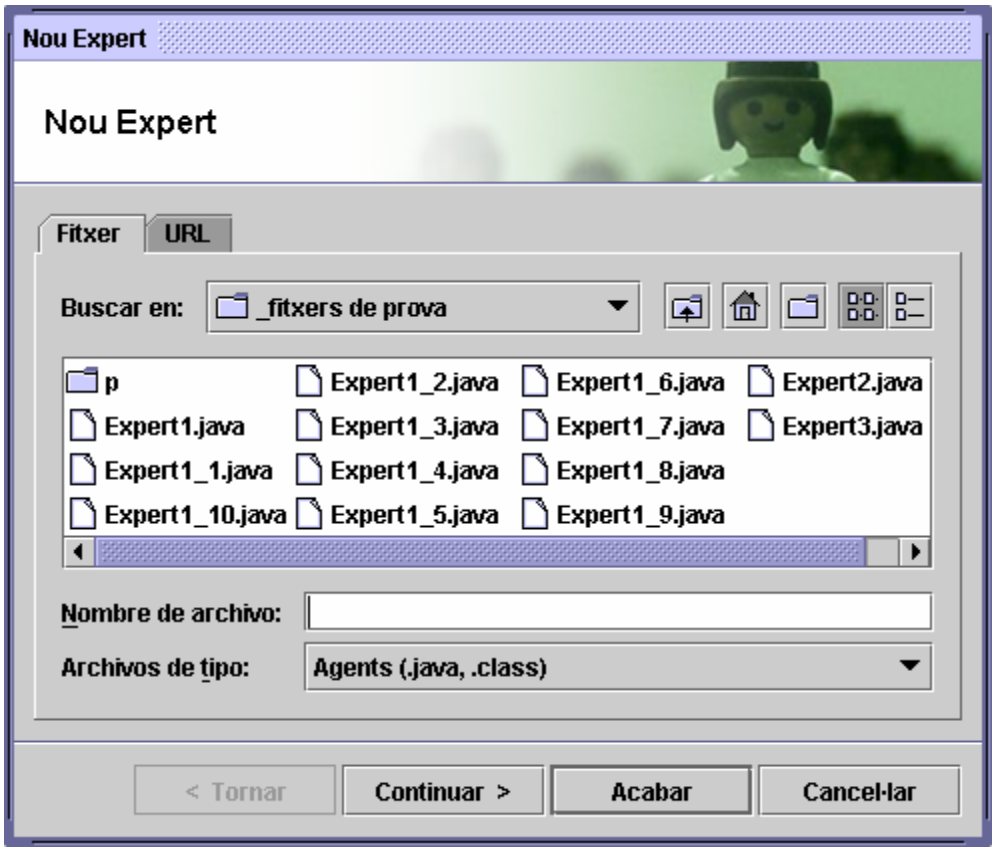


Figura 13: Quadre de diàleg *Nou Expert*

Automàticament, Ninots detecta els fitxers que necessiten ser compilats i els compila. A continuació, afegeix els nous agents experts al sistema.

Si és el primer cop que es compila un fitxer *.java*, Ninots necessita localitzar el fitxer *tools.jar* que acompanya les distribucions del JDK de Java. En aquest cas, es mostrarà un missatge com el de la Figura 14. Prement sobre el botó *Cerca* es pot seleccionar la ubicació del fitxer. Generalment, el fitxer *tools.jar* es troba al subdirectori *lib* del directori d'instal·lació del JDK. Cal remarcar que el fitxer seleccionat ha de ser el que acompanya la versió del JDK amb el que s'està executant Ninots. Altrament, serà impossible utilitzar el compilador de Java.

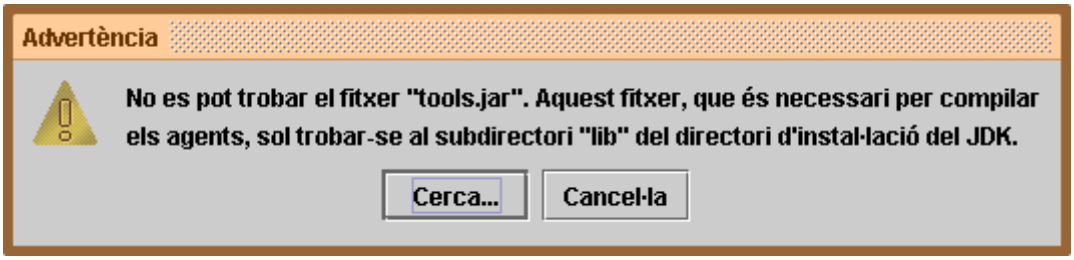


Figura 14: Diàleg per cercar el fitxer *tools.jar*

Cal remarcar que Ninots no proporciona cap mecanisme per modificar la variable *CLASSPATH* abans d'iniciar la compilació. La variable *CLASSPATH*, de forma predeterminada, apunta al fitxer *ninots.jar* i al directori on es troba el fitxer *.java* de l'agent. Conseqüentment, totes les classes que necessiti l'agent per ser compilat han d'estar al mateix directori on es trobi el codi de l'agent. Si es desitja un major control sobre la compilació sempre es pot optar per compilar el codi de l'agent manualment. Si s'opta per aquesta via, cal afegir la llibreria *ninots.jar* al *CLASSPATH*.

Per què un agent expert creat per l'usuari pugui ser afegit al sistema cal que compleixi una sèrie de restriccions:

1) No ha de pertànyer a cap *package*.

2) Ha de derivar de la classe *net.urv.etse.ninots.agents.expert.AgentExpert*.

A l'apartat 3.2 s'explica amb detall com implementar un agent expert.

Si en compilar un agent es produeixen errors, aquests es mostraran en una finestra com la mostrada a la Figura 15. A la part superior es mostra un llistat dels fitxers que tenen errors de compilació. Si es fa clic sobre cadascun d'ells, es mostraran els missatges d'error a la part inferior de la finestra.

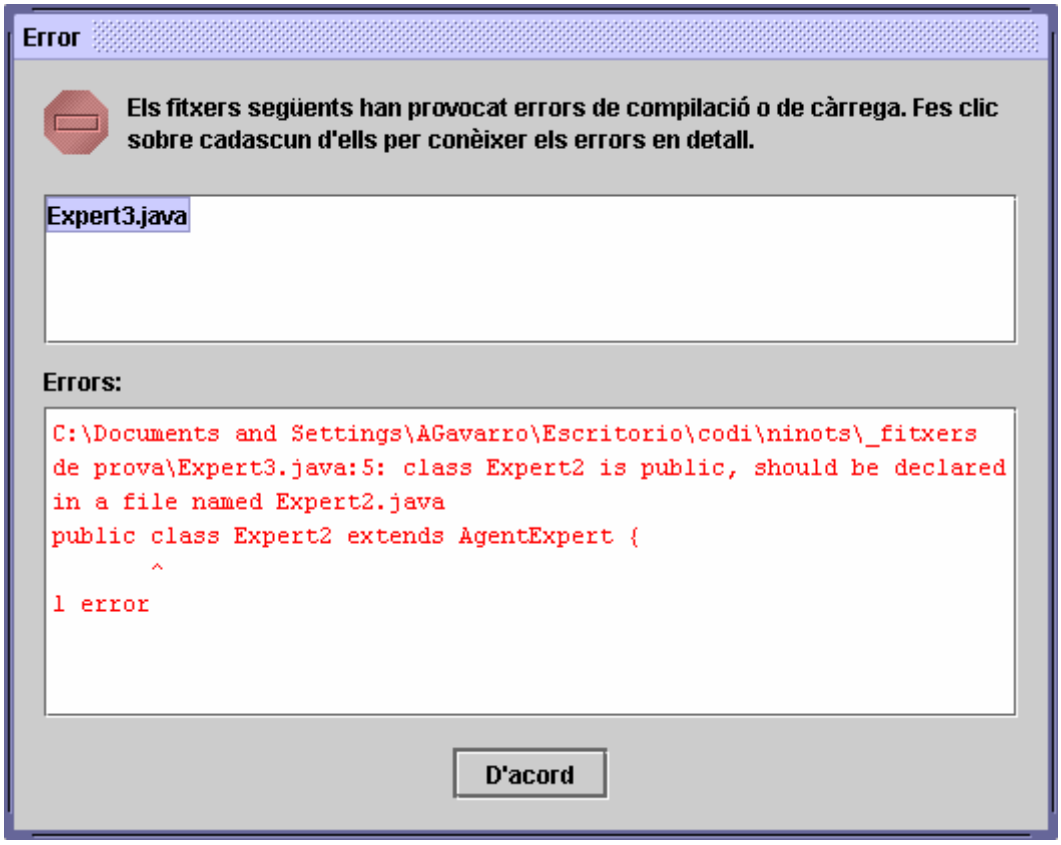


Figura 15: Resum dels errors de compilació dels agents experts

5.5.3 Errors comuns durant la càrrega de nous agents experts

A banda dels errors de compilació, poden haver-hi altres errors que impossibilitin la càrrega de nous agents experts. Els errors més comuns es mostren a la taula següent, acompanyats d'una breu explicació de la seva causa i la seva solució:

Error	Causa	Solució
No es pot carregar l'agent perquè ja existeix un altre agent amb el mateix nom.	Hi ha un altre agent al sistema que s'anomena igual que l'agent que es vol iniciar. Aquest error succeeix quan s'ha especificat manualment un nom per l'agent que ja existeix.	Assignar un nom lliure a l'agent. Comprovar, a través del menú *Veure*, que la GUI de l'agent Gestor estigui mostrant tots els agents.
No es pot determinar la classe de l'agent.	L'agent pertany a algun *package* de Java. Els agents experts no poden pertànyer a cap *package*.	Eliminar la directiva *package* del codi de l'agent expert que es vol carregar. Compilar-lo de nou.
No es pot compilar el fitxer.	No s'ha pogut trobar el fitxer *tools.jar* o el que s'ha seleccionat no és el que acompanya el JDK que està executant Ninots.	Especificar correctament la ruta del fitxer *tools.jar* que acompanya la versió del JDK que està executant Ninots.
Es produeix el següent error de compilació: *Cannot access java.lang.Runnable / bad class file: rt.jar (java/lang/Runnable.class) class file has wrong version 49.0, should be 48.0 / Please remove or make sure it appears in the correct subdirectory of the classpath.*	El fitxer *tools.jar* seleccionat és d'una versió del JDK diferent de la que està executant Ninots.	Utilitzar el JDK correcte per executar Ninots. Si s'està iniciant ninots mitjançant l'script *ninots.bat*, modificar la variable d'entorn JAVA_HOME perquè apunti al directori d'instal·lació del JDK que conté el fitxer *tools.jar* seleccionat.

Taula 23: Errors comuns durant la càrrega de nous agents experts

5.5.4 Iniciar un agent gestor, tutor o intèrpret

A banda de permetre iniciar agents experts, l'agent Gestor també ofereix la possibilitat d'iniciar nous agents gestors, tutors o intèrprets.

Per iniciar un agent gestor, tutor o intèrpret cal anar al menú *Agent* de l'agent Gestor, seleccionar *Nou*, i, a continuació, *Gestor*, *Tutor* o *Intèrpret*, depenent de la preferència.

Immediatament s'afegirà el nou agent al sistema.

5.5.5 Eliminar un agent

De la mateixa manera que es poden afegir nous agents al sistema, també se'n poden eliminar.

Per eliminar un agent del sistema:

1) Seleccionar l'agent que es vol eliminar.

2) Al menú *Agent*, seleccionar *Eliminar*.

O bé:

1) Fer clic amb el botó dret del ratolí sobre l'agent que es vol eliminar.

2) Seleccionar *Eliminar*.

Com que els agents DF i AMS són vitals pel funcionament del sistema no es poden eliminar. Per seguretat, això tampoc és possible amb l'agent Gestor, perquè si s'elimina accidentalment es perd tot el control sobre el sistema.

5.5.6 Mostrar la GUI d'un agent

Alguns agents poden tenir una GUI associada. L'agent Gestor proporciona un mecanisme per mostrar la GUI dels agents propis de Ninots.

Aquesta opció no està disponible no està disponible pels agents externs.

Per veure la GUI d'un agent:

1) Seleccionar l'agent del que es vol mostrar la GUI.

2) Al menú *Agent*, fer clic sobre l'opció *Mostrar la GUI*.

O bé:

1) Fer clic amb el botó dret del ratolí sobre l'agent del que es vol mostrar la GUI.

2) Seleccionar *Mostrar la GUI*.

O més senzill encara:

1) Fer doble clic sobre l'agent del que es vol mostrar la GUI.

Els agents intèrprets no tenen GUI. Per tant, la comanda *Mostrar la GUI* és ignorada per aquests agents.

5.5.7 Agents propis del JADE

L'agent Gestor permet afegir al sistema dos tipus d'agents propis del JADE: agents *Sniffer* i agents *DummyAgent*.

Els agents *Sniffer* permeten inspeccionar de forma selectiva l'intercanvi de missatges entre els agents del sistema. Els agents *DummyAgent* ofereixen una GUI que permet enviar missatges a la resta d'agents i llegir les respostes. El funcionament i les característiques d'aquests agents s'expliquen en detall a l'apartat A.8.

Per iniciar un agent *Sniffer* o *DummyAgent* cal anar al menú *Agent*, seleccionar *Nou*, a continuació, *JADE*, i, finalment, *Sniffer* o *DummyAgent* segons la preferència.

Immediatament s'afegirà el nou agent al sistema.

5.6 L'agent Tutor

Els agents tutor són els que fan treballar la resta d'agents del sistema. La seva missió és la de passar conjunts de dades als agents experts perquè n'extreguin un coneixement o les avaluïn.

Per poder treballar còmodament amb els agents i les dades, l'agent tutor treballa amb bases de dades on emmagatzema els diferents processos d'aprenentatge i de validació, i les dades amb les que opera.

Un procés d'aprenentatge no és res més que un llistat de dades i d'agents. Els agents hauran d'estudiar les dades per extreure'n un coneixement.

Un procés de validació és també un llistat d'agents i de dades, però aquest cop els agents hauran d'estudiar-les per classificar-les. Es diu procés de validació perquè, en certa forma, valida que el procés d'aprenentatge realitzat anteriorment.

Val a dir que les dades utilitzades per l'aprenentatge i la validació han de set compatibles, és a dir, han de satisfer la mateixa plantilla. Si no ho fossin, l'agent expert seria incapaç de classificar les dades perquè pertanyerien a un domini sobre el que no té cap coneixement previ.

5.6.1 Crear una base de dades

Per començar a treballar amb l'agent tutor s'ha de crear una base de dades. Per fer-ho, cal seguir els següents passos:

1) Al menú *BDD*, seleccionar *Nova BDD*, o a la finestra inicial de l'agent tutor, fer clic sobre l'enllaç *Crear una nova base de dades*.

2) A la finestra que s'obre, escriure un nom per la base de dades.

3) Prémer *D'acord*.

Si tot ha anat bé, l'agent Tutor hauria de mostrar una finestra semblant a la de la Figura 16. Inicialment, les bases de dades estan buides. Per començar a treballar cal poblar-les.

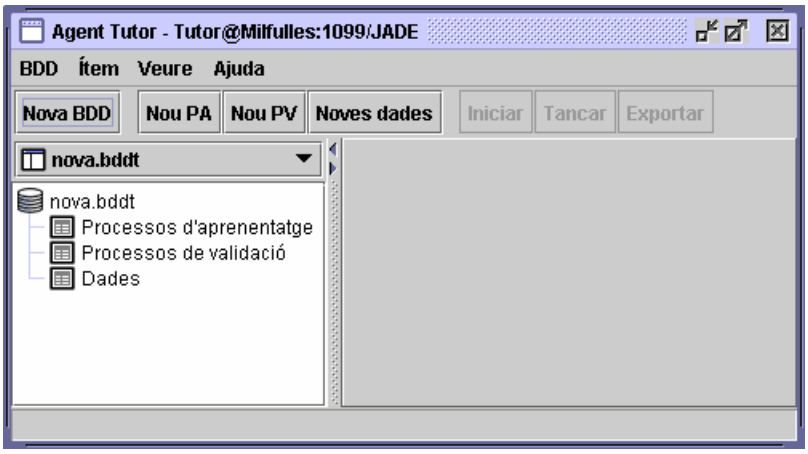


Figura 16: Base de dades buida

5.6.2 Àrea de treball

L'àrea de treball de l'agent Tutor es divideix en tres parts ben diferenciades (veure Figura 17):

1) El selector de la base de dades activa.

2) L'arbre on es mostra el contingut de la base de dades activa.

3) L'àrea d'edició.

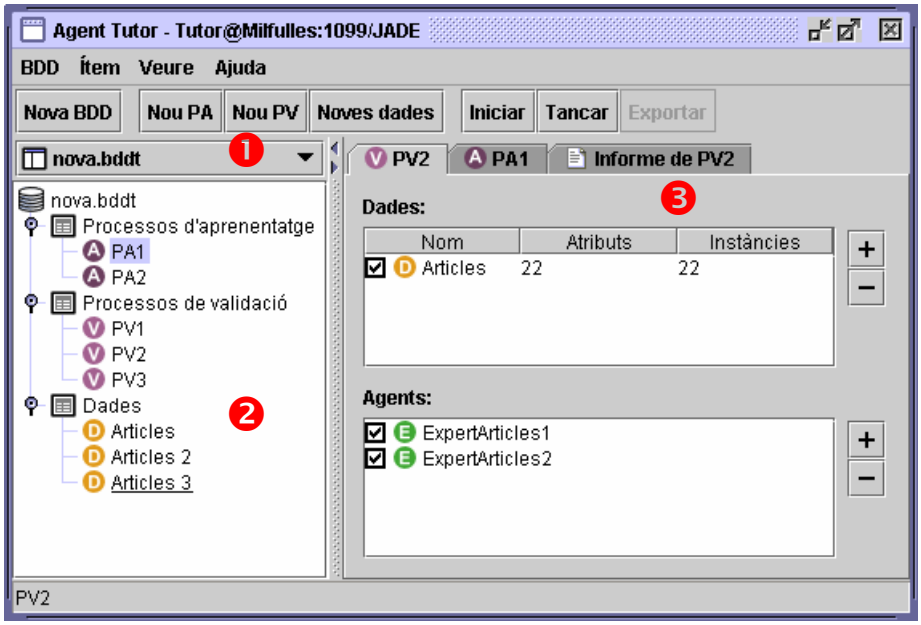


Figura 17: La GUI de l'agent Tutor

El selector de la base de dades activa permet canviar entre les diferents bases de dades que té obertes l'agent Tutor. Cada base de dades té el seu espai de treball propi format per l'arbre de la base de dades i l'àrea d'edició. Aquest espai de treball, com la base de dades en si, és independent de la resta. Conseqüentment, els ítems que trobaven oberts en el moment de canviar d'una base de dades a una altra, tornaran a aparèixer a l'àrea d'edició quan es torni a la base de dades de la que es partia.

Aquesta distribució lògica en àrees de treball permet experimentar amb dades heterogènies sense haver d'obrir i tancar bases de dades contínuament.

L'arbre mostra el contingut de la base de dades actual. A la base de dades s'hi poden trobar tres classes d'ítems:

- Processos d'aprenentatge
- Processos de validació
- Dades

Els processos d'aprenentatge modelen un procediment per instruir a una sèrie d'agents en un domini determinat.

Els processos de validació modelen, justament, el procediment que ha de verificar la qualitat dels agents experts del sistema.

Les dades són, simplement, el domini i els fets del coneixement.

Cada tipus d'ítem està associat a una icona determinada, tal i com es mostra a la TAULA. A més a més, com es pot apreciar a l'arbre, els ítems s'agrupen penjant de la branca que els correspon (veure Figura 17).

Icona	Tipus d'ítem
	Procés d'aprenentatge
	Procés de validació
	Dades

Figura 18: Els tipus d'ítem i les seves icones

A banda d'això, les dades poden aparèixer subratllades o no. Quan estan subratllades, indica que les dades són externes a la base de dades, és a dir, la base de dades només emmagatzema una referència a les dades. Aquestes dades poden trobar-se en qualsevol indret del disc dur o, fins i tot, d'Internet. Per contra, quan no estan subratllades, es diu que les dades estan incrustades, és a dir, que estan físicament dins del fitxer de la base de dades. Les dades incrustades es veuran amb més detall més endavant.

L'àrea d'edició ofereix un espai per editar els diferents ítems que formen la base de dades. Cada editor es mostra en una pestanya separada. A la Figura 17 es poden veure, per exemple, un procés d'aprenentatge i un de validació oberts a l'àrea d'editors.

5.6.3 Afegir dades a la base de dades

Les dades són la peça bàsica del sistema. Sense dades no es poden dur a terme els processos d'aprenentatge i de validació.

Les dades estan constituïdes per un parell de fitxers: un que conté la plantilla que descriu les dades, i un que conté les instàncies.

Per afegir dades al sistema:

1) Al menú *BDD*, seleccionar *Nou* i, finalment, *Dades*.

2) Al quadre de diàleg que es mostra (Figura 19), seleccionar el fitxers que contenen la plantilla i les instàncies.

3) Introduir un nom per identificar les dades dins de la base de dades. Aquest nom ha de ser diferent que el de la resta de dades que ja estiguin a la base de dades.

4) Seleccionar si es volen incrustar o no.

5) Prémer *D'acord*.

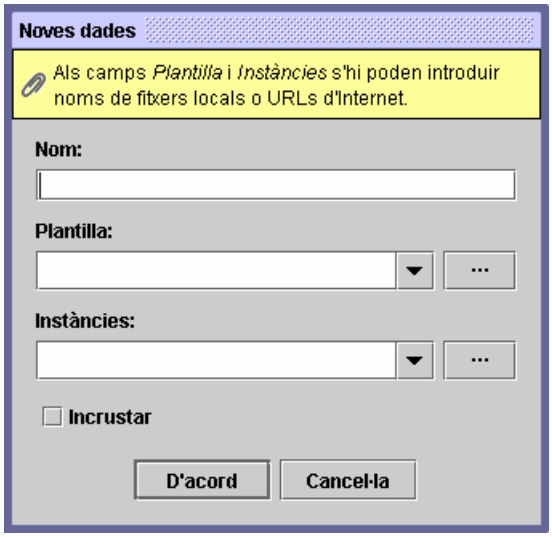


Figura 19: Diàleg Noves Dades

Si es decideix incrustar les dades, aquestes passen a formar part físicament de la base de dades. Sinó, a la base da dades només es guardarà una referència als fitxers. A l'apartat 5.6.11 es parla amb més detall sobre la incrustació de fitxers.

5.6.4 Nou procés d'aprenentatge

Un procés d'aprenentatge permet dirigir un aprenentatge sobre un conjunt d'agents experts.

Per crear un procés d'aprenentatge:

1) Al menú *BDD*, seleccionar *Nou* i, finalment, *Procés d'aprenentatge*.

2) Escriure el nom que del nou procés d'aprenentatge. El nom no ha ser igual al de cap altre procés d'aprenentatge de la base de dades.

3) Prémer *D'acord*.

El nou procés d'aprenentatge es crea immediatament. Aquest apareix sota la branca *Processos d'aprenentatge* de l'arbre de la base de dades.

Cal remarcar que els processos d'aprenentatge, els processos de validació i les dades tenen espais de noms separats. Per tant, un procés d'aprenentatge i unes dades es poden dir igual, però mai es podran dir igual un parell de dades, per exemple.

El contingut de la base de dades es guarda automàticament.

5.6.5 Editar un procés d'aprenentatge

Un procés d'aprenentatge acabat de crear no conté cap mena d'informació. Per especificar quines dades es volen utilitzar per fer l'aprenentatge, i quins agents l'han de realitzar, cal editar-lo.

Per editar un procés d'aprenentatge:

1) Localitzar-lo a l'arbre de la base de dades. Si cal, desplegar la branca *Processos d'aprenentatge*.

2) Fer doble clic sobre el procés.

Immediatament apareix una pestanya a l'àrea d'editors amb el nom del procés i un seguit de controls per poder editar les seves propietats. A la Figura 20 es mostra l'aspecte d'un editor de processos d'aprenentatge.

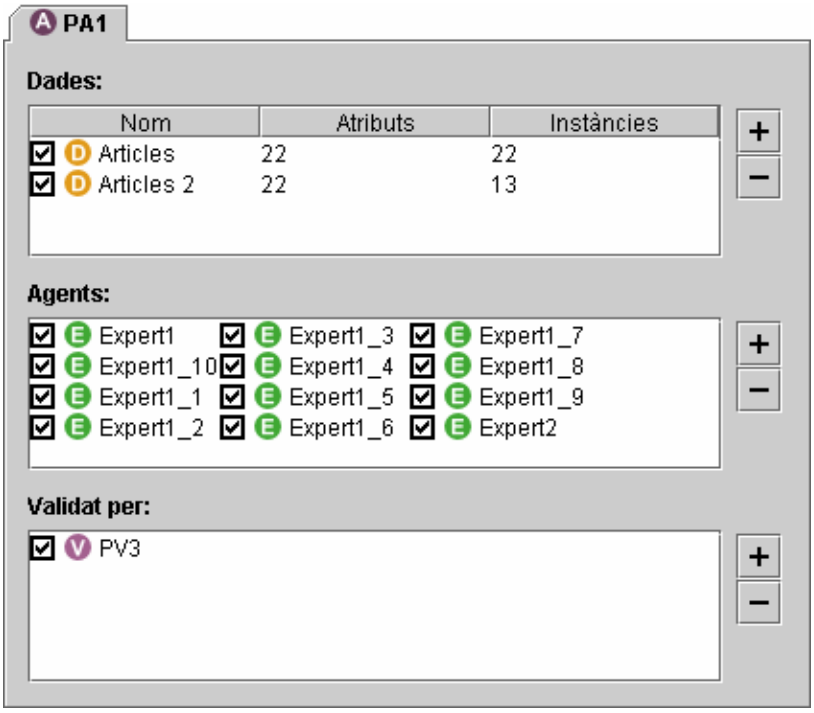


Figura 20: Editor d'un procés d'aprenentatge

Com es pot observar, un procés d'aprenentatge està constituït per tres components fonamentals: dades, agents i processos que el validen. Les dades són enviades als agents perquè n'extreguin un coneixement. Més endavant, aquest coneixement es posarà a prova per mitjà dels procés de validació.

L'editor proporciona tres taules on poder afegir i treure dades, agents i processos de validació. Mitjançant els botons + i − es poden afegir i treure elements de cadascuna de les taules.

La casella de selecció que acompanya a cadascun dels components permet especificat si el component s'ha d'utilitzar o no durant l'aprenentatge. És equivalent a no tenir-lo a la

llista però amb l'avantatge que es pot afegir i treure més ràpidament que no pas mitjançant els botons + i −.

Per què un procés d'aprenentatge es pugui executar, cal que tingui afegits i seleccionats, com a mínim, unes dades i un agent. Els processos de validació són opcionals i només s'associen als processos d'aprenentatge per realitzar aprenentatges i validacions en una sola execució.

Per poder afegir dades i processos de validació a un procés d'aprenentatge cal que aquestes estiguin a la base de dades. A l'apartat 5.6.3 s'explica com afegir dades a la base de dades, i al 5.6.7, com afegir processos de validació.

De forma semblant, per poder afegir un agent a un procés d'aprenentatge cal que l'agent s'estigui executant. A l'apartat 5.5.2 s'explica com afegir agents experts al sistema.

El contingut dels processos d'aprenentatge es guarda automàticament a la base de dades. Tanmateix, només es guarden les dades i els processos que el validen. Els agents, com que són específics de cada execució, no es guarden mai.

5.6.6 Iniciar un procés d'aprenentatge

Per què un procés d'aprenentatge es pugui executar, cal que tingui afegits i seleccionats, com a mínim, unes dades i un agent.

Un procés d'aprenentatge, quan s'executa, realitza les següents accions:

- Recopila totes les dades. Les que encara no ha llegit les sol·licita a algun dels agents intèrprets.
- Envia les dades als agents perquè els hi apliquin un algorisme de mineria de dades.
- Espera la resposta dels agents, confirmant que han processat les dades satisfactòriament.
- Mostra un informe amb els resultats.

Per iniciar un procés d'aprenentatge:

1) Seleccionar el procés d'aprenentatge a l'àrea d'editors o a l'arbre de la base de dades.
2) Al menú *Ítem*, seleccionar *Iniciar*.

Quan el procés acaba, es mostra un informe amb els resultats. A la Figura 21 es pot veure l'aspecte que té. Com es pot apreciar, l'informe consta d'una taula amb una columna per agent, on s'indica si l'agent en qüestió ha completat l'aprenentatge, o no. Si el procés d'aprenentatge tingués afegits i seleccionats un o més processos de validació, els resultats de l'execució d'aquests també apareixerien a l'informe. A l'apartat 5.6.9 es descriu el format dels informes dels processos de validació.

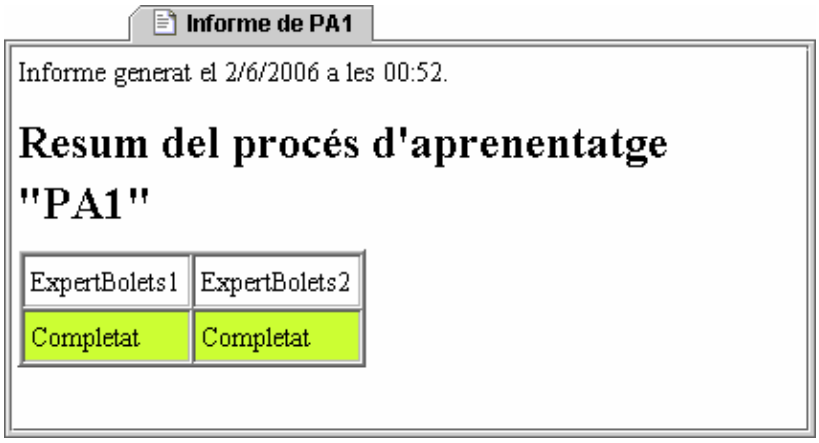


Figura 21: Informe d'un procés d'aprenentatge

Si un procés d'aprenentatge té afegits i seleccionats un o més processos de validació i s'executa, els processos de validació que té associats també s'executaran i s'inclouran una o més seccions amb els seus resultats a l'informe. A l'apartat 5.6.9 s'explica el contingut d'un informe de validació.

5.6.7 Nou procés de validació

Un procés de validació permet dirigir una validació sobre un conjunt d'agents experts que han realitzat un aprenentatge anteriorment.

Per crear un procés de validació:

1) Al menú *BDD*, seleccionar *Nou* i, finalment, *Procés de validació*.

2) Escriure el nom que del nou procés de validació. El nom no ha ser igual al de cap altre procés de validació de la base de dades.

3) Prémer *D'acord*.

El nou procés de validació es crea immediatament. Aquest apareix sota la branca *Processos de validació* de l'arbre de la base de dades.

Cal remarcar que els processos d'aprenentatge, els processos de validació i les dades tenen espais de noms separats. Per tant, un procés d'aprenentatge i unes dades es poden dir igual, però mai es podran dir igual un parell de dades, per exemple.

El contingut de la base de dades es guarda automàticament.

5.6.8 Editar un procés de validació

Un procés de validació acabat de crear no conté cap mena d'informació. Per especificar quines dades es volen utilitzar per fer la validació, i quins agents l'han de realitzar, cal editar-lo.

Per editar un procés de validació:

1) Localitzar-lo a l'arbre de la base de dades. Si cal, desplegar la branca *Processos de validació*.

2) Fer doble clic sobre el procés.

Immediatament apareix una pestanya a l'àrea d'editors amb el nom del procés i un seguit de controls per poder editar les seves propietats. A la Figura 22 es mostra l'aspecte d'un editor de processos de validació.

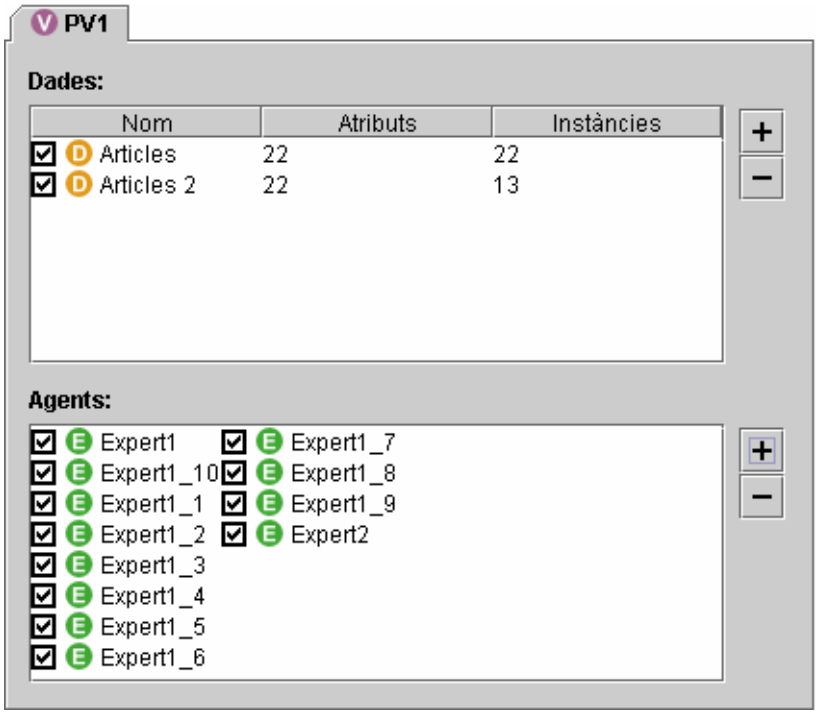


Figura 22: Editor d'un procés de validació

Com es pot observar, un procés de validació és molt semblant a un procés d'aprenentatge: està constituït per dades i agents. Les dades són enviades als agents perquè, mitjançant el coneixement extret d'un aprenentatge anterior, les classifiquin.

L'editor proporciona dues taules on poder afegir i treure dades i agents. Mitjançant els botons + i − es poden afegir i treure elements de cadascuna de les taules.

La casella de selecció que acompanya a cadascun dels components permet especificat si el component s'ha d'utilitzar o no durant la validació. És equivalent a no tenir-lo a la llista però amb l'avantatge que es pot afegir i treure més ràpidament que no pas mitjançant els botons + i −.

Per què un procés de validació es pugui executar, cal que tingui afegits i seleccionats, com a mínim, unes dades i un agent.

Per poder afegir dades a un procés de validació cal que aquestes estiguin a la base de dades. A l'apartat 5.6.3 s'explica com afegir dades a la base de dades.

De forma semblant, per poder afegir un agent a un procés de validació cal que l'agent s'estigui executant. A l'apartat 5.5.2 s'explica com afegir agents experts al sistema.

El contingut dels processos de validació es guarda automàticament a la base de dades. Tanmateix, només es guarden les dades i els processos que el validen. Els agents, com que són específics de cada execució, no es guarden mai.

5.6.9 Iniciar un procés de validació

Per què un procés de validació es pugui executar, cal que tingui afegits i seleccionats, com a mínim, unes dades i un agent.

Un procés de validació es pot iniciar individualment o es pot executar conjuntament amb un procés d'aprenentatge.

Un procés de validació, quan s'executa, realitza les següents accions:

- Recopila totes les dades. Les que encara no ha llegit les sol·licita a algun dels agents intèrprets.
- Envia les dades als agents perquè les classifiquin.
- Espera la resposta dels agents, obtenint la classificació que han generat.
- Mostra un informe amb els resultats.

Per iniciar un procés d'aprenentatge:

1) Seleccionar el procés de validació a l'àrea d'editors o a l'arbre de la base de dades.
2) Al menú *Ítem*, seleccionar *Iniciar*.

Com es pot apreciar, l'informe d'un procés de validació consta de dues parts: la taula d'encerts, que només es mostra si les dades tenen un atribut classificador, i la llista d'instàncies, amb els atributs de cada instància i la classe que li ha atorgat cada agent.

La taula d'encerts té tantes columnes com agents han participat en el procés de validació. Per cada agent es mostra el número d'instàncies que ha classificat correctament i el percentatge que representa sobre el total.

La llista d'instàncies mostra, una a una, totes les instàncies que han participat en el procés de validació. Les instàncies es presenten seguint l'ordre de la taula de dades de l'editor del procés de validació i, dins de cada grup de dades, segons l'ordre que es troben dins del fitxer d'instàncies. Per cada dada es mostren els seus atributs i la classe que li ha atorgat cada agent. La taula d'atributs té una columna per atribut i si algun dels atributs és l'atribut classificador, es ressalta en taronja. La taula que conté la classificació té una columna per agent. Si les dades tenen un atribut classificador i l'agent expert ha classificat la instància correctament, el nom de la classe es ressalta en verd.

Si el procés de validació es realitza conjuntament amb altres processos d'aprenentatge i de validació, els resultats es mostren un rere l'altre a l'informe. A l'apartat 5.6.6 es descriu el format dels informes dels processos d'aprenentatge.

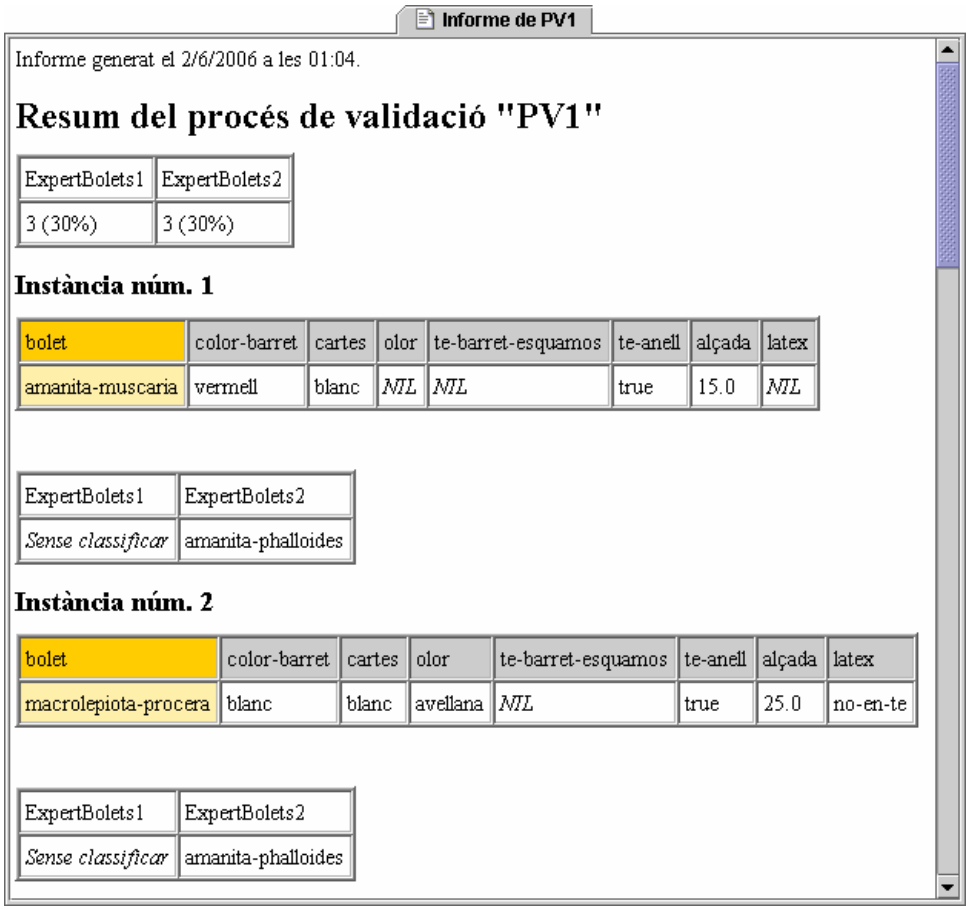


Figura 23: Informe d'un procés de validació

5.6.10 Errors comuns treballant amb processos d'aprenentatge i de validació

Els errors que succeeixen més freqüentment treballant processos d'aprenentatge i de validació es mostren a la taula següent, acompanyats d'una breu explicació de la seva causa i la seva solució:

Error	Causa	Solució
Els agents experts no classifiquen les dades.	Els agents experts no han fet cap aprenentatge anteriorment o l'han fet amb dades que tenen una plantilla diferent de la de les dades proporcionades. Els agents experts han estat eliminats del sistema.	Utilitzar dades que tinguin la mateixa plantilla. Restaurar els agents experts eliminats.
L'agent intèrpret no pot llegir el fitxer.	El fitxer de plantilla o el fitxer d'instàncies no existeixen, han estat moguts de lloc, o no es pot accedir a ells.	Crear unes noves dades assegurant-se que el nom dels fitxers plantilla i d'instàncies sigui correcte. Utilitzar dades incrustades.
El fitxer està mal formatat.	El fitxer de plantilla o el	Corregir el fitxer de

Error	Causa	Solució
	fitxer d'instàncies no s'ajusten a la sintaxi esperada.	plantilla o el fitxer d'instàncies. L'agent Intèrpret proporciona una àmplia informació sobre els errors de sintaxi dels fitxers plantilla o d'instàncies.
No s'ha pogut trobar cap agent intèrpret al sistema.	S'han eliminat els intèrprets del sistema.	Afegir un nou agent intèrpret. A l'apartat 5.5.4 s'explica com fer-ho.

Taula 24: Errors comuns treballant amb processos d'aprenentatge i de validació

5.6.11 Incrustar i exportar

Les dades de la base de dades es poden classificar en dos tipus segons la seva ubicació física: incrustades i externes.

Quan s'afegeixen dades a la base de dades (veure 5.6.3) la interfície ofereix la possibilitat d'incrustar-les. Si s'opta per no fer-ho, la base de dades guarda una referència al fitxer de plantilla i al fitxer d'instàncies. Això significa que la base de dades només guarda la ubicació del fitxers (la ruta del disc o la URL, depenent de si es tracta d'un fitxer local o d'un fitxer d'Internet) i no les dades en si. En canvi, si s'opta per incrustar-les, la base de dades guarda una còpia dels fitxer de plantilla i del fitxer d'instàncies dins de la base de dades.

Aquests dos mecanismes de tenir accés a les dades tenen avantatges i inconvenients. Pel que fa a no incrustar-les, com a avantatges té que els fitxers es poden modificar entre execucions dels diversos processos d'aprenentatge o validació. També genera bases de dades més petites, i l'accés a les dades, en bases de dades grans, és més ràpid. Com a última avantatge té que és equivalent a tenir les dades incrustades si se sap que les dades no canviaran mai d'ubicació. Això pot ser vàlid, per exemple, en dades emmagatzemades en un servidor HTTP o FTP. Com a desavantatges, el més notable és que si es vol distribuir la base de dades no es pot assumir que els fitxers de plantilla i d'instàncies es trobaran en la mateixa ubicació en els ordinadors destí. Típicament, ni tan sols hi seran. En aquest context, es fa obligat distribuir les dades amb la base de dades. Pel que fa a incrustar-les, el seu principal avantatge és que facilita la distribució de la base de dades. Els processos d'aprenentatge, els processos de validació i les dades es troben en un mateix fitxer. En contrapartida, incrustar les dades genera bases de dades més grans i lentes, i impedeix que les dades siguin editades entre successives execucions dels processos d'aprenentatge i de validació.

Aquest seguit de regles poden ajudar a decidir si incrustar o no les dades:

Context	Acció
Es vol distribuir una base de dades entre diferents alumnes perquè puguin sotmetre els seus agents a uns determinats processos d'aprenentatge i validació.	Incrustar.

Context	Acció
Es volen utilitzar unes dades que es troben a Internet.	No incrustar.
Es treballa amb unes dades que encara no són definitives. Poden varia a mesura que es van fent proves amb elles.	No incrustar.
Es vol treballar amb unes dades que es troben al disc local només durant unes poques sessions.	No incrustar.
Es vol conservar la base de dades per un ús futur.	Incrustar.

Taula 25: Regles de decisió sobre la incrustació de dades

Les dades incrustades es poden guardar al disc mitjançant la comanda *Exportar*. Això genera un parell de fitxers (el fitxer de plantilla i el fitxer d'instàncies) que es poden emmagatzemar en qualsevol directori del disc. Per exportar unes dades incrustades:

1) Seleccionar les dades que es volen exportar a l'arbre de la base de dades.

2) Al menú *Ítem*, seleccionar *Exportar*.

3) Seleccionar el directori a on es volen exportar les dades.

4) Prémer *D'acord*.

Al final del procés es creen dos fitxers al directori seleccionat: un amb extensió *.plantilla* que conté la plantilla, l'altre amb extensió *.instancies* que conté les instàncies. El nom dels fitxers és igual al nom que les dades tenien dins de la base de dades.

La comanda *Exportar* també permet guardar els informes al disc en format HTML. Per exportar un informe:

1) Seleccionar l'informe que es desitja exportar.

2) Al menú *Ítem*, seleccionar *Exportar*.

3) Seleccionar el nom del fitxer on es vol guardar l'informe.

4) Prémer *D'acord*.

5.7 Els agents experts

Els agents experts són els encarregats d'aplicar algorismes de mineria de dades sobre conjunts de dades per extreure'n un coneixement. Tot i que són agents d'una gran complexitat interna, la seva interfície és relativament senzilla. A la Figura 24 es mostra l'aspecte que ofereix la GUI dels agents experts.

La interfície presenta un indicador d'estat reforçat amb una barra de progrés i dos botons: *Cancel·lar* i *Més*.

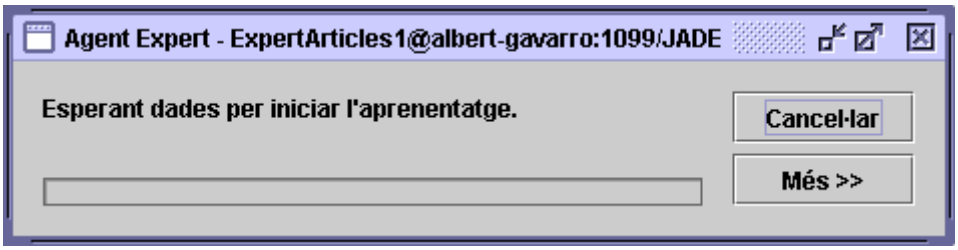


Figura 24: GUI dels agents experts

El botó *Més* desplega la resta de la interfície de l'agent. A la Figura 25 se la pot veure en la seva màxima expressió.

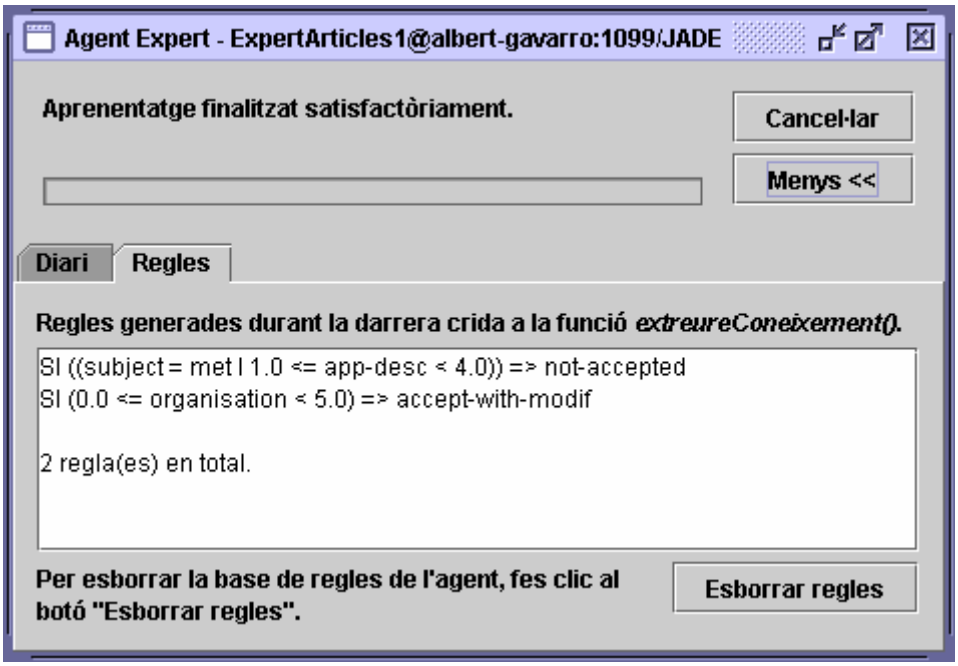


Figura 25: GUI dels agents experts desplegada

Amb la GUI desplegada, prement sobre el botó *Menys* es torna al seu aspecte inicial.

5.7.1 Els estats dels agents experts

Els agents experts passen per diferents estats, depenent de l'estat intern de la seva memòria. L'estat de l'agent es mostra en tot moment a la seva interfície.

Inicialment, qual s'acaba de crear, un agent expert es troba a l'estat NO_EXPERT. Aquest estat es caracteritza per la manca d'informació en el si de l'expert. Bàsicament, encara no ha completat cap aprenentatge i no ha pogut adquirir cap coneixement.

En fer un aprenentatge, l'agent passa a l'estat EXPERT. En aquest estat, l'agent té algun tipus de coneixement relacionat amb les dades amb les que ha fet l'aprenentatge. En aquest estat, les pestanyes *Diari* i *Regles* mostren informació sobre el darrer aprenentatge.

Un aprenentatge fallit o prémer el botó *Esborrar regles* de la pestanya *Regles* torna a l'agent a l'estat NO_EXPERT.

La Figura 26 mostra el diagrama d'estats d'un agent expert.

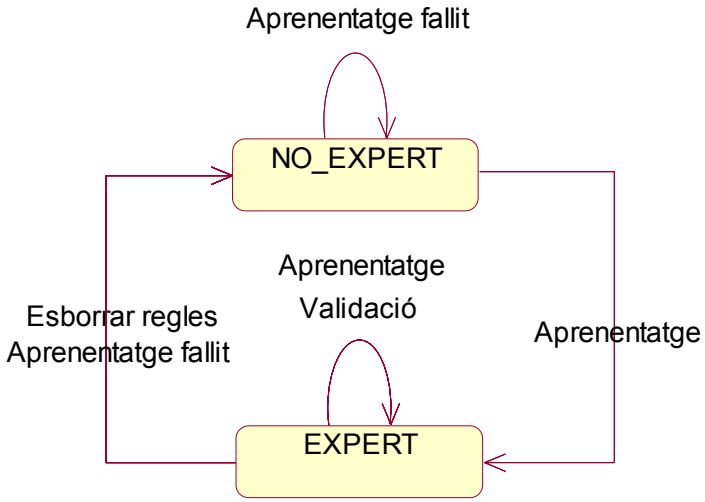


Figura 26: Estats d'un agent expert

5.7.2 La pestanya *Diari*

La pestanya *Diari* mostra el diari generat per l'agent expert durant el darrer procés d'aprenentatge. Cal recordar que el diari el genera cada implementació particular de l'agent expert mitjançant els mètodes *afegirAlDiari()* i *agfegirTraçaAlDiari()* de la classe *net.urv.etse.ninots.agents.expert.AgentExpert*. A l'apartat 3.2 s'explica en detall com programar un agent expert.

5.7.3 La pestanya *Regles*

La pestanya *Regles* mostra les regles generades per l'agent expert durant el darrer procés d'aprenentatge. Es tracta d'un llistat exhaustiu de les regles que constitueixen el coneixement de l'agent.

Les regles s'expressen de la forma:

```
SI (<condició>) => <classe>
```

On <*condició*> és la condició que han de satisfer les instàncies per ser classificades com de la classe <*classe*>.

La resta d'operadors i expressions s'expressen com es mostra a la Taula 26.

Descripció	Notació en C	Notació utilitzada
Operador d'igualtat	a == b	a = b
AND lògic	a && b	(a I b)
Pertinença a un rang	a <= x && x < b	(a <= x < b)
Avaluació d'un booleà	a	a

Taula 26: Notació utilitzada per l'agent expert per mostrar les regles

El botó *Esborrar regles* buida la base de regles de l'agent, tornant a l'agent a l'estat NO_EXPERT.

6 Joc de proves

En aquest capítol es plantejaran cinc de casos de test que permetran verificar la funcionalitat global del sistema. Tot i que es basen exclusivament en la relació Tutor-experts, per dur-los a terme es veuran involucrades moltes altres parts del sistema que, indirectament, quedaran provades.

Cal remarcar que aquests cinc casos de test proven la funcionalitat principal del sistema: que les regles i les classificacions generades pels agents experts, així com els resultats mostrats per l'agent Tutor, siguin coherents amb els resultats obtinguts d'una seqüència d'aprenentatge i de validació.

6.1 Cas de test 1: Dades amb atribut classificador

En aquest cas de test es realitzarà un procés d'aprenentatge i de validació amb dades que tenen definit un atribut classificador. La resposta dels agents es pot comparar amb el valor de l'atribut classificador.

6.1.1 Prerequisits

- Ninots ha d'estar engegat i la interfície de l'agent Gestor, visible.
- Ninots s'ha d'haver iniciat amb una versió del JDK igual a la 1.4 o superior.

6.1.2 Entrades

- El fitxers *ExpertBolets1.java* i *ExpertBolets2.java* que es poden trobar a l'Apèndix D.
- Els fitxers *bolets.plantilla* i *bolets.instancies* que es poden trobar a l'Apèndix D.

6.1.3 Resultats esperats

- Al resum, ha d'aparèixer informació sobre el percentatge d'instàncies que els agents experts han classificat correctament. Es considerarà que una instància està ben classificada si el valor del seu atribut classificador coincideix amb la classe que li ha assignat l'agent. El percentatge d'encerts de cada agent ha de coincidir amb la taula següent:

Agent	Encerts	Percentatge
ExpertBolets1	3	30%
ExpertBolets2	3	30%

- Els resultats de la validació han de coincidir amb el contingut de la taula següent:

Instància	Classe	ExpertBolets1	ExpertBolets2
1	amanita_muscaria	*Sense classificar*	amanita_phalloides
2	macrolepiota_procera	*Sense classificar*	amanita_phalloides

Instància	Classe	ExpertBolets1	ExpertBolets2
3	amanita_muscaria	*Sense classificar*	amanita_phalloides
4	amanita_phalloides	amanita_phalloides	*Sense classificar*
5	lactarius_deliciosus	lactarius_deliciosus	*Sense classificar*
6	lactarius_indigo	*Sense classificar*	lactarius_indigo
7	lactarius_vinosus	lactarius_deliciosus	lactarius_vinosus
8	macrolepiota_procera	*Sense classificar*	macrolepiota_procera
9	amanita_phalloides	*Sense classificar*	*Sense classificar*
10	lactarius_deliciosus	lactarius_deliciosus	*Sense classificar*

6.1.4 Procediment

Pas	Acció	Resultat esperat
1	Al menu *Agent* de l'agent Gestor, seleccionar *Nou*, i, després, *Expert*.	S'obre el quadre de diàleg *Nou Expert*.
2	Seleccionar els agents *ExpertBolets1.java* i *ExpertBolets2*.java i prémer *D'acord*.	S'afegeixen els agents seleccionats al sistema.
3	Fer doble clic sobre l'agent Tutor.	Es mostra la GUI de l'agent Tutor.
4	A la GUI de l'agent Tutor, seleccionar l'opció *Crear una nova base de dades*.	Apareix un quadre de diàleg on es demana el nom de la nova base de dades.
5	Introduir un nom per la base de dades i prémer *D'acord*.	Es mostra una nova base de dades buida.
6	Al menú *BDD*, seleccionar *Nou* i, després, *Dades*.	Apareix el quadre de diàleg *Noves dades*.
7	Al camp *Nom*, introduir "Bolets".	
8	Al camp *Plantilla*, introduir el nom i la ruta del fitxer *bolets.plantilla*. Utilitzar el botó ... per cercar-lo.	
9	Al camp *Instàncies* introduir el nom i la ruta del fitxer *bolets.instancies*. Utilitzar el botó ... per cercar-lo.	
10	Prémer *D'acord*.	S'afegeixen les noves dades a la base de dades.
11	Al menú *BDD*, seleccionar *Nou* i, després, *Procés d'aprenentatge*.	Apareix un quadre de diàleg on es demana el nom del nou procés d'aprenentatge.
12	Introduir "CT1" i prémer *D'acord*.	S'afegeix un nou procés d'aprenentatge *CT1* a la base de dades.
13	Al menú *BDD*, seleccionar *Nou* i, després, *Procés de validació*.	Apareix un quadre de diàleg on es demana el nom del nou procés de validació.
14	Introduir "CT1" i prémer *D'acord*.	S'afegeix un nou procés de validació *CT1* a la base de dades.
15	Desplegar la branca *Processos d'aprenentatge* de l'arbre de la base de dades i fer doble clic sobre el	S'obre l'editor del procés.

Pas	Acció	Resultat esperat
	procés *CT1*.	
16	Afegir a la taula *Dades* les dades *Bolets*.	Les dades *Bolets* s'afegeixen a la taula.
17	Afegir a la taula *Agents* els agents *ExpertBolets1* i *ExpertBolets2*.	Els dos agents s'afegeixen a la taula.
18	Afegir a la taula *Validat per* procés de validació *CT1*.	
19	Desplegar la branca *Processos de validació* de l'arbre de la base de dades i fer doble clic sobre el procés *CT1*.	S'obre l'editor del procés.
20	Afegir a la taula *Dades* les dades *Bolets*.	Les dades *Bolets* s'afegeixen a la taula.
21	Afegir a la taula *Agents* els agents *ExpertBolets1* i *ExpertBolets2*.	Els dos agents s'afegeixen a la taula.
22	Seleccionar el procés d'aprenentatge *CT1* a l'arbre de la base de dades i prémer el botó *Iniciar*.	S'inicia el procés d'aprenentatge i de validació. Quan acaba, es mostra un informe amb els resultats.

6.1.5 Resultats obtinguts

A continuació es mostra l'informe que es genera al final del procediment descrit a l'apartat anterior. Com es pot comprovar, coincideix amb els resultats esperats.

Informe generat el 13/6/2006 a les 00:34.

Resum del procés d'aprenentatge "CT1"

ExpertBolets1	ExpertBolets2
Completat	Completat

Regles generades per l'agent "ExpertBolets1"

SI ((color-barret = vermell I cartes = vermell)) => lactarius-vinosus
SI ((color-barret = taronja I cartes = taronja)) => lactarius-deliciosus
SI ((color-barret = verd I cartes = blanc I olor = farina)) => amanita-phalloides
SI ((color-barret = vermell I te-barret-esquamos I cartes = blanc)) => amanita-muscaria

Regles generades per l'agent "ExpertBolets2"

SI (latex = vermell) => lactarius-vinosus
SI (color-barret = blau) => lactarius-indigo
SI ((cartes = blanc I te-anell)) => amanita-phalloides
SI ((color-barret = blanc I te-barret-esquamos I 20.0 = alçada 40.0)) => macrolepiota-procera

Resum del procés de validació "CT1"

ExpertBolets1	ExpertBolets2
3 (30%)	3 (30%)

Instància núm. 1

bolet	color-barret	cartes	olor	te-barret-esquamos	te-anell	alçada	latex
amanita-muscaria	vermell	blanc	*NIL*	*NIL*	true	15.0	*NIL*

ExpertBolets1	ExpertBolets2
Sense classificar	amanita-phalloides

Instància núm. 2

bolet	color-barret	cartes	olor	te-barret-esquamos	te-anell	alçada	latex
macrolepiota-procera	blanc	blanc	avellana	*NIL*	true	25.0	no-en-te

ExpertBolets1	ExpertBolets2
Sense classificar	amanita-phalloides

Instància núm. 3

bolet	color-barret	cartes	olor	te-barret-esquamos	te-anell	alçada	latex
amanita-muscaria	vermell	blanc	*NIL*	*NIL*	true	12.0	*NIL*

ExpertBolets1	ExpertBolets2
Sense classificar	amanita-phalloides

Instància núm. 4

bolet	color-barret	cartes	olor	te-barret-esquamos	te-anell	alçada	latex
amanita-phalloides	verd	blanc	farina	false	false	10.0	no-en-te

ExpertBolets1	ExpertBolets2

amanita-phalloides	*Sense classificar*

Instància núm. 5

bolet	color-barret	cartes	olor	te-barret-esquamos	te-anell	alçada	latex
lactarius-deliciosus	taronja	taronja	fungica	false	false	*NIL*	taronja

ExpertBolets1	ExpertBolets2
lactarius-deliciosus	*Sense classificar*

Instància núm. 6

bolet	color-barret	cartes	olor	te-barret-esquamos	te-anell	alçada	latex
lactarius-indigo	blau	blau	*NIL*	*NIL*	false	13.0	*NIL*

ExpertBolets1	ExpertBolets2
Sense classificar	lactarius-indigo

Instància núm. 7

bolet	color-barret	cartes	olor	te-barret-esquamos	te-anell	alçada	latex
lactarius-vinosus	taronja	taronja	fungica	*NIL*	false	3.0	vermell

ExpertBolets1	ExpertBolets2
lactarius-deliciosus	lactarius-vinosus

Instància núm. 8

bolet	color-barret	cartes	olor	te-barret-esquamos	te-anell	alçada	latex
macrolepiota-procera	blanc	avellana	*NIL*	true	true	35.0	*NIL*

ExpertBolets1	ExpertBolets2
Sense classificar	macrolepiota-procera

Instància núm. 9

bolet	color-barret	cartes	olor	te-barret-esquamos	te-anell	alçada	latex
amanita-phalloides	blanc	blanc	*NIL*	*NIL*	*NIL*	12.0	*NIL*

ExpertBolets1	ExpertBolets2
Sense classificar	*Sense classificar*

Instància núm. 10

bolet	color-barret	cartes	olor	te-barret-esquamos	te-anell	alçada	latex
lactarius-deliciosus	taronja	taronja	*NIL*	*NIL*	*NIL*	7.0	*NIL*

ExpertBolets1	ExpertBolets2
lactarius-deliciosus	*Sense classificar*

6.2 Cas de test 2: Dades sense atribut classificador

En aquest cas de test es realitzarà un procés d'aprenentatge i de validació amb dades que no tenen definit un atribut classificador.

6.2.1 Prerequisits

- Ninots ha d'estar engegat i la interfície de l'agent Gestor, visible.
- Ninots s'ha d'haver iniciat amb una versió del JDK igual a la 1.4 o superior.

6.2.2 Entrades

- El fitxers *ExpertPisos1.java* i *EspertPisos2.java* que es poden trobar a l'Apèndix D.
- Els fitxers *pisos.plantilla* i *pisos.instancies* que es poden trobar a l'Apèndix D.

6.2.3 Resultats esperats

- Al resum, no ha d'aparèixer informació sobre el percentatge d'instàncies que els agents experts han classificat correctament.
- Els resultats de la validació han de coincidir amb el contingut de la taula següent:

Instància	ExpertBolets1	ExpertBolets2
1	A	A

Instància	ExpertBolets1	ExpertBolets2
2	*Sense classificar*	*Sense classificar*
3	*Sense classificar*	*Sense classificar*
4	A	C
5	C	B
6	*Sense classificar*	A
7	C	C
8	A	C
9	*Sense classificar*	*Sense classificar*
10	B	B

6.2.4 Procediment

Pas	Acció	Resultat esperat
1	Al menu *Agent* de l'agent *Gestor*, seleccionar *Nou*, i, després, *Expert*.	S'obre el quadre de diàleg *Nou Expert*.
2	Seleccionar els agents *ExpertPisos1.java* i *ExpertPisos2.java* i prémer *D'acord*.	S'afegeixen els agents seleccionats al sistema.
3	Fer doble clic sobre l'agent Tutor.	Es mostra la GUI de l'agent Tutor.
4	A la GUI de l'agent Tutor, seleccionar l'opció *Crear una nova base de dades*.	Apareix un quadre de diàleg on es demana el nom de la nova base de dades.
5	Introduir un nom per la base de dades i prémer *D'acord*.	Es mostra una nova base de dades buida.
6	Al menú *BDD*, seleccionar *Nou* i, després, *Dades*.	Apareix el quadre de diàleg *Noves dades*.
7	Al camp *Nom*, introduir "Pisos".	
8	Al camp *Plantilla*, introduir el nom i la ruta del fitxer *pisos.plantilla*. Utilitzar el botó ... per cercar-lo.	
9	Al camp *Instàncies* introduir el nom i la ruta del fitxer *pisos.instancies*. Utilitzar el botó ... per cercar-lo.	
10	Prémer *D'acord*.	S'afegeixen les noves dades a la base de dades.
11	Al menú *BDD*, seleccionar *Nou* i, després, *Procés d'aprenentatge*.	Apareix un quadre de diàleg on es demana el nom del nou procés d'aprenentatge.
12	Introduir "CT2" i prémer *D'acord*.	S'afegeix un nou procés d'aprenentatge *CT2* a la base de dades.
13	Al menú *BDD*, seleccionar *Nou* i, després, *Procés de validació*.	Apareix un quadre de diàleg on es demana el nom del nou procés de validació.
14	Introduir "CT2" i prémer *D'acord*.	S'afegeix un nou procés de validació *CT2* a la base de dades.
15	Desplegar la branca *Processos d'aprenentatge* de l'arbre de la base	S'obre l'editor del procés.

Pas	Acció	Resultat esperat
	de dades i fer doble clic sobre el procés *CT2*.	
16	Afegir a la taula *Dades* les dades *Pisos*.	Les dades *Pisos* s'afegeixen a la taula.
17	Afegir a la taula *Agents* els agents *ExpertPisos1* i *ExpertPisos2*.	Els dos agents s'afegeixen a la taula.
18	Afegir a la taula *Validat per* procés de validació *CT2*.	
19	Desplegar la branca *Processos de validació* de l'arbre de la base de dades i fer doble clic sobre el procés *CT2*.	S'obre l'editor del procés.
20	Afegir a la taula *Dades* les dades *Pisos*.	Les dades *Piso ss*'afegeixen a la taula.
21	Afegir a la taula *Agents* els agents *ExpertPisos1* i *ExpertPisos2*.	Els dos agents s'afegeixen a la taula.
22	Seleccionar el procés d'aprenentatge *CT2* a l'arbre de la base de dades i prémer el botó *Iniciar*.	S'inicia el procés d'aprenentatge i de validació. Quan acaba, es mostra un informe amb els resultats.

6.2.5 Resultats obtinguts

A continuació es mostra l'informe que es genera al final del procediment descrit a l'apartat anterior. Com es pot comprovar, coincideix amb els resultats esperats.

Informe generat el 13/6/2006 a les 01:02.

Resum del procés d'aprenentatge "CT2"

ExpertPisos1	ExpertPisos2
Completat	Completat

Regles generades per l'agent "ExpertPisos1"

SI ((65.0 <= superficie < 80.0 I 200000.0 <= preu < 300000.0)) => B
SI (0.0 <= preu < 180000.0) => C
SI ((te-ascensor I 3.0 <= altura < 9.0)) => A

Regles generades per l'agent "ExpertPisos2"

SI (300000.0 <= preu < 600000.0) => A
SI ((40.0 <= superficie < 70.0 I 5.0 <= altura < 10.0)) => B
SI ((5.0 <= altura < 10.0 I es-lluminos)) => C

Resum del procés de validació "CT2"

Instància núm. 1

superficie	num-habitacions	altura	preu	te-ascensor	es-lluminos
80.0	3.0	3.0	360000.0	true	false

ExpertPisos1	ExpertPisos2
A	A

Instància núm. 2

superficie	num-habitacions	altura	preu	te-ascensor	es-lluminos
100.0	6.0	1.0	250000.0	false	false

ExpertPisos1	ExpertPisos2
Sense classificar	*Sense classificar*

Instància núm. 3

superficie	num-habitacions	altura	preu	te-ascensor	es-lluminos
120.0	5.0	1.0	200000.0	false	true

ExpertPisos1	ExpertPisos2
Sense classificar	*Sense classificar*

Instància núm. 4

superficie	num-habitacions	altura	preu	te-ascensor	es-lluminos
35.0	2.0	7.0	120000.0	true	true

ExpertPisos1	ExpertPisos2
A	C

Instància núm. 5

superficie	num-habitacions	altura	preu	te-ascensor	es-lluminos
42.0	3.0	6.0	130000.0	false	false

ExpertPisos1	ExpertPisos2
C	B

Instància núm. 6

superficie	num-habitacions	altura	preu	te-ascensor	es-lluminos
70.0	4.0	3.0	310000.0	false	true

ExpertPisos1	ExpertPisos2
Sense classificar	A

Instància núm. 7

superficie	num-habitacions	altura	preu	te-ascensor	es-lluminos
65.0	4.0	5.0	163000.0	false	true

ExpertPisos1	ExpertPisos2
C	C

Instància núm. 8

superficie	num-habitacions	altura	preu	te-ascensor	es-lluminos
120.0	5.0	8.0	300000.0	true	true

ExpertPisos1	ExpertPisos2
A	C

Instància núm. 9

superficie	num-habitacions	altura	preu	te-ascensor	es-lluminos
93.0	4.0	1.0	220000.0	false	true

ExpertPisos1	ExpertPisos2
Sense classificar	*Sense classificar*

Instància núm. 10

superficie	num-habitacions	altura	preu	te-ascensor	es-lluminos
65.0	5.0	6.0	218000.0	false	false

ExpertPisos1	ExpertPisos2

B	B

6.3 Cas de test 3: La plantilla de les dades d'aprenentatge no es correspon amb la que els agents esperen

En aquest cas de test es comprovarà que si els agents experts no reconeixen les dades d'aprenentatge, generen una resposta negativa que manega satisfactòriament l'agent Tutor.

6.3.1 Prerequisits

- Ninots ha d'estar engegat i la interfície de l'agent Gestor, visible.
- Ninots s'ha d'haver iniciat amb una versió del JDK igual a la 1.4 o superior.

6.3.2 Entrades

- El fitxer *ExpertBolets1.java* i *ExpertPisos1.java* que es poden trobar a l'Apèndix D.
- Els fitxers *pisos.plantilla* i *pisos.instancies* que es poden trobar a l'Apèndix D.

6.3.3 Resultats esperats

- Al resum de l'aprenentatge, l'agent Tutor ha de mostrar el text "Fallida" sota l'agent *ExpertBolets1*. Sota l'agent *ExpertPisos1* ha de mostrar "Completat".

6.3.4 Procediment

Pas	Acció	Resultat esperat
1	Al menu *Agent* de l'agent *Gestor*, seleccionar *Nou*, i, després, *Expert*.	S'obre el quadre de diàleg *Nou Expert*.
2	Seleccionar els agents *ExpertPisos1.java* i *ExpertBolets1.java* i prémer *D'acord*.	S'afegeixen els agents seleccionats al sistema.
3	Fer doble clic sobre l'agent Tutor.	Es mostra la GUI de l'agent Tutor.
4	A la GUI de l'agent Tutor, seleccionar l'opció *Crear una nova base de dades*.	Apareix un quadre de diàleg on es demana el nom de la nova base de dades.
5	Introduir un nom per la base de dades i prémer *D'acord*.	Es mostra una nova base de dades buida.
6	Al menú *BDD*, seleccionar *Nou* i, després, *Dades*.	Apareix el quadre de diàleg *Noves dades*.
7	Al camp *Nom*, introduir "Pisos".	
8	Al camp *Plantilla*, introduir el nom i la ruta del fitxer *pisos.plantilla*. Utilitzar el botó ... per cercar-lo.	
9	Al camp *Instàncies* introduir el nom i la ruta del fitxer *pisos.instancies*.	

Pas	Acció	Resultat esperat
	Utilitzar el botó ... per cercar-lo.	
10	Prémer *D'acord*.	S'afegeixen les noves dades a la base de dades.
11	Al menú *BDD*, seleccionar *Nou* i, després, *Procés d'aprenentatge*.	Apareix un quadre de diàleg on es demana el nom del nou procés d'aprenentatge.
12	Introduir "CT3" i prémer *D'acord*.	S'afegeix un nou procés d'aprenentatge *CT3* a la base de dades.
13	Desplegar la branca *Processos d'aprenentatge* de l'arbre de la base de dades i fer doble clic sobre el procés *CT3*.	S'obre l'editor del procés.
14	Afegir a la taula *Dades* les dades *Pisos*.	Les dades *Pisos* s'afegeixen a la taula.
15	Afegir a la taula *Agents* els agents *ExpertPisos1* i *ExpertBolets1*.	Els dos agents s'afegeixen a la taula.
16	Prémer el botó *Iniciar*.	S'inicia el procés d'aprenentatge. Quan acaba, es mostra un informe amb els resultats.

6.3.5 Resultats obtinguts

A continuació es mostra l'informe que es genera al final del procediment descrit a l'apartat anterior. Com es pot comprovar, coincideix amb els resultats esperats.

Informe generat el 13/6/2006 a les 01:14.

Resum del procés d'aprenentatge "CT3"

ExpertPisos1	ExpertBolets1
Completat	Fallida

Regles generades per l'agent "ExpertPisos1"

SI ((65.0 <= superficie < 80.0 I 200000.0 <= preu < 300000.0)) => B
SI (0.0 <= preu < 180000.0) => C
SI ((te-ascensor I 3.0 <= altura < 9.0)) => A

Regles generades per l'agent "ExpertBolets1"

Fallida

6.4 Cas de test 4: Les dades de validació i d'aprenentatge no tenen la mateixa plantilla

En aquest cas de test es comprovarà que si les dades de validació tenen una plantilla diferent que les dades d'aprenentatge, els agents experts generen una resposta negativa que manega satisfactòriament l'agent Tutor.

6.4.1 Prerequisits

- Ninots ha d'estar engegat i la interfície de l'agent Gestor, visible.
- Ninots s'ha d'haver iniciat amb una versió del JDK igual a la 1.4 o superior.

6.4.2 Entrades

- El fitxer *ExpertBolets1.java* i *ExpertBolets2.java* que es poden trobar a l'Apèndix D.
- Els fitxers *bolets.plantilla*, *bolets.instancies*, *pisos.plantilla* i *pisos.instancies* que es poden trobar a l'Apèndix D.

6.4.3 Resultats esperats

- Al resum de l'aprenentatge, l'agent Tutor ha de mostrar el text "Completat" sota els agents *ExpertBolets1* i *ExpertBolets2*.
- Al resum de la validació, l'agent Tutor ha de mostrar el text "Fallida" sota els agents *ExpertBolets1* i *ExpertBolets2*, a la taula classificatòria de cada instància.

6.4.4 Procediment

Pas	Acció	Resultat esperat
1	Al menu *Agent* de l'agent Gestor, seleccionar *Nou*, i, després, *Expert*.	S'obre el quadre de diàleg *Nou Expert*.
2	Seleccionar els agents *ExpertBolets1.java* i *ExpertBolets2*.java i prémer *D'acord*.	S'afegeixen els agents seleccionats al sistema.
3	Fer doble clic sobre l'agent Tutor.	Es mostra la GUI de l'agent Tutor.
4	A la GUI de l'agent Tutor, seleccionar l'opció *Crear una nova base de dades*.	Apareix un quadre de diàleg on es demana el nom de la nova base de dades.
5	Introduir un nom per la base de dades i prémer *D'acord*.	Es mostra una nova base de dades buida.
6	Al menú *BDD*, seleccionar *Nou* i, després, *Dades*.	Apareix el quadre de diàleg *Noves dades*.
7	Al camp *Nom*, introduir "Bolets".	
8	Al camp *Plantilla*, introduir el nom i la ruta del fitxer *bolets.plantilla*. Utilitzar el botó ... per cercar-lo.	
9	Al camp *Instàncies* introduir el nom i	

Pas	Acció	Resultat esperat
	la ruta del fitxer *bolets.instancies*. Utilitzar el botó ... per cercar-lo.	
10	Prémer *D'acord*.	S'afegeixen les noves dades a la base de dades.
11	Al menú *BDD*, seleccionar *Nou* i, després, *Dades*.	Apareix el quadre de diàleg *Noves dades*.
12	Al camp *Nom*, introduir "Pisos".	
13	Al camp *Plantilla*, introduir el nom i la ruta del fitxer *pisos.plantilla*. Utilitzar el botó ... per cercar-lo.	
14	Al camp *Instàncies* introduir el nom i la ruta del fitxer *pisos.instancies*. Utilitzar el botó ... per cercar-lo.	
15	Prémer *D'acord*.	S'afegeixen les noves dades a la base de dades.
16	Al menú *BDD*, seleccionar *Nou* i, després, *Procés d'aprenentatge*.	Apareix un quadre de diàleg on es demana el nom del nou procés d'aprenentatge.
17	Introduir "CT4" i prémer *D'acord*.	S'afegeix un nou procés d'aprenentatge *CT4* a la base de dades.
18	Al menú *BDD*, seleccionar *Nou* i, després, *Procés de validació*.	Apareix un quadre de diàleg on es demana el nom del nou procés de validació.
19	Introduir "CT4" i prémer *D'acord*.	S'afegeix un nou procés de validació *CT4* a la base de dades.
20	Desplegar la branca *Processos d'aprenentatge* de l'arbre de la base de dades i fer doble clic sobre el procés *CT4*.	S'obre l'editor del procés.
21	Afegir a la taula *Dades* les dades *Bolets*.	Les dades *Bolets* s'afegeixen a la taula.
22	Afegir a la taula *Agents* els agents *ExpertBolets1* i *ExpertBolets2*.	Els dos agents s'afegeixen a la taula.
23	Afegir a la taula *Validat per* procés de validació *CT4*.	
24	Desplegar la branca *Processos de validació* de l'arbre de la base de dades i fer doble clic sobre el procés *CT4*.	S'obre l'editor del procés.
25	Afegir a la taula *Dades* les dades *Pisos*.	Les dades *Pisos* s'afegeixen a la taula.
26	Afegir a la taula *Agents* els agents *ExpertBolets1* i *ExpertBolets2*.	Els dos agents s'afegeixen a la taula.
27	Seleccionar el procés d'aprenentatge *CT4* a l'arbre de la base de dades i prémer el botó *Iniciar*.	S'inicia el procés d'aprenentatge i de validació. Quan acaba, es mostra un informe amb els resultats.

6.4.5 Resultats obtinguts

A continuació es mostra l'informe que es genera al final del procediment descrit a l'apartat anterior. Com es pot comprovar, coincideix amb els resultats esperats.

Informe generat el 13/6/2006 a les 01:22.

Resum del procés d'aprenentatge "CT4"

ExpertBolets1	ExpertBolets2
Completat	Completat

Regles generades per l'agent "ExpertBolets1"

SI ((color-barret = vermell I cartes = vermell)) => lactarius-vinosus
SI ((color-barret = taronja I cartes = taronja)) => lactarius-deliciosus
SI ((color-barret = verd I cartes = blanc I olor = farina)) => amanita-phalloides
SI ((color-barret = vermell I te-barret-esquamos I cartes = blanc)) => amanita-muscaria

Regles generades per l'agent "ExpertBolets2"

SI (latex = vermell) => lactarius-vinosus
SI (color-barret = blau) => lactarius-indigo
SI ((cartes = blanc I te-anell)) => amanita-phalloides
SI ((color-barret = blanc I te-barret-esquamos I 20.0 <= alçada < 40.0)) => macrolepiota-procera

Resum del procés de validació "CT4"

Instància núm. 1

superficie	num-habitacions	altura	preu	te-ascensor	es-lluminos
80.0	3.0	3.0	360000.0	true	false

ExpertBolets1	ExpertBolets2
Fallida	Fallida

Instància núm. 2

superficie	num-habitacions	altura	preu	te-ascensor	es-lluminos
100.0	6.0	1.0	250000.0	false	false

ExpertBolets1	ExpertBolets2
Fallida	Fallida

Instància núm. 3

superficie	num-habitacions	altura	preu	te-ascensor	es-lluminos
120.0	5.0	1.0	200000.0	false	true

ExpertBolets1	ExpertBolets2
Fallida	Fallida

Instància núm. 4

superficie	num-habitacions	altura	preu	te-ascensor	es-lluminos
35.0	2.0	7.0	120000.0	true	true

ExpertBolets1	ExpertBolets2
Fallida	Fallida

Instància núm. 5

superficie	num-habitacions	altura	preu	te-ascensor	es-lluminos
42.0	3.0	6.0	130000.0	false	false

ExpertBolets1	ExpertBolets2
Fallida	Fallida

Instància núm. 6

superficie	num-habitacions	altura	preu	te-ascensor	es-lluminos
70.0	4.0	3.0	310000.0	false	true

ExpertBolets1	ExpertBolets2
Fallida	Fallida

Instància núm. 7

superficie	num-habitacions	altura	preu	te-ascensor	es-lluminos
65.0	4.0	5.0	163000.0	false	true

ExpertBolets1	ExpertBolets2
Fallida	Fallida

Instància núm. 8

superficie	num-habitacions	altura	preu	te-ascensor	es-lluminos
120.0	5.0	8.0	300000.0	true	true

ExpertBolets1	ExpertBolets2
Fallida	Fallida

Instància núm. 9

superficie	num-habitacions	altura	preu	te-ascensor	es-lluminos
93.0	4.0	1.0	220000.0	false	true

ExpertBolets1	ExpertBolets2
Fallida	Fallida

Instància núm. 10

superficie	num-habitacions	altura	preu	te-ascensor	es-lluminos
65.0	5.0	6.0	218000.0	false	false

ExpertBolets1	ExpertBolets2
Fallida	Fallida

6.5 Cas de test 5: Validació sense aprenentatge

En aquest cas de test es comprovarà que si s'inicia una validació sense haver fet un aprenentatge previ, els agents experts generen una resposta negativa que és manegada satisfactòriament per l'agent Tutor.

6.5.1 Prerequisits

- Ninots ha d'estar engegat i la interfície de l'agent Gestor, visible.
- Ninots s'ha d'haver iniciat amb una versió del JDK igual a la 1.4 o superior.

6.5.2 Entrades

- El fitxers *ExpertBolets1.java* i *ExpertBolets2.java* que es poden trobar a l'Apèndix D.

- Els fitxers *bolets.plantilla* i *bolets.instancies* que es poden trobar a l'Apèndix D.

6.5.3 Resultats esperats

- Al resum de la validació, l'agent Tutor ha de mostrar el text "Fallida" sota els agents *ExpertBolets1* i *ExpertBolets2*, a la taula classificatòria de cada instància.

6.5.4 Procediment

Pas	Acció	Resultat esperat
1	Al menu *Agent* de l'agent Gestor, seleccionar *Nou*, i, després, *Expert*.	S'obre el quadre de diàleg *Nou Expert*.
2	Seleccionar els agents *ExpertBolets1.java* i *ExpertBolets2*.java i prémer *D'acord*.	S'afegeixen els agents seleccionats al sistema.
3	Fer doble clic sobre l'agent Tutor.	Es mostra la GUI de l'agent Tutor.
4	A la GUI de l'agent Tutor, seleccionar l'opció *Crear una nova base de dades*.	Apareix un quadre de diàleg on es demana el nom de la nova base de dades.
5	Introduir un nom per la base de dades i prémer *D'acord*.	Es mostra una nova base de dades buida.
6	Al menú *BDD*, seleccionar *Nou* i, després, *Dades*.	Apareix el quadre de diàleg *Noves dades*.
7	Al camp *Nom*, introduir "Bolets".	
8	Al camp *Plantilla*, introduir el nom i la ruta del fitxer *bolets.plantilla*. Utilitzar el botó ... per cercar-lo.	
9	Al camp *Instàncies* introduir el nom i la ruta del fitxer *bolets.instancies*. Utilitzar el botó ... per cercar-lo.	
10	Prémer *D'acord*.	S'afegeixen les noves dades a la base de dades.
11	Al menú *BDD*, seleccionar *Nou* i, després, *Procés de validació*.	Apareix un quadre de diàleg on es demana el nom del nou procés de validació.
12	Introduir "CT5" i prémer *D'acord*.	S'afegeix un nou procés de validació *CT5* a la base de dades.
13	Desplegar la branca *Processos de validació* de l'arbre de la base de dades i fer doble clic sobre el procés *CT5*.	S'obre l'editor del procés.
14	Afegir a la taula *Dades* les dades *Bolets*.	Les dades *Bolets* s'afegeixen a la taula.
15	Afegir a la taula *Agents* els agents *ExpertBolets1* i *ExpertBolets2*.	Els dos agents s'afegeixen a la taula.

Pas	Acció	Resultat esperat
16	Prémer el botó *Iniciar*.	S'inicia el procés d'aprenentatge i de validació. Quan acaba, es mostra un informe amb els resultats.

6.5.5 Resultats obtinguts

A continuació es mostra l'informe que es genera al final del procediment descrit a l'apartat anterior. Com es pot comprovar, coincideix amb els resultats esperats.

Informe generat el 13/6/2006 a les 01:29.

Resum del procés de validació "CT5"

ExpertBolets1	ExpertBolets2
Fallida	Fallida

Instància núm. 1

bolet	color-barret	cartes	olor	te-barret-esquamos	te-anell	alçada	latex
amanita-muscaria	vermell	blanc	*NIL*	*NIL*	true	15.0	*NIL*

ExpertBolets1	ExpertBolets2
Fallida	Fallida

Instància núm. 2

bolet	color-barret	cartes	olor	te-barret-esquamos	te-anell	alçada	latex
macrolepiota-procera	blanc	blanc	avellana	*NIL*	true	25.0	no-en-te

ExpertBolets1	ExpertBolets2
Fallida	Fallida

Instància núm. 3

bolet	color-barret	cartes	olor	te-barret-esquamos	te-anell	alçada	latex
amanita-muscaria	vermell	blanc	*NIL*	*NIL*	true	12.0	*NIL*

ExpertBolets1	ExpertBolets2
Fallida	Fallida

Instància núm. 4

bolet	color-barret	cartes	olor	te-barret-esquamos	te-anell	alçada	latex
amanita-phalloides	verd	blanc	farina	false	false	10.0	no-en-te

ExpertBolets1	ExpertBolets2
Fallida	Fallida

Instància núm. 5

bolet	color-barret	cartes	olor	te-barret-esquamos	te-anell	alçada	latex
lactarius-deliciosus	taronja	taronja	fungica	false	false	*NIL*	taronja

ExpertBolets1	ExpertBolets2
Fallida	Fallida

Instància núm. 6

bolet	color-barret	cartes	olor	te-barret-esquamos	te-anell	alçada	latex
lactarius-indigo	blau	blau	*NIL*	*NIL*	false	13.0	*NIL*

ExpertBolets1	ExpertBolets2
Fallida	Fallida

Instància núm. 7

bolet	color-barret	cartes	olor	te-barret-esquamos	te-anell	alçada	latex
lactarius-vinosus	taronja	taronja	fungica	*NIL*	false	3.0	vermell

ExpertBolets1	ExpertBolets2
Fallida	Fallida

Instància núm. 8

bolet	color-barret	cartes	olor	te-barret-esquamos	te-anell	alçada	latex
macrolepiota-procera	blanc	avellana	*NIL*	true	true	35.0	*NIL*

ExpertBolets1	ExpertBolets2
Fallida	Fallida

Instància núm. 9

bolet	color-barret	cartes	olor	te-barret-esquamos	te-anell	alçada	latex
amanita-phalloides	blanc	blanc	*NIL*	*NIL*	*NIL*	12.0	*NIL*

ExpertBolets1	ExpertBolets2
Fallida	Fallida

Instància núm. 10

bolet	color-barret	cartes	olor	te-barret-esquamos	te-anell	alçada	latex
lactarius-deliciosus	taronja	taronja	*NIL*	*NIL*	*NIL*	7.0	*NIL*

ExpertBolets1	ExpertBolets2
Fallida	Fallida

7 Conclusions

Tot i ser una paradigma de la programació encara incipient, mancat de l'ampli ventall d'eines de les que gaudeix, per exemple, la programació orientada a objectes, i orfe d'un llenguatge propi, la programació orientada a agents demostra ser perfectament útil a l'hora d'abordar la implementació de qualsevol tipus de sistema. En el cas de Ninots, un programari senzill que permet als alumnes experimentar amb sistemes multi-agent i mineria de dades, l'orientació a agents ha permès construir un sistema molt modular i fàcilment ampliable. Aquestes virtuts eleven el manteniment i la reutilització del codi al seu màxim exponent.

Per altra banda, el Jess s'ha mostrat com una solució senzilla i alhora molt potent per implementar sistemes experts en Java. La integració amb el llenguatge Java és total, fins al punt que el llenguatge del Jess permet accedir als objectes i a les classes de l'entorn d'execució del Java com si fossin vulgars variables del llenguatge. L'estreta relació entre el Java i el Jess permet iteractuar amb el sistema expert amb molta facilitat, simplificant la feina del programador fins a extrems insospitats. A més a més, l'algorisme RETE del Jess és notablement ràpid, tant que fins i tot pot rivalitzar amb altres implementacions escrites en llenguatges compilats. Tot i que, a diferència del CLIPS, el llenguatge del Jess no té construccions per definir objectes, aquesta mancança no desmereix una llibreria rica en característiques, fàcilment integrable i ràpida.

Des del punt de vista de l'experiència personal, aquest projecte m'ha permès treballar en totes les fases del cicle de vida d'un programari: des de l'especificació i la redacció dels requeriments fins a la validació, passant pel disseny i la implementació. Tot i que la meva experiència laboral m'ha donat un cert bagatge en totes aquestes fases, potser és el primer cop que l'especificació la començo des de zero. Generalment, se sol partir d'uns requeriments bastant estrictes del client, que moltes vegades arriben a ser vertaders documents de disseny. En canvi, en el cas de Ninots, tota l'aplicació ha estat dissenyada des de zero, havent-se de tenir en compte aspectes com la facilitat d'ús del programari, el disseny intuïtiu i homogeni de la interfície gràfica, el model didàctic de l'aplicació, el disseny senzill i alhora segur de la interfície de programació dels agents experts, etc. Mil i un aspectes que són la feina real d'un enginyer informàtic i que fins ara havia tingut poques ocasions per abordar-los. En aquest sentit puc dir que he après molt.

Finalment, crec que l'objectiu que s'havia fixat aquest projecte de crear una eina de suport a la docència en matèria de sistemes multi-agent i mineria de dades s'ha superat amb escreix, i que Ninots servirà perfectament per aquest propòsit.

8 Treball futur

El treball futur més immediat al que gairebé obliga aquest projecte és començar a utilitzar Ninots pel que ha estat dissenyat: com a eina de suport a la docència en matèria de sistemes multi-agent i mineria de dades. Així s'esgota l'última etapa del cicle de vida de l'aplicació: la validació per part de l'usuari. A partir dels comentaris dels propis usuaris es poden plantejar futures millores o modificacions, entrant en un cicle de millora la duració del qual dependrà de l'acceptació que tingui Ninots entre els seus potencials usuaris.

A banda d'això, Ninots és una aplicació tancada que no admet un augment massa exagerat de les seves funcionalitats sense canviar la seva finalitat. En tot cas es pot utilitzar l'experiència adquirida durant el desenvolupament de l'aplicació per afrontar nous projectes en matèria de sistemes multi-agent.

Apèndix A JADE

En aquest apèndix es farà una breu introducció al JADE[11], tant des del punt de vista de la programació d'un sistema multi-agent utilitzant les seves llibreries, com de les eines que proporciona per la gestió i, eventualment, depuració d'un sistema d'aquestes característiques. La versió descrita en aquest apèndix és la mateixa que ha estat utilitzada per la realització d'aquest projecte: la 3.2 del 2004/07/26.

A.1 Els paquets del JADE

El JADE està format per una sèrie de paquets que encapsulen les seves funcionalitats:

- ***jade:*** Aquest paquet conté, simplement, les classes per engegar el JADE.
- ***jade.core:*** És el nucli del sistema. Aquest paquet proporciona la classe *jade.core.Agent* que és la classe de la que derivaran els nostres agents.
- ***jade.core.behaviours:*** Aquest paquet és un subpaquet del *jade.core* i conté les classes que solen utilitzar-se per implementar els comportaments (*behaviours*) bàsics dels agents.
- ***jade.domain:*** Conté totes les classes que implementen la plataforma d'agents i els models de domini: les entitats per a l'administració d'agents, llenguatges i ontologies (AMS, DF, ACC, etc).
- ***jade.lang.acl:*** Aquest paquet conté el suport pel *FIPA Agent Communication Language* (ACL) incloent la classe *ACLMessage*, l'intèrpret, el codificador, i les classes auxiliars necessàries per treballar amb missatges ACL.
- ***jade.proto:*** Conté la implementació dels protocols estàndards de la FIPA.
- ***jade.util:*** Aquest paquet conté un conjunt d'eines que ens facilitaran el desenvolupament i l'administració del SMA:
    - ***Remote Management Agent* (RMA):** Agent que mostra de forma gràfica l'estat de la plataforma i que ens permet realitzar tasques d'administració.
    - ***Dummy Agent*:** Agent que ens ajuda a depurar els sistemes. Mitjançant la seva interfície podem crear i enviar missatges ACL a altres agents.
    - ***Sniffer*:** Agent que permet visualitzar el tràfic, total o parcial, d'una plataforma.
    - ***Logger:*** Classe que proporciona capacitats de *logging*.

A més a més, conté altres paquets no tan coneguts però que són vitals en la realització de determinades tasques, com per exemple:

[11] http://jade.cselt.it

- ***jade.domain.FIPAAgentManagement:*** Aquest paquet conté la definició de l'ontologia *FIPA-Agent-Management* tal i com està especificada per l'estàndard *FIPA Agent Management Specification* del a FIPA.

- ***jade.domain.JADEAgentManagement:*** Aquest paquet conté la definició de l'ontologia *JADE-Agent-Management*, el vocabulari amb la llista de símbols comuns, i totes les classes de Java que implementen els conceptes de l'ontologia.

- ***jade.domain.mobility:*** Aquest paquet conté la definició de l'ontologia *JADE-mobility*, el vocabulari amb la llista dels símbols comuns, i totes les classes de Java que implementen els conceptes de l'ontologia.

A.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 27).

Quan s'executa el JADE, es creen immediatament els agents DF i AMS, i el mòdul ACC (*Agent Communitacion Channel*) es posa en marxa per rebre missatges. La plataforma d'agents es pot dividir en diferents ordinadors. Només una aplicació Java, i, per tant, només una JVM (*Java Virtual Machine*), es executada en cada ordinador. Cada JVM és un contenidor simple d'agents que permet al diferents agents executar-se de forma concurrent en un mateix ordinador. El *main-container* és el contenidor on viuen els agents DF i AMS, i on el *RMI Registry*, utilitzat internament pel JADE, s'està executant. La resta de contenidors es connecten al contenidor principal, formant una plataforma distribuïda d'agents.

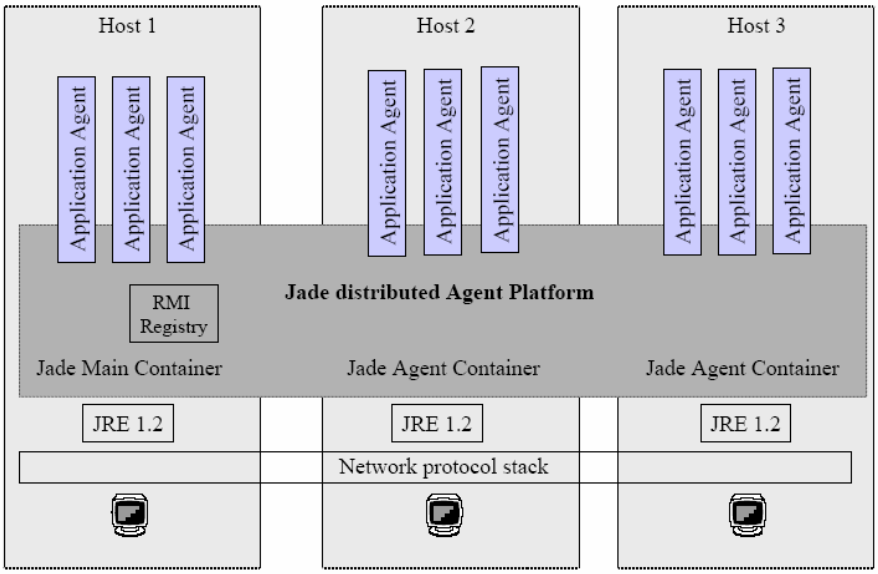


Figura 27: Arquitectura del JADE

A.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 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 de l'agent. Els diferents *behaviours* s'executen de forma pseudo-concurrent dirigits per un planificador no *preemptive* intern. Els *behaviours* es divideixen en compostos i simples, depenent de si poden o no poden tenir fills, respectivament. Anem a veure'ls en detall.

A.3.1 Els *behaviours* 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 *behaviour* ha acabat o no, i el *reset()*, que reinicia el *behaviour*. Els *behaviours* 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.

A.3.2 Els *behaviours* compostos

Els *behaviours* compostos es diferencien dels simples en el fet que poden tenir fills. És a dir, podem associar un o més *sub-behaviours* a un *behaviour* compost, que seran gestionats per un planificador propi i independent del de l'agent.

Tots aquests comportaments deriven de la classe *jade.core.CompositeBehaviour*. Els mètodes a sobrecarregar són l'*onStart()* i l'*onEnd()*. El primer mètode que es cridarà serà l'*onStart()* i serà justament allí on afegirem els *behaviours* fills. Finalitzada l'execució d'aquest mètode es llençaran els *behaviours* fill i, quan acabin tots, s'invocarà el mètode *onEnd()*. La política d'execució dels fills serà la que diferenciarà els tres tipus de *behaviours* compostos:

- ***SequentialBehaviour:*** Els fills s'executen en seqüència i en l'ordre en que s'han creat.
- ***FSMBehaviour:*** Planifica l'execució dels fills segons una màquina d'estats finits (FSM, *Finite State Machine*) definida per l'usuari. Concretament, cada fill representa un estat de la FSM.

- ***ParallelBehaviour:*** Els fills s'executen de forma concurrent i acaba quan una condició particular es dona en algun dels seus fills, per exemple, que tots han acabat, que *N* han acabat o que algun d'ells ha acabat.

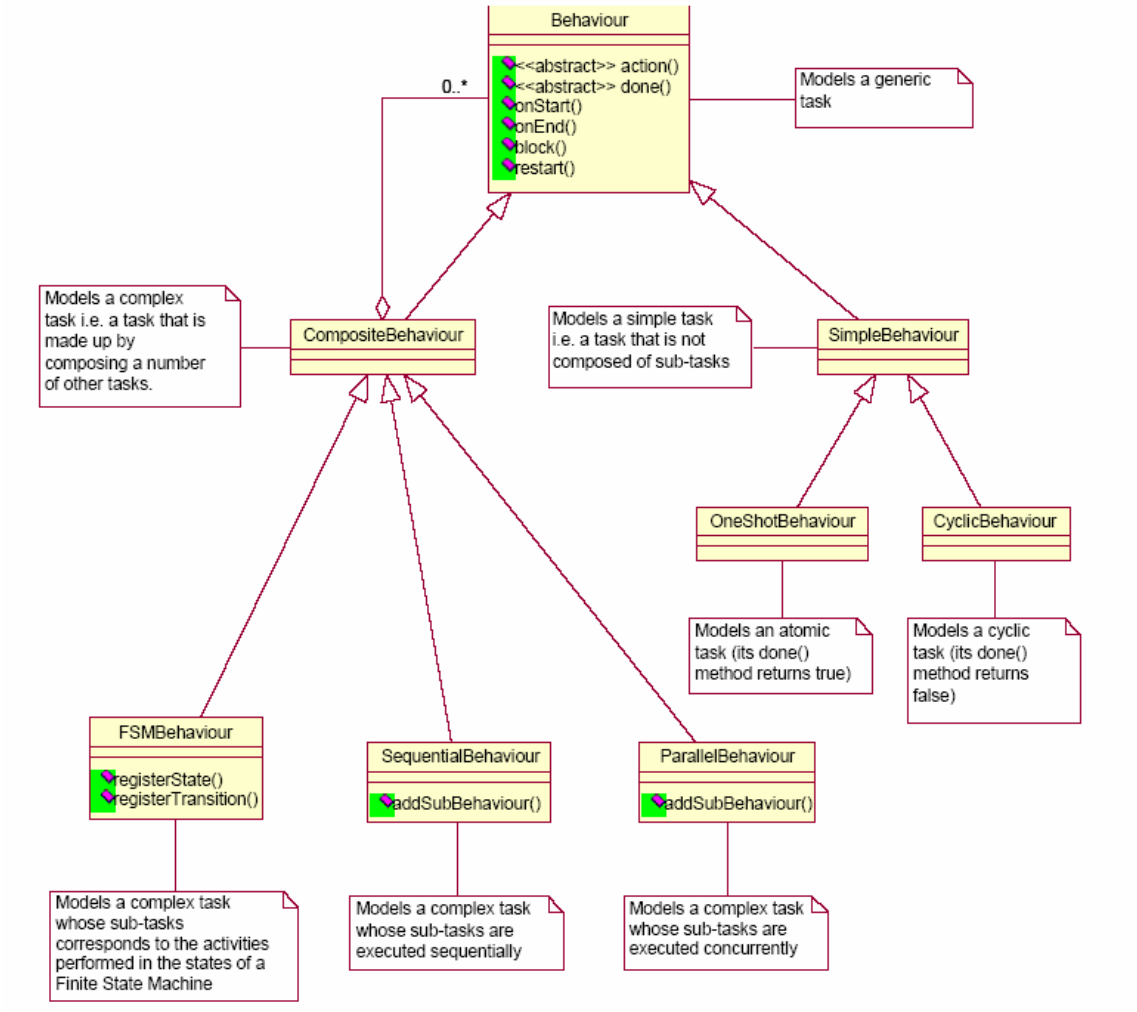


Figura 28: Model UML de la jerarquia de *behaviours* del JADE

A.4 Pas de missatges

En aquest apartat el JADE també segueix les especificacions de la FIPA. Cal recordar que el llenguatge de comunicació és l'ACL, que ha estat descrit a l'apartat 2.4.2.1.

Per poder crear i manipular el contingut dels missatges, el JADE proporciona la classe *jade.lang.acl.ACLMessage*. Per cada paràmetre del missatge tenim els corresponents mètodes *get* i *set* que ens permetran escriure o llegir el valor del paràmetre corresponent. Tanmateix, pel paràmetre *receiver* no tenim cap mètode *set*. Això és així perquè un missatge pot tenir múltiples receptors. Per afegir un destinatari 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()*. Aquest últim no és bloquejant i retorna l'últim missatge

de la cua de missatges si n'hi ha cap. Ambdós mètodes els proporciona la classe *jade.core.Agent*.

A.5 Els protocols de la 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-Request*, *FIPA-query*, *FIPA-Contract-Net*, etc. Aquest conjunt de classes les podem trobar al paquet *jade.proto*.

Cal remarcar que per cada protocol tenim dues classes que implementen els rols de l'agent iniciador i de l'agent participant. Els noms de les classes seran del tipus *xxxInitiator* i *xxxResponder*, on *xxx* dependrà del protocol en particular. Així, el JADE ens proporciona les classes:

- ***AchieveREInitiator* i *SimpleAchieveREInitiator:*** Implementen el rol de l'iniciador dels protocols *FIPA-Request*, *FIPA-query*, *FIPA-Request-When*, *FIPA-Recruiting* i *FIPA-brokering*.

- ***AchieveREResponder* i *SimpleAchieveREResponder:*** Implementen el rol del participant dels protocols *FIPA-Request*, *FIPA-query*, *FIPA-Request-When*, *FIPA-Recruiting* i *FIPA-brokering*.

- ***ContractNetInitiator:*** Implementa el protocol de l'iniciador del protocol *FIPA-Contract-Net*.

- ***ContractNetResponder:*** Implementa el protocol del participant del protocol *FIPA-Contract-Net*.

- ***ProposeInitiator:*** Implementa el protocol de l'iniciador del protocol *FIPA-Propose*.

- ***ProposeResponder:*** Implementa el protocol del participant del protocol *FIPA-Propose*.

- ***SubscriptionInitiator:*** Implementa el protocol de l'iniciador del protocol *FIPA-Subscribe*.

- ***SubscriptionResponder:*** Implementa el protocol del participant del protocol *FIPA-Subscribe*.

Per utilitzar-les només s'han de sobrecarregar els mètodes abstractes que representen els diferents passos del protocol. Així, per exemple, la classe *AchieveREInitiator* proporciona els mètodes *handleInformMessage()* i *handleFailureMessage()*, per gestionar la recepció de missatges del tipus *Inform* i *Failure* respectivament.

A.6 Suport pel llenguatge SL

Per assistir al programador en la creació de textos en llenguatge SL, el JADE proporciona una sèrie de classes que representen a les primitives del llenguatge. Mitjançant els mètodes *get* i *set* de cada un dels paràmetres d'aquestes primitives podem escriure o llegir qualsevol expressió. Així, per exemple, per construir el predicat *(= a b)* utilitzarem la classe *Equals* que representa a l'operador "=". Per especificar cada un dels membres de l'expressió utilitzarem les següents sentències:

```
Equals eq = new Equals();
eq.setLeft(a);
eq.setRight(b);
```

D'una forma semblant es faria per la resta de primitives. Aquestes classes les podem trobar al paquet *jade.content.onto.basic*.

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 corresponent al contingut del missatge del l'objecte *jade.lang.acl.ACLMessage* que li passem com a argument.

A.7 Ontologies definides per l'usuari

Un dels trets més característics dels SMA és l'ús d'ontologies específiques que enriqueixen la comunicació entre els agents. El JADE proporciona una sèrie de classes que permeten crear i treballar amb ontologies específiques.

Una ontologia en JADE és una instància de la classe *jade.content.onto.Ontology*, a la que se li poden associar esquemes que defineixen conceptes, predicats i accions. Aquests esquemes són instàncies de les classes *ConceptSchema*, *PredicateSchema* i *AgentActionSchema*, inclosos al paquet *jade.content.schema*. Aquestes classes tenen mètodes *get* i *set* mitjançant els quals és possible declarar els *slots* que defineixen l'estructura de cada concepte, predicat i acció.

Addicionalment, el JADE proporciona esquemes per:

- Els tipus primitius: *STRING*, *INTEGER*, *FLOAT*...
- Tipus agregats
- Alguns conceptes, predicats i accions genèrics que no pertanyen a cap domini específic.

De forma general:

- Cada esquema de l'ontologia està associat a una classe Java. Aquesta classe de Java té una sèrie de mètodes *get* i *set* que permeten accedir als *slots* definits per l'esquema.
- Cada *slot* d'un esquema té un nom i un tipus. El tipus pot ser un tipus primitiu o un altre esquema.

- Un *slot* pot ser opcional, és a dir, que pot valer *null*. Contràriament, un *slot* és considerat obligatori.

- Un *slot* pot tenir associada una cardinalitat superior a 1. Això significa que l'*slot* és un agregat d'elements d'un tipus determinat.

- Un esquema pot derivar d'un o més esquemes. Això permet definir relacions d'especialització o d'extensió entre esquemes.

Des de la versió 2.5 del JADE, el paquet *jade.content* permet crear ontologies específiques que puguin ser usades independentment del llenguatge de contingut emprat. És a dir, el codi que implementa la ontologia i el codi que envia i rep missatges no depèn del llenguatge emprat per escriure els missatges.

A.8 Eines del JADE

El JADE proporciona un conjunt d'eines d'administració i de depuració destinades a facilitar la implementació i el desplegament de SMA.

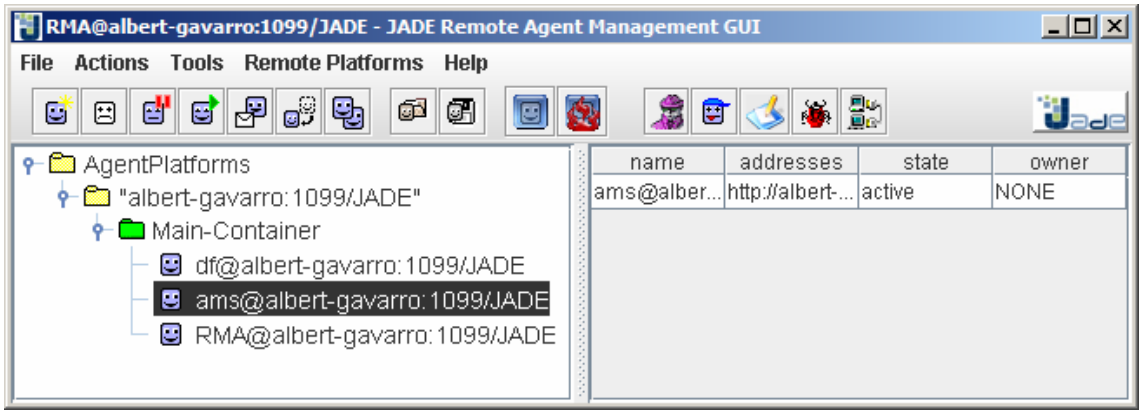


Figura 29: La GUI de l'agent RMA

L'agent RMA és la cara visible del sistema. A partir de la seva interfície es pot accedir a totes les facilitats que proporciona el JADE. A la Figura 29 es mostra l'aspecte que ofereix la seva GUI. A la banda esquerra es pot veure una vista en arbre de la plataforma, amb un llistat dels agents que s'estan executant.

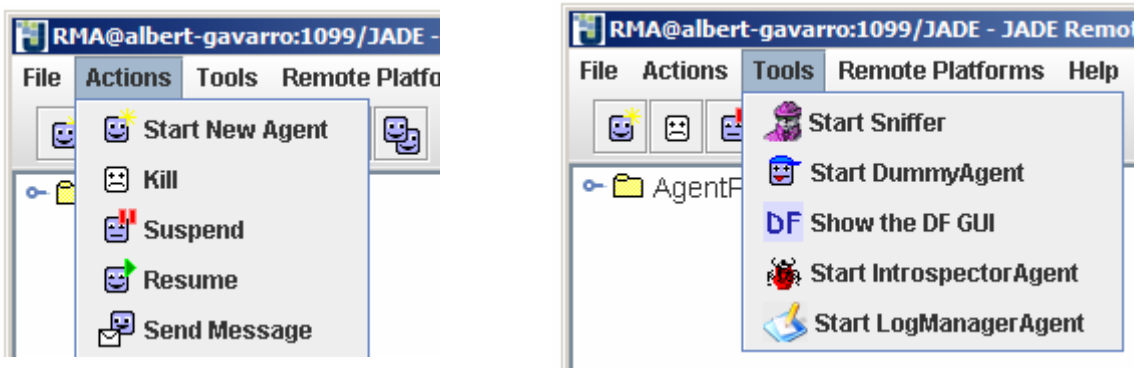


Figura 30: El menús *Actions* i *Tools* de l'RMA

El menú *Actions* (Figura 30) dóna accés a les següents funcions:

- ***Start New Agent:*** Permet afegir un nou agent al sistema. S'ha d'indicar el seu nom, la classe que l'implementa (ha d'estar carregada per la JVM) i el contenidor al que es vol confinar.

- ***Kill:*** Mata a un agent.

- ***Suspend:*** Suspèn l'execució d'un agent.

- **Resume:** Permet reprendre l'execució d'un agent que ha estat suspès prèviament.

- ***Send Message:*** Permet enviar un missatge a un agent. S'ha d'indicar el nom del destinatari i omplir els camps necessaris del missatge. La Figura 31 mostra la finestra de redacció de missatges ACL.

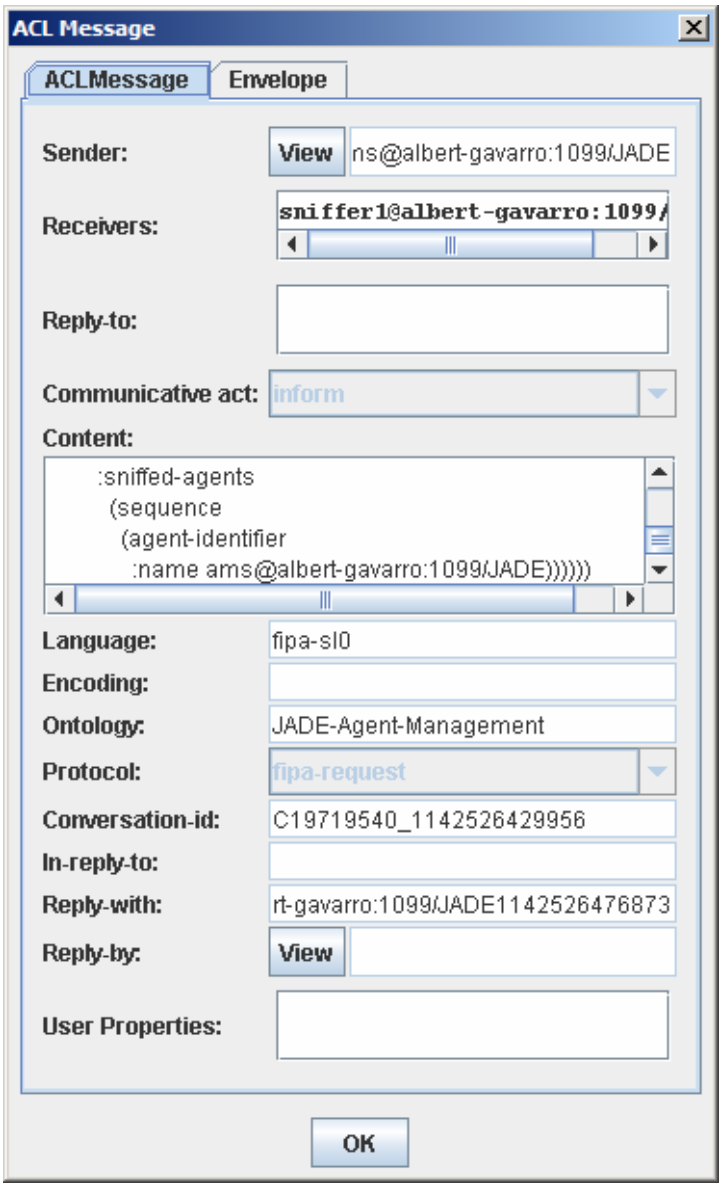


Figura 31: Finestra d'edició de missatges

El menú *Tools* (Figura 30) dóna accés a les següents funcions:

- ***Start Sniffer:*** Inicia una còpia de l'agent Sniffer. L'Sniffer permet monitoritzar el trànsit de missatges dins del SMA. Els resultats es presenten en forma de diagrama de flux on cada enviament es representa com una fletxa que va des de l'agent emissor fins al receptor. Si es fa doble clic sobre qualsevol d'aquestes fletxes es mostra fins a l'últim detall del missatge en una finestra com la de la Figura 31. Val a dir que el conjunt d'agents a monitoritzar és configurable des del menú *Actions*, opció *Add/Remove Agent*. La Figura 32 mostra l'agent *Sniffer* en plena acció.

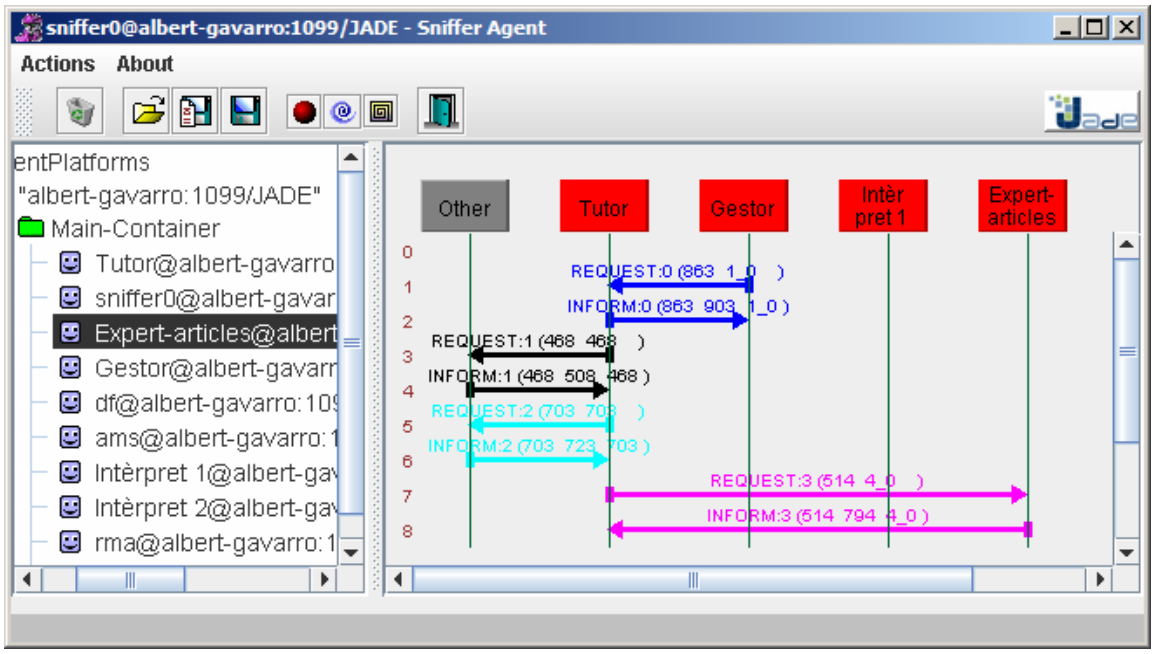


Figura 32: La GUI de l'agent *Sniffer*

- ***Start DummyAgent:*** Inicia l'agent *DummyAgent*. Aquest agent no fa res en particular però permet enviar i rebre missatges a la resta d'agents. És com qualsevol altre agent però està sota el control de l'usuari. Té alguna opció interessant com és la de salvar un missatge al disc. La Figura 33 mostra l'aspecte de la seva interfície.

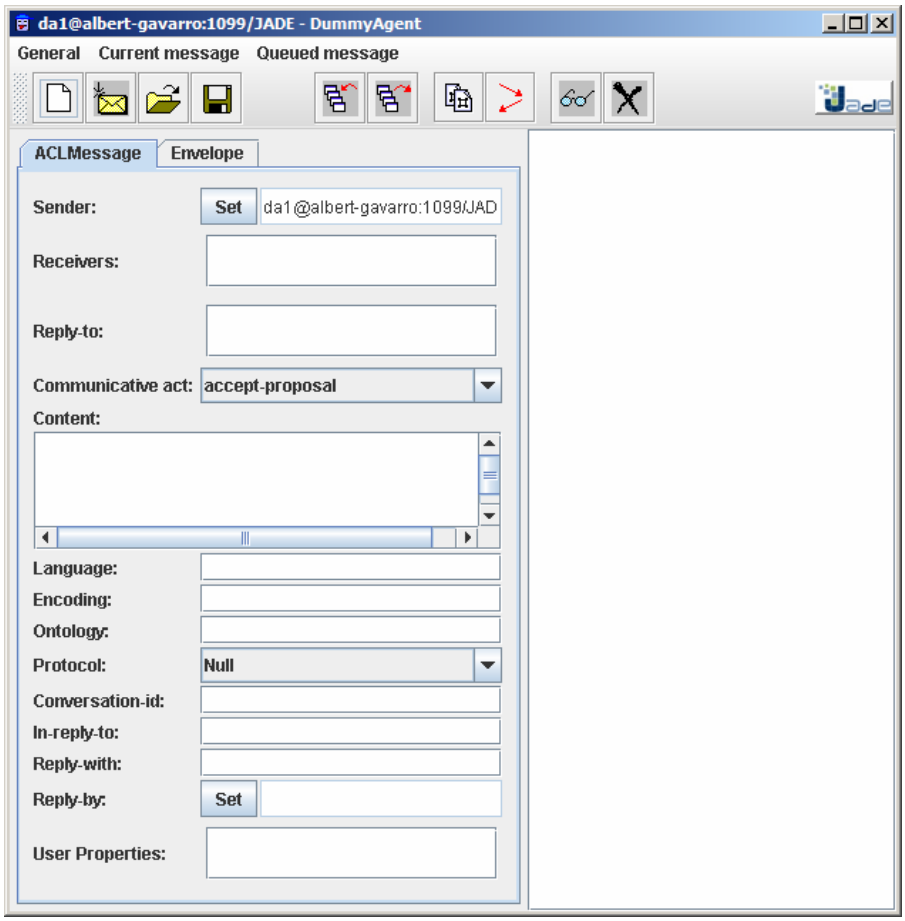


Figura 33: La GUI del *DummyAgent*

- ***Show the DF GUI:*** Mostra la GUI de l'agent DF. En ella es pot 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 es pot obtenir el nom, el tipus, les propietats, etc. També permet afegir o eliminar entrades del seu registre. La Figura 34 mostra la GUI del DF.

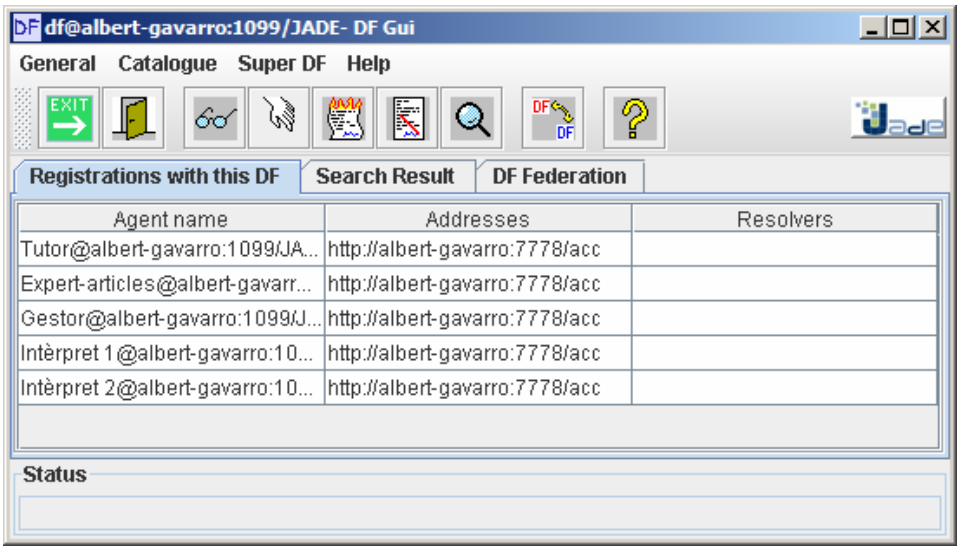


Figura 34: La GUI de l'agent DF

Apèndix B Jess

Jess[12] és una llibreria de programació escrita en Java. La llibreria en si és un intèrpret d'un altre llenguatge – el llenguatge del Jess – que serveix per construir sistemes experts basats en regles. El llenguatge del Jess és molt semblant al llenguatge definit pel CLIPS que, a la seva vegada, és una especialització del LISP.

En aquest apèndix es fa una breu introducció al llenguatge del Jess, i s'explica com utilitzar la llibreria des d'un programa escrit en Java. També es detallen els diferents mecanismes que el Jess proporciona per crear i depurar sistemes experts.

Tot i que no s'espera que el que aquí s'explica variï substancialment en properes versions del Jess, cal remarcar que aquest apèndix ha estat escrit basant-se en les característiques que proporciona la versió 5.1 (24/4/2000) de la llibreria.

Cal remarcar que el Jess és un producte comercial d'Ernest J. Friedman-Hill i Sandia Corporation, l'ús del qual està subjecte al pagament d'una llicència. Tanmateix, el Jess es pot obtenir gratuïtament per un ús no comercial que engloba les finalitats acadèmiques.

B.1 El llenguatge del Jess

El llenguatge del Jess és molt extens i ric en característiques. En aquest punt s'explica únicament com definir regles, fets i plantilles, i es detallen les primitives que ofereix el Jess per controlar el procés d'inferència.

B.1.1 Fonaments

Tot sistema expert escrit en Jess es basa en els següents components:

- **Àtoms:** Construeixen els elements del llenguatge. Un àtom de Jess pot contenir lletres, números i els següents símbols: $*=+/<>_?#. Hi ha tres àtoms predefinits pel llenguatge: *nil*, que és molt semblant al *null* de Java, i *TRUE* i *FALSE*, que són els valors booleans del Jess.

- **Números:** El Jess accepta números de coma flotant i enters. Tanmateix, no accepta la notació científica.

- **Cadenes:** Les cadenes de text del Jess es delimiten amb el caràcter de la doble cometa (“). Si es vol incloure el caràcter de la doble cometa dins d'una cadena, s'ha prefixar amb una contra barra (\). El Jess no accepta els caràcters d'escapament habituals de C i Java.

- **Llistes:** Una llista consisteix en un parell de parèntesis que emmarquen zero o més àtoms, números, cadenes de text o altres llistes. Per exemple:

```
(+ 3 2) (a b c) () (deftemplate nota (slot puntuacio))
```

[12] http://herzberg.ca.sandia.gov/jess

- **Comentaris:** Els comentaris en Jess comencen amb punt i coma (;) i s'estenen fins a final de línia.

B.1.2 Funcions

Com en el LISP, tot codi en Jess té la forma d'una crida a una funció. Les crides a una funció són simples llistes. Per exemple, la següent sentència:

```
(+ 3 2)
```

suma tres i dos.

B.1.3 Variables

Les variables en Jess comencen sempre amb el símbol d'interrogació (?). Una variable pot fer referència a un àtom, a un número a una cadena de text. Per assignar un valor a una variable s'utilitza la funció *bind*:

```
(bind ?x "el valor")
```

Les variables no necessiten ser declarades abans de la seva utilització.

Hi ha un tipus especial de variables que són les variables globals o *defglobals*. És un tipus de variable persistent entre diferents execucions del motor d'inferència. Per declarar-les s'utilitza la construcció *defglobal*:

```
(defglobal [?<nom-global> = <valor>]+)
```

Els noms de les variables globals han de començar i acabar en asterisc. Són variables globals vàlides:

```
?*a*    ?*contador*    ?*ultim-valor*
```

B.1.4 Declaració de funcions

El Jess permet definir funcions a través de la construcció *deffunction*. La seva sintaxi és la següent:

```
(deffunction <nom-funcio> [<comentari>] (<parametre>*)
       <expr>*
      [<valor-retorn>])
```

El valor de retorn de la funció es pot declarar de forma explícita – mitjançant la construcció *return* –, o de forma implícita, siguent l'última expressió avaluada el valor de retorn.

Un exemple de funció és:

```
(deffunction max (?a ?b)
      (if (> ?a ?b) then
            (return ?a)
      else
            (return ?b)))
```

Si s'opta per utilitzar el retorn de funció implícit, la mateixa funció es podria haver escrit:

```
(deffunction max (?a ?b)
      (if (> ?a ?b) then
            ?a
      else
            ?b))
```

B.1.5 Reflexió de Java

Una capacitat interessant que ofereix el Jess és la de poder manipular objectes de Java directament des del seu llenguatge. Es pot fer gairebé de tot, excepte crear noves classes. A continuació es mostra un exemple en que es crea una taula *Hash* (*Hashtable*) i s'afegeixen unes quantes cadenes dins d'ella:

```
(bind ?ht (new java.util.Hashtable))
(call ?ht put "clau1" "element1")
(call ?ht put "clau2" "element2")
```

El Jess també permet accedir als atributs dels objectes de Java mitjançant les funcions *set-member* i *get-member*:

```
(bind ?pt (new java.awt.Point))
(set-member ?pt x 37)
(set-member ?pt y 42)
(printout t "x = " (get-member ?pt x) crlf)
```

Per accedir als membres estàtics de les classes s'ha d'utilitzar el nom de la classe en comptes del objecte en si com a primer paràmetre d'aquestes funcions:

```
(get-member System out)
```

La Taula 27 i la Taula 28 mostren com el Jess tradueix els valors des de Java a Jess i viceversa.

Tipus de Java	Tipus de Jess
Una referència *null*	L'àtom *nil*
Un valor de retorn *void*	L'àtom *nil*
String	*RU.STRING*
Un vector	Un *multifield* de Jess
boolean o java.lang.Boolean	Els àtoms *TRUE* i *FALSE*
byte, short, int, o els seus wrappers	*RU.INTEGER*
long o java.lang.Long	*RU.LONG*
double, float o els seus wrappers	*RU.FLOAT*
char o java.lang.Character	*RU.ATOM*
Qualsevol altra cosa	*RU.EXTERNAL_ADDRESS*

Taula 27: Conversió entre els tipus de Java i els tipus de Jess

Tipus de Jess	Tipus de Java possibles
RU.EXTERNAL_ADDRESS	L'objecte representat
L'àtom *nil*	El *null*
Els àtoms *TRUE* o *FALSE*	*java.lang.Boolean* o *boolean*

Tipus de Jess	Tipus de Java possibles
RU.ATOM, *RU.STRING*	*String*, *char*, *java.lang.Character*
RU.FLOAT	*float, double* i els seus *wrappers*
RU.INTEGER	*long*, *short*, *int*, *byte*, *char* i els seus *wrappers*
RU.LONG	*long*, *short*, *int*, *byte*, *char* i els seus *wrappers*
RU.LIST	Un vector

Taula 28: Conversió entre els tipus de Jess i els tipus de Java

B.1.6 Els *facts*

Un sistema basat en regles manté una col·lecció de bocins de coneixement anomenats *facts*. Aquesta col·lecció és coneguda com a base de coneixement. En Jess hi ha tres tipus de *facts*: els *ordered facts*, els *unordered facts* i els *definstance facts*.

B.1.6.1 *Ordered facts*

Els *ordered facts* són simples llistes, on el primer element és la categoria del *fact*. Els *ordered facts* s'afegeixen a la base de coneixement mitjançant la comanda *assert*. Per exemple, són *ordered facts*:

```
(llista-de-la-compra pa tomaquet oli)
(persona "Anna Musté" Dona 26)
(mare-de pau teresa)
```

Per afegir un *ordered fact* a la base de coneixement:

```
(assert (mare-de pau teresa))
```

B.1.6.2 *Unordered facts*

Els *ordered facts* són clars i molt fàcils d'escriure. Tanmateix, estan desestructurats. Els *unordered facts* són molt semblants als objectes de la programació orientada a objectes. Tenen una sèrie d'atributs, anomenats *slots*, que es diferencien pel seu nom. Per exemple:

```
(persona (nom "Anna Musté") (edat 26) (sexe Dona))
(automobil (marca Ford) (model Fiesta) (any 1982))
```

Abans de poder crear *unordered facts*, cal definir la seva estructura mitjançant la construcció *deftemplate*:

```
(deftemplate <deftemplate-name> [extends <classname>]
                                [<doc-comment>]
             [(slot <slot-name> [(default |
                                  default-dynamic <value>)]
                                [(type <typespec>)]*)
```

<*deftemplate-name*> és la capçalera que han de portar tots els *facts* creats utilitzant aquest *deftemplate*. Poden definir-se un nombre arbitrari d'*slots*. El qualificador *default* especifica el valor per defecte de l'*slot*. Si s'utilitza el modificador *default-dynamic*, el valor especificat serà avaluat cada cop que s'afegeixi un *fact* que utilitzi aquest *deftemplate* a la base de coneixement. Si no s'especifica un valor per defecte aquest serà

nil (sense valor). El tipus de l'*slot* es defineix amb el modificador *type*. Els diferents tipus acceptats són: *ANY*, *INTEGER*, *FLOAT*, *NUMBER*, *ATOM*, *STRING*, *LEXEME*, i *OBJECT*.

Per exemple, un *deftemplate* vàlid és:

```
(deftemplate automobil "Un cotxe determinat."
      (slot marca)
      (slot model)
      (slot any (type INTEGER))
      (slot color (default white)))
```

Per afegir un *unordered fact* a la base de coneixement:

```
(assert (automobil (marca Rover) (model 25) (any 2001)))
```

Cal remarcar que el cotxe serà de color blanc de forma predeterminada. Si no s'especifica un valor predeterminat per un *slot* i, a més a més, no es proporciona cap valor per aquest *slot* quan el *fact* és afegit a la base coneixement, l'*slot* valdrà *nil*.

Normalment, els *slots* d'un *detemplate* només poden contenir un valor. Si es desitja que puguin contenir múltiples valors, s'ha d'utilitzar la paraula clau *multislot* en comptes d'*slot*. El valor per defecte d'un *multislot* és la llista buida ().

Finalment, el modificador *extends* permet definir un *defetemplate* en termes d'un altre. Com en programació orientada a objectes, el *deftemplate* estès hereta els *slots* del *deftemplate* que estén. Per exemple:

```
(deftemplate cotxe-usat extends automobil
      (slot quilometratge)
      (multislot propietaris))
```

Així doncs, el *deftemplate cotxe-usat* tindrà els mateixos *slots* que el *deftemplate automobil* i dos més: el *quilometratge* i els *propietaris*. A més a més, un *cotxe-usat* serà un *automobil* a tots els efectes; Es a dir, que es podrà actuar sobre tots els *deftemplates automobil* (incloent tots els *cotxe-usat*) o només sobre els *cotxe-usat*.

Per simplificar la inserció de *facts* a la base de coneixement el Jess proporciona la construcció *deffacts*. *Deffacts* no és res més que una col·lecció de *facts* amb nom que s'afegirà a la base de coneixement cada cop que s'iniciï el procés d'inferència:

Aquest és un exemple d'utilització de la construcció *deffacts*:

```
(deffacts els-meus-fets ""
      (mare-de pau teresa)
      (caixa (localitzacio garatge) (contingut tisores paper
                                     cola))
      (cotxe-usat (any 1982) (marca Ford) (model Fiesta)
                  (quilometratge 120000) (propietaris albert)))
```

B.1.6.3 *Definstance facts*

Els *definstance facts* són potser el recurs innovador que ofereix el Jess. Els *definstance facts* exploten l'estreta relació que hi ha entre el llenguatge de programació Java i l'intèrpret de Jess.

Es pot dir que els *definstance facts* són una representació de les propietats d'un *Bean* de Java dins de la base de coneixement. A més a més, la representació del *Bean* pot ser estàtica (quan emmagatzema les propietats que tenia el *Bean* en un determinat instant del temps) o dinàmiques (quan la seva representació s'actualitza cada cop que canvia el *Bean*). Les comandes del Jess que permeten treballar amb aquest tipus de *facts* són la *defclass* i la *definstance*.

Suposem que tenim la classe *BeanExemple*:

```
public class BeanExemple {
      private String elMeuNom = "Albert";
      public String getNom() { return elMeuNom; }
      public void setNom(String nom) { elMeuNom = nom; }
}
```

Com es pot apreciar, aquest *Bean* només té una propietat: el nom. Abans de poder inserir aquest *Bean* a la base de coneixement, cal crear un *template* que el representi. Per fer-ho, s'utilitza la construcció *defclass*:

```
(defclass template-bean-exemple BeanExemple)
```

Per afegir una instància de la classe *BeanExemple* a la base de coneixement s'utilitza la comanda *definstance*:

```
(definstance template-bean-exemple ?i static)
```

on *?i* és una referència a la instància. Com que s'ha definit com a estàtica, l'atribut *nom* del *fact* només s'actualitzarà quan es reiniciï la base de coneixement. Si es desitja una actualització constant, s'ha d'utilitzar el modificador *dynamic*. Això exigeix que el *Bean* permeti ser escoltat a través de *java.beans.PropertyChangeListeners*.

B.1.7 Les regles

La base de coneixement no té cap utilitat si no podem prendre accions en conseqüència amb el seu contingut. Això mateix, doncs, és el que fan les regles.

B.1.7.1 *Defrules*

Una regla del Jess és quelcom semblant a un *if ... then* d'un llenguatge procedural. Tanmateix, hi ha una diferència substancial: mentre que en un llenguatge procedural el blocs *if ... then* s'executen en un ordre determinat, en Jess les regles s'executen quan les seves precondicions es satisfan. Aquestes precondicions les comprova i les gestiona el motor de regles mentre s'executa.

Les regles es generen mitjançant la construcció *defrule*:

```
(defrule regla-desplegar-paraigues
         "si plou, desplegar el paraigues"
      (plou)
      =>
      (desplegar-paraigues))
```

Les regles tenen dues parts separades pel símbol "=>": la part de les precondicions *(plou)* i la part de les accions *(desplegar-paraigues)*. La part de les precondicions

consisteix en una sèrie de patrons que seran comparats amb els *facts* de la base de coneixement. Si tots els patrons es satisfan, la regla s'executarà.

La part de les precondicions accepta una sèrie de construccions molt flexibles que doten d'una gran potència al Jess. Els patrons poden incloure comodins i diversos tipus de predicats (comparacions i funcions booleanes).

B.1.7.2 Patrons bàsics

Es poden especificar noms de variables (en comptes d'un valor) per qualsevol *slot* d'un *fact*. Per exemple, la regla:

```
(defrule exemple
      (a ?x ?y)
      =>
      (printout t "a " ?x " " ?y crlf))
```

s'activarà per tots els *facts* que comencin per *a* i que tinguin dos camps qualsevol. Per exemple: *(a b c)*, *(a 1 2)*, etc.

A més a més, cada variable d'un patró pot sotmetre's a una sèrie de tests per determinar quins *facts* satisfaran el patró i quins no. Aquests tests es poden concatenar amb *is* (&) i *os* (|) lògiques. Els tests poden ser:

- Valors literals. L'*slot* només podrà ser igual a un valor:

  ```
  (a b c)
  ```

- Una altra variable. L'*slot* es compararà amb el valor d'una altra variable:

  ```
  (a ?X ?Y)
  ```

- Dos punts (:) seguits per la crida d'una funció. En aquest cas el test tindrà èxit si la funció retorna el valor *TRUE*. Aquest tipus de tests són anomenats *predicate constraints*. Per exemple:

  ```
  (a ?X&:(> ?X 10))
  ```

  serà satisfet pels *facts* de tipus *a* amb un sol *slot* més gran que 10.

- Un signe igual (=) seguit per la crida d'una funció. En aquest cas, l'*slot* ha de coincidir amb el valor de retorn de la funció. Aquests tests són anomenats *return value constraints*. Per exemple:

  ```
  (a ?X=(+ ?X 1))
  ```

  serà satisfet pels *facts* de tipus *a* amb dos *slots* numèrics, el segon dels quals és igual al primer més u.

- Qualsevol dels tests exposats fins ara precedits per "~". L'operador "~" inverteix la lògica de la funció (desigualtat o falsedat). Per exemple:

  ```
  (a ?X ~?X)
  ```

es satisfet per *facts* del tipus *a* amb dos *slots* diferents.

B.1.7.3 Captura de *facts*

De vegades, pot ser necessari obtenir una referència a un dels *facts* que ha activat la regla. Per fer-ho, s'utilitza l'operador "<-":

```
(defrule exemple
      ?fact <- (a ?x)
      =>
      (retract ?fact))
```

Aquesta regla fa un *retract* (esborra de la base de coneixement) els *facts* del tipus *a* amb un sol *slot*.

B.1.7.4 Resolució de conflictes amb *salience*

Tota regla té una propietat anomenada *salience*, que és una mena de prioritat de la regla. Quan dues o més regles es poden activar alhora, s'activaran primer les que tinguin el *salience* més gran. Per exemple:

```
(defrule exemple
      (declare (salience -100))
      (comanda sortir-quan-no-faci-res)
      =>
      (printout t "Sortint..." crlf))
```

B.1.7.5 Patrons *not*

Un patró pot anar precedit d'un *not*. En aquest cas, el patró esdevindrà cert si no hi ha cap *fact* que el satisfaci. Per exemple:

```
(defrule exemple
      (persona ?x)
      (not (casada ?x))
      =>
      (printout t ?x " no està casada" crlf))
```

B.1.7.6 L'element condicional *test*

Un patró precedit de *test* és especial: el cos no consisteix en un patró que s'hagi de comparar amb els *facts* de la base de coneixement, sinó en una funció booleana el valor de retorn de la qual determina si el patró es satisfet o no. Per exemple:

```
(defrule exemple
      (persona (edat ?x))
      (test (> ?x 30))
      =>
      (printout t ?x " té més de 30 anys" crlf))
```

B.1.7.7 L'element condicional *unique*

Quan un patró va precedit de l'element *unique*, el patró només pot ser satisfet per un i només un *fact*. Per exemple:

```
(defrule guanyador
      (unique (participant (nom ?x))
      =>
      (printout t ?x " ha guanyat el joc" crlf))
```

B.1.7.8 L'element condicional *exists*

Quan un patró va precedit per un *exists*, l'element condicional és cert si hi ha algun *fact* que satisfaci el patró. L'*exists* és útil quan es vol disparar una regla un únic cop, tot i haver-hi molts *facts* que la poden activar. Per exemple:

```
(defrule exemple
      (exists (honest ?x))
      =>
      (printout t ?x " és honest" crlf))
```

B.1.8 Control del motor d'inferència

Hi ha tres funcions essencials que permeten controlar l'execució del motor d'inferència:

- ***reset*:** Esborra tots els *facts* de la llista de *facts* i esborra totes les activacions. Llavors afegeix a la llista de *facts* l'*initial-fact* i tots els *facts* dels *deffacts* i dels *definstance*, i (si la variable *set-reset-globals* és igual a *TRUE*) inicialitza tots els *defglobals*.

- ***clear*:** Neteja l'entorn d'execució del Jess. Esborra totes les regles, els *deffacts*, els *defglobals*, els *deftemplates*, els *facts* i les activacions. Les funcions d'usuari escrites en Java no s'esborren.

- ***run*:** Inicia el motor d'inferència. Si no s'especifica cap argument, el Jess s'executarà fins que no quedin més activacions o fins que es cridi a la funció *halt*. Si es proporciona un argument enter, el Jess s'aturarà després d'haver disparat tantes regles com les indicades per paràmetre.

B.2 Jess i Java

En aquesta secció s'explica com utilitzar el Java per estendre les funcionalitats de Jess, i com utilitzar l'intèrpret de Jess des de Java.

B.2.1 La classe *jess.JessException*

Aquest és l'únic tipus d'excepció que generen les funcions del Jess. A banda del típic missatge d'error, una instància d'aquesta classe conté tot tipus d'informació sobre l'error: el nom de la rutina que l'ha provocat, el nom de les construccions de Jess que hi havia a la pila en el moment de la fallida, el codi que s'estava executant i el seu número de línia, etc.

B.2.2 La classe *jess.Value*

La classe *jess.Value* és, probablement, la més utilitzada quan es volen intercanviar dades amb el Jess. Representa a tot àtom del Jess i és auto descriptiva: la funció *type()* proporciona informació sobre el tipus de dades que conté. La funció *type()* retorna un dels valors següents (definits a la classe *jess.RU*):

- ***NONE*:** un valor buit, no *nil*.

- ***ATOM*:** un símbol.

- ***STRING*:** una cadena de text.
- ***INTEGER*:** un enter.
- ***VARIABLE*:** una variable, per exemple, *?x*.
- ***FACT_ID*:** un índex d'un *fact*.
- ***FLOAT*:** un valor real.
- ***FUNCALL*:** una crida a una funció.
- ***FACT*:** un *fact*.
- ***LIST*:** una llista.
- ***DESCRIPTOR*:** valor reservat per ús intern del Jess.
- ***EXTERNAL_ADDRESS*:** una referència a un objecte Java.
- ***INTARRAY*:** valor reservat per ús intern del Jess.
- ***MULTIVARIABLE***: una variable múltiple.
- ***SLOT*:** valor reservat per ús intern del Jess.
- ***MULTISLOT*:** valor reservat per ús intern del Jess.
- ***LONG*:** un valor corresponent a un valor de tipus *long* de Java.

Els objectes *jess.Value* es construeixen especificant les dades i, usualment, el tipus. El Jess proporciona tota una sèrie de constructors per satisfer aquest propòsit:

```
public Value(Object o) throws JessException
public Value(String s, int type) throws JessException
public Value(Value v)
public Value(ValueVector f, int type) throws JessException
public Value(double d, int type) throws JessException
public Value(int value, int type) throws JessException
```

De forma semblant, la classe *jess.Value* té un seguit de mètodes per accedir a les dades que conté:

```
public Object externalAddressValue(Context c) throws
JessException
public String stringValue(Context c) throws JessException
public ValueVector factValue(Context c) throws JessException
public ValueVector funcallValue(Context c) throws JessException
public ValueVector listValue(Context c) throws JessException
public double floatValue(Context c) throws JessException
public double numericValue(Context c) throws JessException
public int descriptorValue(Context c) throws JessException
public int factIDValue(Context c) throws JessException
public int intValue(Context c) throws JessException
```

La classe *jess.Value* té tres subclasses:

- ***jess.Variable*:** que representa a una variable.
- ***jess.FuncallValue*:** que representa a una crida a una funció.
- ***jess.LongValue*:** que representa a un valor *long* de Java.

Quan es vol crear un objecte *jess.Value* que representi a un d'aquests tres tipus de dades, cal utilitzar la subclasse corresponent.

B.2.3 La classe jess.Context

Tot objecte *jess.Value* necessita estar "resolt" abans del seu ús. Això vol dir que cal conèixer-ne el valor. Si bé conèixer el valor d'un valor estàtic és trivial (àtoms, números, cadenes de text), en el cas dels valors dinàmics (variables, crides a funcions) aquest valor ha de ser interpretat en el seu context.

Per exemple, com s'ha vist abans, la classe *jess.Value* té el mètode *intValue()*. Si aquest objecte representa a una funció que retorna un enter, aquesta funció s'executa en el context passat com a paràmetre abans que la funció *intValue()* retorni el resultat.

La classe *jess.Context* representa, doncs, un context d'avaluació de crides a funció o de resolució de variables. L'objecte *jess.Context* no necessita ser creat explícitament, doncs s'obté, generalment, a través d'altres classes del Jess.

B.2.4 La classe *jess.Rete*

La classe *jess.Rete* és el motor de regles en si. Cada instància d'aquest objecte és un motor de regles independent i completament funcional, amb la seva pròpia base de coneixement, agenda, regles, etc.

La classe *jess.Rete* proporciona una sèrie de mètodes que donen accés a les operacions més habituals. Els més destacats són:

- ***run()*:** Inicia el motor d'inferència. Opcionalment pot rebre un paràmetre que indica el nombre de regles que es poden disparar abans que s'aturi el procés. Equival a la funció *run* del llenguatge del Jess.
- ***reset()*:** Esborra tots els *facts* de la llista de *facts* i esborra totes les activacions. Llavors afegeix a la llista de *facts* l'*initial-fact* i tots els *facts* dels *deffacts* i dels *definstance*, i (si la variable *set-reset-globals* és igual a *TRUE*) inicialitza tots els *defglobals*. Equival a la funció *reset* del llenguatge del Jess.
- ***clear()*:** Neteja l'entorn d'execució del Jess. Esborra totes les regles, els *deffacts*, els *defglobals*, els *deftemplates*, els *facts* i les activacions. Equival a la funció *clear* del llenguatge del Jess.
- ***executeCommand()*:** Executa un tros de codi escrit en el llenguatge del Jess al motor de regles actual i retorna l'objecte *jess.Value* resultant. És el mètode preferit per treballar amb el Jess perquè ofereix un accés total sobre el seu llenguatge.

Addicionalment, la classe *jess.Rete* proporciona mètodes per accedir a altres funcions no menys importants: *assert*, *retract*, *defrule*, *deffacts*, etc. Tanmateix, no totes les capacitats del llenguatge estan cobertes, ni les que ho estan ho estan completament. Per aquests casos el mètode *executeCommand()* és la millor alternativa.

B.2.5 Intercanvi de dades entre Jess i Java

El Jess proporciona un mètode molt senzill d'intercanvi de dades amb el Java. La classe *jess.Rete* ofereix els mètodes:

```
public Value store(String name, Value val);
public Value store(String name, Object val);
public Value fetch(String name);
public void clearStorage();
```

que tenen el seu homòleg en el llenguatge del Jess:

```
(store <name> <value>)
(fetch <name>)
(clear-storage)
```

L'ús d'aquest mètodes per l'intercanvi de dades és trivial: els mètodes *store* permeten guardar un valor amb un nom determinat, mentre que els mètodes *fetch* permeten llegir-lo.

B.2.6 Funcions d'usuari

Quan amb les funcions que proporciona el Jess no n'hi ha prou, el llenguatge pot ser ampliat – de forma fàcil – amb noves funcions escrites en Java. Només cal crear una nova classe que implementi la interfície *jess.Userfunction*. Aquesta interfície defineix dos mètodes:

- ***getName()*:** Aquest mètode ha de retornar el nom de la funció.
- ***call()*:** Aquest mètode s'executa cada cop que es crida a la funció i ha de retornar un objecte del tipus *jess.Value* amb el resultat de l'execució de la mateixa. Rep com a arguments el context actual i un vector amb els paràmetres amb els que s'ha cridat a la funció.

A la Figura 35 es mostra un exemple d'una funció d'usuari. La nova funció, que s'anomena *Majuscules* rep una cadena de text com a argument i la retorna en majúscules.

```
import jess.*;

public class FuncioMajuscules implements Userfunction {

      public String getName() {
            return "Majuscules";
      }

      public Value call(ValueVector vv, Context context)
            throws JessException {
            return new Value(vv.get(i).stringValue(context).
                  toUpperCase(), RU.STRING);
      }

}
```

Figura 35: Exemple d'una funció d'usuari

B.3 La consola del Jess

El Jess es pot utilitzar com a intèrpret del seu propi llenguatge sense haver-lo d'integrar en una altra aplicació. En aquest sentit, disposa d'un intèrpret de comandes propi molt semblant al del CLIPS, disponible tant des de la línia de comandes com a través d'una senzilla interfície gràfica. Cal remarcar que és una eina molt útil a l'hora de depurar sistemes experts existents.

Per inicar l'intèrpret des de la línia de comandes, cal executar la comanda:

```
$ java -cp jess.jar jess.Main
```

Per iniciar-lo amb la interfície gàfica:

```
$ javaw.exe -cp jess.jar jess.Console
```

A la Figura 36 es mostra l'aspecte que ofereix la interfície gràfica de l'intèrpret de comandes del Jess.

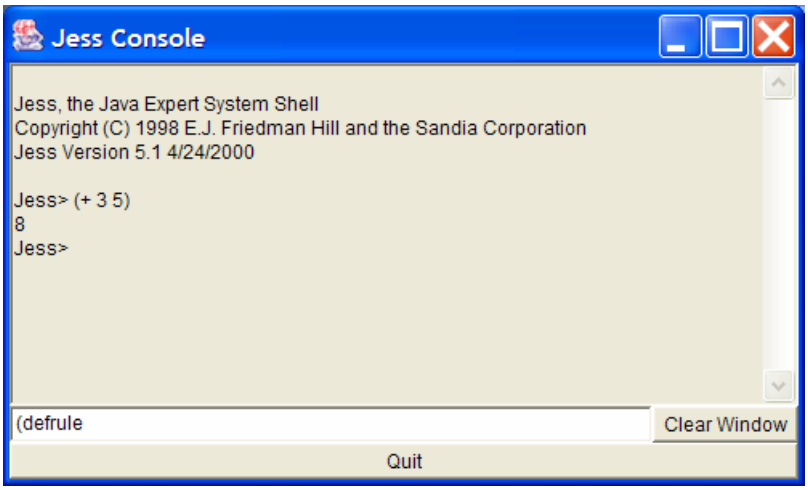


Figura 36: Aspecte de la cònsola gràfica del Jess

Apèndix C Dades d'exemple

Plantilla

```
1  subject:Q:met,app
2  research:Q:preliminary,mature,completed
3  relevance:Q:no,somewhat,quite,very
4  agents:Q:no,doubts,arguable,yes
5  MAS-desc:Q:bad,normal,well:NSNC
6  originality:Q:not,mostly-not,somewhat,mostly,very
7  soundness:Q:no,somewhat,yes:NSNC
8  tech-limits:Q:not-discussed,poorly,briefly,adequately
9  approach:Q:not-discussed,poorly,briefly,adequately
10 app-desc:N:1,5:NSNC
11 app-method:N:1,5:NSNC
12 met-desc:N:1,5:NSNC
13 met-application:N:1,5:NSNC
14 abstract:N:1,7:NSNC
15 introd:N:1,7:NSNC
16 conclusions:N:1,7:NSNC
17 organisation:N:1,7:NSNC
18 readability:N:1,7:NSNC
19 figures:N:1,7:NSNC
20 english:Q:deficient,typogramm,gramm,typo,correct:NSNC
21 references:Q:poor,basic,old,complete
22 overall:CLASSE:not-accepted,doubts,accept-with-modif,
   accept-few-modif,def-accepted
```

Instàncies

```
1  app completed no no NSNC not somewhat poorly briefly 2 2 NSNC NSNC
   1 5 4 3 5 5 typogramm poor not-accepted
2  met mature quite arguable bad very somewhat briefly briefly 3 2 4
   3 6 5 2 6 4 5 correct complete accept-with-modif
3  met mature quite yes normal somewhat yes briefly poorly 4 2 NSNC
   NSNC 7 7 3 5 6 4 typogramm old accept-few-modif
4  app preliminary no doubts NSNC mostly-not somewhat briefly poorly
   4 2 NSNC NSNC 5 4 4 4 5 4 typo basic doubts
5  met preliminary very arguable normal somewhat somewhat briefly
   briefly NSNC NSNC 4 3 6 6 4 3 3 5 correct complete
   accept-with-modif
6  app preliminary quite arguable bad somewhat somewhat poorly
   briefly 4 2.5 NSNC NSNC 6 5 3 5 5 5 typo old
   accept-with-modif
7  app preliminary very yes normal somewhat somewhat poorly
   not-discussed 5 3 NSNC NSNC 6 6 3 5 5 5 gramm old doubts
8  app completed no no NSNC not no not-discussed briefly 4 2 NSNC
   NSNC NSNC NSNC NSNC NSNC NSNC NSNC correct complete not-accepted
9  app mature very arguable bad mostly yes not-discussed briefly 4 3
   NSNC NSNC 6 6 5 4 5 5 typogramm old accept-with-modif
10 app mature very yes normal mostly yes not-discussed adequately 5 4
   NSNC NSNC 7 6 6 6 7 6 typogramm complete accept-few-modif
11 met mature quite arguable bad mostly yes poorly briefly NSNC NSNC
   4 2 6 6 6 7 7 7 typo complete accept-few-modif
12 app preliminary somewhat arguable normal somewhat somewhat briefly
   briefly 4 2 NSNC NSNC 6 6 5 6 6 5 gramm old accept-with-modif
13 met preliminary somewhat yes well mostly yes adequately
   not-discussed NSNC NSNC 5 2 6 5 5 5 4 5 correct complete doubts
14 app mature somewhat doubts bad somewhat NSNC poorly briefly 5 4
```

Instàncies	
	NSNC NSNC 5 6 7 6 5 3 correct complete accept-with-modif
15	app preliminary quite arguable bad somewhat yes poorly not-discussed 2 1 NSNC NSNC 7 6 5 2 2 1 NSNC complete accept-with-modif
16	met mature quite arguable bad mostly somewhat briefly briefly NSNC NSNC 3 3 4 4 5 5 4 4 typo old accept-few-modif
17	app preliminary somewhat doubts NSNC not yes adequately poorly 1 3 NSNC NSNC 5 6 7 7 7 7 correct basic not-accepted
18	met mature very yes well very yes adequately adequately 5 4 5 5 7 7 7 7 6 5 correct complete def-accepted
19	met mature quite arguable normal mostly yes adequately briefly 5 3 3 3 6 6 4 6 6 5 typo old accept-few-modif
20	met preliminary somewhat arguable bad mostly-not no poorly poorly NSNC NSNC 2 2 2 3 4 4 5 3 correct basic not-accepted
21	met mature quite arguable normal mostly yes briefly briefly NSNC NSNC 5 4 7 7 7 7 7 7 typo complete accept-few-modif
22	app mature quite yes well mostly yes briefly adequately 4 5 NSNC NSNC 5 4 6 3 5 6 correct complete accept-few-modif

Apèndix D Fitxers del joc de proves

Les dades que figuren en aquest apèndix han estat generades manualment per poder realitzar el joc de proves.

D.1 Bolets

bolets.plantilla

```
color-barret:Q:blanc,taronja,vermell,gris,blau,verd
cartes:Q:blanc,taronja,vermell,avellana,blau,no-en-te
olor:Q:peix,avellana,farina,fungica:nil
te-barret-esquamos:B:nil
te-anell:B:nil
alçada:N:0,40:nil
latex:Q:blanc,taronja,vermell,no-en-te:nil
bolet:CLASSE:lactarius-vinosus,lactarius-deliciosus,
amanita-phalloides,amanita-muscaria,macrolepiota-procera,
lactarius-indigo
```

bolets.instancies

```
vermell blanc nil nil CERT 15 nil amanita-muscaria
blanc blanc avellana nil CERT 25 no-en-te macrolepiota-procera
vermell blanc nil nil CERT 12 nil amanita-muscaria
verd blanc farina FALS FALS 10 no-en-te amanita-phalloides
taronja taronja fungica FALS FALS nil taronja lactarius-deliciosus
blau blau nil nil FALS 13 nil lactarius-indigo
taronja taronja fungica nil FALS 3 vermell lactarius-vinosus
blanc avellana nil CERT CERT 35 nil macrolepiota-procera
blanc blanc nil nil nil 12 nil amanita-phalloides
taronja taronja nil nil nil 7 nil lactarius-deliciosus
```

ExpertBolets1.java

```
import net.urv.etse.ninots.agents.expert.AgentExpert;
import net.urv.etse.ninots.agents.comu.*;
import net.urv.etse.ninots.agents.expert.regles.*;

public class ExpertBolets1 extends AgentExpert {

    public Regla[] extreureConeixement(Instancia[] instancies) {
        AtributCategoric colorBarret;
        AtributCategoric cartes;
        AtributCategoric olor;
        AtributBoolea teBarretEsquamos;
        Plantilla plantilla;
        ExpressioBooleana exps[];
        Regla regles[];

        plantilla = instancies[0].obtPlantilla();
        regles = new Regla[4];
        try {
            colorBarret = (AtributCategoric)
                    plantilla.obtAtribut("color-barret");
            cartes = (AtributCategoric)
                    plantilla.obtAtribut("cartes");
```

ExpertBolets1.java

```
23              olor = (AtributCategoric)plantilla.obtAtribut("olor");
24              teBarretEsquamos = (AtributBoolea)
25                      plantilla.obtAtribut("te-barret-esquamos");
26
27              // Regla 1: si(color-barret=vermell i cartes=vermell)
28              // => lactarius-vinosus
29              exps = new ExpressioBooleana[2];
30              exps[0] = new Igual(colorBarret, "vermell");
31              exps[1] = new Igual(cartes, "vermell");
32              regles[0] = new Regla(plantilla, new I(exps),
33                      "lactarius-vinosus");
34
35              // Regla 2: si(color-barret=taronja i cartes=taronja)
36              // => lactarius-deliciosus
37              exps = new ExpressioBooleana[2];
38              exps[0] = new Igual(colorBarret, "taronja");
39              exps[1] = new Igual(cartes, "taronja");
40              regles[1] = new Regla(plantilla, new I(exps),
41                      "lactarius-deliciosus");
42
43              // Regla 3: si(color-barret=verd i cartes=blanc i
44              // olor=farina) => amanita-phalloides
45              exps = new ExpressioBooleana[3];
46              exps[0] = new Igual(colorBarret, "verd");
47              exps[1] = new Igual(cartes, "blanc");
48              exps[2] = new Igual(olor, "farina");
49              regles[2] = new Regla(plantilla, new I(exps),
50                      "amanita-phalloides");
51
52              // Regla 4: si(color-barret=vermell i
53              // te-barret-esquamos i cartes=blanc)
54              // => amanita-muscaria
55              exps = new ExpressioBooleana[3];
56              exps[0] = new Igual(colorBarret, "vermell");
57              exps[1] = new VariableBooleana(teBarretEsquamos);
58              exps[2] = new Igual(cartes, "blanc");
59              regles[3] = new Regla(plantilla, new I(exps),
60                      "amanita-muscaria");
61
62              afegirAlDiari("Aprenentatge completat.");
63          } catch (Exception e) {
64              e.printStackTrace();
65              regles = null;
66          }
67
68          return regles;
69      }
70
71  }
```

ExpertBolets2.java

```
1  import net.urv.etse.ninots.agents.expert.AgentExpert;
2  import net.urv.etse.ninots.agents.comu.*;
3  import net.urv.etse.ninots.agents.expert.regles.*;
4
5  public class ExpertBolets2 extends AgentExpert {
6
7      public Regla[] extreureConeixement(Instancia[] instancies) {
```

ExpertBolets2.java

```
 8          AtributCategoric colorBarret;
 9          AtributCategoric cartes;
10          AtributBoolea teBarretEsquamos;
11          AtributBoolea teAnell;
12          AtributNumeric alçada;
13          AtributCategoric latex;
14          Plantilla plantilla;
15          ExpressioBooleana exps[];
16          Regla regles[];
17
18          plantilla = instancies[0].obtPlantilla();
19          regles = new Regla[4];
20          try {
21              colorBarret = (AtributCategoric)
22                      plantilla.obtAtribut("color-barret");
23              cartes = (AtributCategoric)
24                      plantilla.obtAtribut("cartes");
25              teBarretEsquamos = (AtributBoolea)
26                      plantilla.obtAtribut("te-barret-esquamos");
27              teAnell = (AtributBoolea)
28                      plantilla.obtAtribut("te-anell");
29              alçada = (AtributNumeric)
30                      plantilla.obtAtribut("alçada");
31              latex = (AtributCategoric)
32                      plantilla.obtAtribut("latex");
33
34              // Regla 1: si(latex=vermell) => lactarius-vinosus
35              regles[0] = new Regla(plantilla,
36                      new Igual(latex, "vermell"),
37                      "lactarius-vinosus");
38
39              // Regla 2: si(color-barret=blau) => lactarius-indigo
40              regles[1] = new Regla(plantilla,
41                      new Igual(colorBarret, "blau"),
42                      "lactarius-indigo");
43
44              // Regla 3: si(cartes=blanc i te-anell) =>
45              // amanita-phalloides
46              exps = new ExpressioBooleana[2];
47              exps[0] = new Igual(cartes, "blanc");
48              exps[1] = new VariableBooleana(teAnell);
49              regles[2] = new Regla(plantilla, new I(exps),
50                      "amanita-phalloides");
51
52              // Regla 4: si(color-barret=blanc i
53              // te-barret-esquamos i 20 <= mida < 40) =>
54              // macrolepiota-procera
55              exps = new ExpressioBooleana[3];
56              exps[0] = new Igual(colorBarret, "blanc");
57              exps[1] = new VariableBooleana(teBarretEsquamos);
58              exps[2] = new PertanyRang(alçada,
59                      new RangNumeric(20, 40));
60              regles[3] = new Regla(plantilla, new I(exps),
61                      "macrolepiota-procera");
62
63              afegirAlDiari("Aprenentatge completat.");
64          } catch (Exception e) {
65              e.printStackTrace();
66              regles = null;
67          }
```

ExpertBolets2.java

```
68
69              return regles;
70          }
71      }
```

D.2 Pisos

pisos.plantilla

```
1   superficie:N:0,200
2   num-habitacions:N:1,10
3   altura:N:1,15
4   preu:N:0,600000
5   te-ascensor:B
6   es-lluminos:B:NIL
```

pisos.instancies

```
1   80 3 3 360000 CERT FALS
2   100 6 1 250000 FALS FALS
3   120 5 1 200000 FALS CERT
4   35 2 7 120000 CERT CERT
5   42 3 6 130000 FALS FALS
6   70 4 3 310000 FALS CERT
7   65 4 5 163000 FALS CERT
8   120 5 8 300000 CERT CERT
9   93 4 1 220000 FALS CERT
10  65 5 6 218000 FALS FALS
```

ExpertPisos1.java

```
1   import net.urv.etse.ninots.agents.expert.AgentExpert;
2   import net.urv.etse.ninots.agents.comu.*;
3   import net.urv.etse.ninots.agents.expert.regles.*;
4
5   public class ExpertPisos1 extends AgentExpert {
6
7       public Regla[] extreureConeixement(Instancia[] instancies) {
8           AtributNumeric superficie;
9           AtributNumeric preu;
10          AtributBoolea teAscensor;
11          AtributNumeric altura;
12          Plantilla plantilla;
13          ExpressioBooleana exps[];
14          Regla regles[];
15
16          plantilla = instancies[0].obtPlantilla();
17          regles = new Regla[3];
18          try {
19              superficie = (AtributNumeric)
20                      plantilla.obtAtribut("superficie");
21              preu = (AtributNumeric)plantilla.obtAtribut("preu");
22              teAscensor = (AtributBoolea)
23                      plantilla.obtAtribut("te-ascensor");
24              altura = (AtributNumeric)
25                      plantilla.obtAtribut("altura");
26
27              // Regla 1: si(65 <= superficie < 80 i
```

ExpertPisos1.java

```
28              // 200000 <= preu <= 300000) => B
29              exps = new ExpressioBooleana[2];
30              exps[0] = new PertanyRang(superficie,
31                      new RangNumeric(65, 80));
32              exps[1] = new PertanyRang(preu,
33                      new RangNumeric(200000, 300000));
34              regles[0] = new Regla(plantilla, new I(exps), "B");
35
36              // Regla 2: si(0 <= preu < 180000) => C
37              regles[1] = new Regla(plantilla,
38                    new PertanyRang(preu,
39                      new RangNumeric(0, 180000)),
40                    "C");
41
42              // Regla 3: si(te-ascensor i 3 <= altura < 9) => A
43              exps = new ExpressioBooleana[2];
44              exps[0] = new VariableBooleana(teAscensor);
45              exps[1] = new PertanyRang(altura,
46                      new RangNumeric(3, 9));
47              regles[2] = new Regla(plantilla, new I(exps), "A");
48
49              afegirAlDiari("Aprenentatge completat.");
50          } catch (Exception e) {
51              e.printStackTrace();
52              regles = null;
53          }
54
55          return regles;
56      }
57
58  }
```

ExpertPisos2.java

```
 1  import net.urv.etse.ninots.agents.expert.AgentExpert;
 2  import net.urv.etse.ninots.agents.comu.*;
 3  import net.urv.etse.ninots.agents.expert.regles.*;
 4
 5  public class ExpertPisos2 extends AgentExpert {
 6
 7      public Regla[] extreureConeixement(Instancia[] instancies) {
 8          AtributNumeric preu;
 9          AtributNumeric superficie;
10          AtributNumeric altura;
11          AtributBoolea esLluminos;
12          Plantilla plantilla;
13          ExpressioBooleana exps[];
14          Regla regles[];
15
16          plantilla = instancies[0].obtPlantilla();
17          regles = new Regla[3];
18          try {
19              preu = (AtributNumeric)plantilla.obtAtribut("preu");
20              superficie = (AtributNumeric)
21                      plantilla.obtAtribut("superficie");
22              altura = (AtributNumeric)
23                      plantilla.obtAtribut("altura");
24              esLluminos = (AtributBoolea)
25                      plantilla.obtAtribut("es-lluminos");
```

ExpertPisos2.java

```
26
27              // Regla 1: si(300000 <= preu <= 600000) => A
28              regles[0] = new Regla(plantilla,
29                    new PertanyRang(preu,
30                      new RangNumeric(300000, 600000)),
31                    "A");
32
33              // Regla 2: si(40 <= superficie <= 70 i
34              // 5 <= altura < 10) => B
35              exps = new ExpressioBooleana[2];
36              exps[0] = new PertanyRang(superficie,
37                      new RangNumeric(40, 70));
38              exps[1] = new PertanyRang(altura,
39                      new RangNumeric(5, 10));
40              regles[1] = new Regla(plantilla, new I(exps), "B");
41
42              // Regla 3: si(5 <= altura < 10 i es-lluminos) => C
43              exps = new ExpressioBooleana[2];
44              exps[0] = new PertanyRang(altura,
45                      new RangNumeric(5, 10));
46              exps[1] = new VariableBooleana(esLluminos);
47              regles[2] = new Regla(plantilla, new I(exps), "C");
48
49              afegirAlDiari("Aprenentatge completat.");
50          } catch (Exception e) {
51              e.printStackTrace();
52              regles = null;
53          }
54
55          return regles;
56      }
57
58  }
```

Referències

[1] Pàgina web del Termcat, Centre de Terminologia: http://www.termcat.cat.

[2] Pàgina web del JADE, Java Agent DEvelopment framework: http://jade.cselt.it.

[3] Pàgina web del Jess, The Java Expert System Shell: http://herzberg.ca.sandia.gov/jess.

[4] Ernest J. Friedman-Hill (24 d'abril del 2000). *Jess, The Java Expert System Shell* (SAND98-8206). Sandia National Laboratories.

[5] Fabio Bellifemine, Giovanni Caire, Tiziana Trucco i Giovanni Rimassa (14 de juliol del 2004). *JADE Programmer's Guide*. CSELT S.p.A. i Universitat de Parma.

[6] Giovanni Caire (30 de juny del 2002). *Application-Defined Content Languages and Ontologies*. CSELT S.p.A.

[7] Foundation for Intelligent Physical Agents (6/12/2002). *FIPA ACL Message Structure Specification* (SC00061). FIPA.

[8] Foundation for Intelligent Physical Agents (6/12/2002). *FIPA SL Content Language Specification* (SC00008). FIPA.

[9] Foundation for Intelligent Physical Agents (6/12/2002). *FIPA Agent Management Specification* (SC00023). FIPA.

[10] Foundation for Intelligent Physical Agents (6/12/2002). *FIPA Request Interaction Protocol Specification* (SC00026). FIPA.