



# PROPOSTA DE MIGRACIÓ SEGURA D'AGENTS AMB SOL·LICITUD DE CLASSES SOTA DEMANDA PER A LA PLATAFORMA JADE

Memòria del projecte de final de carrera corresponent als estudis d'Enginyeria Superior en Informàtica presentat per Ferran Rovira Tura i dirigit per Guillermo Navarro Arribas.

Bellaterra, Juny de 2007

El firmant, Guillermo Navarro Arribas, professor del Departament d'Enginyeria de la Informació i de les Comunicacions de la Universitat Autònoma de Barcelona

CERTIFICA:

Que la present memòria ha sigut realitzada sota la seva direcció per Ferran Rovira Tura

Bellaterra, Juny de 2007

---

Firmat: Guillermo Navarro Arribas

*Als meus pares, al meu germà, a la meva família i als meus amics.*

# Agraïments

Vull donar les gràcies al meu director de projecte, en Guillermo Navarro, per l'ajut i guia durant tot el desenvolupament i sobretot en la redacció de la memòria i presentació del projecte. A Jordi Cucurull per respondrem tots els meus dubtes que vaig tenir sobre la plataforma durant el desenvolupament dels primers protocols i la predisposició de fer reunions amb el meu director de projecte i amb mi, malgrat estigués fora de Catalunya.

A tot el grup SeNDA i projectistes perquè a més d'aprendre coses noves durant les reunions també fossin divertides.

Als meus amics de la universitat Ruben Dominguez, Guillem Grau, Felipe Pinto i Xavier Sanglas per haver-me fet passar moltes bones estones i fer que els anys de carrera malgrat els exàmens, fossin agradables.

Als meus pares, al meu germà i la meva família pel suport donat durant aquest anys de la meva carrera.

# Índex

# Índex de figures

# Capítol 1

# Introducció

Des de l'aparició d'Internet, s'ha produït un creixement espectacular de les xarxes de comunicacions i això ha comportat l'aparició de nous models de negoci i noves tecnologies. Actualment les tecnologies basades en *sistemes distribuïts* s'estan instaurant cada cop més a tot arreu. Aquests models ofereixen avantatges per la propagació i localització de la informació a més d'establir mètodes de compensació de càrrega, per tal d'eliminar situacions de coll d'ampolla dels sistemes centralitzats. El model client/servidor, és el més estès en l'actualitat, però en els últims temps s'han desenvolupat altres models de sistemes distribuïts de més alt nivell. Podríem fer esment a títol d'exemple un dels models més antics no orientat a objectes, el de crides a procediments remots (*Remote Procedure Call RPC*) o el model una mica més modern, orientat a objectes, d'invocació a mètodes remots (*Remote Method Invocation RMI*).

D'altre banda, els paradigmes de programació van canviant contínuament i des de ja fa uns anys ha anat apareixent la programació basada en agents, un nou model de sistema distribuït, però encara no gaire estès. Els agents són porcions de codi que funcionen de forma autònoma i que interactuen entre ells. Una plataforma d'agents és l'entorn en què aquests es creen, funcionen, es reprodueixen (es *clonen)* i moren. Una altra idea, no tan estesa fins al moment, és que els agents poden viatjar, tant per dins la plataforma que els ha creat com moure's cap a una altra plataforma. Aquesta habilitat dels agents de poder-se moure cap altres platafor-

mes, dona lloc a la possibilitat de dur a terme aplicacions del tipus *mar-de-dades*. En aquest tipus d'aplicacions, no són les dades les que viatgen per a ser tractades sinó que és l'aplicació la que es desplaça fins el lloc on són les dades. Els agents mòbils i les plataformes operen avui en dia sobre xarxes informàtiques : xarxes locals cablejades, xarxes sense fils, xarxes ad-hoc, Internet ...

Cal remarcar, que en l'actualitat la proliferació dels dispositius mòbils ha anat creixent gràcies a l'evolució tecnològica dels sistemes informàtics tant a nivell de programari com de maquinari. Cada vegada més la microelectrònica és capaç de crear microprocessadors més petits, potents i barats dotant-los amb la capacitat de comunicació sense fils. És per aquest motiu, que avui en dia és del tot comú trobar múltiples processadors en dispositius quotidians, des d'ordinadors portàtils, PDAs, telèfons mòbils fins arribar als electrodomèstics. La creixent capacitat de procés d'aquests dispositius, ens dona la possibilitat d'aplicar-hi la tecnologia d'agents mòbils fins avui restringida als ordinadors convencionals. Però transmetre un agent des d'un dispositiu mòbil a un altre dispositiu mòbil o no mòbil, de vegades ens pot resultar econòmicament car depenent del protocol de transferència que utilitzem.

El nostre projecte tractarà sobre la mobilitat dels agents. Per potenciar aquesta habilitat de moures desenvoluparem un protocol de migració sota demanda, el qual es desenvoluparà per la plataforma d'agents JADE (*Java Agent DEvelopment Framework*) [11]. La plataforma JADE és un marc de treball per al desenvolupament i execució d'agents intel·ligents que segueixen els estàndards de la organització *Foundation for Intelligent Physical Agents* (FIPA) [3], que es dedica a la producció d'estàndards per a la interoperabilitat dels agents. Aquest protocol s'inclourà dins del add-on de migració entre diferents plataformes, el *JADE Inter-Platform Mobility Service JIPMS* v.1.97. Aquest mòdul està desenvolupat i integrat dins la plataforma JADE.

La plataforma JADE i els agents estan implementats amb Java, llenguatge orientat a objectes, fortament tipat i que per executar les totes aplicacions utilitza una màquina virtual anomenada *Java Virtual Machine*. El protocol de migració que conté el mòdul JIPMS actualment, conegut com a protocol de migració clàssica,

envia totes les classes dels agents a les plataformes destinatàries. El nostre protocol beneficiarà als agents amb l'habilitat de moure's, perquè ho facin amb la major rapidesa i amb el menor cost econòmic possible. Només enviarem aquelles classes que l'agent necessiti realment per poder-se executar en les altres plataformes que no siguin la que l'hagi creat.

La velocitat de transferència de les dades dels agents ve determinada pel medi de comunicació. Hi han medis en que la velocitat de transmissió és molt baixa comparada amb un xarxa local, com pot ser Internet, una xarxa sense fils seguint l'estàndard IEEE 802.11 o bé *Bluetooth*. Si només enviem les classes necessàries per a la seva execució i no les enviem totes, l'agent migrarà en menys temps.

L'altre benefici que aporta el nostre protocol és la disminució del cost econòmic de les comunicacions. Cada cop més, els dispositius mòbils tenen la capacitat de connectar-se a Internet i és en aquest tipus de connexions on es paga per la quantitat de dades que es transfereixen. Quan els agents duguin a terme les nostres peticions sobre d'altres plataformes només els hi enviarem les classes que els hi són realment necessaries, transferint una quantitat inferior de dades respecte a si enviéssim totes les classes juntes. Per tant s'aconsegueix un estalvi important en el cost de les comunicacions.

Alguns estudis referents al tema economic de les comunicacions conclouen que és molt més barat emmagatzemar les dades i sempre intentar fer els còmputs en el lloc on es trobin, que utilitzar massivament les comunicacions [8]. Els nostres dispositius mòbils o inclús algunes de les computadores que es connectin per Internet tenen quotes de connexió a Internet que són dependents del volum de dades transmeses.

També cal recordar que la seguretat en els sistemes d'agents és un problema encara obert amb un gran abast. Fins el moment, no hi ha cap protocol de control d'accés o d'autenticació creat i integrat dins el mòdul JIPMS v1.97 per la plataforma JADE. La creació de protocols de control d'accés també és objectiu d'aquest projecte i ens han de permetre :

- Autenticar a les plataformes emissores dels agents davant les plataformes destinatàries.

- Autenticar els agents que volen migrar i executar-se a la plataforma destinatària.

Aquests protocols els crearem per tal d'afegir una mínima seguretat perquè les plataformes puguin garantir que estan tractant amb agents que provenen de plataformes de confiança o bé que el propi agent ja és de confiança, malgrat que la plataforma de la qual prové, potser no ho sigui.

Abans d'explicar els objectius que ens plantegem en el nostre projecte, cal remarcar les motivacions de caràcter personal. El nostre interès sobre els agents mòbils prové d'unes pràctiques que vam impartir en l'assignatura de *Xarxes de Computadors II* de la nostra carrera, en les quals treballàvem sobre aquesta tecnologia i ho realitzàvem sobre la plataforma del projecte MARISM-A [14].

Una altre al·licient per escollir aquest projecte és que el nostre treball formarà part d'un projecte més gran on també hi participen altres països, el projecte JADE de la branca de recerca i desenvolupament de *Telecom Italia Lab (TILab)*, que forma part de l'empresa *Telecom Italia*. Destacar sobretot l'ampliació del mòdul JIPMS del *departament d'Enginyeria de la Informació i de les Comunicacions* (*dEIC*) de la UAB, fent una aportació sobre un protocol diferent dels vistos fins ara. A més, el projecte JIPMS forma part de la comunitat de codi lliure del projecte *SourceForge* [13], on es desenvolupen més de cent mil projectes d'*open source* on qualsevol pot participar-hi i fer un seguiment de la seva evolució.

El nostre projecte formarà part del grup *SeNDA* (*Security of Networks and Distributed Applications*) dins del dEIC. Aquest projecte es dur a terme amb set altres projectes del mateix grup però desenvolupant altres temes d'interès com són aplicacions mèdiques, protocols de migració fragmentada, llibreries d'encriptació, jocs, components que afegeixen més funcionalitats als agents i descobriments de serveis. Cada projectista escriurà en una wiki el seu projecte permetent tant els companys com directors fer un seguiment i servir també com ajut per a explicar l'evolució del projecte en les reunions periòdiques de grup amb els nostres directors.

## 1.1 Objectius

Tot seguit presentarem els objectius del nostre projecte que volem desenvolupar :

- Dissenyarem i crearem un Protocol de Migració Sota Demanda seguint els estàndards proposats per FIPA. Aquest és el nostre principal objectiu del projecte i en el qual ens centrarem la major part del temps de desenvolupament. Amb aquest protocol aconseguirem disminuir dins del medi de comunicació la quantitat d'informació en l'enviament i execució de l'agent en un altra plataforma que no sigui la de creació, millorant respecte als protocols de migració actuals.

- Dissenyarem i crearem dos protocols de control d'accés i autenticació per tal de solucionar alguns dels problemes de seguretat dels agents mòbils. Aquests protocols garantiran a la plataforma que rebi els agents una minima confiança en les plataformes que els envien i/o en els propis agents. Al realitzar aquesta autenticació permetem o deneguem a l'agent l'accés a la plataforma i als seus recursos. Aquests protocols també han de seguir els estàndards proposats per FIPA.

- Integrarem tots els nostres protocols de control d'accés i migració sota demanda al mòdul JIPMS v1.97 de la plataforma JADE.

- Crearem un joc de proves per poder saber si s'han aconseguit els objectius de crear i integrar un protocol sota demanda dins del mòdul JIPMS de JADE i disminuir la quantitat d'informació en el medi de comunicació.

  Com a objectius secundaris o opcionals :

- Millorarem el Protocol Sota Demanda per tal que les plataformes guardin les classes que hagin demanat sota demanda, així reduirem encara més la quantitat de dades envers els altres protocols de migració.

Tots els objectius d'aquest projecte, tant els opcionals com els primaris, s'inclouran dins la planificació del nostre projecte per tal de fer-ne un seguiment acurat per saber si ens desviem o no i permetre controlar si podem assolir-los tots.

## 1.2 Estructura de la Memòria

A continuació presentarem el contingut de la memòria i una breu descripció de cada un dels capitols que veurem.

- *Capito 2 : Estudi de Viabilitat* En aquest capitol expliquem breument quins beneficis ens aporten els agents mòbils, les seves fortaleses i debilitats i quin sistema ens és necessari per executar la plataforma JADE i els seus agents. Per últim podrem veure la planificació del nostre projecte i quina metodologia hem seguit. En aquest estudi també explicarem la viabilitat del nostre projecte basant-nos en uns criteris determinats, com són l'operativitat, la legalitat, el cost economic i la tecnologia.

- *Capitol 3 : Entorn i Eines de Desenvolupament* Aquí ens introduirem en el mon dels agents mòbils. Explicarem d'on provenen, quines característiques tenen. Parlarem sobre els estàndards i especificacions de FIPA els quals segueix la plataforma JADE. Veurem com funcionen els serveis de JADE i com integrarem els nostres protocols dins el mòdul de JIPMS. També, explicarem en quins components de Java ens ha estat necessari aprofundir-hi per tal de desenvolupar el nostre projecte. Per últim, veurem les eines que hem utilitzat pel projecte.

- *Capitol 4 : Protocols d'Autenticació i Control d'Accés* Veurem quins protocols d'autenticació hem creat i quins mètodes hem implementat per cada protocol. Explicarem per cada protocol com funciona i com hem fet les fases d'anàlisi i disseny de cada un d'ells. Per últim, podrem veure les dificultats superades d'integració dels protocols.

- *Capitol 5 : Protocol de Migració Sota Demanda V1.0* En aquest capitol presentarem la primera versió del protocol sota demanda i explicarem les seves fases d'anàlisis, disseny i implementació. Veurem les dificultats que ens hem trobat a l'hora d'implementar-lo i integrar-lo i les decisions que hem pres en cada cas. Per últim explicarem la raó per la qual ens és necessari crear la segona versió del protocol.

- *Capitol 6 : Protocol de Migració Sota Demanda V2.0 amb ús de Cache* Presentarem la segona versió del protocol sota demanda i veurem quines fases d'anàlisis, disseny i implementació hem dut a terme per aquesta nova versió. Explicarem les dificultats de implementar la classe que ens permet emmagatzemar (fer *cache*) de les classes sol·licitades sota demanda i com l'hem integrat dins el servei de mobilitat JIPMS. També veurem els resultats dels tests de rendiment d'ambdós protocols Sota Demanda. Explicarem els jocs de proves que utilitzarem i els resultats dels tests de rendiment per comparar-los i veure si l'ús d'una cache es beneficiós o no.

I per finalitzar la memòria hi ha un capitol on mostrem les conclusions extretes de la realització d'aquest projecte. Quins objectius hem complert, quines possibles línies de futur veiem del nostre projecte i una valoració de caràcter personal sobre el projecte.

# Capítol 2

# Estudi de Viabilitat

En aquest estudi de viabilitat presentarem una breu introducció de perquè fem aquest projecte i tot seguit els punt forts i les debilitats d'utilitzar agents mòbils. Explicarem també, la viabilitat del projecte des de els punts de vista d'operativitat, tecnologia, econòmica i legal. Finalment veurem breument la metodologia que utilitzarem i la planificació del nostre projecte.

## 2.1 Raó i Oportunitat del Projecte

Com hem vist en el capitol d'introducció, els agents mòbils proveeixen una nova forma de generar aplicacions que s'adapten millor a problemes o tasques relacionades amb: xarxes, còmput distribuït, treball entre diferents plataformes i arquitectures, xarxes de baixa confiança , desconnexió parcial, sistemes de còmput mòbils sense fils ...

En l'actualitat la tecnologia dels agents mòbils no es gaire utilitzada degut al seu gran desconeixement. Tanmateix es de preveure la seva gradual extensió en els propers anys fins arribar a ser tan important que, inclús, es previsible la veiem aplicada als electrodomèstics.

També hem vist, que el model de sistema distribuït més estés actualment és el client-servidor on les comunicacions acostumen a ésser el coll d'ampolla. Aquest model obliga al client a estar la major part del temps depenent de les respostes

del servidor ja siguin, sol·licitud de dades, obtenció de resultats o errors. Com a problema afegit, el client quan es vol identificar envia per la xarxa el seu nom d'usuari i el seu password que poden ser interceptats per una tercera part, sinó s'utilitza algun mètode d'encriptació.

Vistes aquestes mancances, es presenta una clara oportunitat per crear una migració sota demanda la qual ens permetria no abusar del medi de transmissió i només enviar la informació necessària. També es troba convenient crear protocols d'autenticació per tal de garantir una mínima confiabilitat amb les plataformes i/o els agents, així evitem que el client enviï el seu identificador i password per Internet, ja que amb la identificació de l'agent és suficient.

## 2.2 Beneficis d'Utilitzar els Agents Mòbils

Els beneficis del paradigma de l'ús dels agents mòbils són els següents :

- Facilitat d'implementació. Els agents són fàcils de crear, d'instal·lar i de desinstal·lar.
- Possibilitat que amb el nostre projecte podem ampliar l'àmbit dels agents mòbils i donar-los a conèixer en l'àmbit de les TIC.
- Difondre la nova manera de desenvolupar aplicacions. Per això crear aplicacions o nous protocols per demostrar la utilitat del agents mòbils.
- Assolir que s'estandarditzi els protocols de migració i de control d'accésă per tal de normalitzar la creació, la mobilitat i la seguretat dels agents mòbils.

## 2.3 Especificacions del sistema i de les aplicacions

Un cop hem vist els beneficis, veurem quins són els seus punts forts i debilitats dels agents, quina arquitectura del sistema és necessaria per utilitzar els agents i finalment les seves aplicacions.

### 2.3.1 Fortaleses i Debilitats

Malgrat tot, els punts més importants d'aquesta tecnologia son els següents:

- **Fortaleses** Els agents mòbils consumeixen menys recursos de xarxa ja que ells porten el còmput a on troben les dades i no viceversa. Els agents mòbils no requereixen connexions continues entre màquines.

  Tenen accés als recursos mentre l'usuari estigui desconnectat (*off-line*). A més, gràcies als agents podem paral·lelitzar tasques o càlculs.

- **Debilitats**

  Les debilitats més importants són la seguretat, ja que encara queden molts problemes oberts com impersonalització d'agencies (les plataformes), denegació d'execució, execucions falses, manipulació o espiat de les dades o el codi de l'agent entre moltes altres.

  L'altre debilitat important, és que si un agent s'està executant en una plataforma i aquesta cau, l'agent es perd i tot el que havia estat fent fins el moment en altres plataformes o en la que es trobava també. Però per solventar aquesta debilitat estan creat mètodes per tal que els agents i les plataformes siguin tolerants a fallades (*Fault Tolerance*).

### 2.3.2 Arquitectura Tècnica del Sistema

El desenvolupament d'aquest projecte es realitzarà amb el llenguatge Java, en plataformes JADE. Al desenvolupar en Java hi ha la possibilitat de fer el projecte en qualsevol Sistema Operatiu que interpreti Java.

### 2.3.3 Aplicacions

La migració d'agents es farà de manera més segura que fins ara, com a mínim en l'autenticació de les plataformes i els agents. Les aplicacions de la migració sota demanda són evitar enviar massa quantitat d'informació dels agents que no farà servir la plataforma destí, tot evitant així l'ús excessiu del medi de comunicació

com pot ser una xarxa bluetooth, la xarxa d'Internet, una xarxa local, o bé via xarxa telefònica la qual paguem per la quantitat de bytes enviats i rebuts.

## 2.4 Viabilitat del Projecte

Ara explicarem la viabilitat del projecte en base la tecnologia, la operativitat, la legalitat i el cost econòmic.

### 2.4.1 Viabilitat tècnica

Respecte l'ús de la tecnologia el projecte és viable ja que utilitzem llenguatges coneguts com el Java, i no son necessaris recursos potents en còmput, amb ordinadors estàndards podem arribar a assolir tots els objectius del projecte.

### 2.4.2 Viabilitat operativa

L'operativitat del Protocol de Migració Sota Demanda no sol serveix per les xarxes grans com Internet, sinó també pot servir per a dispositius mòbils com PDAs, telèfons mòbils entre altres dispositius mòbils que tinguessin instal·lada la plataforma JADE. Sobretot en els dispositius mòbils els quals per enviament i rebuda d'agents es pagaria per els bytes, quants menys bytes enviats i només la informació necessària, més barat resultarà la migració d'agents i més extensió podrà tenir.

### 2.4.3 Viabilitat econòmica

No hi han costos afegits d'infraestructura ja que es pot mantenir la que hi havia. Aquest protocol seria un Add-On a la plataforma de JADE, la qual s'adapta a qualsevol Sistema Operatiu. L'únic que s'ha de tenir es el *Java Runtime Environment* 1.5 instal·lat. A més tots aquestes productes pertanyen al programari lliure, per tant no tenen cap cost afegit per comprar llicencies.

Per realitzar la planificació utilitzarem el programa *Microsoft Office Project* i per les etapes de disseny utilitzarem el *Rational Rose*. Ambdós programes estan

instalůlats en les sales de ordinadors de la ETSE i per tant tampoc suposen cap cost afegit per aquest projecte ja que podem disposar d'aquests programes.

### 2.4.4 Viabilitat legal

Un dels problemes principals que tenen els Agents Mòbils és la seguretat i la privacitat tant del codi com de les dades que poden portar amb ells, ja que poden ser sensibles. Aquest projecte no és concentra en aquest abast ja que hi ha altres projectes proposats del departament que solucionen aquesta mancança en la seguretat. Sobretot en els casos d'aplicacions en que s'han de recuperar dades, ja que han de seguir la LOPD (Llei Orgànica de Protecció de Dades).

## 2.5 Abast del Projecte

En aquesta secció veurem quin és l'abast del projecte, tant el que hem d'arribar a implementar com el que no. També veurem alguns possibles desenvolupaments per millorar el projecte base.

### 2.5.1 Requisits Generals del Projecte

Els requisits generals del projecte són crear dos protocols d'autenticació bàsics per controlar l'accés del agents a les plataformes quan migrin. I el Protocol de Migració Sota Demanda es farà entre diferents plataformes i no es contemplarà una migració entre els components interns de les plataformes, excepte entre els components principals d'aquestes.

### 2.5.2 Possibles Millores de Desenvolupament del Projecte

Les possibles millores del projecte un cop complerts els requeriments generals, podrien ser:

Crear protocols de control d'accés més complexes o millorar els bàsics per que esdevinguin d'un nivell més alt de control i autenticació.

Les plataformes podrien emmagatzemar les classes dels agents que hagin sollicitat a la plataforma origen, per futures migracions d'aquests. Els protocols sota demanda podrien contemplar les migracions més complexes, podríem migrar des de qualsevol component que formi la plataforma i no limitar-ho només al principal.

## 2.6 Metodologia

Davant del projecte que s'ens planteja, creiem que la millor opció és una metodologia àgil en front de la clàssica, ja que és molt probable que tinguem de fer versions per millorar les mancances d'alguns protocols i ens aniria bé una metodologia que ens permetés fer desenvolupaments ràpids i fer proves sobre aquests per anar creant diferents entregables dels protocols. Per a cada entregable és molt probable que haurem de tornar a fer una nova fase d'anàlisi, de disseny i per descomptat les fases d'implementació i integració dels protocols dins la plataforma.

En el meu cas la metodologia que aplicaré serà la *eXtreme Programming* [1] (creada per *Kent Beck* per la plantilla del projecte C3 en Chrysler).En la figura 2.1 de [16] podem veure les fases d'aquesta metodologia. Aquesta metodologia l'utilitzarem per les següents raons:

- És una metodologia amb un contracte flexible, és més sensible als canvis.
- El client forma part del equip, com en el nostre cas, ja que el nostre projecte forma part del grup SeNDA on hi participen altres projectistes amb els quals comentem l'evolució dels nostres projectes.
- Serveix per grups reduïts, sol serem una sola persona.
- L'arquitectura no és tant important ja que ens és coneguda.
- Els requeriments emergeixen, canvien molt. Per a cada nova versió és probable que haurem de treure requeriments o afegir-ne
- També perquè té unes pràctiques molt útils com :

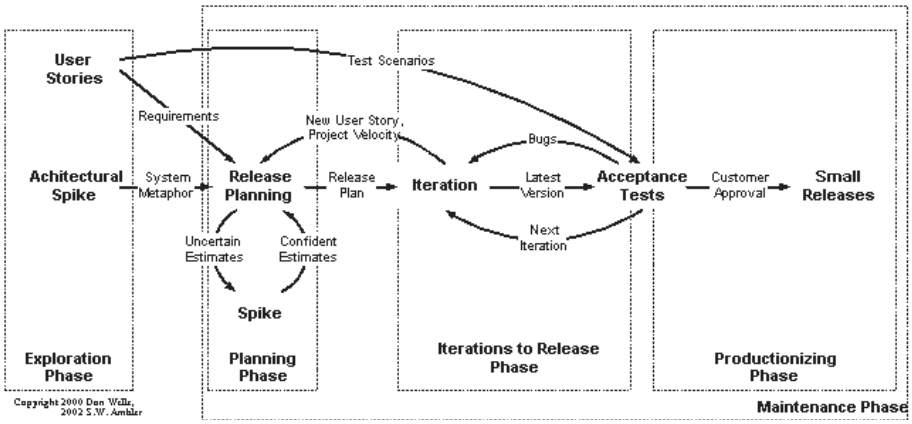


Figura 2.1: Cicle de Vida de la Metodologia eXtreme Programing.

1. *Refactoring*, millora del codi molt freqüentment. Després de les proves és molt possible que millorem el codi potser per millorar el rendiment, per que sigui més eficient o altres paràmetres importants que ha de tenir el nostre projecte.

2. *Integració continua*, cada entrega l'haurem d'integrar dins la plataforma executant un joc complet de proves, per assegurar el seu correcte funcionament.

3. *Entregues petites*, crearem prototips amb les noves funcionalitats per sols per provar allò que hem implementat i integrat.

4. *Molts testos*, en aquesta metodologia s'imparteix més temps en fer proves als entregables complets i parcials, que inclús a la codificació.

- Una altre raó es que en el grup SeNDA cada projecte està exposat en una wiki accessible per la resta de projectistes per tal de col·laborar amb els projectes del altres.

## 2.7 Planificació del Projecte

En la present secció mostrem la planificació temporal i el seguiment del project, realitzada durant la tasca d'inici (*Kick Off*) del projecte. En la figura 2.2 podem veure les tasques i la seva duració real ja que quan escrivim la present memòria ja hem finalitzat totes les tasques i la informació que mostrem és de l'actualització de totes les tasques del projecte.

En la figura 2.2 també podem veure com cada protocol s'haurà de fer les seves pertinents fases d'anàlisis, disseny i implementació. Però on actuarà més la metodologia és en els dos protocols sota demanda. La segona versió, en cas que poguéssim realitzar-la, seria una millora de la primera però haurem de realitzar les fases d'anàlisis, disseny, implementació i proves com si fos un nou protocol malgrat el que realment és només millorar la primera versió.

En la figura 2.3 podem veure el diagrama de Gantt de seguiment del nostre projecte. Podem apreciar un endarreriment del projecte d'una setmana causat per l'implementació de la primera versió del protocol sota demanda explicada al capitol 5 en la secció 5.3.2 sobre les consideracions d'integració del protocol en la plataforma.

## 2.8 Resum

En l'estudi de viabilitat hem pogut saber quins beneficis ens aporten els agents, quines punts forts tenen i quines aplicacions tenen. També hem vist l'abast del projecte tant el que hem d'arribar a implementar com el que no i les parts opcionals possibles del nostre projecte. La viabilitat del projecte en base a termes econòmics, legals, operatius i tècnics.

Quina arquitectura necessita el sistema per tal de poder executar una plataforma JADE. Per últim hem vist quines eines ens són necessaries per assolir el projecte i la planificació que hem fet del projecte. Hem vist les tasques necessaries per realitzar-lo amb els seu temps real, ja que hem fet un seguiment del projecte. També hem vist i comentat el diagrama de Gantt del projecte.

| Id | Nombre de tarea | Duración | Trabajo | Comienzo | Fin |
|---|---|---|---|---|---|
| 1 | **PROJECTE MIGRACIÓ SOTA DEMANDA SEGURA** | **90 días** | **730 horas** | **jue 21/12/06** | **mié 25/04/07** |
| 2 | **Kick Off del Projecte** | 1 día | 8 horas | jue 21/12/06 | jue 21/12/06 |
| 3 | **Estudi de viabilitat** | **17 días** | **72 horas** | **jue 21/12/06** | **vie 12/01/07** |
| 4 | Parlar amb el Director del Projecte i definir el Projecte | 1 día | 8 horas | jue 21/12/06 | jue 21/12/06 |
| 5 | Definir els Objectius del Projecte | 2 días | 16 horas | jue 21/12/06 | vie 22/12/06 |
| 6 | Realització del Document | 12 días | 48 horas | mié 27/12/06 | vie 12/01/07 |
| 7 | **Proves de Concepte** | **24 días** | **78,08 horas** | **jue 21/12/06** | **mar 23/01/07** |
| 8 | **Reconeixement i instal.lació de l'entorn** | 5 días | 10 horas | jue 21/12/06 | vie 29/12/06 |
| 9 | **Crear un Protocol d'Accés Bàsic** | **11 días** | **51,28 horas** | **mar 02/01/07** | **mar 16/01/07** |
| 10 | **Anàlisis** | **1 día** | **5,28 horas** | **mar 02/01/07** | **mar 02/01/07** |
| 11 | Requeriments | 1 día | 2,63 horas | mar 02/01/07 | mar 02/01/07 |
| 12 | Especificacions dels Requeriments | 1 día | 2,63 horas | mar 02/01/07 | mar 02/01/07 |
| 13 | **Disseny** | **6 días** | **24 horas** | **mié 03/01/07** | **mié 10/01/07** |
| 14 | **Control d'Acces Bàsic** | **6 días** | **24 horas** | **mié 03/01/07** | **mié 10/01/07** |
| 15 | Disseny Sencill | 3 días | 12 horas | mié 03/01/07 | vie 05/01/07 |
| 16 | Disseny amb més elements | 2 días | 8 horas | lun 08/01/07 | mar 09/01/07 |
| 17 | Diagrama de Classes | 1 día | 4 horas | mié 10/01/07 | mié 10/01/07 |
| 18 | **Implementació** | 3 días | 18 horas | jue 11/01/07 | lun 15/01/07 |
| 19 | **Proves** | 1 día | 4 horas | mar 16/01/07 | mar 16/01/07 |
| 20 | **Crear un Protocol d'Accés amb Certificats** | **5 días** | **16,8 horas** | **mié 17/01/07** | **mar 23/01/07** |
| 21 | **Anàlisis** | **1 día** | **4 horas** | **mié 17/01/07** | **mié 17/01/07** |
| 22 | Requeriments | 1 día | 2 horas | mié 17/01/07 | mié 17/01/07 |
| 23 | Especificacions dels Requeriments | 1 día | 2 horas | mié 17/01/07 | mié 17/01/07 |
| 24 | **Disseny** | **2 días** | **4,8 horas** | **jue 18/01/07** | **vie 19/01/07** |
| 25 | Creació Certificats | 1 día | 0,8 horas | jue 18/01/07 | jue 18/01/07 |
| 26 | Disseny del Control d'Accés | 1 día | 2 horas | jue 18/01/07 | jue 18/01/07 |
| 27 | Diagrama de Classes | 1 día | 2 horas | vie 19/01/07 | vie 19/01/07 |
| 28 | **Implementació** | 1 día | 4 horas | lun 22/01/07 | lun 22/01/07 |
| 29 | **Proves** | 1 día | 4 horas | mar 23/01/07 | mar 23/01/07 |
| 30 | **Avaluació d'alternatives i selecció** | 14 días | 5,6 horas | jue 21/12/06 | vie 12/01/07 |
| 31 | **Creació del Protocol Sota Demanda Versió 1.0 (Migració entre dues Plataformes)** | **22 días** | **162 horas** | **lun 15/01/07** | **mar 13/02/07** |
| 32 | **Anàlisis** | **2 días** | **8 horas** | **lun 15/01/07** | **mar 16/01/07** |
| 33 | Requeriments | 2 días | 4 horas | lun 15/01/07 | mar 16/01/07 |
| 34 | Especificacions dels Requeriments | 2 días | 4 horas | lun 15/01/07 | mar 16/01/07 |
| 35 | **Disseny** | **2 días** | **10 horas** | **mié 17/01/07** | **jue 18/01/07** |
| 36 | Disseny del ClassLoader | 1 día | 2 horas | mié 17/01/07 | mié 17/01/07 |
| 37 | Disseny del Deserialitzador | 1 día | 2 horas | mié 17/01/07 | mié 17/01/07 |
| 38 | Concretar SubProtocols | 1 día | 2 horas | mié 17/01/07 | mié 17/01/07 |
| 39 | Diagrama de Classes | 1 día | 4 horas | jue 18/01/07 | jue 18/01/07 |
| 40 | **Implementació** | 15 días | 120 horas | vie 19/01/07 | jue 08/02/07 |
| 41 | **Proves d'integració** | 3 días | 24 horas | vie 09/02/07 | mar 13/02/07 |
| 42 | **Creació del Protocol Sota Demanda Versió 2.0 (Ús de Caché)** | **23 días** | **173,28 horas** | **mié 14/02/07** | **vie 16/03/07** |
| 43 | **Anàlisis** | **2 días** | **16 horas** | **mié 14/02/07** | **jue 15/02/07** |
| 44 | Requeriments | 2 días | 8 horas | mié 14/02/07 | jue 15/02/07 |
| 45 | Especificacions dels Requeriments | 2 días | 8 horas | mié 14/02/07 | jue 15/02/07 |
| 46 | **Disseny** | **1 día** | **5,28 horas** | **vie 16/02/07** | **vie 16/02/07** |
| 47 | Disseny de emmagatzemament de les classes en Cachés | 1 día | 2,63 horas | vie 16/02/07 | vie 16/02/07 |
| 48 | Diagrama de Classes | 1 día | 2,63 horas | vie 16/02/07 | vie 16/02/07 |
| 49 | **Implementació** | 11 días | 88 horas | lun 19/02/07 | lun 05/03/07 |
| 50 | **Proves d'Integració** | 4 días | 24 horas | mar 06/03/07 | vie 09/03/07 |
| 51 | **Tests dels Protocols Sota Demanda** | 5 días | 40 horas | lun 12/03/07 | vie 16/03/07 |
| 52 | **Aprenentage del LaTex** | **10 días** | 26,4 horas | **vie 16/02/07** | **jue 01/03/07** |
| 53 | **Aprobació del Sistema** | 1 día | 2 horas | lun 12/03/07 | lun 12/03/07 |
| 54 | **Pla d'integració** | 1 día | 2,63 horas | mar 13/03/07 | mar 13/03/07 |
| 55 | **Realització de la Memòria** | 25 días | 200 horas | jue 22/03/07 | mié 25/04/07 |

Figura 2.2: Tasques del Projecte.

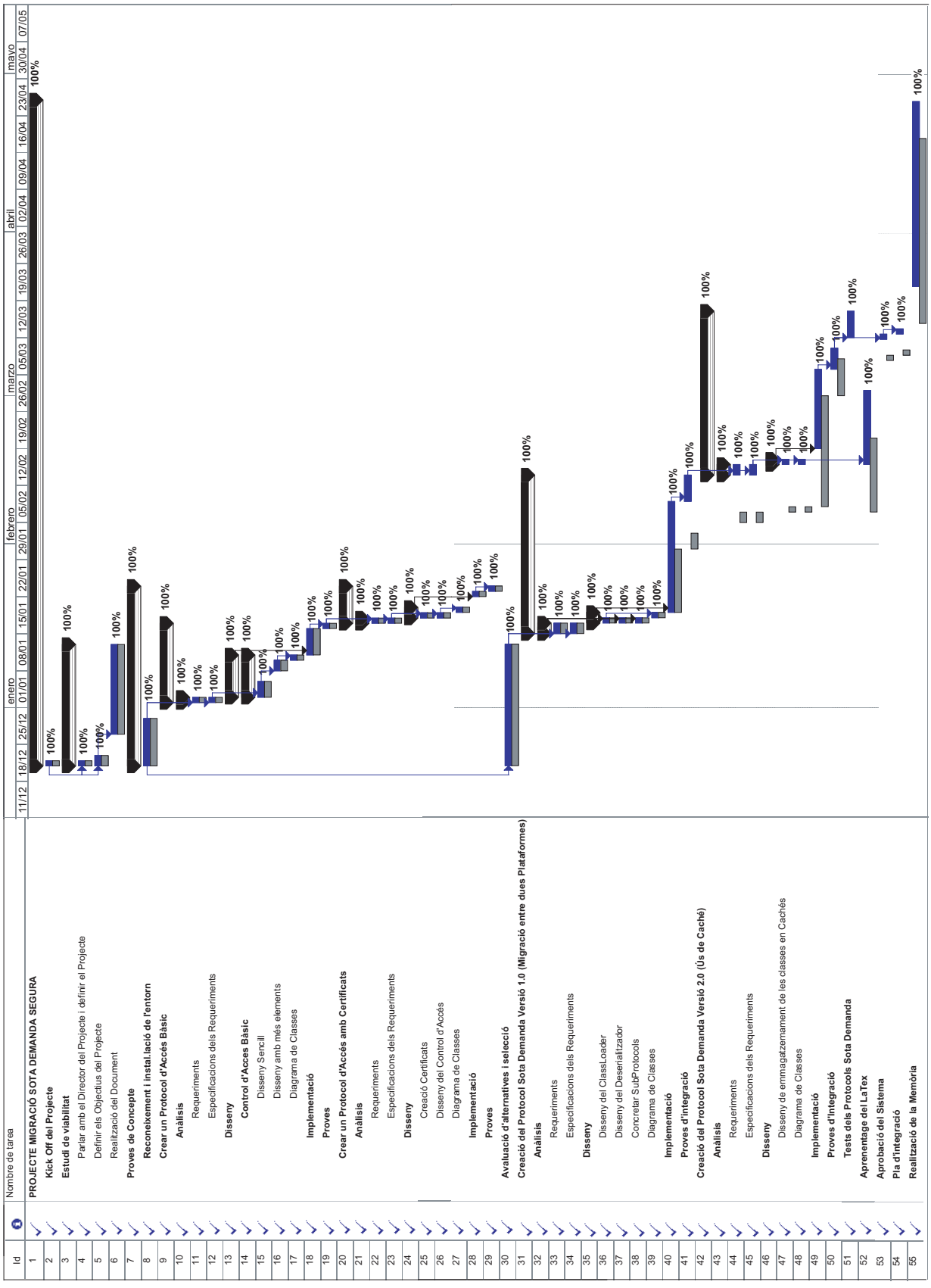


Figura 2.3: Diagrama de Gantt del Projecte.

# Capítol 3

# Entorn i Eines de Desenvolupament

En aquest capitol definirem els conceptes que ens han estat necessaris per la realització del projecte. Primer presentarem que és un agent mòbil, que és FIPA i quins conceptes, especificacions i estàndards d'aquesta organització utilitzem dins del projecte. Explicarem que és la plataforma JADE, quina arquitectura té i en quines parts ens hem centrat per realitzar el projecte. Finalment parlarem de la classe *ClassLoader* de Java que ens ha estat necessari aprendre com funcionava per tal de crear el nostre propi carregador de classes dins del projecte.

## 3.1 Agents Mòbils

El concepte d'agent prové de la intel·ligència artificial en el que es refereix a la capacitat d'aprenentatge, de presa de decisions i la capacitat de comunicació per interactuar amb altres agents.

Un agent és un programa completament autònom que té la capacitat de prendre decisions per ell mateix sense necessitat de què esperi ordres externes. També pot reaccionar davant d'events del seu entorn, com poden ser altres agents. És per això que diem que són sociables i que tenen la capacitat de comunicar-se. A més, cada agent pot donar serveis a la plataforma i qualsevol altre agent dins la plataforma podria contactar amb aquest agent que dona el servei que busca. Per tant, els agents a més de comunicar-se ho poden fer de manera asíncrona amb

qualsevol altre agent. Una altre característica és que els agents són proactius, les seves accions poden arribar a afectar al seu entorn.

Les definicions anteriors podrien definir qualsevol tipus d'agent, però en el nostre cas parlem d'agents mòbils, és a dir, que a més de les anteriors capacitats, tenen la capacitat de moures d'una plataforma a una altre o bé dins una mateixa plataforma. Per gestionar els agents, tinguin o no l'habilitat de moures, ens cal un entorn i això ens ho permet la plataforma JADE que podem veure en la secció 3.3.

## 3.2 FIPA

FIPA (*Foundation for Intelligent Physical Agents*) [3], és una organització sense ànim de lucre fundada el 1996 i dedicada a crear estàndards per la interoperabilitat de sistemes heterogenis d'agents. Ara veurem les especificacions i estàndards necessaris per crear dos tipus protocols de interacció, hi han altres tipus de protocols d'interacció que es poden consultar a [7].

### 3.2.1 Llenguatge de Comunicació dels Agents

Tota comunicació entre agents es dur a terme amb missatges ACL (*Agent Communication Language*). FIPA defineix l'estructura d'aquests missatges especificant els camps que pot arribar a contenir, si són obligatoris o opcionals, de tal manera que la comunicació sempre sigui compatible entre agents de diferents plataformes que segueixin els estàndards de FIPA. Els missatges ACL contenen un paràmetre anomenat *performative*, que el podríem definir com la intenció de l'agent de realitzar algun tipus d'acció.

### 3.2.2 Ontologies

Les ontologies a FIPA, són un model de comunicació per tal de que dos agents involucrats en una conversació, comparteixin una mateixa base de coneixements sobre el contingut dels missatges i interpretin tots els símbols de la mateixa manera. Les ontologies ens serveixen perquè els agents es comuniquin sense ambigüitats.

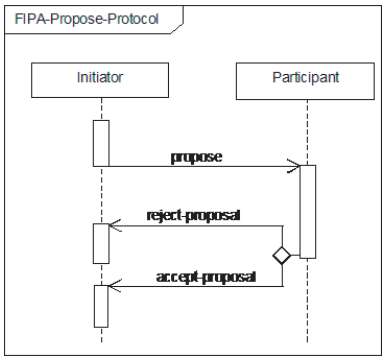


Figura 3.1: Protocol d'Interacció de Tipus Propose.

### 3.2.3 Protocols d'Interacció

Hi han molt tipus de protocols d'interacció, inclús podem fer-ne un de propi. Nosaltres ens centrarem amb els dos que utilitzarem en el nostre projecte :

- *Propose*: Aquest tipus de protocol d'interacció ens serveix per comunicacions en les quals només volem fem una pregunta sobre l'agent o alguna informació que conté el missatge ACL i només ens és necessaria una resposta de si o no, (*Accept*) o *Reject* respectivament. A la figura 3.1 extreta de [4] podem veure la seqüència de etapes del protocol.

- *Request*: Al contrari que el protocol d'interacció de tipus Propose, aquest ens serveix per una comunicació en la qual volem demanar alguna acció a la plataforma destí, com pot ben ser una petició de migració.

  Segons FIPA i com veiem a la figura 3.2 extreta de [5], primer s'envia un missatge amb el valor del paràmatre *performative* del missatge ACL a *Request*. L'agent qui rep aquest missatge respondrà amb un *Agree* si accepta la petició o un *Reject* si la denega.

  Si ha acceptat, pot torna a respondre amb un ACL amb el *performative* amb el valor *Inform* per saber com ha anat la nostra petició o bé amb un *Failure* si la petició ha fallat quan es duia a terme adjuntant un missatge de la causa.

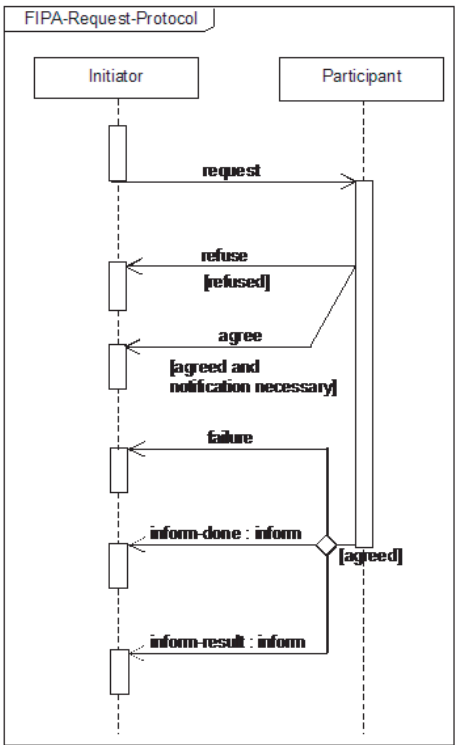


Figura 3.2: Protocol d'Interacció de Tipus Request.

Malgrat FIPA especifica que s'hagi d'enviar com a primera resposta els missatges de tipus *Agree* o *Reject*, nosaltres no hem implementat aquest pas intermig en els protocols que utilitzin aquest model. Considerem que el temps entre l'Agree i la resposta de la petició és molt curt i per tant l'ignorem a favor d'un estalvi d'ample de banda de la xarxa.

Perquè existeixi un protocol d'interacció ens són necessaries les classes que afegeixen certs comportaments als agents. L'agent que ens interessa en pels nostres protocols és l'agent que gestiona la migració, l'AMM (*Agent Migration Manager*). I les classes que afegeixen comportaments són els Behaviours. Per crear els protocols d'interacció ens són necessaris dos Behaviours un Iniciador i un Receptor.

Pels protocols d'interacció, els nostres Behaviours han heretat de les següents classes predefinides de JADE :

- Tipus Propose
    1. *ProposeInitiator*
    2. *ProposeResponder*
- Tipus Request
    1. *SimpleAchieveREInitiator*
    2. *SimpleAchieveREResponder*

## 3.3 Plataforma JADE

La plataforma JADE (*J*ava Agent DEvelopment Framework) [11] és la plataforma d'agents utilitzada en aquest projecte, en la seva versió 3.4.1. Aquesta plataforma ha estat desenvolupada per TILAB (Telecom Italia Lab). És de codi obert i es distribueix amb llicencia *Lesser General Public License LGPL.*

La característica fundamental d'aquesta plataforma és que compleix amb els estàndards de FIPA, a més de proporcionar un entorn d'execució d'agents i una serie d'eines per desenvolupar sistemes distribuïts multi-agents.

L'únic que necessitem per arribar a poder executar la plataforma JADE, és un sistema operatiu que suporti el *Java Runtime Environment 1.5._X* i tingui les llibreries de JADE.

### 3.3.1 Arquitectura de la Plataforma JADE

Les plataformes de JADE segueixen l'estàndard de FIPA que podem veure en la figura 3.3 de [6].

Ara explicarem cada un dels components principals que composen una plataforma i que es creen un cop s'inicii:

*Agent Management System (AMS)* : És un component obligatori de la plataforma, el qual només existeix un per plataforma. Representa la autoritat de gestió i el responsables de diverses operacions com poden ser la creació d'agents, la

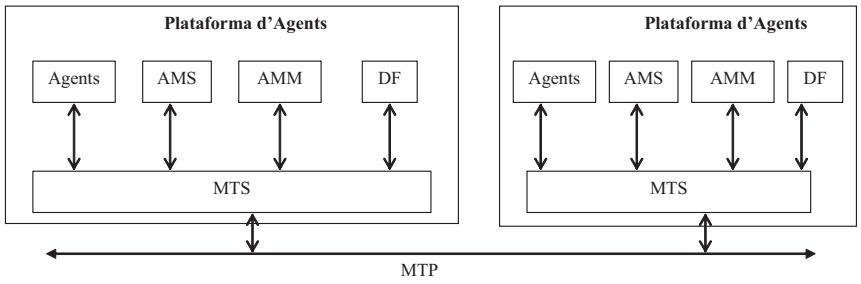


Figura 3.3: Model de Plataforma segons FIPA.

eliminació d'agents. El AMS té a més la funcionalitat de controlar l'accés i l'ús de la plataforma, manté un directori on registra els agents amb els seus identificadors (*Agent IDentificator AID*) corresponents. Oferint així un servei de pàgines blanques. Tot agent, per tal de complir el model de FIPA s'ha de registrar al AMS.

*Directory Facilitator (DF)* : Aquest component opcional ofereix un servei de pàgines grogues. Els agents registren els seus serveis disponibles al DF i qualsevol altre agent que necessiti un servei pot consultar-ho i saber amb quin agent comunicar-se.

*Message Transport System (MTS)* : El MTS correspon al sistema de comunicació de la plataforma que permet l'intercanvi de missatges ACL entre agents. Aquesta comunicació pot ser tant entre agents de la mateixa plataforma (aquesta comunicació o de vegades migració s'anomena *Intra-Plataforma*), o bé entre agents de plataformes diferents ( aquesta comunicació o inclús migració s'anomena *Inter-Plataforma*). Si el missatge és per un agent de dins la plataforma el MTS directament l'envia al seu destinatari. Per altre banda, si el missatge és per un agent d'una altre plataforma, el MTS envia el missatge a l'altre MTS de la plataforma de l'agent destinatari. Aquesta comunicació entre MTS es realitza amb algun protocol de transport de missatges (*Message Transport Protocol MTP*).

A més d'aquests components contenen agents dels usuaris de la plataforma o

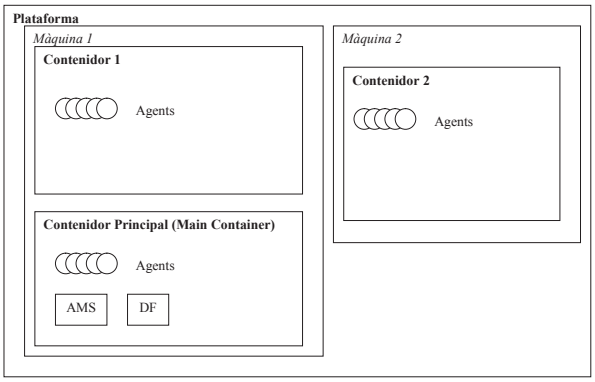


Figura 3.4: Arquitectura Interna d'una plataforma JADE.

bé que hagin immigrat d'altres plataformes. Ara cada plataforma conté contenidors, tants com vulguin. Cada contenidor conté els agents que migrin o bé que s'hagin creat dins del contenidor. L'únic contenidor que conté el AMS i el DF és el contenidor principal, el qual es crea automàticament quan engeguem la plataforma. La resta de contenidors poden estar en una mateixa màquina o bé cada contenidor pot estar iniciat en una màquina per separat. En la figura 3.4 podem veure una representació d'una plataforma.

### 3.3.2 Serveis de JADE

Els serveis de JADE són un dels components més importants que formen part de la plataforma. Els serveis ens proporcionen funcionalitats i control sobre les peticions dels agents. Per exemple hi ha el servei de mobilitat Intra-Plataforma per migracions d'agents dins la pròpia plataforma, el servei de missatgeria i entre altres, el servei Inter-Plataforma que és amb el que ens centrarem en el nostre projecte i és el que ens permet migrar els agents cap a una altre plataforma.

Cada contenidor que formi part de la plataforma de JADE, té uns serveis. Qui sap de tots els serveis de tots els contenidors de la plataforma és el gestor de serveis (*ServiceManager*) que es troba al contenidor principal (*Main-Container*). Quan algun agent necessita d'un servei sempre es posarà en contacte amb ell.

Dins un mateix contenidor, els agents es comuniquen amb els serveis mitjan-

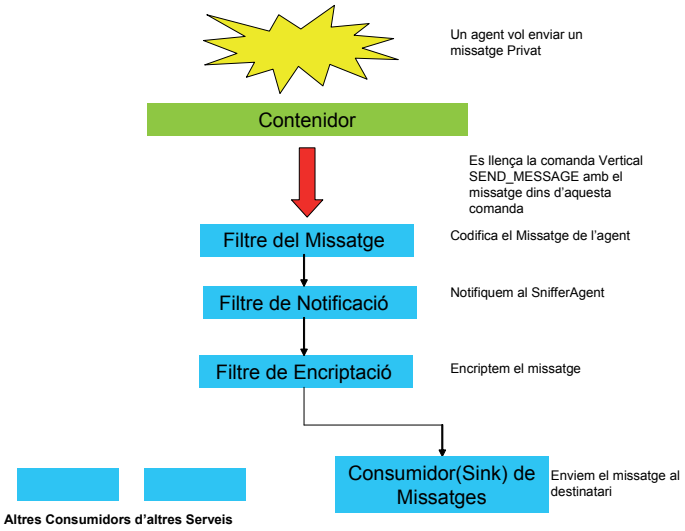


Figura 3.5: Etapes d'una comanda Vertical.

çant les anomenades comandes Verticals. Tots els serveis tenen designats unes comandes. Quan una comanda pertany a un servei passarà per certes etapes, que es poden veure en la figura 3.5 de [10].

Les comandes Verticals primer són processades per els filtres (anomenats dins la plataforma *Outgoing Filter*) del Servei, del qual pertany aquesta comanda Vertical. Els filtres només actuaran amb les comandes que els interessin i amb la resta no faran res. Un cop passats els filtres qui realment consumeix/llegeix la comanda Vertical i no la propaga són els consumidors de comandes verticals, els *sinks*. Cada sink consumeix les comandes verticals que li pertanyin i farà una certa funció amb les dades de la comanda.

Per contra, si un agent d'un contenidor o node es posa en contacte amb un servei d'un altre contenidor diferent del seu, llavors la comunicació es farà mitjançant comandes horitzontals. Llavors intervenen un altre cadena de filtres anomenada *Incoming Filter*, a més d'un *Slice* per filtra les comandes horitzontals. L'Slice de vegades pot consumir la comanda horitzontal i llençar una de vertical amb les mateixes dades o bé directament consumir la comanda horitzontal, tot depenent del cas de cada petició.

El cas de crear una comanda vertical a partir d'una horitzontal es fa servir

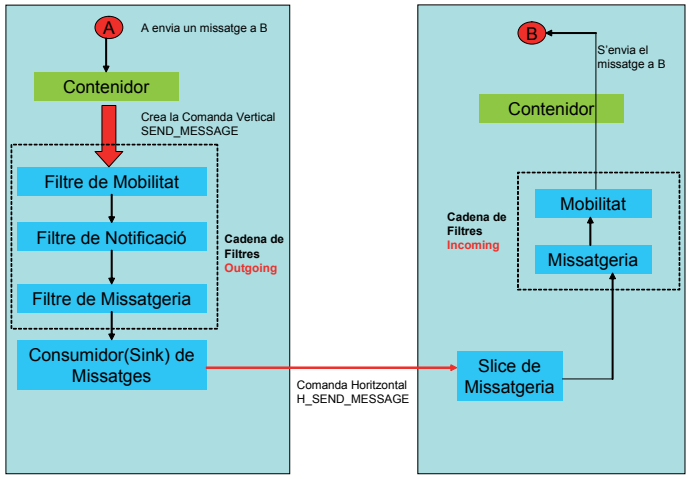


Figura 3.6: Etapes d'una comanda Vertical.

per exemple per enviar un missatge d'un agent d'un contenidor a un altre. Per comunicar-se els dos contenidors utilitzen la comanda horitzontal i per que el contenidor destí pugui enviar el missatge a l'agent desitjat, és necessari crear una comanda vertical per tal que el servei de missatgeria del contenidor destí entregui el missatge a qui correspon. En la figura 3.6 podem veure aquest exemple de [10].

### Servei de Mobilitat Intra-Plataforma de JADE

La mobilitat interna de la plataforma la proporciona el servei de mobilitat d'agents (*Agent Mobility Service*). Aquest servei s'activa cada cop que un agent vol migrar cap a un contenidor de la mateixa plataforma. Aquesta petició es realitzarà mitjançant comandes verticals de tipus *INFORM_MOVED* i la capturarà el servei, ja que l'identifica coma pròpia, tal i com hem vist en la figura 3.5 de les etapes de comandes verticals. Les comunicacions amb altres contenidors o nodes, es realitzen mitjançant les comandes horitzontals, la figura 3.6 que tenim d'exemple, mostra tota la seqüència de etapes per realitzar-ho.

El protocol d'aquesta migració és sota demanda, és a dir, els contenidors destinataris del agents que no tinguin les classes d'aquests, les sol·licitaran del contenidor origen. Però els protocols de migració són propis de JADE i no segueix els

estàndards de FIPA, per tant no ens val com a model ni exemple per fer el nostre protocol sota demanda Inter-Plataforma. Aquest protocol de migració el porta a terme l'agent de gestió del sistema d'agents (*Agent Management System agent AMS*).

### Servei de Mobilitat Inter-Plataforma de JADE V0.97

El servei encarregat de realitzar aquest tipus de migració és el servei de mobilitat Inter-Plataforma (*JADE Inter-Platform Mobility Service JIPMS*).

Quan un agent vol migrar es generarà una comanda vertical *INFORM_MOVED*, el primer filtre que la captura és el del servei Inter-Plataforma i si no la consumeix perquè l'agent vol moures per dins la plataforma, la comanda serà capturada pels altres serveis interns de la plataforma JADE que esperin aquest tipus de comanda. En el cas d'una migració el següent servei intern seria el de migracions Intra-Plataforma. El primer que comprovarà el filtre del servei Inter-Plataforma és si es tracta d'un contenidor extern o intern. Si és intern la deixarà passar perquè el servei Intra-Plataforma resolgui la petició de migració. Si per contra no és una migració interna aquesta comanda vertical és consumida i es llença una altre comanda vertical per tal de suplantar-la. Aquesta nova comanda serà de tipus *INFORM_MIGRATED*. Les tres diferències a primera vista d'aquest tipus de migració externa comprant-la amb la migració interna són :

1. El protocol de migració Inter-Plataforma segueix els estàndards de FIPA, el protocol Intra-Plataforma no.

2. El protocol de migració Inter-Plataforma necessita enviar totes les classes de l'agent al destí encapsulades en un *JAR (Java ARchive)* per tal de executar l'agent. En canvi el protocol Intra-Plataforma és sota demanda.

3. Ara els protocols de migració cap a una altre plataforma els dur a terme no el AMS, sinó l'agent de gestió de mobilitat (*Agent Mobility Manager AMM*). Aquest agent només està present en el contenidor principal, igual que el AMS.

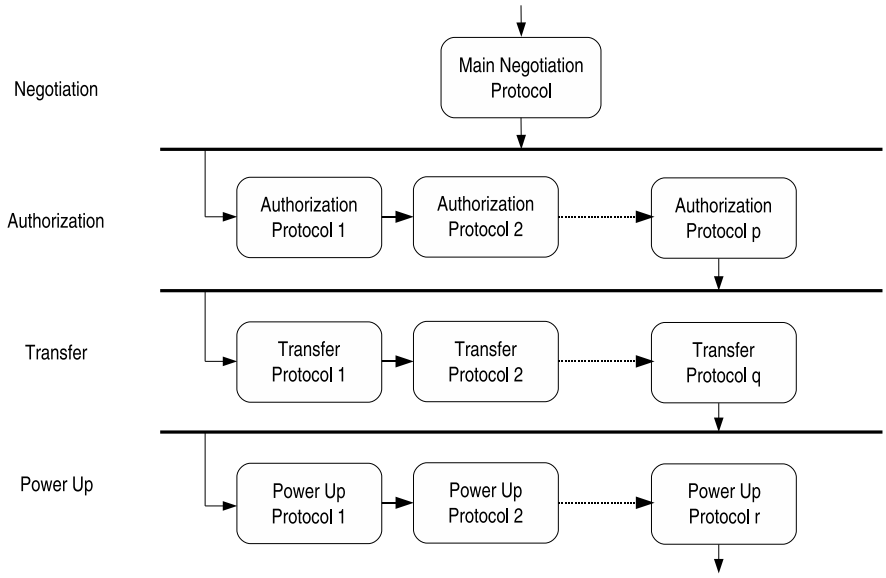


Figura 3.7: Arquitectura Multi-Protocol per la Migració d'Agents.

Per gestionar el *JAR (Java ARchive)* que encapsula les classes de l'agent que ha de migrar, el protocol i el servei utilitzen una classe que és diu *CodeLocator*. Aquesta classe ens serveix per registrar els agents creats o que han immigrat a la plataforma i també hi guardem on està localitzat el seu jar amb les seves corresponents classes. A més, ens serveix per desregistrar els agents migrats o que han finalitzat la seva execució.

El servei Inter-Plataforma segueix una seqüència de protocols necessaris abans de realitzar el protocol de transferència i enviar l'agent. La seqüència la podem veure a la figura 3.7. Aquesta seqüència es realitzarà cada cop que un agent demani migrar a una altre plataforma. Les etapes són les següents :

1. Primer s'executa el protocol de negociació principal el qual servirà perquè les dues plataformes es posin d'acord quins protocols utilitzaran per el control d'accés, per la transferència de l'agent i finalment el d'execució de l'agent anomenat *PowerUp*.

2. Tot seguit s'executen els protocols de Control d'Accés que hagin triat entre les dues plataformes

3. Si hem superat tots els protocols d'accés, realitzarem la transferència de l'agent amb els protocols que que han triat les plataformes en la negociació prèvia.

4. Finalment si tot ha anat correctament, s'inicia el protocol que sol·licita l'execució de la'gent a la plataforma destí.

## 3.4 Java 1.5_X

Ens ha estat necessari, per desenvolupar els protocols de migració sota demanda, aprendre com funcionava la classe de Java que ens permet carregar classes en memòria. Aquesta classe és el *Classloader* (Carregador de Classes). A més d'aprendre el seu funcionament hem hagut de saber com es creava una delegació de classloaders, per tal de poder afegir el nostre en la delegació existent de classloaders de la màquina virtual de Java. Hem hagut de saber com era la seqüència des de que ens falta una classe fins que l'arribem a carregar tot afegint un classloader creat pel l'usuari. En la figura 3.8 podem veure aquest passos i els classloaders de la màquina virtual de Java.

Per saber com realitzar una delegació ens va ser mol útil l'API de Java [17] i per entendre com funcionava i per a que servien els carregadors de classes interns de la màquina virtual de Java de la delegació de ClassLoaders que conté, vam consultar [12].

## 3.5 Eines de Desenvolupament

Explicarem breument quines eines hem utilitzat pel desenvolupament del projecte, com la seva planificació i el seu disseny.

### 3.5.1 Fase d'Anàlisis

Per realitzar la planificació del projecte hem utilitzat l'eina *Microsoft Project 2003*. És un planificador de projectes molt complet i és capaç de generar tant

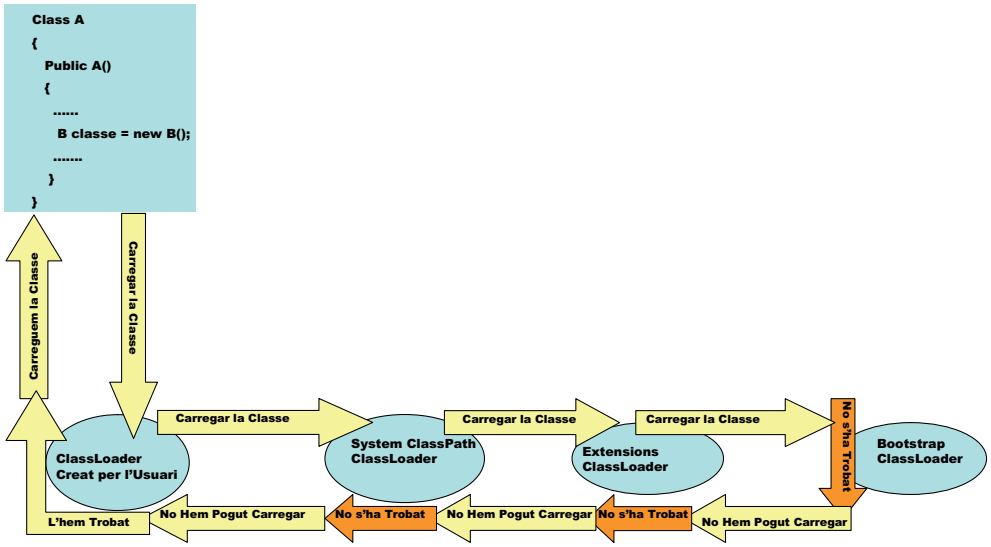


Figura 3.8: Seqüència de passos dins una delegació de ClassLoaders amb Java.

gràfics de Gantt, com l'ús dels recursos, entre moltes altres eines i gràfics. A més, ens permet fer un seguiment del projecte sabent així en quines fases ens hem demorat més i si anem bé de temps o no respecte la planificació principal amb el seguiment.

### 3.5.2 Fase de Disseny

Per realitzar els diagrames de classes i de seqüència hem utilitzat el *Rational Rose Java Professional Edition V7.0.0.0*. És un programa molt complet que segueix el llenguatge de modelat UML [15], a més de poder crear altres tipus de diagrames com els de casos d'ús i d'activitat.

### 3.5.3 Fase d'Implementació

En la fase d'implementació hem utilitzat una eina molt comuna per desenvolupar codi Java. L'*Eclipse 3.2*, és un entorn de desenvolupament integrat (IDE) per la programació de aplicacions Java a més de que és software lliure.

## 3.6 Resum

Hem explicat quin és el significat d'un agent mòbil i també quines són les especificacions de *FIPA*, les classes de protocols d'interacció que utilitzem en el nostre projecte, que són els missatges *ACL* i les ontologies.

Hem explicat també, com funcionen els serveis de JADE i com es comuniquen els agents amb els serveis de la pròpia plataforma i els d'altres plataformes. Per la comunicació entre agent-plataforma hem vist que ens són necessaries comandes Verticals (si és un servei de la pròpia plataforma) o bé comandes Horitzontals (si es tracta de serveis d'una plataforma externa).

Per últim hem vist quina classe de Java hem hagut de buscar informació per poder-la utilitzar adequadament. Aquesta classe és el *ClassLoader* (el carregador de classes) i el més important és com funciona una de legació de carregadors de classes.

# Capítol 4

# Protocols d'Autenticació i Control d'Accés

Tot seguit veurem dos protocols que ens ajuden a fer un control d'accés d'agents dins les plataformes autenticant als propis agents o bé les plataformes. Per introduir-nos a les diferents maneres de autenticar i els dos protocols que hem creat per assolir aquesta autenticació, està la secció **??**. Per a cada un dels protocols veurem les fases d'anàlisi i disseny. També explicarem certes consideracions importants sobre l'implementació dels protocols.

Hem creat dos tipus de protocols, cada un autentica les dues principals entitats dins les migracions, agents i plataformes. L'autenticació sobre els agents l'hem dut a terme amb llistes de control d'accés (*ACL Access Control List*). En canvi l'autenticació de plataformes l'hem implementat mitjançant certificats X.509.

Val a dir, que malgrat en els nostres protocols ho hem implementat així, podríem haver-ho fet a l'invers, és a dir, podríem tenir una llista de control d'accés sobre plataformes i que els agents s'autentiquessin amb certificats de la plataforma o bé de l'usuari que l'hagués creat. A més de poder executar ambdós protocols, un rera l'altre per controlar l'accés tant l'agent com de la plataforma origen.

## 4.1 Protocol Bàsic de Control d'Accés

Aquest protocol està orientat a utilitzar llistes de control d'accés només als identificadors del agents. Una altre cas que podríem haver implementat amb llistes de control d'accés, seria al control dels identificadors de les plataformes. Abans d'explicar les fases per crear el protocol, introduirem que són les *ACLs Access Control List o Llistes de Control d'Accés*.

### 4.1.1 Llistes de Control d'Accés

Les llistes de control d'accés, és un concepte de seguretat per determinar tant privilegis com permisos d'accés. A partir d'aquesta llista podríem guardar tant els usuaris que creen els agents mantenint els rols o permisos d'aquests usuaris transmesos als agents, a la plataforma destí.

També podríem controlar les plataformes autoritzades guardant els seus privilegis o els seus permisos d'accés.

### 4.1.2 Fase d'Anàlisis

Aquí mostrarem la primera fase, on veurem els requisits funcionals necessaris pel protocol.

#### Requisits Funcionals

- Integrar el protocol de control d'accés dins l'arquitectura multiprotocol que podem veure a la figura 3.7 del capitol 3 sobre la tecnologia utilitzada.

- Donar o prohibir accés a agents depenent del seu identificador, l'AID (*Agent IDentificator*).

- Gestió l'accés dels agents mitjançant una Llista de Control d'Accés.

### 4.1.3 Fase de Disseny

En aquesta segona fase podrem observar com està format el nostre protocol i quines classes el componen.

#### Gestió de la Llista de Control d'Accés

Un cop encenem la plataforma carregarà la llista de control en una Hashtable on tenim guardades les tuples <Nom de l'Agent, Permís d'Accés>. El nom de l'agent no és tota la classe AID, només ens fixem amb el nom de l'agent que ens ho proporciona el AID. El permís d'accés només és un String que ens indica Permit o Deny.

El fitxer on tenim guardada la taula, s'afegirà dins del fitxer de configuració de la plataforma, que es passa com a fitxer de propietats quan engeguem la plataforma.

El protocol l'hem fet de tipus Propose, ja que només preguntem si un cert agent pot instal·lar-se en la plataforma i pot accedir als recursos d'aquesta. Només ens han de contestar amb un missatge de denegació o d'acceptació.

#### Com Funciona el Protocol?

En la figura 4.1 podem veure els passos que segueix el protocol de control d'accés quan un agent a sol·licitat migrar.

Primer enviem l'identificador de l'agent a la plataforma destí. Aquesta comprovarà que s'hagi enviat aquest identificador i tot seguit buscarà si aquest identificador li està permès l'accés a la plataforma o no. Si no el trobem dins la llista es considerarà com una negació d'accés.

#### Diagrama de Classes del Protocol Bàsic de Control d'Accés

En la figura 4.2 veiem el diagrama de classes del protocol. Tot seguit explicarem les classes més importants.

Del diagrama de classes podem veure que hi han dos Behaviours un iniciador i un de resposta. Amb aquests dos Behaviours podem crear el protocol. De la

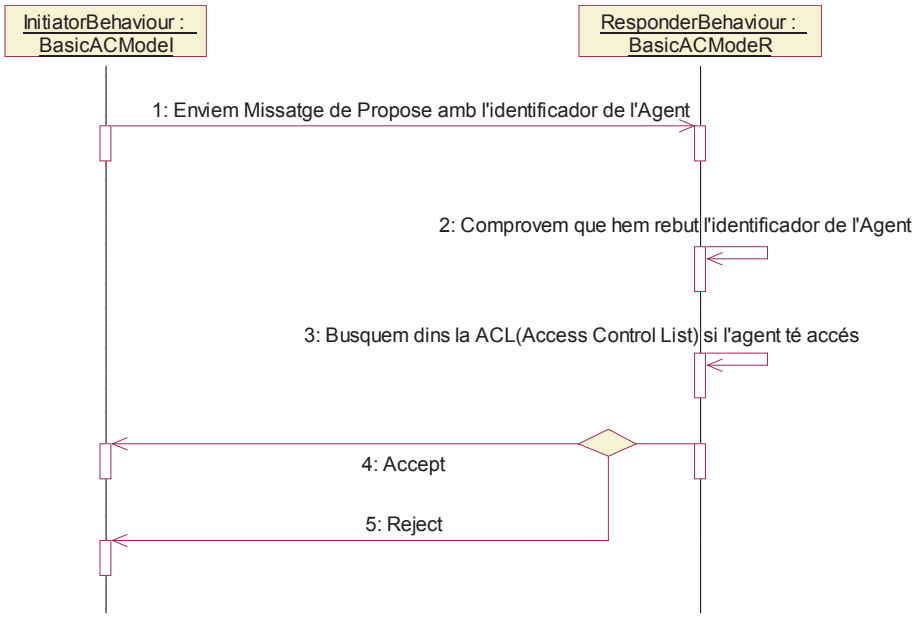


Figura 4.1: Diagrama de Seqüència – Protocol Basic de Control d'Accés.

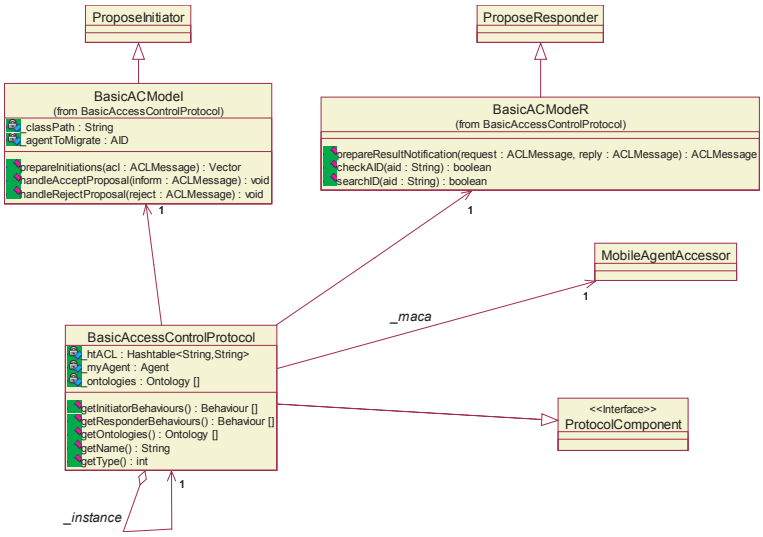


Figura 4.2: Diagrama de Classe – Protocol Basic de Control d'Accés.

figura 4.1 on hem vist el diagrama de seqüència que ens ha mostrat quins passos segueix el protocol, tot seguit els explicarem més detalladament i quin Behaviour participa a cada pas.

- *BasicACModeI*

  Aquest és el Behaviour Iniciador, el qual envia l'identificador de l'agent a la plataforma que ha demanat migrar.

- *BasicACModeR*

  Aquest és el Behaviour un cop rebut el missatge d'inici del protocol de Control d'Accés, comprovarà si el AID del missatge està dins la llista de control o no.

  Si no ho està o bé l'identificador té accés denegat, enviarem un missatge de tipus *Reject*, indicant el motiu de la negació d'accés.

  Si trobem el nom de l'agent i es permet accedir dins de la plataforma enviarem un missatge de tipus *Accept*.

**Ontologies del Protocol**

Perquè els Behaviour de Resposta reconeguin una petició del Behaviour Iniciador ens cal crear una ontologia. Enviarem només l'AID de l'agent a la plataforma destí. Per poder implementar aquesta ontologia ens calen tres classes. En el diagrama de classes de la figura 4.3 podem veure les seves relacions. Tot seguit descriurem quina informació guarden les classes i per a què ens serveixen.

- *BasicAccessControlProtocolAction*

  Aquesta classe que hereta de AgentAction, en aquest cas, ens implementa la gestió de l'identificador de l'agent, l'AID. Aquesta és la classe que afegirem en el missatge *ACL*, a més d'haver-li definit prèviament al missatge quina ontologia emprarem.

- *BasicAccessControlProtocolOntology*

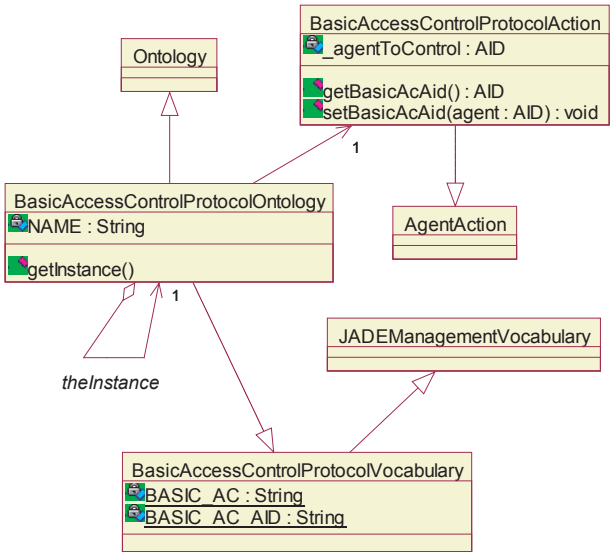


Figura 4.3: Diagrama de Classes de la Ontologia del Primer subprotocol.

Aquesta classe hereta de la classe Ontology de JADE, i ens descriu quin contingut portarà el missatge. El nostre cas hi portarà un esquema d'acció d'agent.

- *BasicAccessControlProtocolVocabulary*

  Aquesta classe hereta JADEManagementVocabulary. Aquí definim el vocabulari que utilitzarem en l'Ontologia. Només hem definit dos conceptes:

  - L'AID de l'agent.
  - L'acció de l'agent.

## 4.2 Protocol de Control d'Accés amb Certificats X.509

A diferència de l'anterior protocol, aquest està orientat a utilitzar certificats X.509 amb una infraestructura de clau pública (*PKI*) per autenticar les plataformes. Hem utilitzat aquesta infraestructura, ja que és la més estesa. Tot seguit explicarem els tipus de certificats que hi han actualment en la secció 4.2.1. Després d'aquesta secció veurem les fases que hem passat per arribar a crear el protocol.

### 4.2.1 Certificats X.509

Els certificats basats amb una PKI *(Public Key Infrastructure o Infraestructura de Clau Pública)* clau pública, és el més estès avui dia. Són els certificats X.509. Sempre hem de confiar en les *CAs (Certificate Authorities o Autoritats Certificadores)*. Aquest certificats ens serveixen per autenticar.

Per utilitzar aquests tipus de certificats és cert que no sempre ens és necessari una PKI i inclús podem crear els nostres propis certificats sense necessitat d'una CA. Però aquests dos elements ens serveixen per poder gestionar i confiar amb els certificats, ja que qualsevol pot fer-se el seu propi.

### 4.2.2 Fase d'Anàlisis

En aquesta fase inicial veurem els requisits funcionals que caracteritzen el protocol amb certificats.

#### Requisits Funcionals

- Integrar el protocol de control d'accés dins l'arquitectura multiprotocol que podem veure a la figura 3.7 del capitol 3 sobre la tecnologia utilitzada.

- Donar o prohibir accés als agents depenent de la plataforma de la qual migren.

- Les plataformes que rebin agents, els donaran accés si confien amb el certificat X.509 de la plataforma origen.

### 4.2.3 Fase de Disseny

Aquí podrem veure la segona fase que ens explicarà quines classes componen el nostre protocol amb certificats.

### Gestió dels Certificats

Un cop engeguem la plataforma, crearem les classes trustStore (magatzem on guardem els certificats en que confia la plataforma) i keyStore (magatzem on guardem els certificats propis de la plataforma). La plataforma obtindrà l'informació per instanciar aquestes classes a partir dels fitxers salvats d'aquestes especificats en el fitxer de configuració que es passa com a fitxer de propietats quan engeguem la plataforma. Dins d'aquest fitxer s'especificarà el *path* dels fitxers dels dos magatzems.

### Com Funciona el Protocol?

La figura 4.4 és un diagrama de seqüència on veiem els passos que segueix el protocol de control d'accés quan un agent a sol·licitat migrar.

Primer extraurem el certificat de la plataforma origen, que emmagatzema el nostre magatzem de certificats de la plataforma, anomenat *KeyStore*. Enviem el certificat a la plataforma destí. Aquesta comprovarà que s'hagi enviat aquesta informació dins el missatge ACL i tot seguit buscarà si aquest certificat està en dins del seu magatzem de certificats en que confia, anomenat *TrustStore*.

Si no el trobem dins el magatzem es considerarà com una negació d'accés, ja que no confia amb aquest certificat.

### Diagrama de Classes del Protocol de Control d'Accés amb Certificats

El protocol l'hem fet de tipus Propose, ja que només preguntem si la plataforma destí permet i confia amb els agents de la plataforma origen a partir del certificat X.509 d'aquesta. Només ens han de contestar amb un missatge de denegació o d'acceptació.

En la figura 4.5 veiem el diagrama de classes, d'on podem extreu-re que hi han dos Behaviours un iniciador i un de resposta. Amb aquests dos Behaviours ja podem crear el protocol.

- *CertificateACModeI*

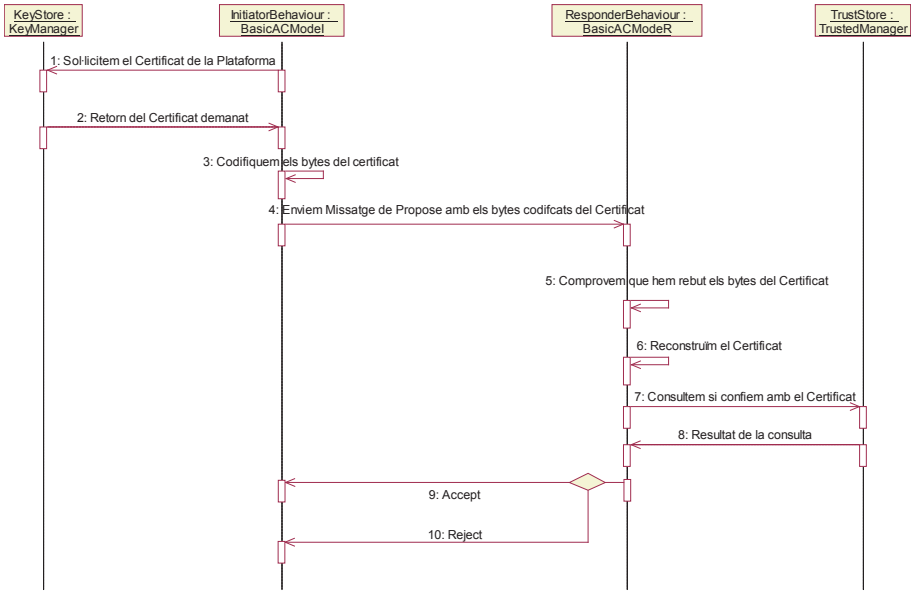


Figura 4.4: Diagrama de Seqüència – Protocol de Control d'Accés amb Certificats.

Aquest és el Behaviour Iniciador, el qual envia l'array de bytes del certificat de la plataforma. El certificat el tenim emmagatzemat dins del keyStore de la plataforma.

- *CertificateACModeR*

  Aquest és el Behaviour que un cop rebut el missatge d'inici del protocol de Control d'Accés, comprovarà si el certificat està dins del trustStore de la plataforma. Si no ho està enviarem un missatge de tipus *Reject*, indicant el motiu de la negació d'accés. Si el trobem enviarem un missatge de tipus *Accept*, ja que confiem amb la plataforma origen.

### Ontologies del Protocol

Perquè els Behaviour de Resposta reconeguin una petició del Behaviour Iniciador ens cal crear una Ontologia. Enviarem els bytes del certificat de la plataforma origen a la plataforma destí. Per poder implementar aquesta ontologia ens calen quatre classes. En el Diagrama de Classes de la figura 4.6 podem veure les seves

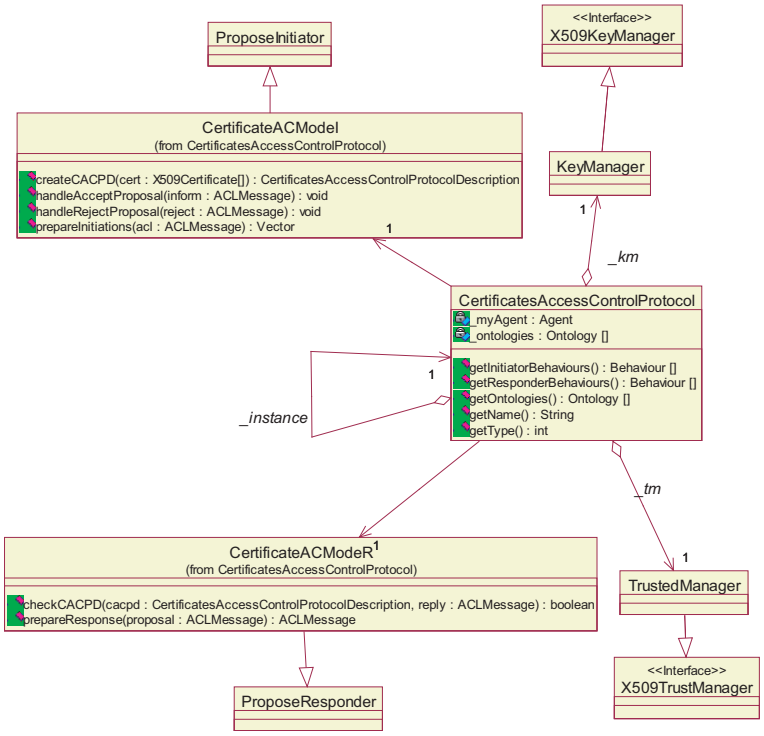


Figura 4.5: Diagrama de Classe – Protocol de Control d'Accés amb Certificats.

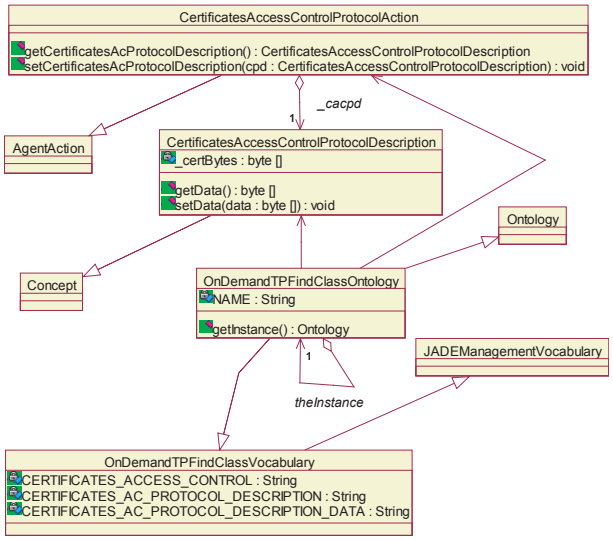


Figura 4.6: Diagrama de Classes de la Ontologia pel Protocol de Control d'Accés amb Certificats.

relacions. Tot seguit descriurem que contenen les classes i per a què ens serveixen.

- *CertificatesAccessControlProtocolAction*

  Aquesta classe que hereta de AgentAction, ens gestiona la Descripció de la ontologia, definida per la classe CertificatesAccessControlProtocolDescription. Aquesta és la classe que afegirem en el missatge ACL, a més d'haver-li definit prèviament al missatge quina Ontologia emprarem.

- *CertificatesAccessControlProtocolDescription*

  Aquesta classe hereta de Concept i ens encapsularà els bytes codificats en Base64 del Certificat.

- *CertificatesAccessControlProtocolOntology*

  Aquesta classe hereta de la classe Ontology de JADE, i ens descriu quin contingut portarà el missatge. En el nostre cas hi portarà un esquema d'acció d'Agent, i dins de l'acció hi incorporem la classe Descripció, que serà un esquema de Concepte.

- *CertificatesAccessControlProtocolVocabulary*

  Aquesta classe hereta JADEManagementVocabulary. Aquí definim el vocabulari que utilitzarem en l'Ontologia. Només hem definit quatre conceptes que emprem dins la nostra Ontologia :

  - La Descripció.
  - L'acció de l'agent.
  - L'array de bytes del Certificat per autenticar la plataforma origen.

## 4.3 Consideracions dels Protocols

Amb l'única dificultat que ens hem trobat al implementar els protocols ha estat poder-los integrar dins del servei i dins l'arquitectura multiprotocol. Això té una raó i és que eren els primers que vàrem fer. Ens han servit per tenir un contacte ferm amb com realitzar i integrar protocols al servei de Inter-Plataforma existent, a més de crear dos protocols per autenticar plataformes i agents i permetre executar els agents dins les plataformes destí, un cop autenticat l'agent i/o la plataforma origen.

Per arribar a poder desenvolupar adequadament el protocols amb certificats vam utilitzar el paquet *IAIK* [9]. Aquest paquet ens permetia més funcionalitats amb els certificats X.509 que els propis que conté l'API de Java.

## 4.4 Resum

En l'inici d'aquest capítol hem explicat els dos tipus de controls d'accés que durien a terme els protocols. Cada protocol creat l'hem basat en un tipus de control d'accés i sobre només un objecte concret, l'identificador de l'agent o les plataformes.

Ara explicarem breument els dos tipus protocols :

- Un està basat sobre la gestió d'una llista de control d'accés del identificadors dels agents.

- L'altre protocol està bastat amb Certificats X.509 per autenticar les plataformes amb una infraestructura de clau pública o *PKI (Public Key Infrastructure)*.

Podríem haver aplicat els controls de manera inversa, si ho volguéssim podríem haver controlat els noms de les plataformes amb llistes de control d'accés i l'autenticació amb certificats podríem haver-la dut a terme sobre els agents.

Després d'aquesta breu introducció als tipus de controls d'accés que hem explicat, hem aprofundit en cada un dels protocols per separat veient cada una de les fases d'anàlisis , disseny i d'implementació corresponents a cada protocol.

En les fases d'anàlisis de cada un, hem descrit les característiques generals que tenen cada protocol i que ens era necessari per fer la seva implementació.

En la fase de Disseny hem pogut explicar a partir de diagrames de seqüència com funcionaven cada protocol. També hem vist diagrames de classes, els quals ens han ajudat a saber quines classe ens eren necessaries i com es relacionaven entre sí. A més de poder veure detalladament quines funcions vistes als diagrames de seqüència duien a terme cadascuna de les classes.

En la fase d'implementació de cadascun dels protocols hem explicat quines dificultats d'integració o de necessitat de paquets de Java externs que ens eren necessaris i perquè.

# Capítol 5

# Protocol de Migració Sota Demanda V1.0

En aquest capitol explicarem com hem desenvolupat el protocol Sota Demanda, com funciona el protocol i quines dificultats ens hem trobat en la implementació.

En la secció 5.1 veurem la Fase d'Anàlisis, explicarem les característiques generals que ha de contemplar el nostre protocol. La secció 5.2, hi presentarem la Fase de Disseny, mostrarem les classes que ens són necessaries per crear el protocol i què ens cal per poder integrar-lo dins del servei de mobilitat Inter-Plataforma. En la secció 5.3 hi explicarem els problemes i les solucions que ens hem trobat al fer les proves d'integració. Sabrem amb quines dificultats ens hem trobat tant a nivell de codi, com a nivell més conceptual i com les hem solucionat. Veurem també dins aquesta secció, les possibles millores que podria tenir aquest protocol, tot introduint-nos en la segona versió del protocol sota demanda.

## 5.1 Fase d'Anàlisis

En aquesta primera fase veurem els requisits funcionals i no funcionals, i les especificacions que deriven dels objectius del projecte, esmentats en la Introducció. Aquest Requisits seran els que caracteritzaran el nostre protocol.

### 5.1.1 Requeriments Funcionals

Ara veurem els requeriments funcionals del protocol que són els següents:

- Crear un Protocol de Migració Sota Demanda que s'integri dins del Add-On de Servei de Mobilitat Inter-Plataforma versió 0.97, creat per la plataforma Jade.
- Seguir l'arquitectura multi-protocol de la migració d'agents.
- Adaptar les comunicacions del Protocol de Migració Sota Demanda v1.0 als estàndards de FIPA.
- No utilitzar la classe CodeLocator per Localitzar les classes de l'agent (veure secció 5.3.2 ).
- Crear un Carregador de Classes (ClassLoader) propi per aquest nou Protocol.
- Crear dues noves Ontologies.
- Sempre es sol·licitaran les classes a la Plataforma que ha creat l'agent.

### 5.1.2 Requeriments No Funcionals

Els requeriments no funcionals són :

- Amb aquest protocol decrementarem la quantitat d'informació enviada entre plataformes, respecte la migració clàssica d'agents.

### 5.1.3 Especificacions

Considerem oportú inicialment centrar-nos en fer el protocol sota demanda només pel cas de migracions entre Main Containers. En cas de tenir suficient temps, ho ampliarem perquè contempli el supòsit de migració des de qualsevol container de la plataforma origen i/o destí.

La sol·licitud i enviament de les classes ho farem mitjançant els missatges ACL entre els AMM (*Agent Migration Manager*) de les dues plataformes, tot seguint els estàndards de FIPA.

Aquest protocol l'adaptarem a l'arquitectura multi-protocol per la migració d'agents [2] i que hem vist al capitol 3 sobre la **Entorn i Eines de Desenvolupament** en la figura 3.7.

El nostre Protocol Sota Demanda és un dels protocols de transferència que veiem en l'etapa de transferència (Transfer). Per tant, els passos previs i posteriors al nostre protocol són :

- En primer lloc s'executarà el protocol principal de negociació per tal de saber si la plataforma destí accepta rebre o no agents.
- En segon lloc, s'iniciaran els protocols de control d'accés.
- En tercer lloc s'executaran els protocols de transferència triats, un dels quals és el Protocol Sota Demanda.
- per últim s'executaran els protocols de PowerUp.

En la plataforma origen, tindrem un procés de cerca de les classes sol·licitades de l'agent executant-se a la plataforma Destí. Com que l'únic lloc on es contenen les classes és en la plataforma origen, podem dir que no hi ha persistència d'aquesta informació per migracions futures. És per això que cada cop que un agent migri, ens veurem obligats a demanar les classes a la plataforma creadora de l'agent.

## 5.2 Fase de Disseny

En aquest apartat explicarem les decisions de Disseny presses per la primera versió del Protocol Sota Demanda. Abans d'aprofundir en les classes, veurem un diagrama conceptual que ens mostrarà el funcionament general del protocol (veure figura 5.1).

En aquesta secció veurem diagrames de classes i diagrames de seqüència, que hem creat en aquesta fase d'anàlisis i que ens ajudaran a entendre com funciona el protocol sota demanda.

En la figura 5.1 podem veure un diagrama que de manera molt genèrica i conceptual que ens introduirà esquemàticament al funcionament del protocol.

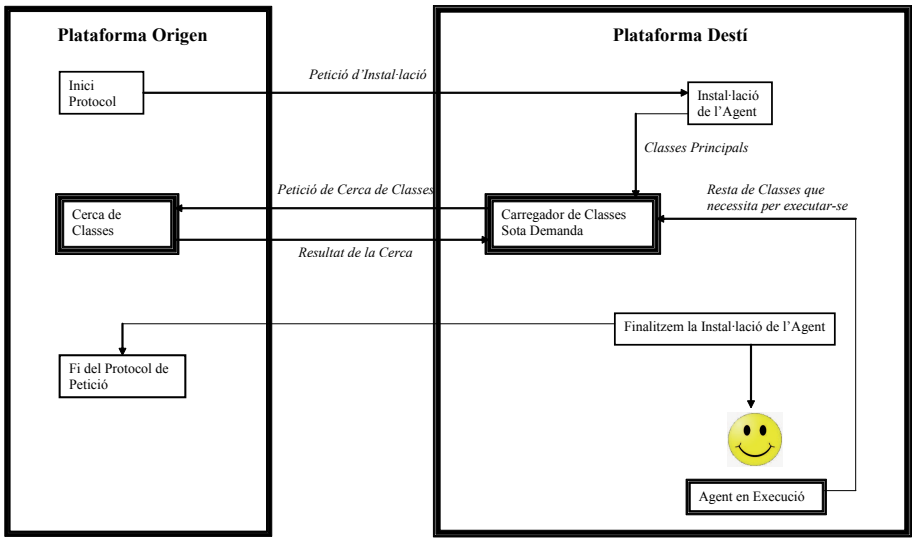


Figura 5.1: Diagrama de Conceptual del Protocol Sota Demanda V1.

### 5.2.1 Protocol Sota Demanda

La primera decisió de Disseny que hem pres és que només enviarem la instància de l'agent al destí, i totes les classes les anirem sol·licitant sota demanda a la plataforma Origen.

Per presentar el Protocol hem creat uns diagrames de seqüència per mostrar el funcionament global del protocol. Tot seguit veurem un Diagrama de Classes que reflecteix les relacions entre les classes que intervenen en el protocol i que hem vist en els diagrames de seqüència en la següent secció.

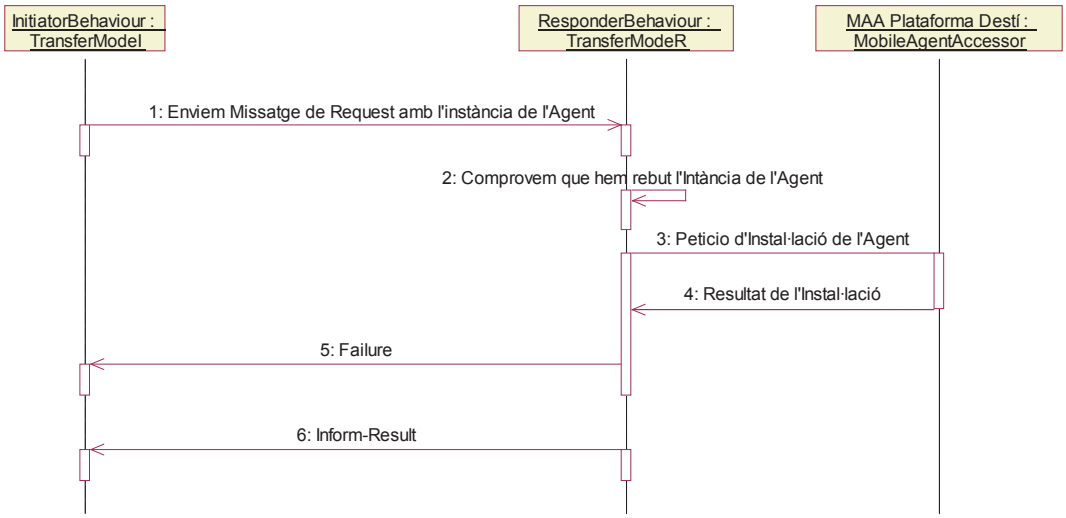


Figura 5.2: Diagrama de Seqüència – Inici del Protocol Sota Demanda V1.

### Com Funciona el Protocol?

Com hem vist en la figura 3.7, referent a l'arquitectura multiprotocol, el primer protocol que s'activa és el de Negociació, per tal de que les dues plataformes es comuniquin amb els mateixos protocols tant de migració com de control d'accés.

El Segon pas comença un cop s'ha finalitzat el protocol de negociació principal el qual ens indicarà quins protocols de control d'accés ,transferència i powerUp s'han triat. En el cas que ens interessa, ens centrarem en el protocol de sota demanda.

Dins aquest segon pas iniciarem el subprotocol que envia a la plataforma destí la instància de l'agent que vol migrar i aquest no acabarà fins que la instal·lació de l'agent no finalitzi. Aquesta seqüència de passos els podem veure en la figura 5.2.

Un cop s'hagi verificat a la plataforma destí que el missatge contenia la instància de l'agent, l'agent s'instal·larà dins la plataforma. Per això és necessari la creació d'una comanda Vertical que llençarà la classe *MobileAgentAccessor* i la capturarà la classe *CommandSourceSink*. Aquesta última classe començarà la instal·lació utilitzant el *Deserializador* i el *carregador de classes* exclusiu pel Protocol Sota Demanda. Tot aquest procés ho podem veure a la figura 5.3.

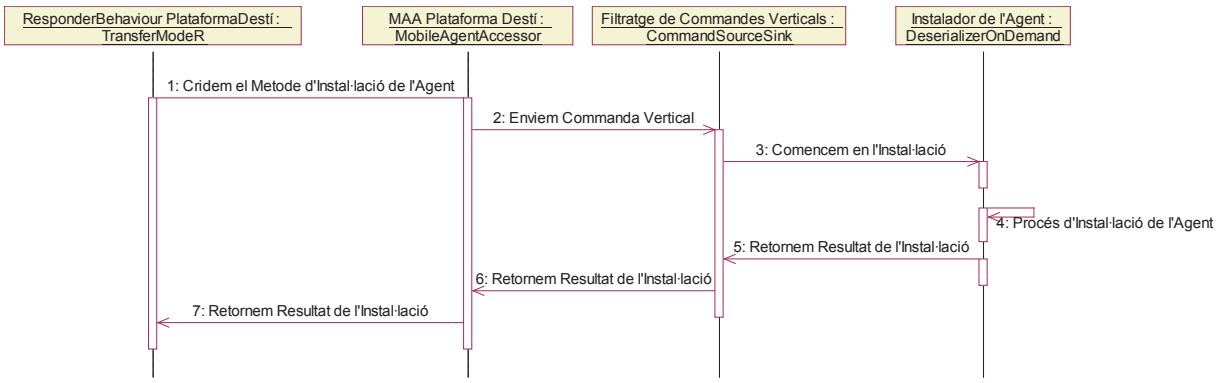


Figura 5.3: Diagrama de Seqüència – Instal·lació de l'Agent.

Mentre instal·lem l'agent en la plataforma destí és molt probable que es sol·licitin classes a la plataforma Origen. Aquesta sol·licitud de classes la satisfarà el segon subProtocol. Aquest és el protocol sota demanda pròpiament dit, ja que és qui implementa la sol·licitud i enviament dels bytes de les classes.

La sol·licitud de classes que hem vist a la figura 5.4, també es pot arribar a iniciar durant l'execució, ja que no només demanem classes a la plataforma origen durant la deserialització de l'agent, sinó que també poden produir-se en temps d'execució.

Quan finalitzem la instal·lació i no hi ha hagut cap error, informarem a la plataforma origen que l'agent ha migrat correctament. Un cop hem rebut aquest missatge, finalitzarem el primer subprotocol. Cas contrari també finalitzarem el primer subprotocol, però avortant la migració de l'agent, tot informant de l'error que s'ha produït. Si la migració ha estat correcte, la plataforma origen que envia la petició d'inici del protocol de PowerUp, per tal de que comenci l'execució de l'agent.

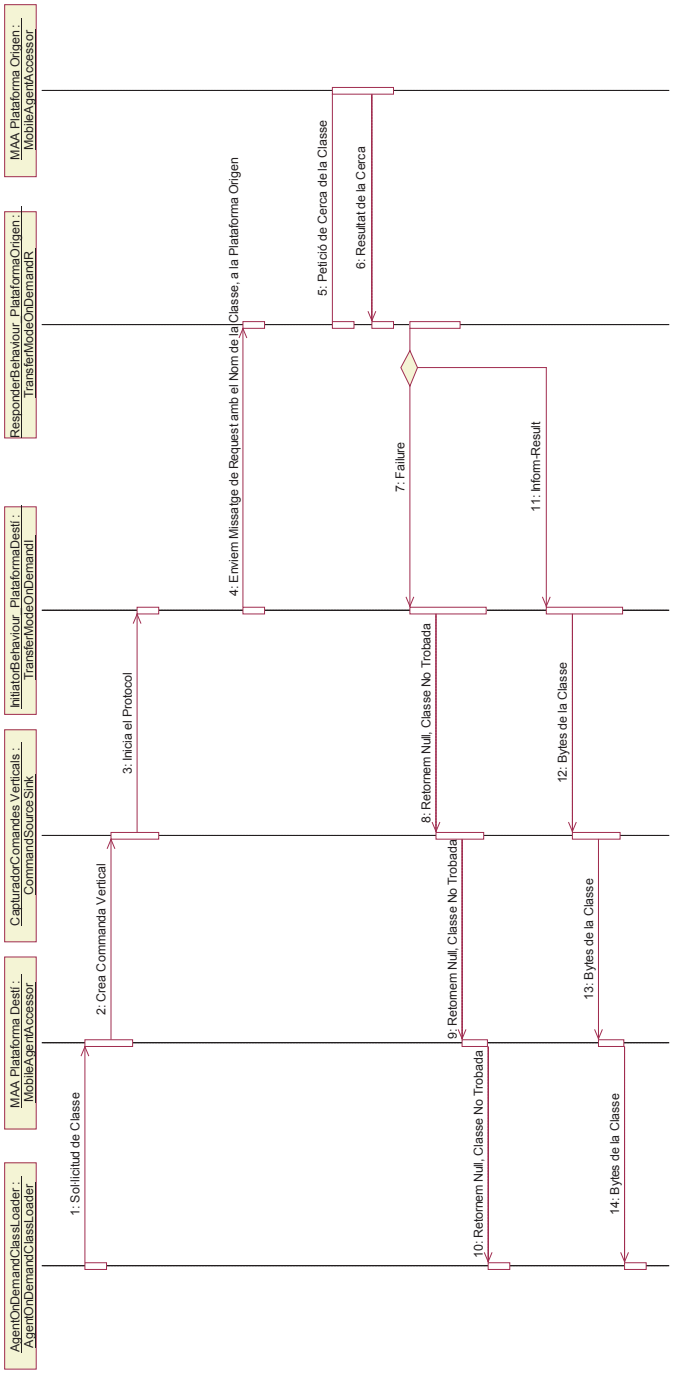


Figura 5.4: Diagrama de Seqüència – Sol·licitud de Classes.

**Diagrama de Classes**

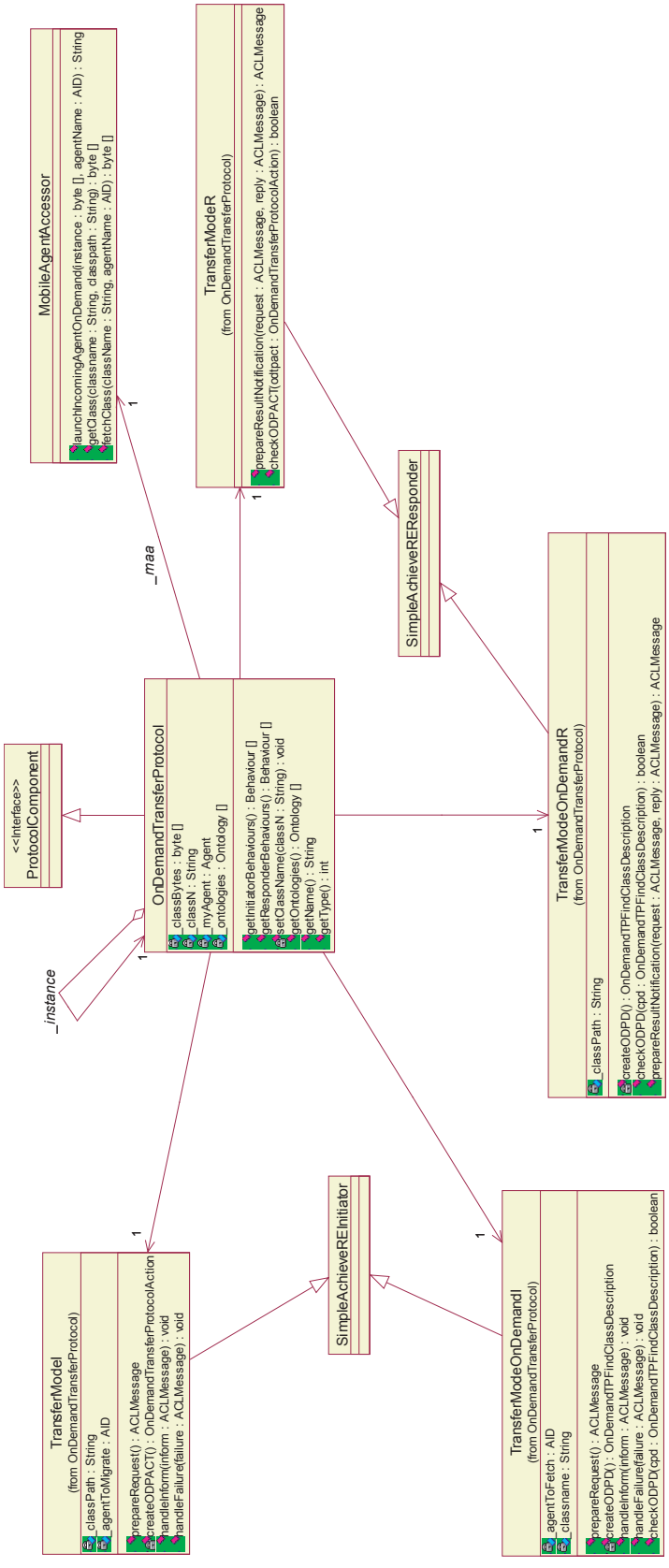


Figura 5.5: Diagrama de Classes del Protocol Sota Demanda V1.

Del Diagrama de classes del protocol (veure figura 5.5) veiem que el Protocol Sota Demanda està format per quatre Behaviours, dos iniciadors i dos receptors. Cada parella de Behaviour iniciador/receptor forma un subprotocol.

Ara descriurem perquè serveix cada un dels subprotocols i els seus corresponents Behviours:

- *Primera Fase, la instal·lació de l'agent*

  Aquest subprotocol l'hem fet de tipus Request (tipus de protocol vist en el capitol 3 sobre la tecnologia utilitzada en aquest projecte) ja que estem fent una petició de migració i instal·lació de l'agent en la plataforma destí.

  - *TransferModeI*

    És el Behaviour Iniciador. Amb aquest Behaviour només enviem la instància de l'agent a la plataforma destí.

  - *TransferModeR*

    Aquest Behaviour ens serveix per informar a la plataforma origen si hem rebut la instància i com ha anat la instal·lació de l'agent a la plataforma destí.

    Si no rebem la instància o hi ha algun problema en la instal·lació de l'agent, s'enviarà una resposta de Failure (d'Error) a la plataforma origen. D'altra banda si tot ha anat correctament també informarà a la plataforma origen perquè continuï en la migració de l'agent.

    Una de les característiques d'aquest Behaviour és que l'hem de dur a terme en un fil de execució paral·lel, si no és possible que ens trobéssim en un cas de *DeadLock* entre la plataforma origen i destí, és a dir, que ambdues plataformes esperen resposta de l'altre i cap contesta.

- *Segona Fase, cerca i enviament de classes*

  Aquest subprotocol també l'hem creat de tipus Request, ja que és el que s'ajusta millor en aquest cas, perquè la plataforma destí *sol·licita* una classe a la plataforma creadora de l'agent. La plataforma origen respondrà a aquesta

petició enviant els bytes de la classe que ha sol·licitat la plataforma on s'està instal·lant o executant l'agent.

- *TransferModeOnDemandI* En aquest Behaviour iniciador només enviem el nom de la classe que ens fa falta. També gestiona la rebuda dels bytes de la classe de la plataforma origen.
- *TransferModeOnDemandR* Aquest Behaviour respondrà a les peticions de classes sota demanda i ens enviarà els bytes de la classe que hem sol·licitat.

Per poder realitzar el Protocol Sota Demanda ens cal carregar les classes en memòria, per realitzar aquest procés crearem un carregador de classes dedicat per aquest protocol.

També necessitem poder instal·lar l'agent a la plataforma destí i és per això que crearem un Deserializtzador exclusiu pel Protocol Sota Demanda, el qual incorporarà el Carregador de Classes creat per aquest Protocol.

### Ontologies del Protocol Sota Demanda

Perquè els Behaviour de Resposta reconeguin una petició d'un Behaviour Inicador ens cal crear una Ontologia. En el nostre Protocol ens calen dues, una per cada subprotocol.

1. **Ontologia del Primer subprotocol**

   Enviament de la instància i instal·lació de l'agent a la plataforma destí. Per poder implementar aquesta ontologia ens calen tres classes.

   En el Diagrama de Classes de la figura 5.6 podem veure les seves relacions. Tot seguit descriurem que contenen les classes i per a què ens serveixen.

   - *OnDemandTransferProtocolAction*

     Aquesta classe que hereta de AgentAction, en aquest cas, ens implementa la gestió de la instància de l'agent. Aquesta és la classe que

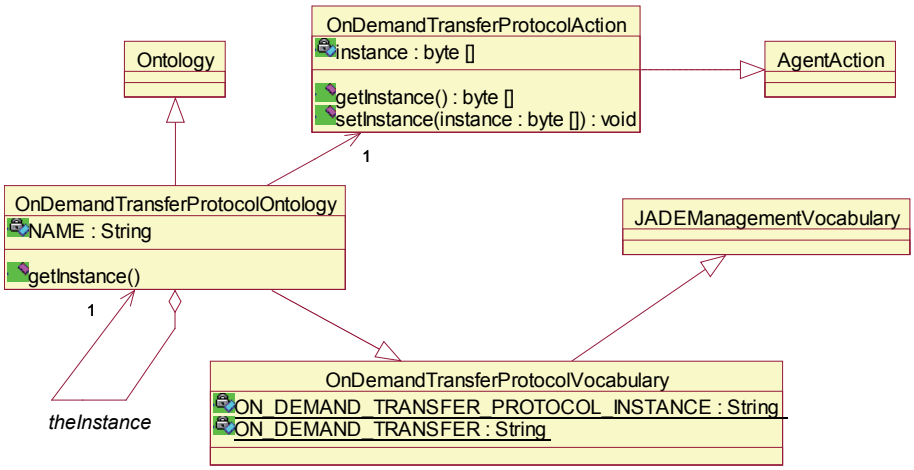


Figura 5.6: Diagrama de Classes de la Ontologia del Primer subprotocol.

afegirem en el missatge *ACL*, a més d'haver-li definit prèviament al missatge quina Ontologia emprarem.

- *OnDemandTransferProtocolOntology*

  Aquesta classe hereta de la classe Ontology de JADE, i ens descriu quin contingut portarà el missatge. El nostre cas hi portarà un esquema d'acció d'Agent.

- *OnDemandTransferProtocolVocabulary*

  Aquesta classe hereta JADEManagementVocabulary. Aquí definim el vocabulari que utilitzarem en l'Ontologia. Només hem definit dos conceptes :

  – La instància de l'agent.

  – L'acció de l'agent.

2. **Ontologia del Segon SubProtocol**

   Petició de la classe que ens cal, enviament i posterior recepció dels bytes de la classe. Per poder implementar aquesta ontologia ens calen quatre classes. En el Diagrama de Classes de la figura 5.7 veurem les seves relacions, i tot seguit descriurem les classes.

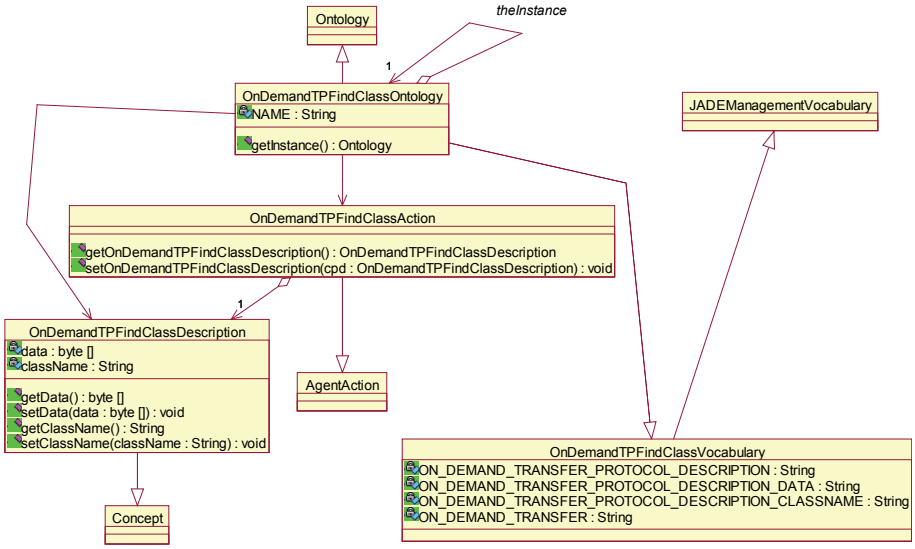


Figura 5.7: Diagrama de Classes de la Ontologia del Segon subprotocol.

- *OnDemandTPFindClassAction*

  Aquesta classe que hereta de AgentAction, ens gestiona la Descripció de la ontologia, definida per la classe OnDemandTPFindClassDescription. Aquesta és la classe que afegirem en el missatge ACL, a més d'haver-li definit prèviament al missatge quina Ontologia emprarem.

- *OnDemandTPFindClassDescription*

  Aquesta classe hereta de Concept, i ens encapsularà el nom i els bytes de la classe que sol·licitem.

- *OnDemandTPFindClassOntology*

  Aquesta classe hereta de la classe Ontology de JADE, i ens descriu quin contingut portarà el missatge. En el nostre cas hi portarà un esquema d'acció d'Agent, i dins de l'acció hi incorporem la classe Descripció, que serà un esquema de Concepte.

- *OnDemandTPFindClassVocabulary*

  Aquesta classe hereta JADEManagementVocabulary. Aquí definim el vocabulari que utilitzarem en l'Ontologia. Només hem definit quatre

conceptes que emprem dins la nostra Ontologia :

- La Descripció
- L'acció de l'agent
- El nom de la classe
- L'array de bytes de la classe sol·licitada

### 5.2.2 Servei de Mobilitat Inter-Plataforma de Jade

Per acabar d'Integrar el nostre protocol dins del servei de mobilitat en que treballem, és necessari que es pugui comunicar amb el servei de mobilitat. Per assolir això hem de Crear/Implementar mètodes per les classes ja existents del servei de mobilitat Inter-Plataforma.

Aquestes classes són :

- *MobileAgentAccessor*
- *CommandSourceSink*

Perquè la plataforma executi les nostres peticions, ens es necessari crear dues *Comandes Verticals* :

- Una ens servirà per les peticions d'instal·lació d'agents.
- L'altre serà per la sol·licitud de classes del protocol.

Hem de crear dues classes més que el servei no conté, que són:

- *AgentClassLoaderOnDemand*
- *DeserializerOnDemand*

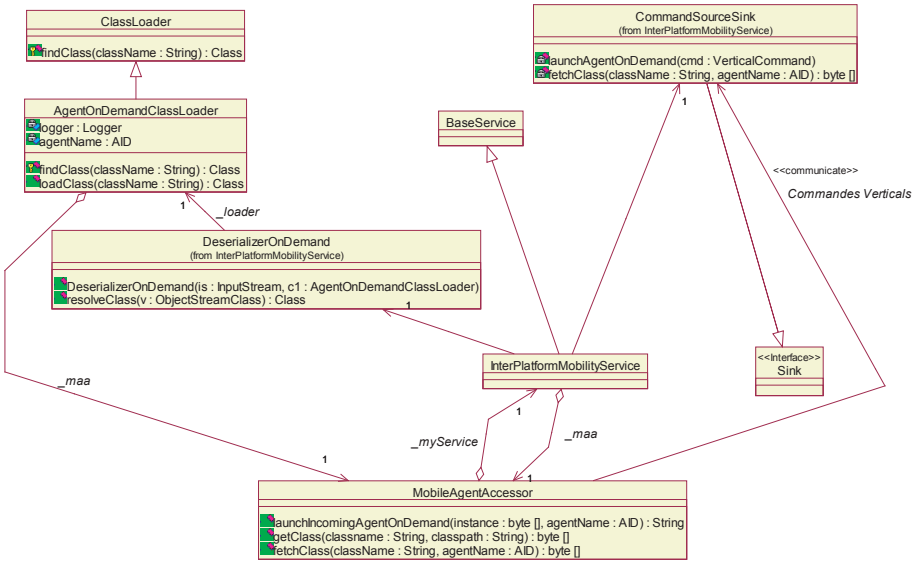


Figura 5.8: Diagrama de Classes – Servei de Mobilitat Inter-Plataforma.

### Diagrama de Classes

### Creació/Implementació necessaris de mètodes a les classes del Servei

En aquesta secció veurem els mètodes i classes que hem hagut de crear per tal de poder integrar el nostre protocol dins el servei de mobilitat Inter-Plataforma.

- *MobileAgentAccessor*

  - Hem creat un Mètode exclusiu per la Migració Sota Demanda anomenat *launchIncomingAgentOnDemand*. En aquest mètode crearem la comanda Vertical, amb el nom *LAUNCH_AGENT_ONDEMAND* ,per la migració sota demanda, la qual indicarà al servei de mobilitat que ha d'instal·lar l'agent.

  - Creem el mètode *fetchClass*. En aquest mètode és on creem la comanda Vertical amb el nom *FETCHCLASSFILE*, per tal de que la classe *CommandSourceSink* la capturi i ens inicia el protocol de sol·licitud de classes.

- Implementació d'un Mètode exclusiu per la Migració Sota Demanda *getClass*. Aquest és el mètode en el qual farem la cerca de la classe sol·licitada, dins la plataforma origen.

- *CommandSourceSink de la classe InterPlatformMobilityService*

  - Hem creat un mètode exclusiu per la Migració Sota Demanda anomenat *launchIncommingAgentOndemand*. En aquest mètode instal·larem l'agent dins la plataforma. Per realitzar-ho utilitzarem el *Deserialitzador* i el *Carregador* de classes que hem creat per la migració Sota Demand. Les classes són el *DeserializerOnDemand* i *AgentOnDemandClassLoader* respectivament.
  - Creació d'un mètode anomenat *fetchClass*, el qual afegirà el Behaviour Iniciador corresponent al protocol de sol·licitud de classes sota demanda a l'*AMM*.
  - El *CommandSourceSink* captura les comandes verticals. En el nostre cas, les nostres dues comandes verticals les crea i les llença el *MobileAgentAccessor*.
  - Quan la classe captura la comanda vertical dedicada a la Migració Sota Demanda,*LAUNCH_AGENT_ONDEMAND*, crida el mètode *launchIncommingAgentOndemand*.
  - Ara bé, quan capturi la comanda vertical per la sol·licitud de classes, *FETCHCLASSFILE*, cridarà el mètode *fetchClass*.

**Creació de noves classes per integrar el Protocol Sota Demanda**

- *DeserializerOnDemand de la classe InterPlatformMobilityService*

  - Aquesta classe hereta de *ObjectInputStream*, i ens serveix per poder llegir els bytes de la instància de l'agent, i poder instal·lar l'agent dins

la plataforma. Cal destacar que l'atribut *_loader*, de tipus *AgentOndemandClassLoader*, és el carregador de classes per la migració Sota Demanda, que tot seguit detallarem.

  - sobrecarreguem el mètode *resolveClass*, de la classe *ObjectInputStream*, i dins del d'aquest mètode cridem el mètode *findClass*, del nostre Carregador de classes, l'atribut *_loader*.

- *AgentOnDemandClassLoader*

  - Aquesta classe la farem heretar de *ClassLoader*.
  - Com a atribut, li hem afegit una classe de tipus MobileAgentAccessor.
  - Sobrecarregarem el mètode *findClass* del *ClassLoader*.
  - Al buscar una classe cridarem el mètode *fetchClass* de la classe *MobileAgentAccessor* que tenim com a atribut, per tal de que ens crei la comanda Vertical i ens retorni els bytes de la classe demanada.

## 5.3 Consideracions de la Fase d'Implementació

En aquesta secció veurem quins problemes hem tingut en la implementació del protocol i explicarem com els hem solucionat. Ho hem dividit en dues subseccions més. Una que ens parlarà de les dificultats al implementar-ho a nivell de codi i una altre sub-secció de les dificultats d'implementació a nivell més conceptual. Aquesta última secció, conté un apartat dedicat a possibles millores d'aquests protocol i una petita introducció a la creació de la següent versió del protocol Sota Demanda.

### 5.3.1 Consideracions a nivell de Codi

Aquí parlarem de les dificultats a nivell de codi que hem tingut al implementar el protocol Sota Demanda.

### Creació del fil d'execució paral·lel pel Behaviour de Resposta

El Behaviour de resposta del segon subProtocol, tal i com hem dissenyat, ha de executar-se en un fil a part. Per això l'hem implementat en un tipus Híbrid, Beahviour-Thread, amb l'ajut de la classe ThreadedBehaviours. Per poder transformar-lo ens cal la classe ThreadedBehaviourFactory. Si no ho implementem així, tal i com vam preveure en la fase de Disseny, l'AMM bloqueja el protocol per sol·licitar i rebre les classes sota demanda, i de retruc les comunicacions que puguin haver d'altres plataformes.

### Codificació de la sincronització al sol·licitar una classe

Una de les dificultats importants que hem tingut en la implementació és la sincronització entre l'inici del protocol de sol·licitud de classe i la resposta de la plataforma origen, en la qual tenim els bytes de la classe que hem demanat.

El problema és que quan afegíem el Behaviour iniciador al AMM en el mètode fetchClass del CommandSourceSink, l'execució seguia i no esperava la resposta de la plataforma origen. La solució va ser sincronitzar un element comú entre el mètode fetchClass del CommandSourceSink, i el Behaviour iniciador del SubProtocol de sol·licitud de classes.

Aquests element a sincronitzar, és el DataStore que hem creat abans de demanar la referència del Behaviour iniciador del segon subprotocol a partir de la instància del Protocol Sota Demanda. Per sol·licitar els Behaviours utilitzem el mètode.

```
getBehaviours(DataStore ds)
```

Per saber quina classe hem de sol·licitar ho passem com a paràmetre quan demanem la instància del protocol:

```
public static OnDemandTransferProtocol getInstance(Agent
myAgent,..... ,String classn)
```

El DataStore l'utilitzem per guardar l'identificador de l'agent que ens ha sol·licitat la classe. Aquest identificador d'agent el necessitem per saber a quina pla-

taforma hem d'enviar el missatge, és a dir, quina és la plataforma origen que ha creat l'agent. Aquesta informació la conté l'AID(Agent Identificator).

La sincronització l'hem dut a terme cridant els mètodes que tot objecte té en Java, el mètode wait() i notify(). Dins del mètode fetchClass, cridem amb el DatStore el mètode wait(). Per complementar-ho i retornar l'execució de la plataforma destí, necessitem un notify(). Aquest mètode ens desbloqueja el wait() fet anteriorment, el qual ens serveix per avisar de que la variable ja es pot llegir i que té contingut.

Aquesta sincronització l'hem implementat així :

*mètode fetchClass de la classe CommandSourceSink*

```
synchronized(ds)
{
        try {
        ds.wait();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
```

*mètode handleInform de la classe TransferModeOnDemandI*

```
    synchronized (_ds)
    {
        _ds.notify();
    }
```

### 5.3.2 Consideracions de la Integració del Protocol dins la Plataforma

Aquesta sub-secció expliquem les dificultats de nivell més conceptuals que hem tingut al implementar el protocol i un cop estàvem fent les proves d'integració dins el servei de mobilitat Inter-Plataforma.

### Causes del problema de sincronització al sol·licitar una classe

Un dels problemes que ens ha enrederit una setmana de la planificació, ha estat la implementació del Behaviour receptor del primer subrotocol, el qual havíem d'implementar-lo en un fil d'execució paral·lel. Ens vam enrederir més del compte ja que no sabíem com sincronitzar-ho de manera eficient, sense aturar la plataforma del tot.

### Problema de sol·lictar classes en temps d'execució de l'agent

La implementació del Disseny proposat fins aquí funciona correctament però, com s'especifica a l'apartat de Disseny, durant l'execució també s'haurien de esperar peticions de classes sota demanda. El problema és que en el Segon subprotocol, utilitzàvem el *CodeLocator* per localitzar el codi de l'agent en la plataforma origen, però quan estem executant l'agent a la plataforma destí, el *CodeLocator* ja ha eliminat l'agent i les seves classes de la plataforma.

Per evitar utilitzar el *CodeLocator* i l'AID de l'Agent en la cerca de classes, hem decidit afegir en el *classpath* un directori on hi estaran les classes del agents, que volem que la plataforma executi. Aquest directori, s'especificarà en un fitxer de propietats (de configuració) que crearem per el nostre protocol, i que al engegar la plataforma el passarem com a paràmetre.

### Raons i Millores del Protocol

Com que hi han classes que s'hauran d'enviar sempre,com la classe principal de l'agent, altres classes definides dins la classe principal de l'agent com poden ser Behaviours entre altres. Per evitar que les tornem a sol·licitar a la plataforma origen i ens les enviï, es dissenyarà la següent solució a aquest problema.

Crearem un protocol sota demanda amb memòria de les classes que han sollicitat les plataformes dels Agents que han migrat, així només caldrà demanarles sota demanda un sol cop, perquè les tindrà la plataforma Destí. Per fer això crearem el protocol de migració sota demanda segura versió 2, que contemplarà l'emmagatzematge de les classes en caches. Dins la cache, per identificar les clas-

ses de manera única i segura farem ús de funcions resum (Hash) amb l'algorisme MD5.

## 5.4 Resum

En aquest capitol hem vist les fases de anàlisis , disseny i implementació que hem dut a terme per crear el protocol.

En la fase d'anàlisis hem vist les característiques generals que té el protocol derivades dels objectius del capitol d'Introducció.

En la fase de Disseny hem pogut veure com funcionava el protocol a partir de diagrames de seqüència que ens mostraven els passos que segueix. També hem vist les classes que hi participen i com es relacionen entre elles. A més, de poder veure quants subprotocols conté i les ontologies que ens són necessaries.

La secció 5.3.2 de consideracions en la integració del protocol, fa referència a la fase d'implementació. Aquí hem pogut apreciar com hem dut a terme alguns dels requisits i com hem solucionat alguns problemes de sincronisme i d'integració dins del servei de mobilitat Inter-Plataforma.

# Capítol 6

# Protocol de Migració Sota Demanda amb ús de Cache V2.0

Com hem vist en el protocol sota demanda V1, en el capitol 5, teníem una mancança. Aquesta mancança és que no guardem les classes que els agents sol·liciten des de la plataforma destí a la plataforma que els ha creat. Si les guardéssim només hauríem de fer aquesta sol·licitud una vegada. Per tant guanyaríem substancialment en temps d'instal·lació i d'execució dels agents. És per això, que crearem una nova versió del protocol en la qual contemplarem l'ús d'una cache de les classes demandes.

La fase d'anàlisis la detallarem en la secció 6.1. Explicarem les característiques generals que haurem d'afegir a la nova versió per tal d'integrar una cache, així com també la gestió d'aquesta.

En la secció 6.2 veurem la fase de disseny, on mostrarem les classes que ens són necessaries per crear la nova versió del protocol i què ens cal per poder integrar-lo de nou, dins del servei de mobilitat Inter-Plataforma.

I per finalitzar veurem la fase d'implementació, en la secció 6.3, on sabrem amb quines dificultats ens hem trobat i com les hem solucionat, per tal de emmagatzemar les classes i gestionar de manera eficient la cache.

## 6.1 Fase d'Anàlisis

Aquesta primera fase veurem els Requisits funcionals i no funcionals i les especificacions que deriven de millorar el protocol Sota Demanda V1, del capitol 5. Aquest Requisits seran els que caracteritzaran la segona versió del nostre protocol.

### 6.1.1 Anàlisis de Requeriments

**Requeriments Funcionals**

Els requeriments del protocol són els següents :

- Emmagatzemar les classes sol·licitades sota demanda dels Agents que han migrat a la plataforma destí.
- Verificar que les classes enviades sota demanda són correctes.
- Gestionar les classe de la cache.
- Crear una petita aplicació per tal de crear el fitxer de configuració tant per la cache de la segona versió del protocol, com per la resta de paràmetres necessaris i comuns per la primera versió del protocol Sota Demanda i altres paràmetres de configuració pels protocols de Control d'Accés.
- Al iniciar el protocol enviarem en una taula, les tuples (hash, nom de la classe), per tal que la plataforma destí pugui saber a priori els hashes actuals i els noms de les classes que possiblement demani l'agent.
- Per evitar l'ús de l'AID, que podria ser manipulat, enviarem també dins la taula anterior de hashes i classes, el nom de la plataforma origen i l'adreça HTTP, la qual l'AMM estarà esperant missatges ACL.

**Requeriments No Funcionals**

Ara presentem els requeriments no funcionals del nou protocol sota demanda :

- Millorar substancialment els temps d'execució i instal·lació dels agents en la migració presentada en l'anterior versió del protocol Sota Demanda.

### 6.1.2 Especificacions

Les especificacions del nostre protocol són :

- El protocol sota demanda versió 2, sempre es demanaran les classes a la plataforma origen, la qual ha creat l'agent, independentment dels *salts* que hagi fet l'agent. Idènticament a l'anterior versió del protocol.

  Aquesta sol·licitud es farà si :

  - En la plataforma destí es requereix una classe de l'agent per instal·lar-lo o executar-lo.
  - En la cache de la plataforma destí no existeixi cap classe amb el mateix Hash. Si existís significaria que la classe ja la tenim, ja que considerem que el Hash és únic per cada classe.

- Guardem les noves classes enviades sota demanda a la cache.

- Si el hash no coincideix però el nom de la classe està emmagatzemat dins la cache, és *sobrescriurà* la classe, ja que podem dir que és una actualització i la classe emmagatzemada era una versió antiga de la classe que ens demanen.

- Eliminació de les classes més antigues. Això ens ho indicarà un paràmetre de límit de temps dins el fitxer de configuració esmentat als requeriments ( veure secció 6.1.1). Aquest paràmetre serà un enter seguit d'una cadena de caràcters indicant que significa, hores,dies,mesos o anys.

- Dins el fitxer de configuració tindrem un paràmetre que ens indicarà la mida màxima de la cache. Si superem un cert llindar eliminarem les classes més antigues encara que no hagin sobrepassat el límit de temps, fins arribar a tenir altre cop la mida adient.

- Podem verificar si la classe demanada sota demanda prové de l'origen o no, és a dir, si no ha estat modificada en l'enviament. Això ho podem saber comprovant els hashes de les classes. Malgrat tot, no és pas un mètode

completament segur ja que, es possible que s'hagi suplantat la identitat al enviar la taula de classes de l'agent amb els hashes corresponents. En cas que no s'hagués suplantat la identitat des de l'inici del protocol, llavors la plataforma ho podria saber si s'intenta després de l'inici. O bé, també podem saber si la classe no arribat correctament per algun error en el medi de comunicació.

- Aquest protocol també està integrat dins l'arquitectura multiprotocol tal i com ho estava la versió previa. Les etapes d'aquesta arquitectura les podem veure en la figura 3.7.

## 6.2 Fase de Disseny

Aquí comentarem les decisions de Disseny presses per aquesta segona versió del Protocol Sota Demanda. Primer de tot veurem un diagrama conceptual que ens mostrarà el funcionament general del protocol(veure figura 6.1).

En les següents apartats d'aquesta secció, podrem veure els diagrames de classes i de seqüència que hem creat i que ens ajudaran a entendre com el protocol gestiona i interactua amb la cache. També ens ajudara a veure quines noves funcionalitats té la segona versió del protocol sota demanda respecte la primera.

Primer presentarem la segona versió del protocol sota demanda, mitjançant diagrames de seqüència i de classe. Després veurem la classe cache, les seves característiques i com la gestionem. I per últim veurem com s'integra la nova versió del protocol i la cache dins el servei de mobilitat Inter-Plataforma.

Abans d'entrar en detall de les classes que componen el protocol, veurem un diagrama conceptual que de manera molt genèrica ens introduirà esquemàticament al funcionament del protocol.

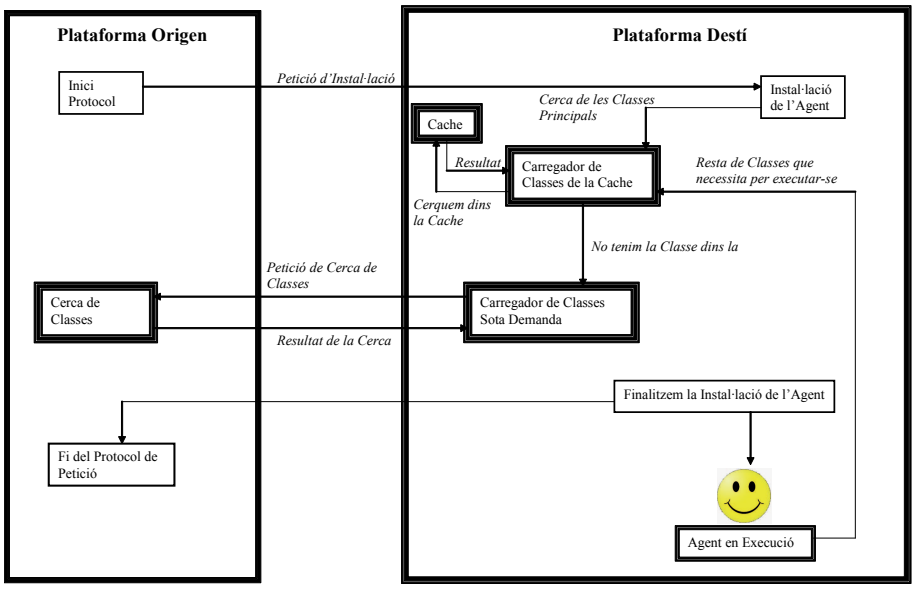


Figura 6.1: Diagrama de Conceptual del Protocol Sota Demanda V1.

### 6.2.1 Protocol Sota Demanda

Per saber els Hash de les classes actuals de l'agent, s'enviarà en el primer missatge, juntament amb la instància, les tuples <Nom de la Classe,Hash>.

També hi hauran les tuples de el nom de la plataforma origen de l'agent, i l'adreça HTTP, en la qual s'han d'enviar els missatges de sol·licitud de classes sota demanda i l'AMM de la plataforma creadora estarà esperant els missatges.

#### Com Funciona el Protocol?

Recordem que dins la migració d'un agent hi han una seqüència de protocols que hem pogut veure en el capitol 5 en la figura 3.7. La primera etapa és el protocol Principal de Negociació. La segon etapa comença un cop s'ha finalitzat el protocol de negociació principal el qual ens indicarà quins protocols de control d'accés ,transferència i powerUp s'han triat. En el cas que ens interessa, ens centrarem en el protocol de sota demanda amb cache, que forma part dels protocols de transferència.

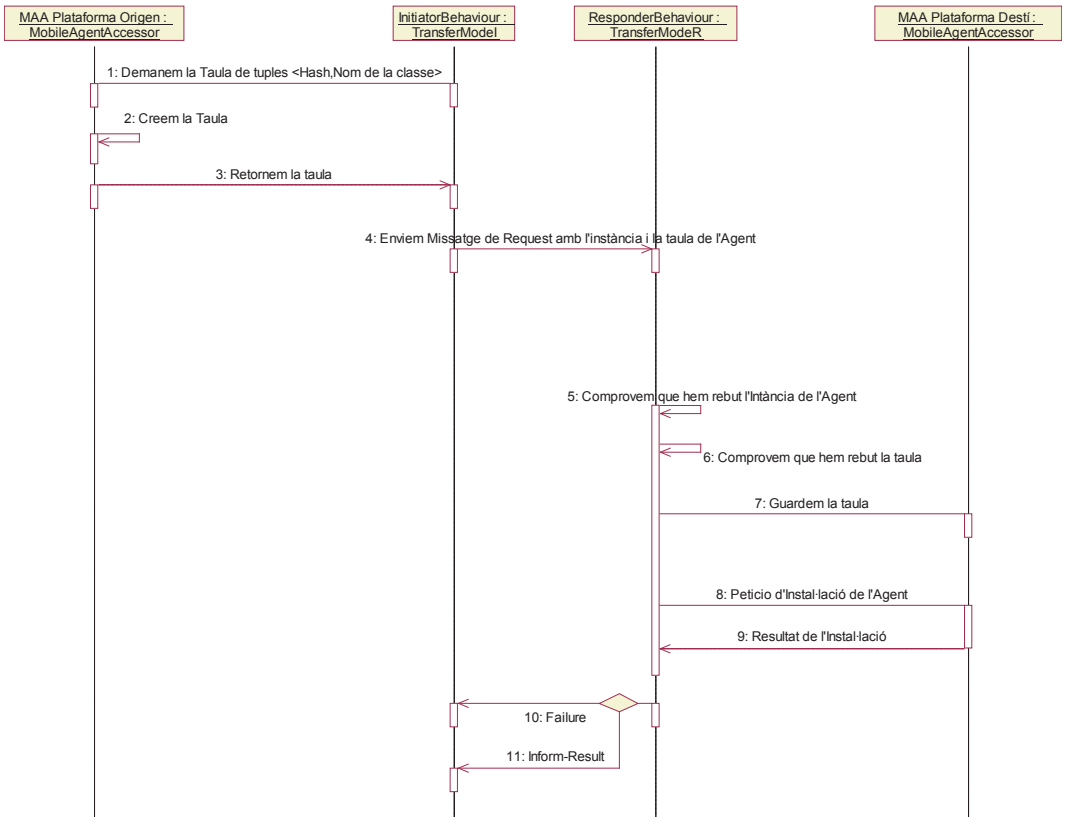


Figura 6.2: Diagrama de Seqüència – Inici del Protocol Sota Demanda V2.

Dins aquesta segona etapa iniciarem el subprotocol que envia a la plataforma destí la instància i la taula de classes (on hi han emmagatzemades les tuples <nom-de la classes, hash>) de l'agent que vol migrar. La taula de classes de l'agent, la guardarem dins la plataforma destí en cas de que l'agent volgués tornar a migrar. Així no hem de demandar la taula a la plataforma origen en cas de que l'agent vulgui migrar.

Aquesta segona etapa no acabarà fins que la instal·lació de l'agent no finalitzi. Aquesta seqüència de passos els podem veure en la figura 6.2.

Un cop s'hagi verificat a la plataforma destí que el missatge contenia la instància i la taula declasses de l'agent, iniciarem la instal·lació de l'agent dins la plataforma.

Per això és necessari l'ús de la comanda Vertical que hem creat en l'anterior

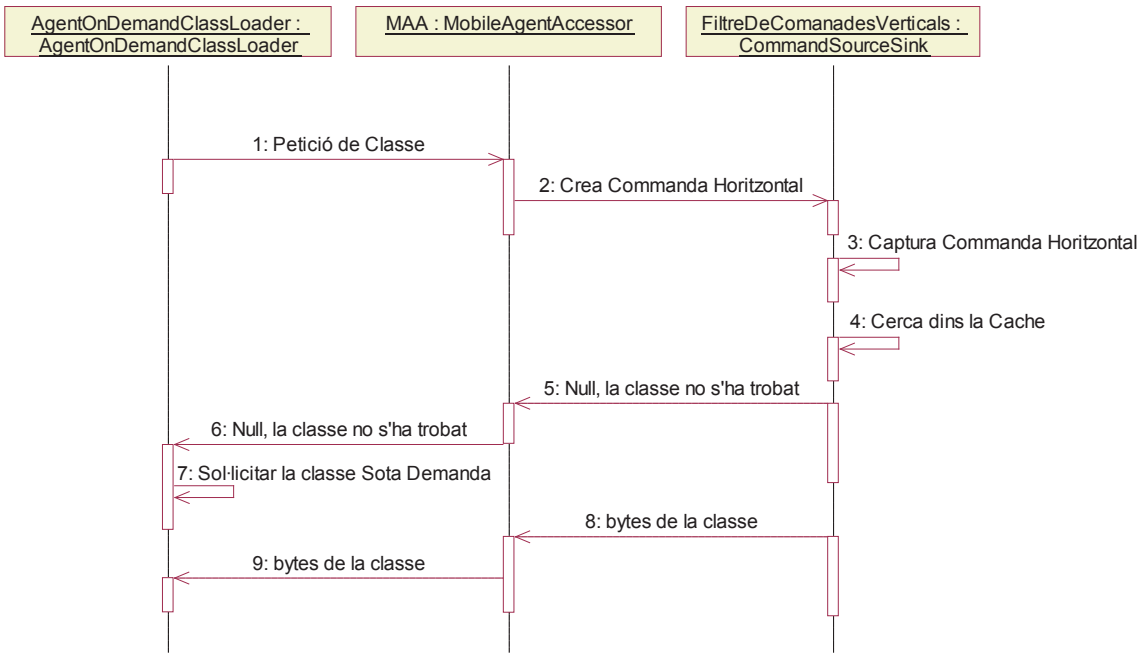


Figura 6.3: Diagrama de Seqüència – Sol·licitud de Classes en la Cache.

protocol. Aquesta comanda la llençarà la classe *MobileAgentAccessor* i la capturarà la classe *CommandSourceSink*. Aquesta última classe començarà la instal·lació utilitzant la classe *Deserialitzador* i el *carregador de classes* exclusius pel Protocol Sota Demanda i creats en l'anterior versió del protocol. Tot aquest procés ho podem veure a la figura 5.3 de l'anterior versió del protocol en el capitol 5.

Mentre instal·lem l'agent en la plataforma destí és molt probable que es sol·licitin classes. A diferencia de l'anterior versió, aquesta sol·licitud primer es farà dins la pròpia plataforma, per tal de saber si la classe desitjada la conté la cache (veure figura 6.3). Si no hi és, llavors la demanarem a la plataforma origen.

Aquesta sol·licitud de classes cap a la plataforma origen, la satisfarà el segon subProtocol. Aquest protocol podem veure el seu funcionament en el capitol 5 en la figura 5.4. La sol·licitud de classes tant a la cache (figura 6.3), com la sol·licitud sota demanda (figura 5.4 del capitol 5), es poden arribar a iniciar durant l'execució de l'agent.

Quan finalitzem la instal·lació i no hi ha hagut cap error, informarem a la pla-

taforma origen que l'agent ha migrat correctament. Un cop hem rebut aquest missatge, finalitzarem el primer subprotocol. Cas contrari també finalitzarem el primer subprotocol, però avortant la migració de l'agent, tot informant de l'error que s'ha produït.

Per altre banda, si la migració ha estat correcte, la plataforma origen envia la petició d'inici del protocol de PowerUp, per tal de que comenci l'execució de l'agent.

Un cop esborrem l'agent de la plataforma, tant sigui perquè ha migrat o bé si ja ha finalitzat la seva tasca, esborrarem de dins la plataforma la taula emmagatzemada al principi del protocol sota demanda.

### Diagrama de Classes

Del Diagrama de classes de la figura 6.4 del protocol sota demanda amb cache, si ho comparem amb el diagrama de classes de l'anterior versió del protocol (veure figura 5.5), podem veure que aquesta versió del Protocol Sota Demanda continua estant formada per quatre Behaviours, dos iniciadors i dos receptors. Cada parella de Behaviour iniciador/receptor forma un subprotocol.

Tot seguit descriurem les funcionalitats afegides respecte el protocol anterior, de cada un dels subprotocols i els seus corresponents Behviours:

- *Primera Fase, la instal·lació de l'agent*

  Aquest subprotocol l'hem fet de tipus Request (tipus de protocol vist en el capitol 3 sobre la tecnologia utilitzada en aquest projecte) ja que estem fent una petició de migració i instal·lació de l'agent en la plataforma destí.

  - *TransferModeI*

    Aquest és el Behaviour Iniciador. Aquest Behaviour envia la instància de l'agent i la taula de classes, que conté tuples <nom de la classe – hash>, a la plataforma destí.

  - *TransferModeR*

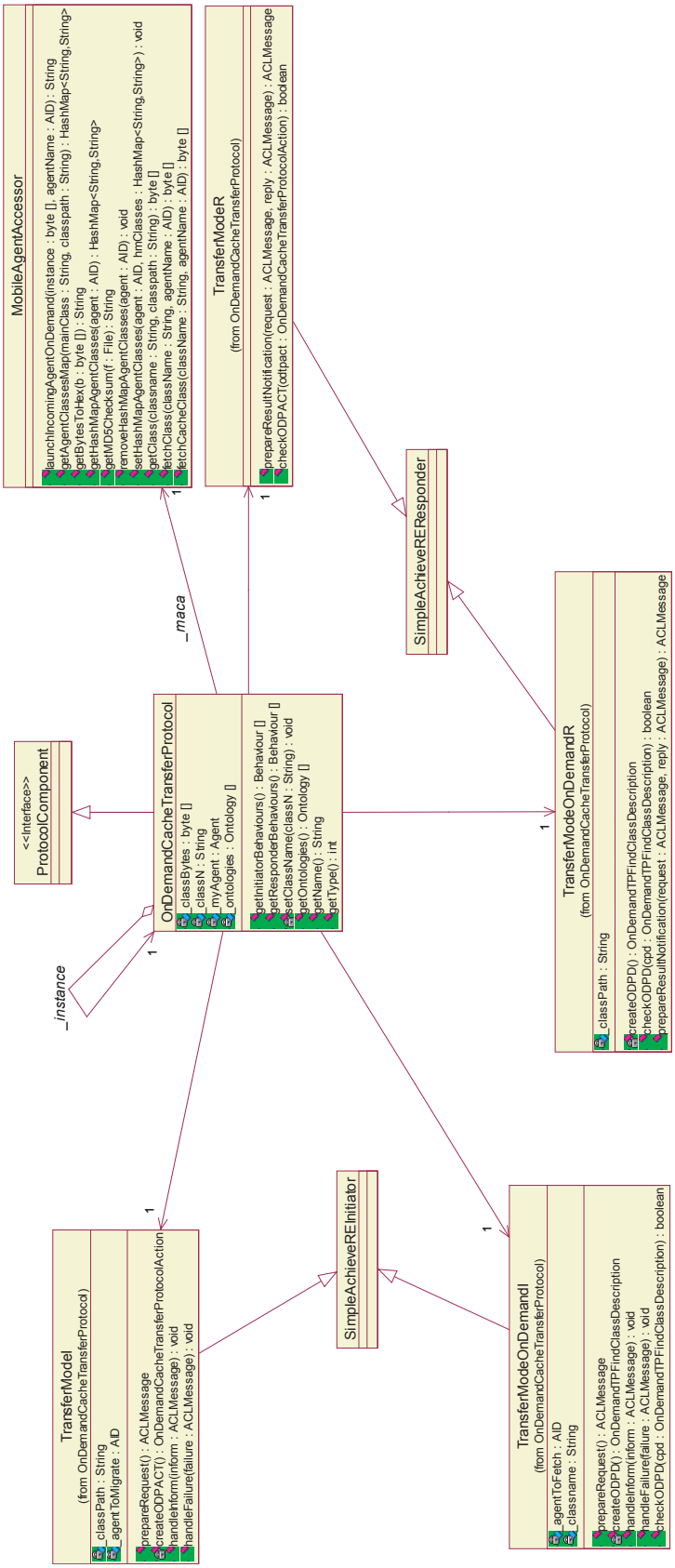


Figura 6.4: Diagrama de Classes del Protocol Sota Demanda V2.

Aquest Behaviour continua informant a la plataforma origen si hem rebut la instància, la taula de classes i com ha anat la posterior instal·lació de l'agent a la plataforma destí.

Un cop revem la petició d'instal·lació, comprovem que tinguem la instància i la taula de classes. Si ho tenim la taula de classes la reconstruïm al destí, ja que en el missatge per seguir l'estàndard de FIPA, no podem enviar un Objecte no atòmic. Podem enviar una classe que pertanyi al package *java.lang* com la classe String, Double, Float, Integer, ... o bé podem enviar qualsevol variable atòmica, que no sigui classe, com per exemple un enter, bytes , booleans, float, double, ...

És per això que la taula s'envia amb una classe String. La cadena de caràcters es pot obtenir a partir del mètode *toString()* que la Hastable té, entre d'altres Objectes de Java que també el tenen.

Si no seguíssim els estàndards de FIPA, en un futur el nostre protocol no es podria comunicar amb altres plataformes ja que de ben segur hi hauran plataformes i agents codificats en qualsevol altre llenguatge que no sigui Java, com per exemple C++.

Si no rebem la instància, la taula de classes o hi ha algun problema en la instal·lació de l'agent, llavors s'enviarà una resposta de Failure (d'error) a la plataforma origen. D'altra banda si tot ha anat correctament també informarà a la plataforma origen perquè continuï en la migració de l'agent.

- *Segona Fase, cerca i enviament de classes sota demanda*

  Aquest subprotocol no ha estat modificat respecte l'anterior versió vista en el capitol 5 en la secció 5.2.1 i només serà iniciat en cas de que no trobem la classe en la cache.

**Ontologies del Protocol Sota Demanda amb Cache**

Com hem vist tenim dos subprotocols però un és exactament el mateix que el de la versió anterior, el subprotocol de sol·licitud de classes. Per tant d'ontologies noves

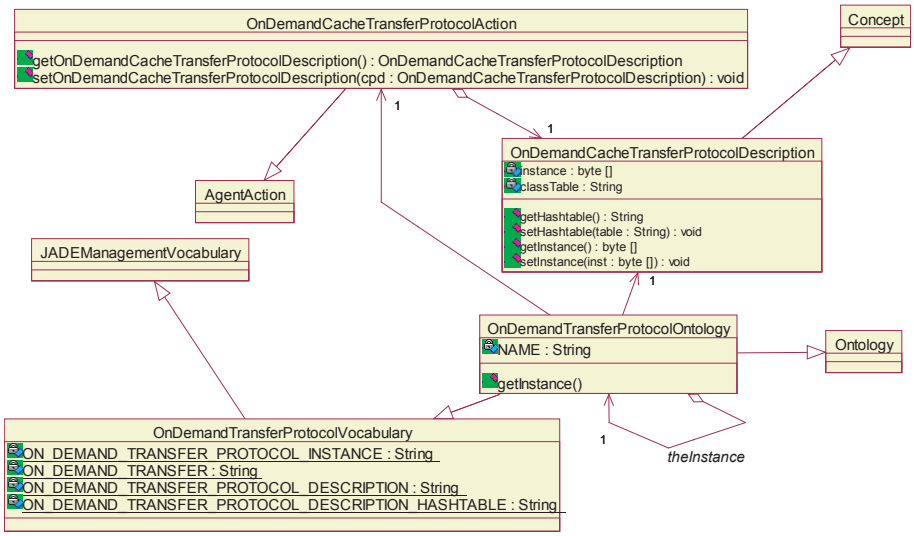


Figura 6.5: Diagrama de Classes de la Ontologia del Primer subprotocol.

només en veurem una, la del primer subprotocol (*Primera Fase, la instal·lació de l'agent*).

L'altre ontologia l'hem deixa't exactament igual i la podem veure en la secció 5.2.1 d'Ontologies del capitol 5 sobre la primera versió del protocols sota demanda.

**Nova Ontologia pel Primer subprotocol**

Com que hem canviat la petició del primer protocol, ara ja no només enviem la instància sinó que també enviem la taula de classes. A més de continuar sol·licitud d'instal·lació de l'agent a la plataforma destí. Per poder implementar aquesta ontologia ens calen tres classes.

En el diagrama de classes de la figura 6.5 podem veure les seves relacions. Tot seguit descriurem que contenen les classes i per a què ens serveixen.

- *OnDemandCacheTransferProtocolAction*

  Aquesta classe que hereta de AgentAction, ens gestiona la descripció de la ontologia, definida per la classe OnDemandCacheTransferProtocolDescription. Aquesta és la classe que afegirem en el missatge ACL, a més d'haver-li definit prèviament al missatge quina Ontologia emprarem.

- *OnDemandCacheTransferProtocolDescription*

  Aquesta classe hereta de Concept, i ens encapsularà la taula de classes en un String i els bytes de la instància de l'agent.

- *OnDemandCacheTransferProtocolOntology*

  Aquesta classe hereta de la classe Ontology de JADE, i ens descriu quin contingut portarà el missatge. En el nostre cas hi portarà un esquema d'acció d'agent, i dins de l'acció hi incorporem la classe Descripció, que serà un esquema de Concepte.

- *OnDemandCacheTransferProtocolVocabulary*

  Aquesta classe hereta JADEManagementVocabulary. Aquí definim el vocabulari que utilitzarem en l'Ontologia. Només hem definit quatre conceptes que emprem dins la nostra Ontologia :

  - La Descripció
  - L'acció de l'agent
  - La taula de Classes
  - L'array de bytes de la instància

### 6.2.2 La classe Cache

Tot seguit presentarem en detall la classe Cache, la qual ens gestiona les classes guardades a dics.

#### Diagrama de Classe

En la figura 6.6, podem veure un diagrama de classes on esta representada la classe Cache.

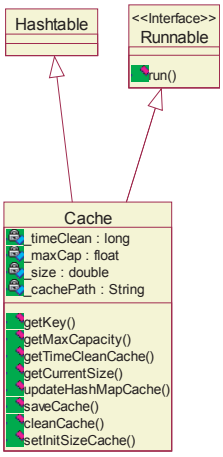


Figura 6.6: Diagrama de Classe de la cache

**Característiques Generals de la classe Cache**

En el diagrama de classes representat en la figura 6.6, veiem que la classe cache hereta de Hashtable. La classe Hashtable ens ajuda a gestionar una estructura de tuples <clau única, valor de la clau>, justament la informació que volem emmagatzemar nosaltres dins la cache <nom de la classe, hash>.

Una altre característica que ens proporciona, és que sabem que mai cap clau ni valor serà null. A més podem guardar la informació que volem dins aquesta classe i gestionar-la amb molta facilitat.

Del diagrama de classes de la cache podem observar que implementa Runnable. La raó és que la cache porta un fil d'execució apart que cada 5 minuts guarda el seu contingut en un fitxer.

**Gestió de les classes dins la Cache**

Les classes es guardaran en un directori, el Directori de la cache. Aquest no estarà inclòs al classpath, però el directori d'on es creen i es guarden els agents els agents hi continuarà essent com a la versió anterior del protocol. Aquests directoris s'especificaran en el fitxer de configuració de la plataforma, en el arxiu *çonfig.properties"*.

També, dins el fitxer de configuració hi especificarem una ruta a un fitxer de

recolzament, on guardarem les tuples de les classes i hashes que té la cache. És en aquest fitxer que cada cinc minuts guardem l'estat de la cache a partir del fil d'execució paral·lel que hem parlat en l'apartat **Característiques Generals de la classe Cache** en la sub-secció 6.2.2.

El paràmetre de temps per esborrar les classes que hagin sobrepassat aquest límit de temps, estarà contingut dins el fitxer de configuració.

Al engegar la plataforma s'especificarà com una Propietat d'aquesta. Cada cop que engeguem la plataforma, per tal de poder recuperà les classes ja emmagatzemades es llegirà el directori de la cache especificat en el fitxer de configuració i es carregaran les tuples <Hash, Nom de la Classe>.

Si existeix el fitxer on tenim la copia de recolzament (*backup*), al guardar dins la cache, les tuples <nom de classe, hash> de les classes llegides del directori, comprovarem que el hash sigui el mateix i la classe no ha estat modificada.

Amb el fitxer on salvem les dades de la cache sempre hi confiarem, ja que es un fitxer dins del disc local i suposem que està adequadament protegit.

Com que el fitxer es salva cada 5 minuts, podrien haver-hi incongruències. Per tant hem de contemplar que :

- Si el fitxer està buit o li falten classes i en el directori on tenim les classes emmagatzemades existeixen, aquestes seran eliminades, ja que abans de apagar la plataforma s'havia salvat el fitxer sense aquestes classes. Per tant podem pensar que són classes que s'haurien d'haver eliminat.

- Si per contrari tenim una classe de més dins el fitxer de la cache i no la tenim en el disc, serà eliminada de la llista.

**Algorismes pel Càlcul del Hash**

Els hash de les classes es calcularan amb l'algorisme MD5 (de 128 bytes). També es possible calcular-ho amb qualsevol altre algorisme, però llavors el protocol consumirà més temps en el càlcul dels hashes que en la propi enviament de l'agent. Vam provar d'utilitzar l'algorisme SHA-512 i se'ns van duplicar els temps, amb el SHA-256 s'augmentava el temps en un 50% i amb el SHA1 (de 160 bytes)

se'ns augmentava un 20%. Vist aquests resultats i malgrat que l'algorisme MD5 actualment sabem que pot tenir atacs mitjançant col·lisions, hem considerat que és suficient i creiem que la probabilitat de col·lisions és molt petita ja que les classes de Java són de mides molt reduïdes, entre 3K i 18K bytes.

Tot i això, dependrà de si considerem que hem de protegir més les classes per garantir que provenen de la plataforma origen i que no han estat manipulades. Si això no ho considerem crític millor utilitzat MD5.

**Guardar les classes en la Cache**

Quan hàgim demanat una classe sota demanda perquè el hash no s'ha trobat en la cache, aquesta s'afegirà o es modificarà directament a la taula, i es guardarà en disc la nova classe.

En el cas que dins la cache hi hagués un nom de classe que tenim guardat dins, però diferent hash respecte la taula que ens han enviat al principi del protocol, llavors es demanarà a l'origen.

Aquesta classe es sobrescriurà a disc, ja que es pot entendre com una nova versió de la classe i a la cache l'actualitzarem.

**Mètodes Claus en la gestió de la Cache**

*saveCache*

Guardem la cache a disc dins el fitxer especificat, per poder recuperar la cache en cas de que la plataforma caigués i es tornés a engegar.

*cleanCache*

Aquest mètode ens serveix per esborrar les classes que hagin passat el període establert en les propietats de la plataforma si la cache està més del 75% de la seva capacitat màxima. Si malgrat l'esborrat d'aquestes classes, la cache està per sobre de 90% de la seva capacitat, esborrem les classes emmagatzemades més antigues fins arribar al límit.

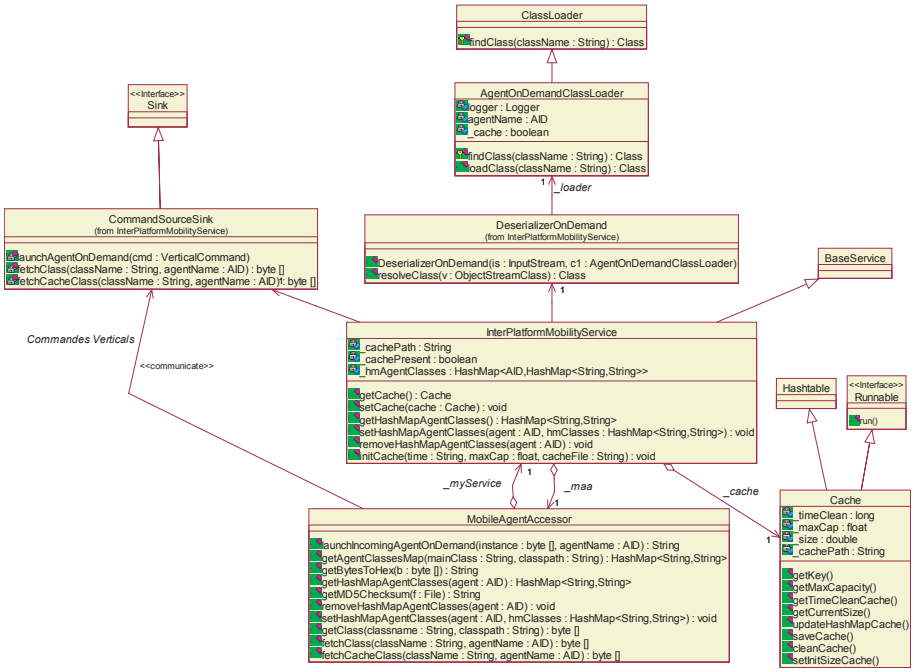


Figura 6.7: Diagrama de Classe del Servei Inter-Plataforma

*updateHashMapCache*

Afegeix dins la cache una nova classe que no teníem. A més comprovem que no estiguem en un 85% de càrrega de la cache. Si és així, cridem el mètode cleanCache. Ho comprovem en el 85% perquè està entre el 75% i el 90% dels límits de càrrega de la cache que comprova el mètode cleanCache, així ens assegurem que esborrarem classes per necessitat de espai en cache.

### 6.2.3 Servei de Mobilitat Inter-Plataforma de Jade

Del Diagrama de classes de la figura 6.7 podem veure els nou mètodes i atributs que s'han hagut d'afegir en el Servei per tal d'incorporà la cache dins la plataforma, comparant amb com havien quedat les classes del servei en la primera versió del protocol( veure figura 5.8 del capitol 5).

- *MobileAgentAccessor*

  Aprofitem el mètode exclusiu per la migració Sota Demanda launchInco-

mingAgentOnDemand, creat en l'anterior versió del protocol. Així com també utilitzem la comanda Vertical i el Mètode exclusiu per la Migració Sota Demanda getClass, creats anteriorment.

Creem el mètode Hashtable<String,String> getAgentClassesMap(String mainClass,String classpath). Aquest ens retorna una classe de tipus Hashtable, la taula de classes, on hi guardem el hash i el nom de les classes que l'agent pot arribar a sol·licitar a les plataformes que no són la origen.

Hem creat el mètode fetchCacheClass el qual crea una comanda Vertical *FETCHCACHECLASSFILE* per tal de que la classe CommandSourceSink la capturi.

- *InterPlatformMobilityService*

  Crearem el mètode initCache, el qual ens inicialitza la cache amb els paràmetres obtinguts del fitxer de configuració, com pot ser el temps limit d'esborrat de les classes, la capacitat màxima de la caché en MegaBytes i la ubicació del fitxer de la cache.

  Dins el mètode, mirem si prèviament existeix el fitxer de copia de recolzament de la cache. Un cop sabem aquesta informació, seguirem els passos que hem explicat a la secció 6.2.2 a l'apartat de *Gestió de les classes dins la Cache*.

  També crearem un mètode d'obtenció de la cache el mètode getCache(), el qual ens retorna la cache actual que tenim.

  Per gestionar les taules de classes dels agents hem creat els següents mètodes :

  – El mètode setHashMapAgentClasses. Aquest mètode ens guarda la taula de classes de l'agent, de tipus Hashtable<String,String>, dins un altre taula de tipus HashMap <AID,Hashtable<String,String», amb l'AID de l'agent corresponent, per tal de si després volem fer consultes poder identificar les taules de manera única dins la plataforma en que ens trobem.

  - El mètode getHashMapAgentClasses, ens retorna la taula de classes de l'agent a partir del AID.
  - El mètode removeHashMapAgentClasses , aquest mètode elimina la taula de classes de l'agent, un cop hagi migrat cap una altre plataforma o bé s'hagi cridat el mètode removeAgent del CommandSourceSink, perquè s'hagi acabat l'execució de l'agent.

- *CommandSourceSink*

  Utilitzarem el del mètode creat en l'anterior versió, el launchIncommingAgentOndemand. Recordem que el CommandSourceSink captura les comandes Verticals creades del MobileAgentAccessor. Quan capturi la comanda Vertical anomenada *FETCHCACHECLASSFILE*, llavors cridarem el mètode *fetchCacheClass*.

  Quan aquest captura la Comanda Vertical de Migració Sota Demanda, crida el mètode launchIncommingAgentOndemand.

  Hem afegit la crida al mètode removeHashMapAgentClasses del InterPlatformMobilityService, en el mètode removeAgent(AID agent), per tal de esborrar la taula de classes de l'agent en cas de finalitzar la seva execució.

  Creació d'un mètode anomenat **fetchCacheClass**, el qual buscarà la classe sol·licitada en la cache de la plataforma. Aquest mètode ens retornarà la classe demanada si existeix en la cache. Per saber si existeix la classe dins la cache, només comprovem si el hash, l'identificador únic de cada classe, està dins la cache. Si el conté directament retornem els bytes de la classe que té aquest hash. No comprovem que el nom sigui el mateix ja que , amb el hash podem identificar de manera única les classes. Si no existeix la classe dins la cache, aquest mètode retornarà un valor null.

  Cal remarcar que hem hagut de modificar el mètode *fetchClass* de l'anterior protocol (veure 5.2.2), per tal de que quan revem una nova classe sota demanda, tot seguit la guardem dins la cache.

- *DeserializerOnDemand*

Utilitzarem sense cap modificació el DeserializerOnDemand creat en la versió anterior.

- *AgentOnDemandClassLoader*

  Hem afegit dins el mètode sobrecarregat findClass, la crida de cerca de classes dins la cache. Aquesta crida la fem mitjançant la classe MobileAgentAccessor executant el mètode fetchCacheClass d'aquesta classe.

## 6.3 Consideracions de la Fase d'Implementació

Presentarem tot seguit alguns dels problemes i les seves respectives solucions sobre la gestió de la classe Cache.

### 6.3.1 Problemes de com guardar a dics les classes

Un dels problemes que vam tenir va ser com guardar les classes a disc. Poden haver-hi noms de classes iguals contant, però de diferents orígens. Dins la cache, no hi havia cap problema, la clau única era el hash i el nom era el valor i no passava res que estigues repetit. El problema estava en com guardar-ho a disc, si tenia el mateix nom, es sobrescriurien i si provenien de diferents orígens, les dues classes s'haurien de mantenir.

Podríem haver guardat la correspondència de nom real i nom del fitxer guardat dins el disc, de les classes. Però per no fer més complicada la classe Cache i complicar la gestió d'aquesta, vam decidir guardar a disc amb la següent formula "*[Nom de la Plataforma Origen]+Nom de la classe*>". Així podem mantenir dues classes amb el mateix nom, si i només si tenen diferent hash, sinó podem dir que les dues classes malgrat de plataformes diferents, són la mateixa.

### 6.3.2 Creació de nous mètodes per satisfer certes mancances

La classe cache hereta de Hastable, però necessitàvem una funcionalitat que aquesta classe no ens satisfeia. Algun cop era necessari l'obtenció del nom de la classe,

la clau dins la taula. Per obtenir-ho vam crear el mètode getKey(String value) , que a partir d'un valor ens retorna la seva clau, en el nostre cas aquest valor és a el hash.

### 6.3.3 Problemes d'esborrat de classes

Un altre problema obtingut en la creació del protocol sota demanda amb caches, ha estat que per fer l'esborrat de classes en un limit de temps, havíem de guardar també quan s'havia creat o utilitzat les classes. Aquesta informació s'havia d'emmagatzemar la cache. Però com en cas anterior, volíem fer-la el més simple possible i per no guardar l'últim dia i hora d'ús o creació de les classes, modifiquem la propietat de dia i hora de tot fitxer directament, amb la classe File i el mètode setLastModified(long milliseconds).

## 6.4 Aplicació per crear el fitxer de configuració del Protocols

Després d'haver creat el protocols sota demanda amb cache, com que hi havien força paràmetres de configuració no sols contant els d'aquest protocol, si no que contant també l'altre protocol de migració sota demanda i els dos protocols de control d'accés, vàrem pensar de crear una petita interfície gràfica on hi haguessin tots els paràmetres necessaris de configuració per els protocols. Així aquesta GUI facilita a l'usuari la configuració dels paràmetres necessaris de tots els protocols que hem integrat dins la plataforma, tant els dos de migració sota demanda com els dos de d'autenticació i control d'accés.

En la figura 6.8 veiem la interfície gràfica d'aquesta aplicació. No hem considerat oportú deixar triar a l'usuari l'algorisme de hash, ja que totes les plataformes han de fer el mateix algorisme, si no mai cap hash coincidiria. Si en un futur es volgués fer híbrid, unes plataformes per defecte utilitzen un algorisme diferent entre elles, es podria afegir en el protocol de Negociació aquest paràmetre a negociar entre les plataformes origen i destí.

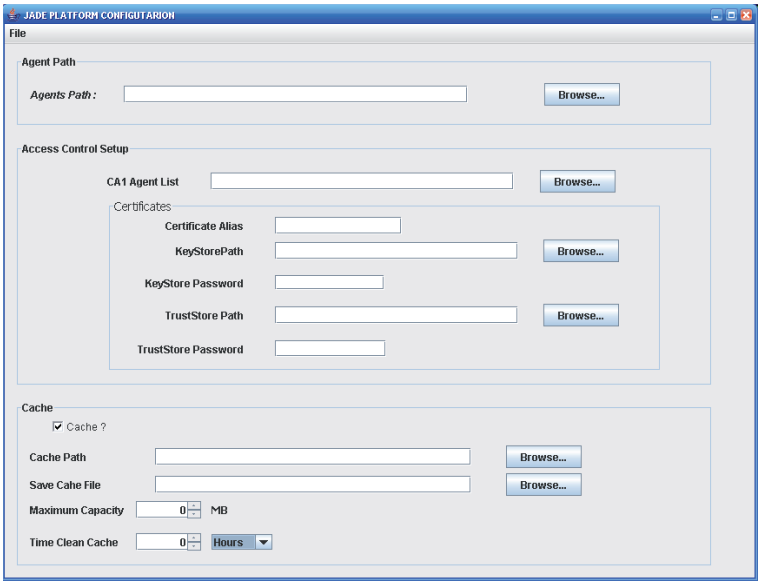


Figura 6.8: Interfície gràfica de l'aplicació.

## 6.5 Tests de Rendiment dels Protocols de Migració Sota Demanda

Ara veurem els tests de rendiment aplicats als dos protocols sota demanda i n 'explicarem els resultats obtinguts per tal de poder veure si la segona versió el protocol ens beneficia o no. Abans presentarem els jocs de proves utilitzats per dur a terme els tests.

### 6.5.1 Joc de Proves

Hem dut a terme els testos amb uns agents determinats del paquet *performance* que hi ha en el mòdul JIPMS. Els agents del joc de proves es configuren amb un fitxer XML en el qual indiquem quin codi o codis volem utilitzar, quantes instàncies volem de cada codi, quantes iteracions de l'itinerari volem i quantes repeticions de tot el procés desitgem. Cal també indicar als agents un fitxer en el qual conté quin itinerari han de seguir. El codi dels agents l'hem creat de tal manera que només creïn (instanciïn) classes i les carreguin en memòria però no

pas que les manipulin, ja que depenent de la classes afectaria als tests. Tots els agents demanaran com a mínim dues classes, en temps d'instanciació (d'instal·lació), sota demanda la classe principal i el seu Behaviour. La resta d'agents a més de les dues classes esmentades anteriorment, uns sol·licitaran en execució 5, altres 10 i finalment altres 25 classes.

Els agents salten (migren) entre tres màquines amb una configuració molt similar, totes tres tenen una CPU Pentium IV 2.4 GHz i 756Mb de Memòria RAM.

Els agents fan el següent itinerari :

1. Es creen els agents a la plataforma origen.

2. Salten al primer destí, s'executen i demanen les totes les classes necessaries.

3. Salten al segon destí, s'executen i demanen les totes les classes necessaries.

4. Finalment tornen al origen.

Per a cada prova és farà servir un únic codi d'agent, el qual demanarà 2,7,12 o 27 classes. Per a cada prova, es crearan 10 agents diferents, 10 instàncies, del mateix codi. Cada agent s'itera 1000 cops per fer l'itinerari descrit anteriorment i finalment tot aquest procés es repeteix 5 vegades.

Com que en memòria tenim 10 instàncies de cop en una plataforma i cada instància, en el cas màxim, tindrà 25 instàncies de les classes que sol·licitades, això fa que podem tenir 270 classes per sol·licitar, és a dir, 270 missatges. És per això que no hem fet proves amb 52 o més classes, ja que eren unes 520 o més classes i al enviar una quantitat tant elevada de missatges la plataforma s'arribava a col·lapsar.

Els agents sempre demanaran totes les classes a l'origen a cada salt que facin, per així poder provar el pitjor dels casos, tests d'estrès. Si només demanéssim algunes classes segur que els resultats no reflectirien la realitat i no podríem determinar en certesa quin protocol podria ser millor.

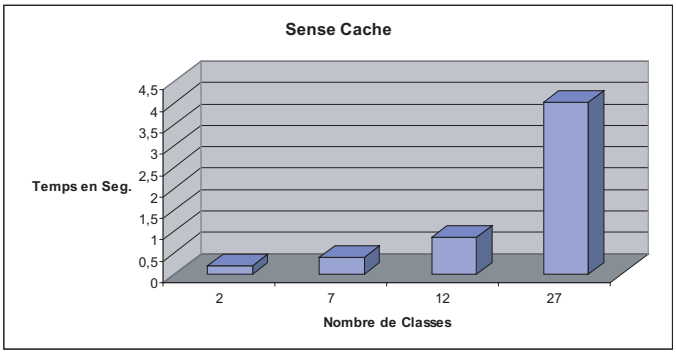


Figura 6.9: Resultats dels Tests Sense Cache.

### 6.5.2 Resultats sense Cache

Podem apreciar, en la figura 6.9, que els resultats són bons, però quantes més classes hi afegim a demanar un agent, es duplica i fins i tot gairebé quadruplica el temps. Això és a causa de que s'estan demanant totes les classes a la plataforma origen des de les altres dues plataformes de l'itinerari i les comunicacions arriben a ésser el coll d'ampolla. A més, que les plataformes tenen un limit de missatges a gestionar, si es sobre passa aquest limit s'alenteixen.

### 6.5.3 Resultats amb Cache

En la figura 6.10, podem veure que els resultats són millors que els anteriors, però que quantes menys classes hi afegim a demanar un agent, pitjor és el rendiment. Aquesta penalització és a causa del temps de càlcul dels hashes de les poques classes que tenim que enviar. A més cal recordar que sempre que migra l'agent enviem la taula de classes i hashes de l'agent. En canvi quantes més classes arribem a enviar els temps s'incrementen mínimament. Si tenim que enviar poques classes el temps de càlcul és superior, en canvi si enviem moltes més classes, el temps de càlcul ens és inferior al temps dedicat a les comunicacions i minimitzem el risc de que la plataforma es col·lapsi al haver de gestionar un cert límit de missatges.

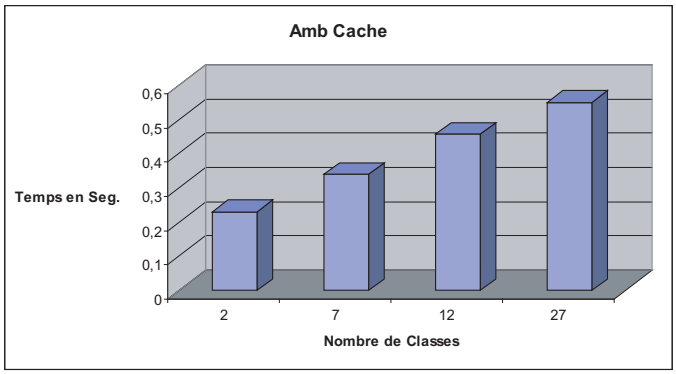


Figura 6.10: Resultats dels Tests Amb Cache.

### 6.5.4 Comparacions entre Migracions Sota Demanda

En la figura 6.11, hi tenim un gràfic on podem apreciar millor les diferencies entre els dos protocols sota demanda. Veiem que quantes menys classes té l'agent surt a compte la versió sense cache, ja que no tenim necessitat dels càlculs dels hashes. Per contra, quantes més classes demanem si no tenim cache, gairebé tripliquem els temps. En canvi amb la cache ens mantenim. Augmenta el temps però molt poc, un increment d'un 25%.

La primera versió del protocol, quantes més classes sol·licita més missatges de petició de classes es creen. La plataforma té suport fins un límit de missatges alhora i en el cas de demandar 25 classes és supera en escreix aquest límit i la plataforma va més lenta al processar les peticions i les respostes. És per això, que quan demanem 25 classes el temps no es duplica sinó que s'augmenta 4,7 vegades els temps de demanar 10 classes.

### 6.5.5 Comparacions entre la Migracions Sota Demanda amb Cache i la Clàssica

Com hem vist en la figura 6.11 la millor migració sota demanda és la segona versió del protocol. Aquesta migració la comparem amb la migració clàssica. En la figura 6.12 podem veure els resultats de comparar-les. Si l'agent necessita 27

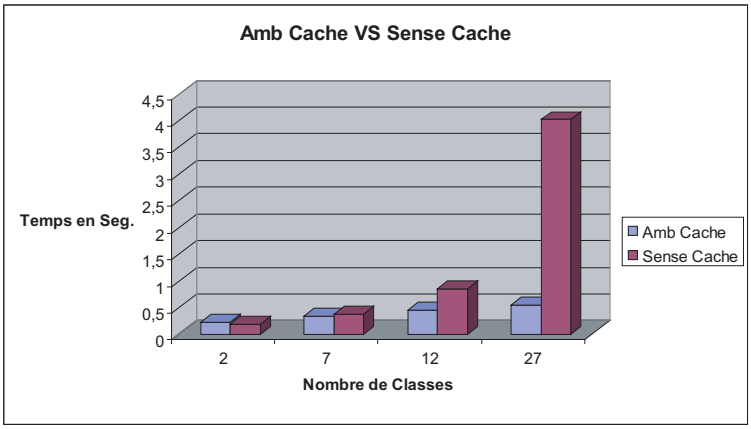


Figura 6.11: Comparació de Resultats dels Tests Amb i Sense Cache.

classes i les demana a totes les plataformes en que migra, podem veure que la migració sota demanda amb cache continua donant un millor temps que la clàssica. Recordem que la migració clàssica sempre que l'agent migra envia totes les classes juntament amb la seva instància. En canvi, quan un agent té 12 classes, que és el nombre promig de classes que durà un agent, veiem que la migració clàssica dona millors temps que la cache. Però aquests resultats aparentment negatius són perquè sempre es demanen totes les classes i la cache penalitza pel fet de comprovar i calcular els hashes de les 12 classes.

En la figura 6.13 podem veure els resultats d'un altre tipus de test. Ara l'agent demanarà també 12 classes, però no totes de cop, sinó que primer en demanarà 5 classes en una plataforma i 5 classes més diferents en una altre. Els agents per a cada migració necessitaran 5 classes diferents més la classe principal de l'agent i el seu Behaviour, en total 7 classes per a cada migració però només dues classes són sempre les mateixes. Al fer aquesta prova podem apreciar que els temps amb cache són millors que si enviem totes les classes i tenint en compte que la meitat no s'utilitzen. Aquesta prova és més realista que les anteriors ja que els agents no sempre necessitaran totes les classes allà on migrin.

Els testos d'estrés, per provar el pitjor dels casos, ens han estat un bon indicatiu per saber quin és el millor protocol de migració sota demanda, però al realitzar

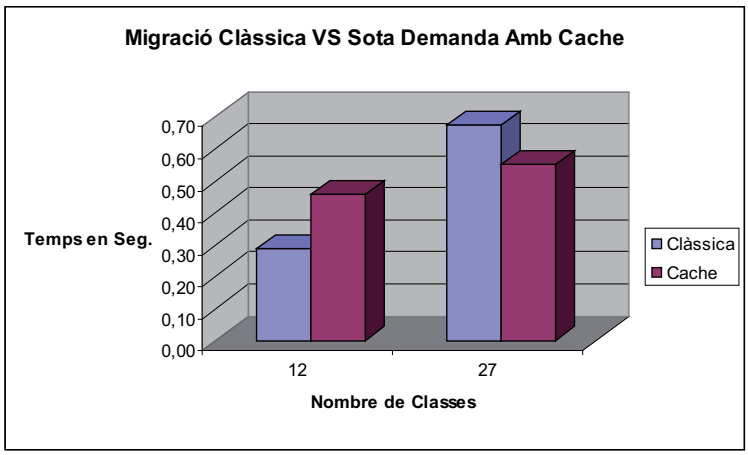


Figura 6.12: Comparació de Resultats dels Tests entre Migració Clàssica i Sota Demanda.

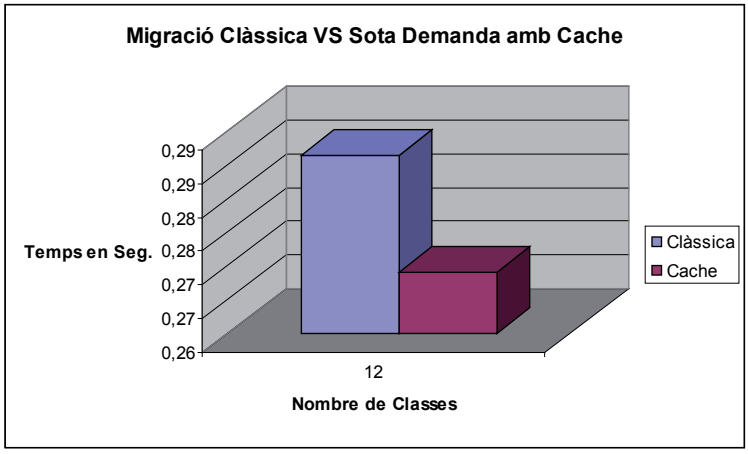


Figura 6.13: Comparació de Resultats dels Tests entre Migració Clàssica i Sota Demanda Amb Cache.

aquest tipus de test per compar el protocol de migració sota demanda amb cache amb la migració clàssica ens ha estat necessari crear aquest anterior test que s'aproximés més a la realitat on els agents no sempre demanen totes les classes. Només ho provem en aquests casos ja que menys de 10 classes era millor la primera versió del protocol sota demanda, a més que un agent que utilitzi poques classes, 5 o menys, resulta millor utilitzar un altre protocol de migració com el clàssic o la migració fragmentada que el protocol sota demanda ja que creem més missatges que la migració clàssica si ho comparem amb la primera versió o bé penalitzem pel càlcul dels hashes si ho comparem amb la cache.

## 6.6 Resum

En aquest capitol hem vist les fases de anàlisis , disseny i d'implementació.

En la fase d'anàlisis hem vist les característiques generals que té el protocol sobre la gestió i la integració d'una cache dins el protocol i el servei de mobilitat Inter-Plataforma.

En la fase de Disseny hem pogut veure com funcionava el protocol a partir de diagrames de seqüència que ens mostraven els passos que segueix, no tots ja que alguns són idèntics a la primera versió del protocol vist en el capitol 5. També hem vist les classes que hi participen i com es relacionen entre elles i molt detalladament la classe cache. A més, hem vist que el nombre de subprotocols no ha variat respecta l'anterior versió, però si la implementació de la ontologia del primer dels subprotocols (Fase d'instal·lació de l'agent).

En la fase d'implementació, vista en la secció 6.3 de consideracions en la creació i gestió de les classes dins la cache, hem pogut apreciar com hem dut a terme alguns dels requisits i com hem solucionat alguns problemes per gestionar de manera eficient i simple la cache.

També hem explicat veure en aquest capitol, els jocs de proves utilitzats per provar els nostres protocols sota demanda. Hem vist els resultats d'aquests test, i podem afirmar que l'ús de cache augmenta el rendiment dels agents ja que podem executar-los amb menys temps.

# Capítol 7

# Conclusions

En aquest últim capitol explicarem les conclusions a les que hem arribat un cop hem finalitzat el projecte, començant per una breu descripció del treball fet i un repàs a l'assoliment dels objectius plantejats al principi. Seguidament parlarem sobre el compliment de la planificació realitzada i el seu seguiment i per finalitzar analitzarem possibles línies de treball futur del nostre projecte completant-lo amb una valoració personal.

## 7.1 Descripció del Projecte

El paradigma dels agents mòbils ha aparegut fa uns anys i ens dona la possibilitat de fer els còmputs allà on es trobin les dades. Així evitem que el coll d'ampolla siguin les comunicacions ja que amb aquest paradigma intentem reduir-ne el seu ús. Només utilitzem les comunicacions quan sigui estrictament necessari com per exemple viatjar d'una plataforma a una altra, sol·licitar classes sota demanda, etc.

El nostre projecte s'ha basat en la mobilitat dels agents i hem creat dos protocols de migració sota demanda per tal de migrar de manera diferent de com s'estava fent actualment. El protocol de migració que estava implementat enviava totes les classes de l'agent juntament amb seva instància a la plataforma destinatària.

La primera versió del protocol de migració sota demanda que hem creat, no-

més envia les classes que els agents necessitin per executar-se i així ens aporta un estalvi en les comunicacions a més d'un millor rendiment en l'execució. Amb la segona versió del protocol de migració sota demanda, utilitzem una cache per tal d'emmagatzemar les classes sol·licitades sota demanda dels agents i evitar els possibles casos en que s'haguessin d'enviar totes les classes dels agents provocant un ús excessiu de les comunicacions.

En la plataforma no hi havia cap protocol de control d'accés dels agents i és per això que també hem creat dos protocols d'autenticació per tal de controlar l'accés dels agents i garantir una mínima confiabilitat amb les plataformes que envien agents i/o amb els propis agents que migrin.

Els protocols creats dins del nostre projecte els hem desenvolupat i integrat dins el mòdul que conté el servei de mobilitat Inter Plataforma, el JIPMS *JADE Inter-Platform Mobility Service* v1.97, per a la plataforma JADE(*Java Agent DEvelopment Framework*). El mòdul JIPMS proporciona als agents mòbils, el servei de mobilitat d'una plataforma cap a una altra.

## 7.2 Assoliment d'Objectius

El primer objectiu que ens vàrem marcar, va ser dissenyar i crear un protocol sota demanda per tal que els agents migressin amb aquest tipus de protocol a qualsevol altra plataforma tot seguint els estàndards de comunicació que proposa FIPA. Aquest objectiu s'ha acomplert i ara els agents migren a qualsevol plataforma i sol·liciten les classes a la plataforma que els havia creat. A més l'hem dissenyat de tal manera que també segueix els esquemes dels protocols d'interacció de tipus request proposats per FIPA.

Un altre objectiu complert és el referent a la creació de protocols de control d'accés. Ne'm creat dos per autenticar agents i plataformes. El primer està orientat a controlar els identificadors dels agents, els *AIDs*, mitjançant una llista de control d'accés ( *Control Access List ACL*) en la qual tenim tuples de noms d'agents i la seva autorització d'execució a la plataforma destí. L'altre protocol autentica les plataformes utilitzant els certificats X.509. Podríem haver creat aquests

protocols intercanviat-se les dues eines de control esmentades i inclús aplicar els dos protocols en serie davant de migracions d'agents.

Addicionalment hem complert amb un altre objectiu que era integrar tots els nous protocols tant els de control d'accés com el protocol de migració sota demanda, dins de la plataforma JADE, més concretament dins del mòdul *JADE Inter-Platform Mobility Service JIPMS* v.1.97.

En la primera iteració del desenvolupament del protocol de migració sota demanda no s'havia assolit l'objectiu de reduir la quantitat d'informació dins del medi de comunicació. Aquesta situació es donava perquè en el pitjor dels casos hauríem de demanar i enviar totes les classes de l'agent, per tant crearíem més flux de missatges que si ho féssim amb la migració clàssica, en la qual s'enviaven totes les classes amb només un missatge.

En la segona iteració per arribar a complir aquest objectiu s'ha millorat la funcionalitat del protocol sota demanda. S'ha resolt guardant les classes que demanem sota demanda dels agents, és a dir, fem cache d'aquestes classes. Amb el disseny i posterior creació d'aquesta segona versió hem acomplert amb un objectiu principal, que era reduir la quantitat de informació dins del medi de comunicació a més de millorar molt el rendiment del primer protocol i hem complert amb un objectiu opcional, que era crear aquest protocol sota demanda fent ús d'una cache de classes. Aquest segon protocol continua seguint els estàndards de FIPA a més d'estar integrat dins del mòdul JIPMS.

## 7.3 Compliment de la Planificació

Respecte a la planificació val a dir que la vam ajustar molt bé al nostre projecte. Han aparegut petites variacions en la durada de determinades tasques sense afectar en absolut el camí crític del projecte ni a la data d'entrega. La tasca d'integració de la primera versió del protocol sota demanda va tenir una duració addicional d'una setmana pels problemes apareguts en la fase d'integració. En canvi es va produir un avançament de cinc dies en l'implementació i integració de la cache, en la segona iteració. Va ser fruit del grau de coneixement adquirit.

La resta de tasques es van realitzar gairebé sempre en el temps previst a la planificació. Les desviacions en tasques d'anàlisis i disseny dels protocols s'han mitigat durant la fase d'implementació utilitzant tres pràctiques de la metodologia *eXterm Programming*, lliuraments curts i continua integració dels protocols i millora del codi, *Refactoring*. La memòria fa estar finalitzada a finals d'abril tot i així hem aprofitat els dies restants del lliurament per fer-hi unes mínimes modificacions i aportacions que hem cregut oportunes per tal de completar-la.

## 7.4 Línies de Treball Futur

Les principals línies futures d'actuació sobre el projecte poden focalitzar-se en dues branques clarament separades.

- Possibles millores i creació de nous protocols de control d'accés.
- Millorar els protocols sota demanda.

Les possibles millores sobre els protocols de control d'accés, consistiria en implementar un o varis protocols utilitzant els certificats d'atributs per així garantir autoritzacions a recursos concrets de la plataforma com podrien ser dades privades, accés a una base de dades entre altres. Per assolir-ho s'haurien de crear protocols basats en polítiques i certificats d'atributs per garantir un control d'accés als recursos de les plataformes o inclús un altre model de control d'accés basat en rols com pot ésser el *RBAC (Role-based Acces Control)*.

Les millores possibles referents al protocol sota demanda serien varies. La principal seria fer més complexa la cache on ara només guardem el nom de les classes que tenim a disc i el hash. Podríem complicar-la tot guardant el temps de creació o últim ús de la classe dins la cache per si el sistema de fitxers no ho contempla o no ens refiem de la data que marqui. També hi podríem guardar l'origen, enlloc de concatenar-lo al nom de la classe.

Es podrien millorar els protocols de negociació per tal de que suportessin sistemes híbrids de protocols de transferència. Potser hi ha plataformes que no farien mai cache de classes, altres que no utilitzarien el protocol sota demanda. En la

situació actual totes les plataformes han de implementar el mateix protocol, cosa que no reflecteix les possibles realitats ja que cada plataforma serà un món amb les seves limitacions i regles.

Una altra millora per fer un bon protocol sota demanda amb cache seria implementar la cache com un servei més de la plataforma i així aplicar la funcinalitat de descobriment de serveis que proporcionen JADE. Les plataformes quan no tinguessin una classe en la seva cache, podrien fer un descobriment de servei de cache a les plataformes de més a la vora que ho proporcionessin. Llavors, les classes dels agents podrien ser gairebé independents de la plataforma creadora. Aquesta independència ens ajuda a resoldre casos com per exemple que la plataforma origen pot arribar a estar molt llunyana respecte a la sol·licitant o bé inoperativa. Això permetria a l'agent continuar executant-se.

Dins el grup de projectistes del *SeNDA* hi ha un projecte sobre els descobriments de serveis en xarxes de tipus ad-hoc amb tecnologia *Bluetooth*. També hi ha un altre projecte respecte descobriment de serveis en xarxes P2P (*Peer To Peer*), concretament seguint el model de JXTA, per les plataformes JADE creat l'any 2006. Altres possibles projectes serien fer-ho amb altres tipus de xarxes P2P com pot ser GNutella. Amb aquesta tecnologia es podria arribar a crear una xarxa de plataformes i ajudaria a saber quins nodes tenim més a prop per descobrir els serveis que proporcionen.

## 7.5 Valoració Personal del Projecte

A nivell personal, projecte ha estat molt beneficiós pel fet d'aprendre una nova tecnologia utilitzant els agents mòbils i conèixer en profunditat com funcionen els serveis de la plataforma JADE, més concretament el servei creat per el grup *SeNDA* el *JADE Inter-Platform Mobility Service (JIPMS)*. Addicionalment hem aprés més coses sobre el llenguatge Java com són els carregadors de classes, els anomenats *ClassLoaders*. Hem hagut de saber com estructura internament la màquina virtual de Java els carregadors de classes del sistema i quins mètodes tenen i saber perquè serveixen cada un i entendre'ls bé abans de decidir quins mètodes

sobrecarregar i quins no.

Actualment el nostre prototip de protocol sota demanda amb cache està en fase d'integració dins del projecte JIPMS v1.98 (18/04/2007), que és una versió més nova d'aquest mòdul on està actualment integrat el protocol, que era el JIPMS v1.97(22/01/2007). JIPMS és un projecte de codi obert, d'àmbit internacional i es pot fer el seu seguiment o participar-hi en el projecte *SourceForge* [13]. A més, el projecte JIPMS forma part del projecte JADE de la companyia TILab el qual també és de codi obert, és encara més ampli i també s'hi està treballant a nivell internacional.

Altrament cal destacar, que bona part del treball en el nostre projecte està contribuint en la elaboració d'un article d'investigació que serà publicat en una conferència internacional i s'està realitzant conjuntament amb personal del *departament d'Enginyeria de la Informació i de les Comunicacions* (*dEIC*) de la UAB.

D'altre banda ens ha estat molt enriquidor haver pogut treballar en un equip com és el grup SeNDA, ja que la seva metodologia de treball i de seguiment dels projectes és molt bona facilitant les tasques de presentació dels nostres progressos davant d'altres professors a més del nostre director de projecte i també davant dels altres projectistes. Saber que el teu projecte podrà ser útil un cop integrat dins la plataforma per algun altre projectista és un altre dels al·licients. Saber treballar en equip i exposar les teves idees davant d'altre persones és una tasca que no dona temps de practicar durant la carrera i que gràcies a aquest projecte hem pogut millorar.

La realització del projecte ens ha permès d'aplicar sobre un cas concret una metodologia d'avaluació i planificació de projectes, *eXtreme Programming* inclosa dintre de les anomenades metodologies àgils les quals cada cop s'estan utilitzant més per desenvolupar projectes amb èxit.

# Bibliografia


[1] Kent Beck and Cynthia Andres. *Extreme Programming Explained: Embrace Change (2nd Edition)*. Addison-Wesley Professional, 2004.

[2] Jordi Cucurull, Ramon Martí, Sergi Robles, Joan Borrell, and Guillermo Navarro. FIPA-based interoperable agent mobility. In *Central and East European Conference on MultiAgent Systems*. Springer, 2007.

[3] FIPA. Foundation for intelligent physical agents (FIPA). `http://www.fipa.org/`, 2002.

[4] FIPA. Propose interaction protocol. technical report 36, foundation for intelligent physical agent (FIPA). `http://www.fipa.org/specs/fipa00036/`, 2002.

[5] FIPA. Request interaction protocol. technical report 26, foundation for intelligent physical agent (FIPA). `http://www.fipa.org/specs/fipa00026/`, 2002.

[6] FIPA. FIPA abstract architecture specification. `http://www.fipa.org/specs/fipa00001/`, 2007.

[7] FIPA. Foundation for intelligent physical agents (FIPA) specifications. `http://www.fipa.org/specifications/`, 2007.

[8] Jim Gray. Distributed computing economics. Technical report, Microsoft Research , Microsoft Corporation, `http://research.microsoft.com/research/pubs/view.aspx?tr_id=655`, Juliol 2003.

[9] Institute for Applied Information Processing and Communication. API del package IAIK. `http://javadoc.iaik.tugraz.at/iaik/_jce/current/index.html`, 2007.

[10] JADE. JADE services architecture. `http://jade.tilab.com/papers/Jade-the-services-architecture.pdf`, 2004.

[11] JADE. JADE (java agent development framework) an open source platform for peer-to-peer agent based applications. `http://jade.tilab.com/`, 2007.

[12] JADE. Java ClassLoader. Understanding the Java Classloading Mechanism. `http://www2.sys-con.com/ITSG/virtualcd/java/archives/0808/chaudhri/index.html`, 2007.

[13] Jordi Cucurull Juan, Ferran Rovira, Guillermo Navarro Arribas, MCarmen de Toro Valdivia. JADE Inter-Platform Mobility Service JIPMS. http://sourceforge.net/projects/jipms/, 2007.

[14] MARISM-A. MARISM-A. an architecture for mobile agents with recursive itinerary and secure migration. `http://www.marisma.org`, 2007.

[15] OMG. UML, unified modeling language. `http://www.uml.org/`, 2007.

[16] Scott W. Ambler. Agile modeling throughout the XP lifecycle. `http://www.agilemodeling.com/essays/agileModelingXPLifecycle.htm`, 2007.

[17] Sun Microsystems. Java api. `http://java.sun.com/j2se/1.5.0/docs/api/`, 2007.

Firmat: Ferran Rovira Tura

Bellaterra, Juny de 2007

## Resum

Aquest projecte consisteix en la implementació d'una migració d'agents mòbils amb sol·licitud de classes sota demanda de manera segura per la plataforma JADE. En aquest projecte hem desenvolupat dos protocols de control d'accés i autenticació sobre agents i/o plataformes. Però la línia central de desenvolupament d'aquest projecte radica en la creació d'un protocol de migració sota demanda d'agents. Hem implementat dues versions d'aquest protocol. La primera versió del protocol de migració es centra en la sol·licitud de classes sota demanda i la segona versió, per tal de millorar-ne el rendiment de la primera, emmagatzema les classes que ha demanat sota demanada en una cache de la plataforma.

## Resumen

Este proyecto consiste en la implementación de una migración de agentes móviles con solicitud de classes bajo demanda de manera segura para la plataforma JADE. En este proyecto hemos desarrollado dos protocolos de control de acceso i autenticación sobre agentes i/o plataformas. Pero la línea central de desarrollo de este proyecto radica en la creación de un protocolo de migración bajo demanda de agentes móviles. Hemos implementado dos versiones de este protocolo. La primera versión del protocolo de migración se centra en la solicitud de clases bajo demanda i la segunda versión, para mejorar el rendimiento de la primera, almacena las clases que ha pedido bajo demanda en una cache de la plataforma.

## Abstract

This project deals with the implementation of a mobile agent migration with request of classes on demand for the JADE platform. In this project we have developed two access control and authentication protocols involving the mobile agents and/or the platforms. However, the main line of work of this project is the creation of a mobile agent migration protocol on demand. We have developed two versions. The first version consist in the request of classes on demand, and the second version consist in the improvement of the performance of the first version, saving all classes requested on demand, into a cache of the platform.