MOBILITAT INTER-PLATAFORMA D'AGENTS SOFTWARE: PROTOCOL DE MIGRACIÓ FRAGMENTADA

Memòria del projecte de final de carrera corresponent als estudis d'Enginyeria Superior en Informàtica presentat per Víctor Sales Barberà i dirigit per Jordi Cucurull Juan.

Bellaterra, Juny de 2007

El firmant, Jordi Cucurull Juan , 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 Víctor Sales Barberà

Bellaterra, Juny de 2007

Firmat:

Al Jose, la Dolors i la Ruth.

Agraïments

M’agradaria donar les gràcies a tota aquella gent que ha fet possible que jo hagi arribat fins aquí.

Gràcies al meu director de projecte Jordi Cucurull, que m’ha ajudat molt en l’elaboració d’aquest projecte i per la paciència que ha tingut amb mi.

Gràcies al grup SENDA per l’oportunitat que m’han brindat de treballar amb ells.

Moltes gràcies al companys del laboratori Franc, Abraham, Jaume i Carlos pels moments que hem passat durant tot l’any.

Per últim, vull agraïr especialment als meus pares que m’han recolzat durant els anys de carrera i que sempre han cregut amb mi. Gràcies.

Índex

Índex de figures

Capítol 1

Introducció

L'evolució de noves tecnologies, que permeten accelerar l'intercanvi d'informació, fa que s'intensifiqui la recerca de models de comunicació pel seu ús en la implementació d'aplicacions distribuïdes.

Aquesta evolució ha esdevingut en un increment de l'ús de paradigmes de programació basats en agents. Els agents són components software autònoms i socials. Tenen un determinat objectiu i reaccionen als canvis del seu entorn. L'entorn en el qual els agents es creen s'executen i moren s'anomena plataforma.

Un agent amb la capacitat de suspendre la seva execució, viatjar a una altra plataforma i reprendre allà les seves tasques és un agent mòbil. Aquesta capacitat anomenada mobilitat permet la creació de noves aplicacions. Un tipus d'aplicació és la de "Mar de Dades", on els agents es desplacen on es troben les dades les dades per tractar-les. Així s'evita la seva transferència, que en dades molt pesants, consumeix molts recursos de la xarxa i pot arribar a congestionar-la. Una altra aplicació, és la de fer ús d'agents intel·ligents per explorar els diversos components d'una xarxa, per realitzar el manteniment d'aquesta o fer front a atacs. Una altre tipus de dades d'aplicació és la negociació, on es poden programar agents que facin la compra, visitant diferents botigues i comparant preus.

En l'àmbit general dels agents s'han desenvolupat moltes plataformes diferents que no poden interactuar entre elles. Per això s'ha fet necessari definir uns criteris de disseny dels agents i les plataformes, per tal d'aconseguir interoperabi-

litat entre ells.

D'aquesta necessitat neix FIPA (Foundation for Intelligent Physical Agents), una organització que recolza les tecnologies basades en agents i promou la creació d'estàndards per la interoperabilitat d'aquests. Una de les plataformes d'agents que segueix aquests estàndards és JADE (Java Agent DEvelopment Framework). Aquesta plataforma segueix l'estructura definida per FIPA i els seus estàndards de comunicació per a agents.

La plataforma JADE, no contempla la migració d'agents d'una plataforma a una altra, per això, el grup de recerca SENDA del Departament d'Enginyeria de la Informació i de les Comunicacions, ha dut a terme un projecte que es tracta d'un servei flexible basat en l'ús de diversos protocols que utilitzen missatges per la migració dels agents. Aquest projecte sorgeix de la recerca en la mobilitat d'agents entre plataformes, basats en les especificacions de FIPA, anomenat Inter-Platform Mobility Service (IPMS). Les diverses implementacions disponibles es poden descarregar de la pàgina de SourceForge [IPMS].

La necessitat del present projecte comença a partir d'un estudi realitzat l'any 2004 [JMT04] sobre l'eficiència de la missatgeria de JADE, que va demostrar que l'enviament de missatges amb un tamany superior a 25 Kb, fei augmentar el temps de transferència de forma exponencial. En la migració implementada al IPMS, el protocol que s'encarrega de la transferència de l'agent, es recolza en l'enviament de l'agent en un sol missatge. Això és un problema, ja que en agents on el seu tamany és molt gran, el temps de migració podria augmentar considerablement. Això probablement es podria solucionar enviant la informació repartida en diversos missatges.

El propòsit principal del projecte consisteix en l'estudi i implementació d'un protocol fragmentat dins l'arquitectura multi-protocols del IPMS v1.96, on s'enviarà l'agent repartit en diversos missatges. La realització del protocol es farà sobre la plataforma d'agents JADE v3.4.1 i serà implementat amb el llenguatge Java v1.5.

Per tal d'assolir aquest propòsit s'han marcat un conjunt d'objectius més concrets que poden ajudar al desenvolupament del projecte:

- Estudiar el funcionament del servei de mobilitat inter-plataforma (IPMS) i els protocols implementats.
- Analitzar i dissenyar un protocol de migració fragmentada fent ús dels estàndards de FIPA.
- Implementar el disseny realitzat seguint l'estructura dels protocols de migració del IPMS.
- Realitzar un estudi de rendiment respecte el protocol que ja hi ha implementat al IPMS i el fragmentat.
- Optimitzar el tamany de fragment.

A continuació farem una breu descripció de com s'ha estructurat la memòria, explicant breument el contingut que podem trobar a cada capítol:

- **Capítol 2: Entorn de desenvolupament**. En aquest capítol es veurà el marc on es desenvolupa el projecte. Es parlarà de FIPA i dels estàndards envers la plataforma d'agents i el sistema de missatgeria. Es veurà la plataforma JADE que està implementada segons les especificacions de FIPA i sobre la qual s'està en desenvolupant el servei de migració IPMS.
- **Capítol 3: Inter-Platform Mobility Service**. Aquí es comentarà quines són les motivacions per la realització d'un servei de mobilitat inter-plataforma i com s'implementa dins l'arquitectura de serveis de JADE. S'explicarà com funciona l'arquitectura multi-protocols i en què es basa l'estructura interna d'aquests protocols.
- **Capítol 4: Anàlisi**. En aquest capítol s'explicarà quina ha estat la planificació del projecte, des del punt de vista de la repartició temporal de les tasques. S'argumentarà més a fons les motivacions per la realització del projecte i es definiran els requisits que aquest ha de complir. S'explicarà amb més detall aquells aspectes del projecte que són rellevants pel seu disseny i implementació.

- **Capítol 5: Disseny**. Es comentarà quines han estat les decisions de disseny que s'han pres respecte els diferents requisits que s'havien marcat. S'explicarà a fons el funcionament del protocol tenint en compte les accions que s'hauran de dur a terme en les plataformes origen i destí per la transferència de l'agent. Es veurà quin ha estat el disseny de les ontologies, les quals permetran estructurar la informació que s'intercanvien les plataformes implicades en la migració.

- **Capítol 6: Implementació**. En aquest capítol es veuran els diagrames de classes escollits per la implementació del protocol i de les ontologies. S'explicarà com s'ha implementat el disseny escollit i entrarem en detall en les classes més importants. Finalment, es veurà com ha estat el procés d'integració del protocol fragmentat al servei, i de quina manera es pot utilitzar.

- **Capítol 7: Test**. Aquí es coneixerà quin ha estat el conjunt de proves que s'ha realitzat per medir el rendiment del protocol. S'analitzaran els resultats, valorant la migració d'agent en diferents situacions, i es compararan amb el rendiment del protocol clàssic.

Per acabar la memòria s'ha fet una valoració del projecte, on es comenta com s'han assolit els objectius que s'havien plantejat. També s'avalua com ha estat d'encertada la planificació temporal que s'havia planejat. I es veuran quines són les ampliacions que es podrien aplicar al projecte i quines podrien ser les possibles línies d'investigació futures.

Capítol 2

Entorn de desenvolupament

En aquest capítol es veurà el marc on es desenvolupa el projecte. Es parlarà de FIPA i dels estàndards envers la plataforma d'agents i el sistema de missatgeria. Es veurà la plataforma JADE que està implementada segons les especificacions de FIPA i sobre la qual s'està en desenvolupant el servei de migració IPMS.

2.1 FIPA

L'existència de múltiples sistemes d'agents, cadascun d'ells amb implementacions totalment diferents, fa que la interacció entre aquests sigui difícil. Per aquesta raó, neix la necessitat de crear estàndards per a la interoperabilitat dels agents. Una unió de criteris, facilitaria aquesta possible interacció.

FIPA (Foundation for Intelligent Physical Agents) és una organització sense ànim de lucre que promou les tecnologies basades en agents i la producció d'estàndards per a la interoperabilitat amb altres tecnologies. El principal objectiu d'aquesta és la definició d'un conjunt d'estàndards per la implementació de sistemes on els agents puguin executar i comunicar i oferir serveis.

2.1.1 Esquema general

Com s'explica en [FIPA23], per a que un agent existeixi i s'executi és necessari que corri sobre una plataforma d'agents. Una plataforma és la infraestructura on

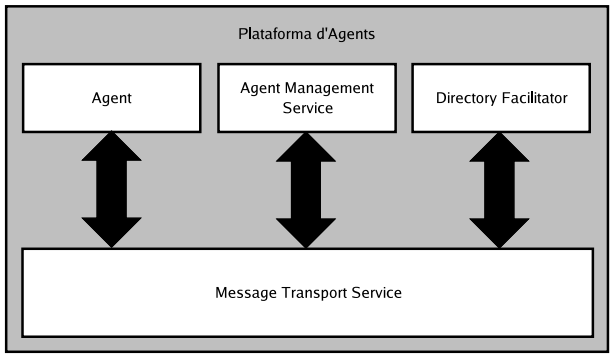

Figura 2.1: Plataforma FIPA-compatible

els agents poden funcionar. Perquè aquesta plataforma segueixi les especificacions de FIPA ha de contenir els següents components (Figura 2.1): agents, Agent Management System, Agent Transport System i, opcionalment, Directory Facilitator. FIPA no especifica com ha de ser el disseny intern de la Plataforma d'Agents mentre aquesta compleixi les seves especificacions.

Seguidament, comentarem breument la funcionalitat de cadascun d'aquests elements:

- **Agent:** És un procés computacional que implementa una aplicació de forma autònoma. Per a la distinció d'agents s'usa l'AID (Agent Identifier) que els identifica de manera única els Agents.

- **Agent Management System (AMS):** L'AMS és un component obligatori que supervisa l'ús i l'accés a la plataforma. Manté una llista d'AIDs juntament amb la seva adreça de transport, oferint així un servei de pàgines blanques. Per obtenir un AID vàlid, cada agent s'ha d'haver registrat a aquest component. Només n'hi pot haver un per plataforma.

- **Message Transport System (MTS):** Es tracta del sistema de comunicació entre agents, tant a nivell intra-plataforma com inter-plataforma.

- **Directory Facilitator (DF):** És un component opcional de la plataforma que funciona com a guia de serveis, com les pàgines grogues. Els agents

poden registrar serveis o demanar-ne d'altres oferts per altres agents. En una plataforma pot haver-hi tants DF com es vulgui.

2.1.2 Missatgeria

Com hem comentat en l'apartat anterior, una de les prestacions que ofereix la plataforma és la comunicació entre agents. L'estructura del transport de missatges es divideix en tres nivells [FIPA67]:

- Message Transport Protocol (MTP)
- Message Transport Service (MTS)
- Agent Communication Language (ACL)

L'estructura del missatge està basat en dues parts: el sobre (o Envelope), que conté la informació adicional de transport, i el contingut (o Payload) on es troba el cos del missatge. FIPA no conté cap especificació sobre com ha de ser l'Envelope, en canvi si que proposa l'estructura del Payload que es representa amb l'estàndard ACL. Els missatges ACL contenen una sèrie de paràmatres que faciliten l'intercanvi d'informació entre agents. El fet d'estandarditzar el cos dels missatges fa que plataformes diferents puguin interaccionar amb facilitat.

El Message Transport Service (MTS) s'encarrega de dur a terme la transferència de missatges ACL entre agents. Aquest servei serveix tant per a l'enviament intern de plaforma com per a l'extern.

Si l'enviament de missatges és entre màquines diferents, es requereix l'ús d'un Message Transport Protocol (MTP), que són mòduls que implementen el transport d'una màquina a una altra. El que fa l'MTP és encapsular el missatge en un protocol d'aplicació per a ser enviat. Actualment existeixen dos estàndards que són el MTP-HTTP i MTP-IIOP.

2.1.3 Protocols d'Interacció

En general, assumim que els agents no s'envien missatges un als altres sense esperar o demanar alguna cosa, sinó que normalment la comunicació requereix una

sèrie de missatges d'anada i tornada per assolir un propòsit. Per això, es poden utilitzar Protocols d'Interacció estàndards de FIPA.

Els protocols d'interacció són models de comunicació que defineixen un esquema d'intercanvi de missatges ACL per facilitar la interacció entre agents. El que pretenen és dur la comunicació entre agents a un nivell més alt. Estan formats per una part que fa d'iniciador i un o més participants. Cada protocol conté una sèrie de passos, però ha de ser el programador qui l'adeqüi a les seves necessitats. Actualment hi ha casi una dotzena de protocols estandarditzats, però els més utilitzats són els següents:

- FIPA Request
- FIPA Query
- FIPA Contract Net
- FIPA Subscribe
- FIPA Propose

A continuació explicarem el funcionament dels protocols d'interacció FIPA-Propose i FIPA-Request, els quals són rellevants pel nostre projecte.

FIPA Propose

El protocol d'interacció FIPA-Propose [FIPA36] permet a un agent realitzar una proposta a un altre. Aquest ha de respondre si accepta la proposta o no. Com s'observa en la Figura 2.2, l'iniciador envia un missatge ACL de tipus "propose" al participant. Aquest, després d'estudiar la proposta, retorna un missatge de tipus "reject-proposal" o "accept-proposal", per indicar si accepta o no la proposta.

FIPA Request

El protocol d'interacció FIPA-Request [FIPA26] permet que un agent sol·liciti a un altre que realitzi una certa acció. El participant pren la decisió i li comunica a

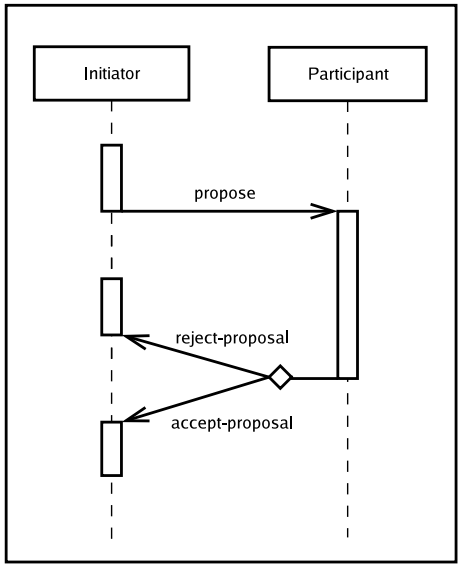

Figura 2.2: FIPA Propose Interaction Protocol

l'iniciador si accepta o refusa la petició (com s'observa a la Figura 2.3). Després de realitzar l'acció, s'informa a l'iniciador de quin ha estat el seu resultat.

2.2 JADE

JADE (Java Agent DEvelopment Framework) [JADE] és una plataforma multi-agent que segueix els estàndards i les especificacions de FIPA. Està totalment implementat en llenguatge Java i conté un entorn gràfic que ajuda en les fases de desenvolupament i correció d'errors. Es tracta d'un software totalment gratuït distribuït per l'empresa Telecom Italia, sota els termes de la llicència LGPL.

2.2.1 Arquitectura

Com s'ha dit la plataforma JADE cobreix els estàndards que proposa FIPA en la seva implementació. JADE implementa els agents com a threads de Java. L'A-

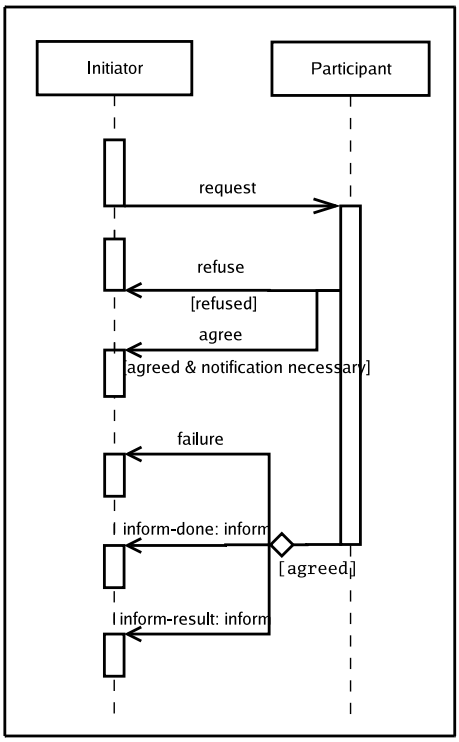

Figura 2.3: FIPA Request Interaction Protocol

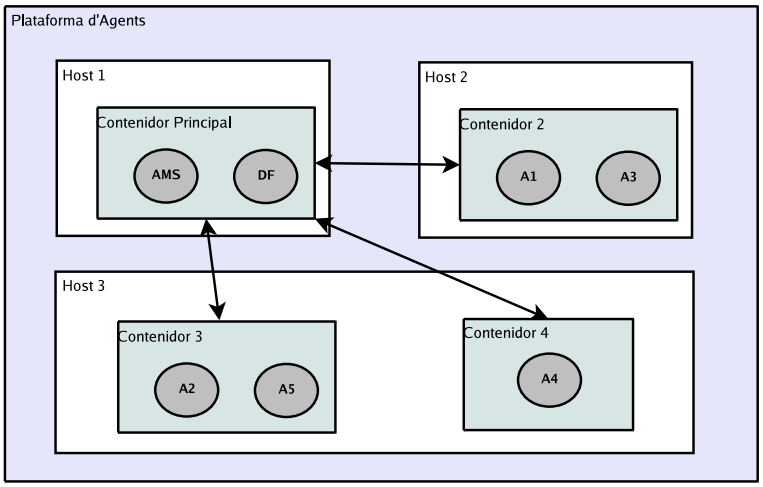

Figura 2.4: Plataforma d'agents JADE

gent Management System (AMS) s'implementa com un agent amb el nom "ams" i, només n'hi pot haver un per plataforma. El Directory Facilitator (DF), que aporta servei de pàgines grogues, s'implementa també amb un agent de nom "df". La comunicació de JADE està basada, també, en les especificacions de FIPA, i s'implementa el Message Transport Service (MTS). La plataforma també permet afegir mòduls de Message Transport Protocol (MTP), per a la comunicació entre màquines.

A més a més, JADE inclou una sèrie de característiques no especificades per FIPA. La més significativa és el concepte de contenidors (Figura 2.4). Aquests s'utilitzen per estendre la plataforma a diverses màquines. Per tant, una plataforma pot tenir un o més contenidors dispersats en una o més màquines. La limitació que comporta aquesta arquitectura és que hi ha d'haver un contenidor principal (main-container) del qual depenen la resta de contenidors.

JADE inclou un servei de mobilitat intra-plataforma, es a dir, que per defecte els agents només es poden moure entre els contenidors d'una mateixa plataforma.

2.2.2 Behaviours

Un agent en JADE s'implementa amb un thread [DMA07], per aconseguir així l'execució de més d'un en paral·lel. Perquè un agent pugui realitzar més d'una tasca, s'introdueix el concepte de behaviour.

Un behaviour representa una tasca que un agent pot du a terme. A un agent se li pot afegir més d'un behaviour que executarà de forma seqüencial. Això es produeix gràcies al planificador de behaviours que conté cada agent.

El problema es troba quan un behaviour executa una operació blocant, això fa bloquejar tot l'agent i no permet l'execució d'altres behaviours que pugui haver instal·lats. JADE dóna solució a això, executant behaviours en threads dedicats de Java. Això permet que un agent executi diverses tasques de forma concurrent.

2.2.3 Els serveis de JADE

JADE ofereix la possibilitat d'afegir serveis per als agents residents a la plataforma [GC04]. Aquests poden ser creats per qualsevol programador i instal·lats en aquesta. Això permet una gran flexibilitat, a l'hora d'afegir o modificar funcionalitats. L'arquitectura permet la distribució dels serveis entre els contenidors de la plataforma.

Per manegar aquests serveis es necessita un gestor (Service Manager) instal·lat al contenidor principal, on serán instal·lats tots els serveis que ofereix la plataforma. A la resta de contenidors hi haurà un gestor que farà de servidor intermediari amb el contenidor principal i que oferirà alguns o tots els serveis que hi hagin a la plataforma.

Els serveis están formats per diversos components que es comuniquen entre si mitjançant comandes. Les comandes són una estructura de dades que conté un nom i una sèrie d'arguments. Són les que aporten flexibilitat a la infraestructura de serveis, ja que permeten comunicar els components d'un servei i interactuar amb d'altres. Són la base de la distribució d'aquests en diferents contenidors. N'hi ha de dos tipus:

- **Comandes Verticals**: s'envien entre diferents components d'un servei en

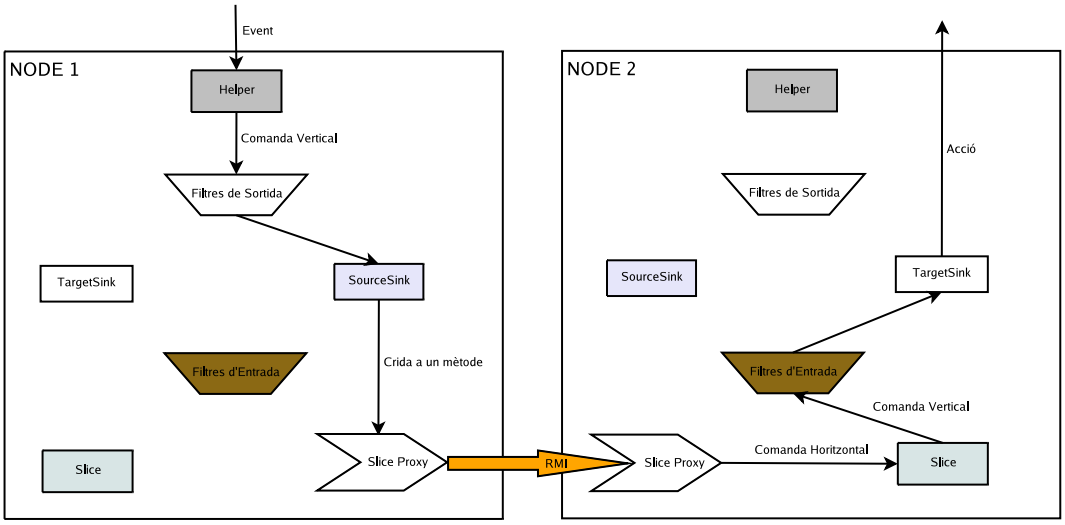

Figura 2.5: Serveis de JADE

un mateix contenidor.

- **Comandes Horitzontals**: s'envien entre components que es troben en diferents contenidors.

Els components que formen un servei són el Helper, el SourceSink i TargetSink, els filtres d'entrada i sortida, i els Slice i SliceProxy:

- **Helper**: És la part que permet a l'usuari (agent) utilitzar el servei, accedint als seus mètodes públics. Quan aquests siguin cridats es generarà una comanda vertical cap al Source Sink del servei.

- **Sinks**: Són els components que implementen la pròpia funcionalitat del servei. S'encarreguen de la recepció de comandes verticals, provinents únicament del seu servei, i en funció d'aquestes realitzen una determinada acció. Hi ha dos tipus de Sinks:

  - **Source Sink**: Està alerta de les comandes verticals que li arriben procedents del propi contenidor (normalment Helper o filtres).

  - **Target Sink**: Atent a les comandes verticals procedents del component Slice, que són les que provenen d'un altre contenidor.

- **Filtres**: N'hi ha d'entrada i de sortida segons l'origen de les comandes. Permeten capturar les comandes verticals de tots els serveis que hi ha en un contenidor.

  - **Filtre de sortida**: Captura les comandes dirigides al SourceSink.
  - **Filtre d'entrada**: Captura les comandes dirigides al TargetSink.

- **Slice i SliceProxy**: Permeten la comunicació i interacció del servei entre diferents contenidors mitjançant les comandes horitzontals.

  - **SliceProxy**: És una classe que implementa els mètodes interns accessibles entre contenidors, i que genera un comanda horitzontal.
  - **Slice**: S'encarrega de rebre i processar la comanda horitzontal generada per l'SliceProxy. Aquesta es traduïda a comanda vertical perquè el TargetSink dugui a terme l'acció.

En la Figura 2.5 es pot veure la relació entre els components explicats i la interacció amb altres nodes. Es un exemple del recorregut de les crides i comandes que es generen en una conversa entre contenidors.

Capítol 3

Inter-Platform Mobility Service

Aquí es comentarà quines són les motivacions per la realització d'un servei de mobilitat inter-plataforma i com s'implementa dins l'arquitectura de serveis de JADE. S'explicarà com funciona l'arquitectura multi-protocols i en què es basa l'estructura interna d'aquests protocols.

3.1 Introducció

La plataforma d'agents JADE implementa la mobilitat dels agents només entre contenidors de la mateixa plataforma. Aquesta mobilitat està integrada com un servei dins la infraestructura de serveis de JADE explicada a la secció 2.2.3.

El servei proporciona una migració amb la transferència del codi sota demanda, ja que primer es transfereix la instància de l'agent serialitzada, i en el procés de deserialització es demana el codi si és necessari.

Inter-Platform Mobility Service (IPMS) [IPMS] és un add-on de JADE que s'ha dissenyat i implementat per dur a terme la mobilitat d'agents entre diferents plataformes. Aquest servei ha estat construit per ser el més transparent possible al programador d'agents, en el sentit que, migrar agents entre plataformes sigui tan fàcil com fer-ho entre contenidors de JADE. Això dóna una nova perspectiva per al desenvolupament d'aplicacions basades en agents mòbils sobre JADE.

El que pretén aquest servei és afegir un nova funcionalitat a l'agent, com és la

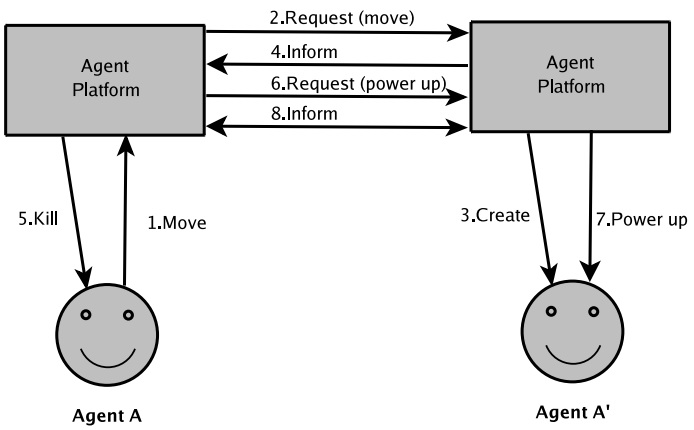

Figura 3.1: Protocol de mobilitat basat en ACL

mobilitat entre plataformes. El seu desenvolupament se serveix de les especificacions de FIPA.

3.2 Arquitectura estàtica

En la primera versió d'aquest servei [DMA07], es va implementar un arquitectura de protocols basada en l'enviament d'agents sobre missatges ACL. El protocol es dividia en dos passos: el primer on es transferia la instància i el codi, i en el segon es posava en marxa l'agent en la plataforma. En la Figura 3.1 podem veure com, utilitzant el protocol d'interacció FIPA-Request, és fa la migració d'un agent entre dues plataformes.

Aquesta versió quedà obsoleta per diverses raons. L'arquitectura no permetia una negociació prèvia sobre els paràmetres de la migració. Afegir nous protocols era una tasca restringida, i finalment, l'agent s'enviava en el primer missatge traient tota possibilitat de rebutjar la migració abans de rebre'l.

3.3 Arquitectura Multi-Protocol

Per intentar millorar els inconvenients anteriors es va dissenyar una nova arquitectura molt més flexible [MPA]. Aquesta es basa en un conjunt de protocols distribuïts en 4 etapes ben diferenciades: Negociació, Control d'accés, Transferència i Power Up. L'ordre d'execució és el mateix amb el que s'han nombrat. Anem a explicar en què consisteix cada etapa:

- En la Negociació es decideix quin serà el conjunt de protocols que s'utilitzarà en la resta de passos. Com es obvi, en aquesta part s'utilitza un protocol únic (Main Negotiation Protocol) per garantir un diàleg inicial estàndard.
- El Control d'accés accepta un conjunt de protocols per decidir i garantir que només els agents autoritzats puguin migrar a la plataforma destí. Això permet afegir un grau de seguretat a la plataforma a nivell d'autorització.
- En la Transferència és on es du a terme la transmisió del codi i de la instància de l'agent cap a la plataforma destí. Després d'aquesta transmisió, l'agent es registra a la plataforma. En aquesta part, també es pot escollir el protocol a utilitzar, i actualment n'existeix un que s'explicarà més tard.
- El Power Up és el pas on, després que l'agent ha estat acceptat a la plataforma, s'activa per començar l'execució en la nova plataforma.

Cal afegir que la migració, com la primera versió, es basa en l'enviament de missatges ACL. En la Figura 3.2 podem veure l'esquema del funcionament d'aquesta arquitectura multi-protocol, i com s'escull un protocol a cada etapa segons la decisió presa en l'etapa de negociació.

3.4 Estructura general

Com que està implementat en forma de servei, l'estructura del IPMS [CMS] està condicionada a l'estructura tradicional dels serveis de JADE. Ja que JADE ja té un servei de mobilitat intra-plataforma, l'IPMS funciona conjuntament amb aquest i,

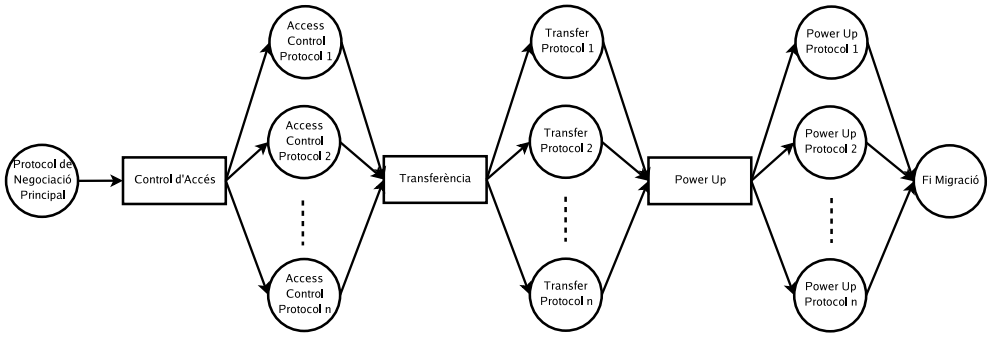

Figura 3.2: Arquitectura Multi-Protocol

en ocasions hi coopera. Cal afegir que la migració sempre es du a terme entre els contenidors principals de les plataformes implicades. Podríem dir que el servei està dividit en sis parts, cadascuna amb la seva funcionalitat:

- **Controlador del Servei**

  És el nucli del sistema, d'acord amb l'estructura típica de serveis de JADE. S'encarrega de coordinar les altres parts per acabar duent una migració inter-plataforma o per la recepció d'un agent extern.

  El controlador implementa un filtre que esta pendent de les comandes verticals del servei intra-plataforma. Si es detecta d'una migració on la destinació és una altra plataforma, en comptes d'un contenidor, s'anul·la la comanda i s'en genera una altra per al servei inter-plataforma. És d'aquesta manera com es du a terme la migració de forma totalment transparent a l'usuari.

  També coopera amb el servei intra-plataforma per saber en quin contenidor està el codi de l'agent que es vol transferir. Instala els "behaviours participants" corresponents a l'agent AMS per poder rebre agents. A l'hora d'enviar, serialitza l'agent, a partir del codi que haurà rebut, i arrenca el "behaviour" corresponent.

- **Analitzador de codi compilat**

Per saber les dependències de classes que té un agent, i per tant, saber quin és el codi que s'ha d'incloure per l'enviament és necessari estudiar la taula ClassPool. Aquesta taula es troba present en el "bytecode" de totes les classes de Java. Així que, aquest mòdul esta format per un conjunt de classes que proveeixen mètodes per accedir a la informació interna del "bytecode".

- **Analitzador de classes i empaquetador**

  S'encarrega d'analitzar les classes de l'agent i recursivament determinar les dependències. Per dur a terme aquesta tasca utilitza l'analitzador de codi. Un cop s'han determinat totes les classes, amb l'ajuda del carregador de classes del sistema, s'afegeix a un fitxer JAR el "bytecode" de cada classe.

- **Repositori del codi**

  Els fitxers JAR contenen el codi de l'agent i s'han d'utilitzar quan l'agent vol migrar o quan l'usuari ho especifica. Per tant, aquest component està inclòs en l'AMS, i manté una llista de fitxers JAR associats a un o més agents per poder manegar-los quan convingui. Les operacions bàsiques que realitza són: registrar un agent, obtenir el codi d'un agent i eliminar la referència a un agent.

- **JAR Classloader**

  El carregador de classes, que es troba a l'AMS, s'encarrega de carregar les classes del fitxer JAR.

- **Gestor de protocols i "behaviours participants"**

  Aquest component és el que s'encarrega del diàleg entre les dues plataformes. És el component que més ha evolucionat del servei. En la primera versió, estava format pel protocol que s'ha explicat abans (Figura 3.1). Com ja s'ha dit la seva evolució va dur a aquesta part del servei a implementar una arquitectura multi-protocol.

A la Figura 3.3 es pot veure l'esquema d'aquesta estructura amb tots els seus components. Com que l'agent pot sol·licitar la migració a un altra plataforma

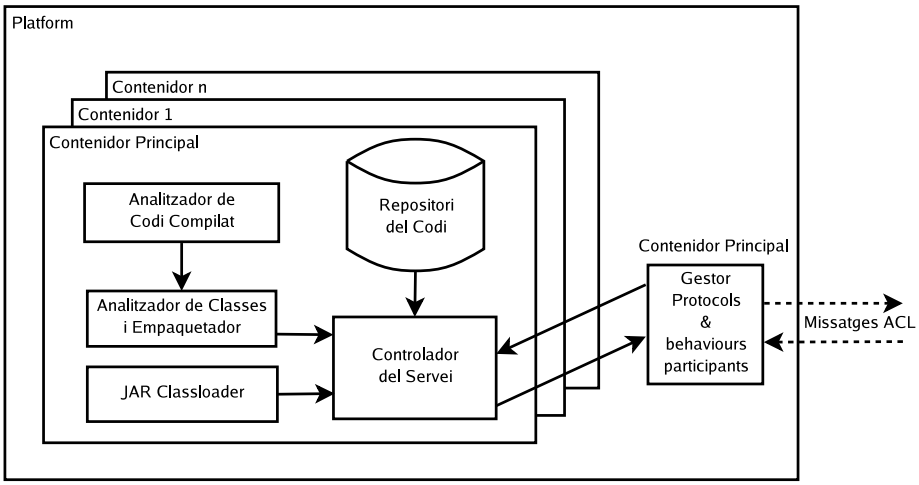

Figura 3.3: Estructura del servei de mobilitat inter-plataforma

desde qualsevol contenidor, els 5 primers components es troben instal·lats a tots ells. El que s'encarrega del diàleg entre plataformes només es troba al contenidor principal.

3.5 Protocols de Migració

L'arquitectura multi-protocol [MPA] permet afegir nous protocols en cadascuna de les seves etapes. Això dóna una gran flexibilitat al servei per escollir el protocol més adequat per a cada situació. Aquest protocol ha de tenir un estructura per poder-se integrar en el servei. Es podria dir que un protocol de migració està basat en tres parts:

- **Behaviours Iniciadors**: Un o més behaviours que s'encarreguen d'iniciar la conversa entre les parts interesades. S'instal·len al Protocols Manager que s'explica en la següent secció.

- **Behaviours Participants**: Són instal·lats a l'agent AMS, i es tracta d'un o més behaviours que estàn a l'espera de rebre missatges dels iniciadors per respondre'ls.

- **Ontologies**: Per facilitar la conversa entre les dues parts, s'utilitzen una o més ontologies en els missatges que s'intercanvien. Les ontologies permeten estructurar la informació sobre els missatges ACL.

En la implementació, els protocols són encapsulats perquè la seva integració resulti el més senzill possible. Per a cada protocol es defineix una classe que conté internament tots els recursos necessaris per a l'execució del protocol.

3.6 Gestió de Protocols

Protocols Manager és un "behaviour" que s'instal·la a l'agent AMS. És l'encarregat de gestionar totes les etapes de l'arquitectura multi-protocol i les instàncies dels protocols que la formen.

Un punt important, que afecta al rendiment del servei, és que el servei arrenca més d'una instància de cada protocol en els seus behaviours iniciadors. Això serveix per poder migrar diversos agents de forma concurrent. Aquest és un paràmetre que l'usuari pot gestionar segons les seves necessitats. També cal tenir en compte que les instàncies es reutilitzen en diferents migracions per no sobrecarregar l'agent AMS.

Els behaviours participants, és un cas diferent i només s'executa una instància per cadascun. Això es fa junt amb l'arrencada de la plataforma, per preparar al servei per rebre migracions.

S'ha volgut ser més explícit amb l'estructura dels protocols i la seva gestió, perque aquest projecte tracta de dissenyar i implementar un protocol per l'etapa de transferència.

Capítol 4

Anàlisi

En aquest capítol s'explicarà quina ha estat la planificació del projecte, des del punt de vista de la repartició temporal de les tasques. S'argumentarà més a fons les motivacions per la realització del projecte i es definiran els requisits que aquest ha de complir. S'explicarà amb més detall aquells aspectes del projecte que són rellevants pel seu disseny i implementació.

4.1 Introducció

Aquest projecte està basat en la creació d'un protocol de migració fragmentada inter-plataforma d'agents. La realització del protocol s'ha de fer dins el marc del servei Inter-Platform Mobility Service (IPMS) de la plataforma d'agents de JADE. I més concretament ha de ser un protocol de l'etapa de Transferència, vista a la secció 3.3. Actualment hi ha un protocol implementat en aquesta etapa, que consta de l'enviament de la instància de l'agent i, si es necessari, el codi del mateix.

Com s'ha explicat en el capítol anterior, la mobilitat inter-plataforma es basa en l'ús de missatges ACL. Al 2004 es va fer un estudi sobre el rendiment de l'enviament d'aquests missatges amb diferents tamanys [JMT04]. Es va demostrar que quan el tamany del missatge augmentava, el temps de generació i transferència es veia afectat considerablement. Quan es superava el tamany de 25 Kb, el temps deixava d'incrementar de forma linial per fer-ho de forma exponencial.

En una aplicació real, els agents poden tenir un tamany de codi que superi el llindar de 25 Kb. Aquest fet, com s'ha vist en l'estudi, fa augmentar força el temps de migració de l'agent. Això esdevé en una disminució del rendiment temporal de l'aplicació, i segons l'estudi esmentat aquest baix rendiment podria arribar a ser insostenible. Per tant, la principal motivació del projecte és mantenir un bon rendiment en totes les migracions, desenvolupant un protocol on els missatges ACL no siguin tan grans.

Per la realització del projecte cal fer una bona planificació per assegurar que es compleixin els terminis establerts per l'entrega de la memòria i la presentació oral. A la següent secció s'explicarà quina ha estat la planificació del projecte.

4.2 Planificació del Projecte

El projecte requereix la realització d'un anàlisi per definir les diverses tasques que s'han de realitzar i estructurar-les en el temps. Per això el primer que s'ha fet és definir les tasques i distribuir-les en una planificació temporal, estimant el temps de duració.

S'han dividit les tasques en set apartats, on els dos primers tracten de la realització de la planificació i dels estudis previs que cal dur a terme. S'ha pensat que abans de la pròpia realització del projecte és important fer un estudi sobre l'estat de l'art per situar-se en l'entorn on es desenvoluparà el projecte, i adquirir aquells coneixements que són imprescindibles abans d'iniciar la resta de tasques. En la figura 4.1 es pot observar la distribució d'aquestes tasques en el temps.

Les quatre tasques següents són les pròpies d'un projecte d'enginyeria: anàlisi, disseny, implementació i test. Tot i que la distribució d'aquestes és seqüencial, a la pràctica es pensa que adquiriran un comportament evolutiu i en ocasions s'haurà de fer backtracking en les etapes, per solucionar els possibles problemes que puguin sorgir. Per últim, la darrera tasca consisteix en la redacció de la memòria.

L'inici del projecte està datat en 19 de Febrer de 2007, i la seva finalització per la última setmana abans de l'entrega de la memòria.

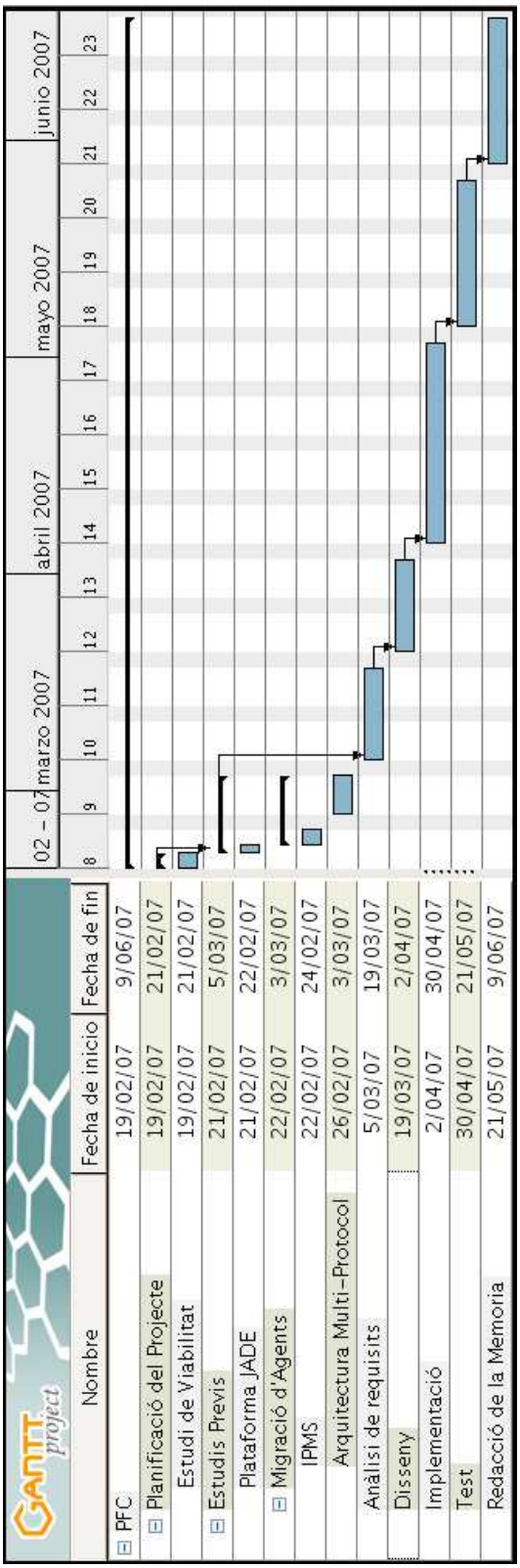

Figura 4.1: Diagrama de Gantt

4.3 Requisits

En aquesta secció s'anomenaran els requisits que s'han cregut convenients que ha de complir el projecte. En les seccions següents s'entrarà més en detall en alguns d'aquests requisits.

- Realitzar un protocol per a l'etapa de transferència que s'adapti a l'arquitectura multi-protocol.
- El protocol haurà de funcionar de forma concurrent en cas d'haver varies migracions a l'hora.
- Només s'enviarà el codi quan faci falta.
- El protocol enviarà la instància sencera i el codi fragmentat en diferents missatges ACL.
- El tractament dels fitxers JAR serà amb streams.
- Enviament de confirmació de fragment.
- Creació d'un conjunt d'ontologies per l'intercanvi d'informació entre l'iniciador i el participant.
- Recerca del tamany de fragment òptim depenent de la situació.

4.4 Fragmentació del codi

Per la transferència d'un agent d'una plataforma a una altra cal enviar, mitjançant missatges ACL, la seva instància serialitzada i el seu codi. Com hem comentat abans, si el missatge ACL es molt gran, el rendiment temporal disminueix bastant. Per tant, en la migració d'un agent, es pot veure el rendiment afectat quan el codi i/o la instància ocupen molt.

Com s'explica a [JMT04], el baix rendiment en l'enviament de missatges ACL majors de 25 KB, pot ser perquè la plataforma no està preparada per tractar missatges amb aquests tamanys. D'aquí sorgeix el problema de la migració d'agents que

ocupin aquestes quantitats de dades. Aquest fet, planteja l'enviament de l'agent en més d'un missatge ACL per evitar el baix rendiment que provoca l'enviament de missatges de gran tamany.

Les dades a enviar en una migració són la instància serialitzada i el codi de l'agent. Per tant, tenim varies opcions: dividir en diversos missatges ACL el codi, la instància o totes dues.

La instància, en una aplicació Java, és la que ens dóna el valor de les variables de l'agent. Tot i això, per aquesta primera aproximació de la migració fragmentada, no es planteja la mobilitat d'agents amb instàncies excessivament grans. La funcionalitat dels agents pot ser tot el complexa que es vulgui. L'agent pot estar format per un conjunt de classes gran, això fa que el tamany del codi augmenti. Es pensa que aquest últim fet pot ser més comú en els agents, per tant aquest projecte planteja la migració fragmentada com l'enviament del codi de l'agent en diversos missatges.

La questió que sorgeix ara és com tractar els fitxers JAR per no carregar-los completament en memòria. Tant en la part de l'emissor, que llegeix el JAR de disc i el posa en un missatge ACL, com en el receptor, que llegeix el codi del missatge ACL i el guarda a disc, s'hauria de fer un tractament d'aquestes lectures i escriptures mitjançant "streams" per evitar carregar el fitxer sencer a memòria.

En aquest últim punt apareix un problema. Com s'ha explicat al capítol anterior, quan un agent vol migrar a una altra plataforma, això sempre es du a terme des del contenidor principal. Si un agent vol migrar a una altra plataforma i es troba en un contenidor secundari, pot ser que el seu codi estigui en aquest contenidor. Per tant, si volem accedir mitjançant streams a aquest codi des del contenidor principal, no podrem. Això es pot solucionar transferint primer el codi al contenidor principal o simplement, no permetre la migració fragmentada desde contenidors secundaris.

4.5 Existència del codi

El protocol clàssic de transferència que ja hi ha implementat, contempla una millora quan el codi de l'agent ja es troba a la plataforma destí. El protocol es divideix en dos passos, en el primer dels quals es pregunta si el codi ja està en la plataforma on vol migrar l'agent. Si es així, no cal enviar-lo.

Aquesta sembla una bona característica per la transferència de l'agent, ja que es pot evitar l'enviament del codi en múltiples ocasions. En el protocol d'aquest projecte, aquesta millora també faria disminuir els temps de migració dels agents, sobretot en agents amb un alt tamany de codi. Per això, es pensa que enviar el codi només quan cal és una bona prestació per tenir en compte.

Aquesta prestació es possible gràcies a una funcionalitat del servei de mobilitat inter-plataforma. L'AMS guarda una taula amb tots els agents que s'estan executant en la plataforma i una altra taula amb tots el fitxer JAR registrats en aquell moment. S'estableix una relació entre aquestes dues taules on s'indica el nombre d'agents que utilitzen cada fitxer JAR. En el moment que cap agent utilitza un codi, s'elimina l'entrada de la taula i s'esborra del disc.

4.6 Confirmació de fragments

En el protocol clàssic de transferència, s'envia un missatge del receptor a l'emissor quan la transferència i el registre de l'agent a la plataforma ha estat satisfactòria. En aquest projecte, això també pot ser així, però cal tenir en compte que hi haurà tota una série de missatges intermitjos on només s'enviaràn fragments de codi. Aquests missatges es poden perdre o pot ser que no arribin bé.

Per això cal plantejar-se alguna manera de saber si els missatges han arrivat correctament a la plataforma destí. Una solució és informar a la plataforma origen que el fragment ha arribat bé, enviant un missatge de confirmació (missatges ACK). Aquesta confirmació es pot donar de diverses maneres. Seguidament es comentarà quines són aquestes possibilitats i els seus avantatges i inconvenients.

Una opció tracta de pensar en enviar un ACK per cada fragment de codi rebut.

Això pot derivar en tres tipus d'implementacions:

- La més senzilla, es haver d'esperar a rebre l'ACK del fragment anterior per enviar el següent. Aquesta manera, sembla relativament més segura, perquè s'informa de la rebuda de cada fragment després d'enviar-lo, però més lenta, ja que mentre s'espera a que arribi l'ACK no s'està enviant res i es desaprofita la xarxa. Tot i això la implementació sembla més senzilla i intuïtiva.

- L'altra possible manera de pensar-ho, és enviar fragments sense la necessitat d'haver rebut l'ACK anterior. Sembla que el rendiment augmentaria respecte la configuració anterior, però s'hauria d'anar controlant quins fragments s'han confirmat i quins s'han enviat.

- Una possible combinació d'aquests dos esquemes és deixar d'enviar fragments quan s'arriba a un nombre determinat de fragments no confirmats. Això permetria no sobrecarregar en excés la xarxa si els ACKs tarden en arribar.

L'altra opció és no enviar missatges de confirmació per a tots els fragments. Això ens permet enviar un conjunt de fragments i només rebre un ACK que els confirma tots. Sembla que aquest esquema no carregaria la xarxa amb enviaments múltiples de ACKs. El problema es troba quan la confirmació no arriba, perquè s'ha perdut algun fragment, no es podrà saber quin fragment s'ha perdut i s'haurien de tornar a enviar tots.

Aquest anàlisi sobre la confirmació de fragments, fa plantejar que fer quan es perd un fragment. L'opció més ràpida és abortar tota la migració retornant un error. També es pot pensar amb alguna política de retransmissió de fragments, per no haver d'abortar la migració per la pèrdua d'un sol fragment.

4.7 Tamany de fragments

Un dels paràmetres que més influiràn en el rendiment del protocol serà el tamany dels seus fragments. A continuació es farà un petit anàlisi de com es podria com-

portar el protocol davant de variacions d'aquest paràmetre. Cal tenir en compte també, que el codi de l'agent més simple que es pot implementar ronda els 2KB de tamany de JAR. Això vol dir que fer hipòtesis per a fitxers JAR més petits no té sentit.

- Si s'escull un tamany de fragment petit (no superior a 1 Kb), el que fa és enviar molts missatges ACL, desaprofitant el throughput de la xarxa. Per a JARs inferiors a 25 Kb és possible que no es millori gaire respecte al protocol clàssic. En canvi, en tamanys de fitxer més grans, s'hauria de veure si és possible obtenir millora del rendiment. Com que la plataforma no està preparada per tractar missatges molt grans, és possible que s'aconsegueixi reduir el temps encara que es fragmenti en tamanys molt petits.

- Si el tamany de fragment no és tan petit (no superior a 25 Kb), amb JARs més petits, no empitjorarà en res respecte al protocol clàssic, perquè s'enviarà en un sol missatge. En canvi si el tamany del fitxer és més gran, s'hauria de mirar per a quina mida de JAR es millora el rendiment respecte al protocol clàssic.

- En el cas que el tamany de fragment sigui superior a 25 KB poden passar varies coses. Si el tamany del JAR és inferior, un altre cop no perdrem res. En canvi si és més gran, possiblement millorem el temps, respecte el protocol clàssic, però no tot el que es podria.

Aquestes hipòtesis s'hauran de comprovar empíricament al finalitzar la implementació. És on es veurà en quines situacions val la pena la utilització del protocol fragmentat i en quines no. També es podrà observar quina és la mida de fragment que ens dóna millor rendiment, si pensem en tamanys de JAR grans.

Tot això fa pensar que seria important que l'usuari, pugués escollir el tamany del fragment per poder adaptar-lo a cada situació.

Capítol 5

Disseny

Es comentarà quines han estat les decisions de disseny que s'han pres respecte els diferents requisits que s'havien marcat. S'explicarà a fons el funcionament del protocol tenint en compte les accions que s'hauran de dur a terme en les plataformes origen i destí per la transferència de l'agent. Es veurà quin ha estat el disseny de les ontologies, les quals permetran estructurar la informació que s'intercanvien les plataformes implicades en la migració.

5.1 Decisions de disseny

Després de l'anàlisi de diferents aspectes del projecte i de la definició dels requisits que haurà de complir el protocol, en aquesta secció es comentaran les decisions de disseny que s'han près per la realització del projecte. Aquestes les s'ampliaran amb més detall a les seccions posteriors i el que es preten és que el protocol compleixi els requisits que ens hem imposat.

- El protocol es dividirà en dues parts. En la primera part es pregunta pel codi i s'envia la instància, i en la segona part s'envia el codi si així ho ha especificat el receptor.

- S'ha decidit dissenyar dues ontologies, una per cada sub-protocol, per estructurar la informació que s'intercanviaràn entre les plataformes implica-

des en la migració.

- Pel que fa a la confirmació dels fragments, s'ha decidit enviar un ACK per cada fragment. L'emissor haurà d'esperar a rebre l'ACK d'un fragment per poder enviar el següent. Finalment s'ha pensat en aquesta opció, encara que no sigui la més òptima, perquè fa la implementació més senzilla i fiable.

- En cas de qualsevol tipus de fallida la migració serà abortada informant a les dues plataformes. Per tant, la retransmissió de fragments en cas de pèrdua no es durà a terme simplificant-ne així la implementació.

Després d'aquesta aproximació a les principals decisions de disseny que s'han pres, es veurà més a fons el disseny del funcionament del protocol i les ontologies utilitzades per estructurar la informació que s'envia.

5.2 Funcionament del Protocol

El protocol s'ha d'adaptar a l'estructura del servei de mobilitat inter-plataforma. Tot i això, el disseny del seu funcionament és independent a aquesta estructura. Aquest disseny estableix quines seràn les accions que es realitzaran, tan en l'emissor com al receptor, i quin serà l'intercanvi de missatges entre ells.

Com s'ha comentat a l'anàlisi, resulta molt interessant no enviar el codi si no és necessari. Això es durà a terme fent una negociació prèvia a la transferència. Per això, es dividirà el protocol en dues parts on la segona dependrà completament de la primera. Així que el primer sub-protocol s'encarregarà d'enviar la instància i de fer aquesta negociació i el segon, enviarà el codi, si és necessari, i el registrarà a la plataforma.

Seguidament es descriurà més a fons la funcionalitat del protocol en les seves dues parts:

- A la primera part serà on es negociarà l'enviament del codi. Aquesta es basa en l'esquema de comunicació del protocol d'interacció FIPA-Propose (Figura 2.2), ja que s'adapta força bé al propòsit de la negociació. En aquest

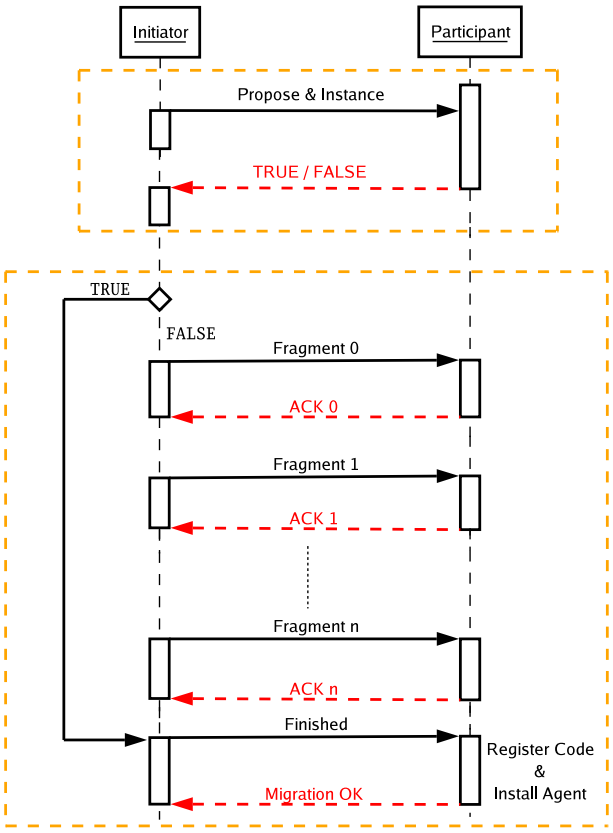

Figura 5.1: Diagrama de seqüència del protocol

esquema, l'iniciador proposa alguna cosa al participant i aquest respon si l'accepta o la refusa.

Com es veu a dalt de la Figura 5.1, l'iniciador envia un missatge al participant amb la seva proposta, que com hem dit es tracta de saber si necessita el codi. En aquest missatge s'inclou també l'enviament de la instància de l'agent ja que aquesta s'enviarà sempre. El participant respon TRUE o FALSE, que vol dir que ja té el codi o no, respectivament.

- La segona part del protocol és més complicada que la primera. La seva execució depèn de l'anterior en gran part, ja que si el codi no s'ha d'enviar l'esquema de comunicació consistirà només en un missatge d'anada i un de tornada. S'utilitza el protocol d'interacció FIPA-Request (Figura 2.3),

on l'iniciador sol·licita al participant que realitzi alguna acció i aquest si l'accepta informa del resultat després de la seva realització. Aquest protocol d'interacció s'adapta força bé a l'esquema de comunicació que volem dissenyar.

Com es veu en el requadre inferior de la Figura 5.1, existeixen dos possibles fluxes d'execució, depenent de la resposta de la negociació prèvia. Dels dos, el fil d'execució més senzill es produeix quan no cal l'enviament del codi. En aquest cas, s'envia un missatge demanant al receptor que instal·li l'agent a la plataforma amb el codi que té i la instància que ha rebut previament. Si tot ha anat bé, el participant respon informant que la migració ha estat satisfactòria.

Per al cas on la resposta ha estat FALSE, i per tant s'hagi d'enviar el codi, veiem en la Figura 5.1 com es l'intercanvi de missatges i la seqüència d'enviament. L'iniciador envia un fragment al receptor amb un missatge ACL. Si la recepció ha estat satisfactòria, el participant guarda el fragment a disc i respon amb un missatge ACK del fragment rebut. Si ens fixem amb l'ús del FIPA-Request, representa que l'iniciador demana al participant que accepti el fragment i que se'l guardi. El participant informa de si la transferència del tros de codi ha estat correcta contestant amb un ACK.

Aquests passos d'enviament i confirmació es repeteixen tantes vegades com el nombre de fragments en què s'ha dividit el codi. Els fragments van numerats per poder identificar-los de manera única i que la confirmació no sigui incoherent. En l'enviament de tots els fragments, s'informa al receptor si es tracta o no de l'últim fragment del codi. Quan la transferència de tot el codi s'hagi dut a terme satisfactòriament, l'iniciador envia un missatge demanant que es registri el codi enviat i s'instal·li l'agent. Com abans, si ha estat tot correcte, el participant informa a l'iniciador d'aquest fet.

Cal afegir que durant tot el procés, si es produeix algun error, el participant ho comunicarà, mitjançant un missatge, a l'iniciador i automàticament s'abortarà la migració.

Frame	fragment-protocol-description		
Ontology	fragment-protocol-ontology		
Paràmetre	**Descripció**	**Presencia**	**Tipus**
code-uid	Identificador únic del codi de l'agent	Obligatori	String
data	Instància serialitzada de l'agent	Obligatori	Byte Sequence

Taula 5.1: Fragment Protocol Ontology

5.3 Ontologies

Les ontologies serveixen per estructurar la informació dins els missatges ACL i, així facilitar l'intercanvi d'informació. En el nostre protocol, les ontologies permetran transmetre les dades necessàries per la negociació prèvia i la transferència de l'agent.

Les ontologies estan formades per un conjunt de conceptes i accions. Els conceptes contenen tots aquells camps amb la informació que es vol transmetre. Les accions indiquen al receptor quina tasca ha de du a terme.

S'ha decidit dissenyar dues ontologies, una per cada sub-protocol. S'ha afegit un concepte a cadascuna, una acció per la primera i dues per la segona. Seguidament s'explicarà més detalladament el disseny d'aquestes ontologies.

5.3.1 Ontologia de Negociació

En la part de negociació prèvia, el flux d'execució és sempre el mateix, per tant, només fa falta definir una acció. Recordem que la tasca que ha de fer el participant és la de rebre la instància de l'agent i mirar si ja té el seu codi. Per tant, el concepte que inclou aquesta ontologia consta de dos camps: "data" i "code-uid" (Taula 5.1).

El camp "data" conté la instància de l'agent serialitzada. El participant després d'extreure-la del missatge l'haurà de guardar, ja que s'usarà al final de la transferència de l'agent per instal·lar-lo a la plataforma.

El camp "code-uid" guarda l'identificador únic del codi de l'agent. Aquest servirà al participant per comprovar si té aquest codi registrat a la plataforma, i així poder respondre a l'iniciador.

5.3.2 Ontologia de Transferència

Per la part de transferència i registre del codi a la plataforma, els paràmetres necessaris no seràn els mateixos. Com s'explica en el funcionament d'aquest subprotocol, conté dos fluxes d'execució i per tant s'haurà de saber quan realitzar cadascun. Per això s'afegirà a l'ontologia dues accions que indicaran quina tasca s'haurà de du a terme.

El concepte d'aquesta ontologia contindrà els següents camps: "code-uid", "code-fragment", "fragment-id" i "more-fragment" (Taula 5.2). Els últims tres camps no seran obligatoris ja que només seran usats per l'acció pertanyent a l'enviament d'un fragment de codi. En canvi, l'identificador únic del codi haurà d'estar present en les dues accions de l'ontologia.

El camp code-fragment conté la seqüència de bytes corresponent al fragment de codi que s'està enviant. L'iniciador haurà d'omplir aquest camp amb el tros del codi llegit de disc, i el participant després d'extreure'l l'emmagatzemarà immediatament a disc, per no carregar la memòria.

El camp de fragment-id conté un valor sencer, que serveix per identificar els fragments del codi que es creen. Aquest valor permetrà al receptor comprovar si el fragment que està rebent és l'esperat. Aquest valor s'utilitzarà per enviar la confirmació a l'iniciador, perquè pugui enviar el següent fragment. Per simplicitat s'ha decidit que aquest valor comenci desde zero i s'incrementi en u per cada nou fragment.

Finalment, el camp de more-fragments conté un booleà que indica si el missatge en questió conté l'ultim fragment del codi de l'agent. Per tant, informa al participant de que no rebrà més missatges amb fragments d'aquell codi. Aquest camp està inspirat amb el de la capçalera IPv4 que s'usa en la fragmentació de datagrames.

Hi ha dos tipus de missatges que també formen part de l'etapa de transferència del codi. Es tracta de missatges que el participant envia a l'iniciador per informar-lo de la tasca que ha realitzat, i que s'afegeix al contingut del missatge ACL. Després de rebre un fragment, el receptor ha d'informar a l'emissor perquè aquest pugui enviar-ne més. Aquest missatge consisteix en una cadena de caràcters on

Frame	fragment-protocol-code-description		
Ontology	fragment-protocol-code-ontology		
Paràmetre	**Descripció**	**Presencia**	**Tipus**
code-uid	Identificador únic del codi de l'agent	Obligatori	String
code-fragment	Conté el fragment de codi	Opcional	Byte Sequence
fragment-id	Identificador de fragment	Opcional	Integer
more-fragment	Indica si es tracta de l'últim fragment del codi	Opcional	Boolean

Taula 5.2: Fragment Code Protocol Ontology

s'indica el fragment rebut: "ACK"+ identificador del fragment confirmat. Per altra banda, després del registre del codi i de l'instal·lació de l'agent, el participant envia un missatge informant a l'iniciador que tot ha anat bé. Aquest missatge consisteix amb la següent cadena de caràcters: "migration-ok".

Capítol 6

Implementació

En aquest capítol es veuran els diagrames de classes escollits per la implementació del protocol i de les ontologies. S'explicarà com s'ha implementat el disseny escollit i entrarem en detall en les classes més importants. Finalment, es veurà com ha estat el procés d'integració del protocol fragmentat al servei, i de quina manera es pot utilitzar.

6.1 Introducció

La implementació del protocol s'ha realitzat seguint el disseny del capítol anterior i l'estructura especificada per l'arquitectura multi-protocols del servei de mobilitat inter-plataforma de JADE. Això vol dir que conté un conjunt de behaviours iniciadors amb el seus corresponents behaviours participants i un conjunt d'ontologies per la comunicació.

Seguint aquesta estructura, s'ha dividit la implementació en dues parts. El mòdul principal del protocol que conté els behaviours, i el conjunt de classes que implementen les dues ontologies que s'han explicat al capítol anterior.

6.2 Mòdul principal

Seguint l'estructura d'encapsulament del protocol, el mòdul principal consta d'una classe, que es troba al paquet *jade.core.migration.protocols*, anomenada FragmentProtocolTransfer. Aquesta classe hereda de la interfície ProtocolComponent, que conté una sèrie de mètodes declarats que han d'implementar tots els protocols per al tractament i la manipulació dels seus behaviours i les seves ontologies.

Com podem veure a la Figura 6.1, la classe FragmentProtocolTransfer conté 4 classes internes. Aquestes pertanyen als behaviours que implementen els dos subprotocols explicats en el capítol anterior. Per tant, hi han dos behaviours iniciadors i dos participants.

A continuació s'explicarà la funcionalitat d'aquests behaviours i les classes que els implementen:

- **ConfigModeI**

  Aquesta classe hereda de ProposeInitiator, behaviour que implementa la part iniciadora del protocol d'interacció FIPA-Propose. La funcionalitat de ConfigModeI és de part iniciadora de la negociació del codi, i la transferència de la instància de l'agent.

  El mètode prepareInitiations s'encarrega de la construcció del primer missatge del protocol FIPA-Propose. S'ha sobreescrit aquest mètode per crear el missatge de tipus PROPOSE, que preguntarà al participant si té el codi de l'agent. En aquest missatge també s'afegeix la instància serialitzada de l'agent, utilitzant l'ontologia Fragment Protocol Ontology.

  S'han implementat un conjunt de mètodes per tractar els diferents fets que poden passar després de l'enviament de proposta: acceptada o refusada, temps d'espera superat, etc.

  En cas d'acceptament de proposta, el valor que indica si s'ha d'enviar el codi o no, es guarda per a que el següent sub-protocol conegui la resposta i actui en conseqüència.

- **ConfigModeR**

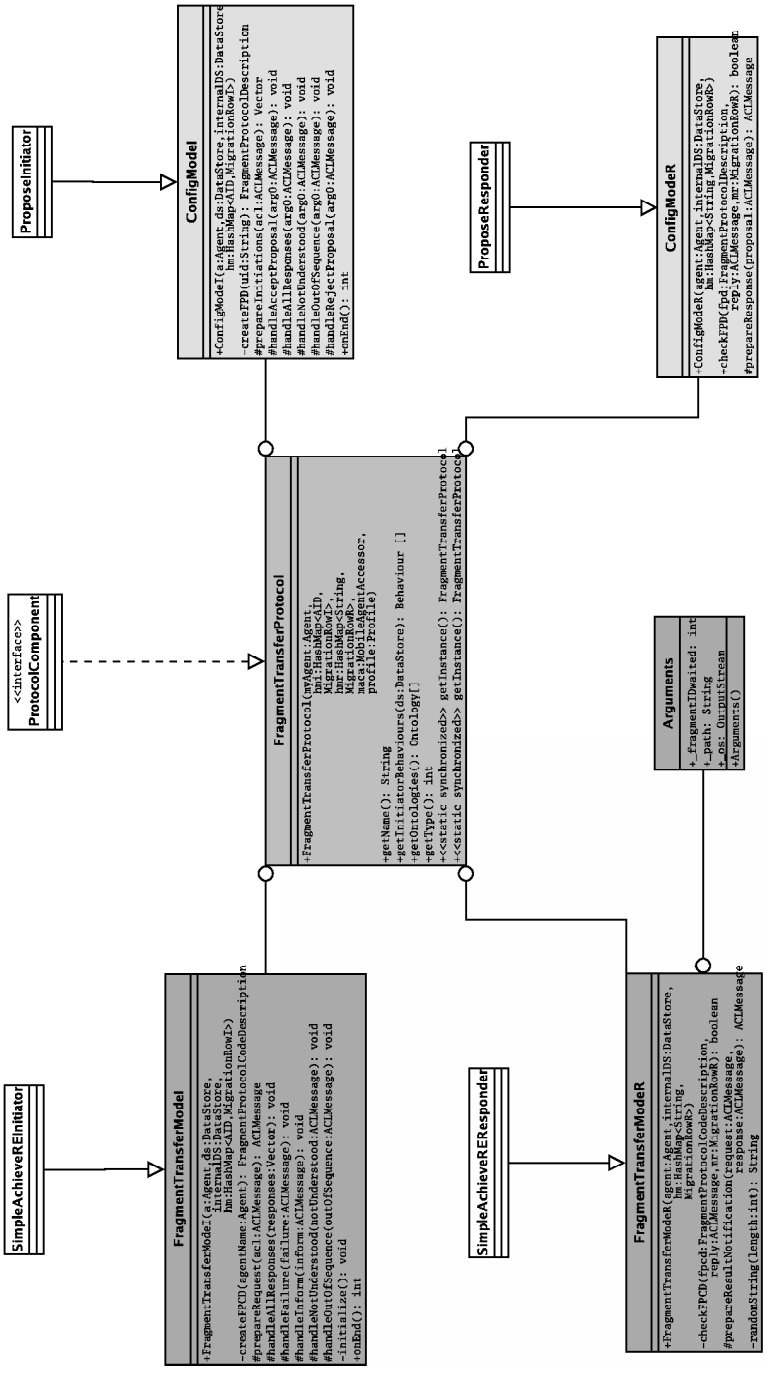

Figura 6.1: Diagrama de Classes del Protocol

Aquesta classe hereda de ProposeResponder, behaviour pertanyent a la part participant del protocol d'interacció FIPA-Propose. Funciona com la part complementària de l'anterior i realitza una sèrie de tasques.

Mitjançant el mètode prepareResponse que es crida quan es rep el missatge de tipus PROPOSE, es comprova que aquest té el format esperat i que els paràmatres són correctes. Seguidament, s'extreu la instància i s'emmagatzema per ser utilitzada posteriorment. A continuació, amb l'ús de l'identificador de codi únic rebut, es fa una consulta al servei per saber si el codi es troba a la plataforma. Finalment, si no hi ha hagut cap error es crea un missatge de tipus ACCEPT-PROPOSAL, on s'inclou en el contingut el valor TRUE o FALSE per indicar a la plataforma iniciadora si la participant té el codi o no, respectivament.

En cas de qualsevol tipus d'error, s'envia un missatge de tipus REJECT-PROPOSAL indicant el motiu de l'error com a contingut.

- **FragmentTransferModeI**

  FragmentTransferModeI hereda de SimpleAchieveREInitiator, que implementa la part iniciadora del protocol d'interacció FIPA-Request. Aquesta classe és l'encarregada de realitzar les tasques corresponents a la segona part del protocol de migració fragmentada.

  El seu funcionament s'inicia amb el mètode PrepareRequest, d'on surtirà el missatge ACL que s'enviarà al participant. Com s'ha explicat en el disseny d'aquesta part del protocol, hi ha dos fluxes d'execució. L'elecció d'aquest s'aconsegueix consultant el valor booleà, emmagatzemat a l'etapa anterior, que indica si s'ha de transferir el codi o no.

  El missatge és crea amb l'ontologia Fragment Protocol Code Ontology. Al concepte d'aquesta s'afegiran els paràmatres necessaris per poder du a terme l'acció. Aquesta acció pot ser de dos tipus:

  – **Transferència del codi**: En cas d'haver d'enviar el codi, és necessari accedir al contingut del fitxer JAR. Per això s'usa un InputStream que

proporciona el servei, just abans de l'enviament del primer fragment, i que es guarda fins al final de la transferència del codi. El tamany de fragment pot estar especificat per defecte o per l'agent, per tant s'haurà de consultar si l'agent hi ha introduit un valor.

Després d'enviar el fragment, s'espera la resposta del participant. Si tot ha estat correcte, es rebrà un missatge ACK amb l'identificador del fragment. Això voldrà dir que ja es pot enviar el següent fragment, si en queden.

– **Registre de l'agent**: Acció de quan el codi ja s'ha enviat o simplement no ha fet falta fer-ho. S'envia un missatge al participant per demanar-li que ja pot registrar el codi i instal·lar l'agent. Si tot ha anat bé, es rebrà un missatge de tipus INFORM, llavors s'avisarà a la plataforma que la migració ha estat satisfactòria.

Per a qualsevol de les dues accions anteriors, si la resposta del participant es tracta d'un missatge de tipus FAILURE, s'informarà d'aquest fet i s'abortarà la migració.

En la secció 3.5 es parla sobre la concurrència del servei de mobilitat inter-plataforma a l'hora de l'enviament d'agents. Per aconseguir-ho es creen varies instàncies de les parts iniciadores dels protocols, i són reutilitzades per futures migracions. Perquè això sigui possible en el protocol fragmentat, s'han hagut d'inicialitzar els atributs de la classe per deixar-los preparats per la següent migració. Això s'ha de tenir en compte també quan es produeix qualsevol tipus de fallida que faci abortar la migració.

- **FragmentTransferModeR**

FragmentTransferModeR hereda de SimpleAchieveREResponder, que implementa la part participant del protocol d'interacció FIPA-Request. Aquesta classe implementa la part complementària de l'anterior.

El mètode prepareResultNotification s'executa quan es rep un missatge de la part iniciadora i realitza dos tipus de tasques depenent de l'acció que du

el missatge:

- Si l'acció pertany a la rebuda d'un fragment de codi, aquest s'emmagatzema amb l'OutputStream que es crea amb l'arribada del primer fragment. Si tot ha estat correcte, es crea un missatge de tipus INFORM on el seu contingut és l'ACK amb l'identificador del fragment rebut.
- Quan l'acció comporta el registre de l'agent a la plataforma, es registra el codi de l'agent, i amb la instància que havia emmagatzemat el behaviour ConfigModeI s'instal·la l'agent a la plataforma. Si el procés transcorre correctament es genera un missatge de tipus INFORM per indicar-ho a la plataforma iniciadora.

Si succeïx qualsevol tipus d'error, s'envia un missatge de tipus FAILURE on el contingut és l'error que s'ha produït.

Com s'explica en la secció 3.6 només s'executa una instància d'aquesta classe a cada plataforma. Això significa que s'ha d'implementar per que funcioni en cas d'haver varies migracions a l'hora, i preparar-ho per a futures migracions. Per això s'ha creat una estructura de dades (classe Arguments), que guarda els paràmetres necessaris per cobrir la transferència del codi d'un agent. Cada migració que s'està produint consta d'una instància d'aquesta estructura. Aquesta es guarda en una HashTable que s'accedeix pel nom de l'agent. D'aquesta manera és poden executar diferents etapes de diverses migracions de forma concurrent sense cap tipus de conflicte entre elles.

6.3 Ontologies

S'ha implementat les ontologies seguint el disseny realitzat a la secció 5.3. Com ja s'ha comentat s'ha dissenyat una ontologia per cada sub-protocol: Fragment Protocol Ontology i Fragment Protocol Code Ontology. Aquestes i la resta de classes pertanyents a les ontologies es troben al paquet *jade.core.migration.ontology*.

A JADE les ontologies s'implementen mitjançant classes que segueixen unes deteterminades interfícies (com podem veure a [JTO04]).

Les ontologies que s'han implementat estan basades en 4 elements: el vocabulari, els conceptes, les accions i la classe principal de l'ontologia que uneix els altres elements. Es possible implementar altres elements, com són els predicats, però en aquest protocol no fa falta perque només es vol du a terme accions. Seguidament es veurà que implementen aquests elements (Figura 6.2):

- **Vocabularis**: Contenen les cadenes de text que formaran les ontologies. Es troben a les classes FragmentProtocolVocabulary i FragmentProtocolCodeVocabulary en forma de variables estàtiques. Aquestes classes són interfícies que hereden de JADEManagementVocabulary i que són implementades per FragmentProtocolOntology i FragmentProtocolCodeOntology respectivament.

- **Conceptes**: En aquest projecte s'han utilitzat els conceptes per estructurar la informació que albergarà en l'ontologia, com s'observa en les taules de la seccions 5.3.1 i 5.3.2. Els conceptes a JADE són classes que hereden de Concept i contenen mètodes per accedir a la informació que vulguem posar a l'ontologia.

- **Accions**: Indiquen accions que ha de realitzar l'agent que rebi el missatge. Les accions són classes que implementen la interfície AgentAction. En el aquest protocol hem implementat tres accions: una per la part de negociació i dues pel segon sub-protocol.

- **Ontologies**: Són classes que hereden de Ontology i que implementen els vocabularis que s'han explicat. Com ja s'ha dit, s'en han implementat dos: FragmentProtocolOntology i FragmentProtocolCodeOntology. Són les classes principals de l'ontologia, per això s'ha hagut d'incloure les accions i els camps del concepte que s'han definit.

Aquesta implementació assegura que el contingut del missatge és congrüent amb l'ontologia. També permet canalitzar els missatges entrants cap al behaviour

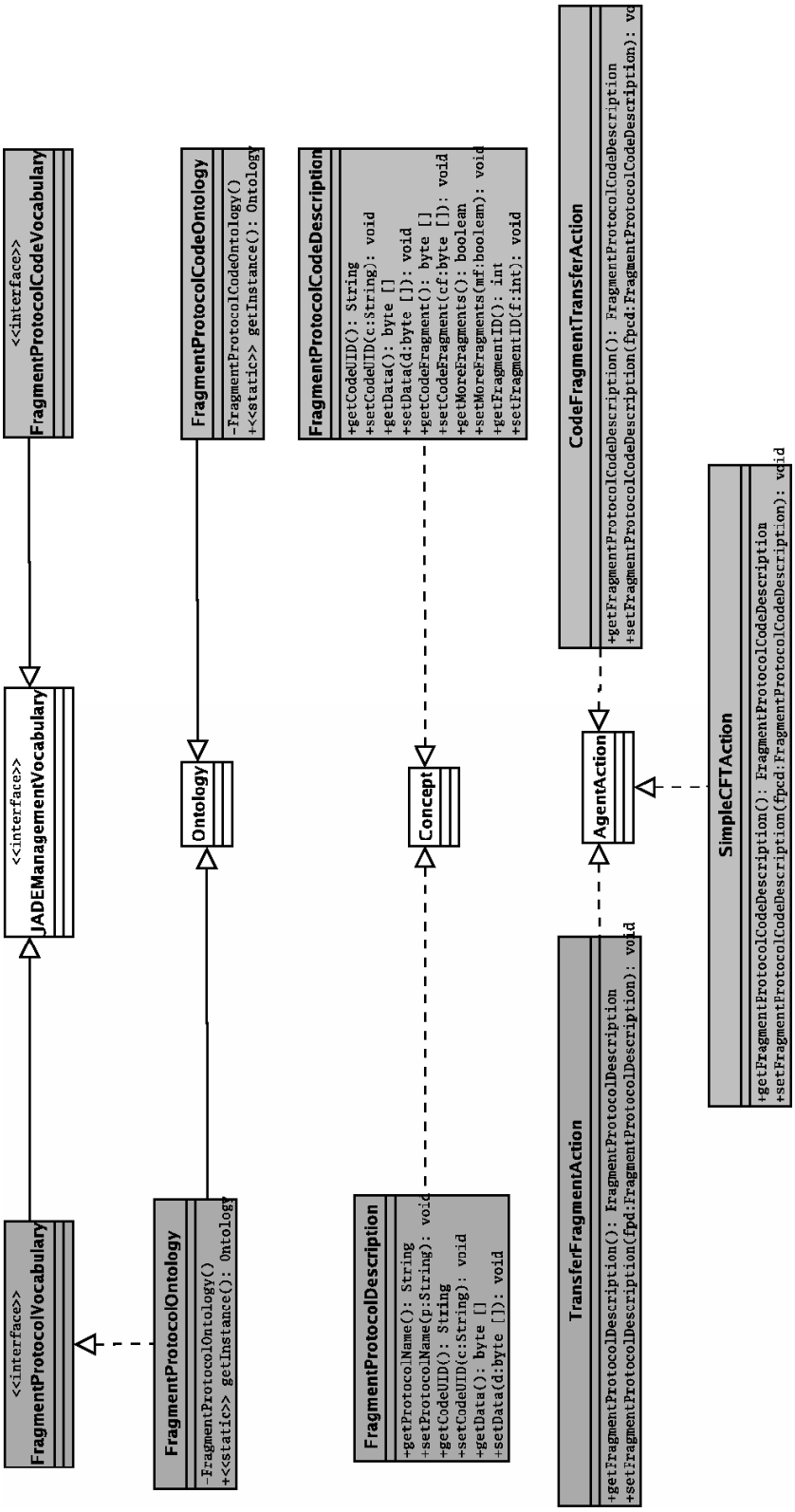

Figura 6.2: Diagrama de Classes de les Ontologies

dedicat a tractar-lo.

6.4 Integració amb el Servei

Després de la implementació del protocol i de les seves ontologies, cal afegir una sèrie de modificacions al codi del servei de mobilitat inter-plataforma. Aquests canvis es produeixen en tres classes del servei.

S'ha afegit la instància de FragmentTransferProtocol a la llista de protocols de l'etapa de transferència al mètode installAMSBehaviours de la classe CommandOutgoingFilter. També s'ha registrat les ontologies que usarà el protocol i finalment s'ha afegit els behaviours participants a l'agent AMS per què s'executin al iniciar la plataforma.

S'ha definit les transicions que seguirà la màquina d'estats del protocol fragmentat i s'han registrat els seus behaviours iniciadors, al constructor de la classe AMSInitiator. Aquesta classe és el behaviour que s'encarrega d'iniciar i controlar la migració d'agents desde la plataforma origen.

A la classe MobileAgentAccessor, que conté mètodes per gestionar el codi de l'agent, s'ha implementat el mètode getJarIS(AID agent) que retorna el InputStream d'on es troba el codi de l'agent. També s'ha escrit el mètode registerAgent(AID agent, String classCode, String codeSource) que registra el codi de l'agent passant-li la ruta del seu fitxer JAR (codeSource).

Per fer una migració amb el Protocol Fragmentat, l'agent demana a la plataforma la instància del Helper del servei, i amb el mètode *setDesiredSubProtocols* especifica el protocol de transferència amb el que vol migrar. Vegem com s'implementa això:

```
InterPlatformMobilityHelper sh=
        (InterPlatformMobilityHelper)
        getHelper(InterPlatformMobilityHelper.NAME);
ProtocolsList pl = new ProtocolsList();
pl.addProtocol("transfer_protocol");
sh.setDesiredSubProtocols
```

```
(ProtocolComponent.TYPE_TRANSFER, pl);
```

Per escollir el tamany de fragment s'han implementat dues possibilitats per facilitar a l'usuari la seva elecció:

- En l'arrencada de la plataforma podem especificar el tamany de fragment per defecte amb l'opció:

  -jade_core_migration_IPMS_FTP_transfer_fragment_size *[nova mida]*

- L'agent pot especificar el tamany de fragment utilitzant la instància del Helper del servei i el mètode *addSubProtocolProfile*:

```
ProtocolProfile pp = new ProtocolProfile
                ("fragment-transfer-protocol-v1");
pp.setProperty("fragmentSize", 15000);
sh.addSubProtocolProfile
        ("fragment-transfer-protocol-v1", pp);
```

Capítol 7

Test

Aquí es coneixerà quin ha estat el conjunt de proves que s'ha realitzat per medir el rendiment del protocol. S'analitzaran els resultats, valorant la migració d'agent en diferents situacions, i es compararan amb el rendiment del protocol clàssic.

7.1 Introducció

La última part del projecte és basa en comprovar el nivell d'èxit que s'ha aconseguit amb la implementació feta. Per tant, s'ha decidit comprovar el rendiment temporal del protocol, i per això s'ha realitzat una sèrie de proves per tal d'avaluar la seva resposta.

El que es vol fer és una comparació entre el protocol clàssic, que consistia en enviar l'agent en un únic missatge ACL, i el fragmentat, i en aquest últim observar com la variació del tamany de fragment afecta al seu temps de resposta. Per fer una bona comparació s'hauran de posar a prova migrant agents amb diferents tamanys de JAR.

La insfraestructura que s'ha utilitzat per a les proves es basa en un clúster de dues màquines. Cada màquina inclou una CPU Pentium IV 2.0 GHz, amb una memòria cache de 512 Kb i 755 Mb de RAM. Funcionen amb un sistema operatiu Linux, distribució Fedora Core 5, un kernel 2.6.17-1.2145 i un SWAP de 2047 Mb. Les màquines es conecten mitjançant un switch 3COM Gigabit Ethernet dedicat.

7.2 Eines per la creació de proves

Per la realització dels tests s'ha utilitzat un agent anomenat PerformanceAgent. Aquest s'encarrega de crear els agents i calcular els temps de migració. Aquest agent permet establir un conjunt de tests a través d'un fitxer de configuració XML. En aquest fitxer es pot indicar els agents i el nombre d'instàncies que es volen crear. A més també podem indicar el fitxer de propietats, on entre altres coses es troba l'itinerari a seguir i el nombre d'iteracions que s'en realitzarà. Internament, aquestes dues darreres propietats es passaran a l'agent com a paràmetre. Per tant, els agents que creem hauràn de contemplar aquesta condició. L'agent Performance calcula el temps de vida de l'agent i estima el de migració.

S'han creat 8 agents on els seu funcionament és el mateix: extreuen del fitxer de propietats l'itinerari a seguir i el protocol de transferència amb el que migraràn, si es tracta del protocol fragmentat també obtindran el tamany de fragments. Llavors, realitzen l'itinerari tantes vegades com ho indica el nombre d'iteracions. Finalment, abans de morir, envien un missatge ACL al PerformanceAgent per a que aquest enregistri el temps.

L'itinerari és per tots els agents el mateix, i fa fer anar l'agent de la plataforma actual fins a l'altra i tornar. La raó per la qual s'itera varies vegades aquest itinerari, és per despreciar el temps de creació i destrucció de l'agent, i només fixar-se amb el de migració.

El que es preten és aconseguir que cada agent tingui un tamany de JAR diferent. Per això s'han creat 8 fitxers JAR cadascún amb un tamany diferent: 3Kb, 10Kb, 25Kb, 50Kb, 100Kb, 250Kb, 500Kb i 1000Kb. També s'han creat 8 fitxers de propietats, un que especifica l'us del protocol de transferència clàssic i els altres 7 el protocol fragmentat amb tamany de fragment diferent: 1Kb, 3Kb, 5Kb, 10Kb, 15Kb, 20Kb i 25Kb. No s'ha volgut incrementar aquests 25Kb, ja que se sap que el rendiment disminueix a partit d'aquest tamany de missatge ACL.

7.3 Configuració dels tests

S'ha dividit el conjunt de test pel tamany del fitxer JAR de l'agent. Aquesta configuració permet comparar els diferents protocols en igualtat de condicions i veure com responen per un determinat tamany de codi.

S'han creat 8 fitxer de configuració XML que es passaran a l'agent Performance. En cada fitxer es defineixen les proves per agents del mateix tamany. Per tant, cada fitxer albergarà una prova amb el protocol de transferència clàssic i 7 amb les diferents combinacions del protocol fragmentat.

El nombre d'instàncies escollit per a cada agent ha estat 10. D'aquesta manera es pot comprovar que la concurrència del protocol funiciona davant de múltiples migracions. Hi ha hagut un test que llença una sola instància, ja que amb més d'una donava un error de memòria en la màquina virtual de Java. Ha estat la migració de l'agent de 1000 Kb amb el protocol clàssic.

Respecte al nombre d'iteracions escollit, aquest ha variat segons el test. Si en fixavem un que fos significatiu per a totes les proves, el temps de durada augmentava massa. Per tant, hem decidit escollir un nombre d'iteracions que despreciés el temps de creació i destrucció dels agents, tenint en compte el seu tamany i sense que la duracció fós massa gran.

Això ha fet falta, ja que al migrar diferents agents amb el mateix codi, en ocasions no s'hagués enviat.

Al migrar diferents agents amb el mateix codi, hi ha vegades que aquest no s'envia perque ja es troba en la plataforma destí. Per tant, s'ha hagut d'alterar la negociació prèvia sobre si el codi ja es trobava a la plataforma destí. Simplement, s'ha forçat que la resposta fós negativa, indicant que no es tenia el codi i haverlo d'enviar. Aquesta modificació l'hem hagut de fer tant pel protocol fragmentat com pel clàssic.

També s'ha hagut d'augmentar els timeouts dels protocols ja que els que hi havia per defecte eren superats per algunes migracions, sobretot les d'agents amb tamany de codi major.

	Clàssic	FP 1kb	FP 3kb	FP 5kb	FP 10kb	FP 15kb	FP 20kb	FP 25kb
JAR 3kb	76	105	92					
JAR 10kb	144	162	126	114				
JAR 25kb	419	315	194	175	175	201	241	
JAR 50kb	1035	527	291	245	241	277	322	710
JAR 100kb	2853	926	493	389	673	884	1192	1404
JAR 250kb	12167	2902	1918	1935	1836	2147	2826	3428
JAR 500kb	42048	5635	3732	3240	3592	4467	5523	6636
JAR 1000kb	338953	11094	7144	6317	7176	8756	11880	12197

Taula 7.1: Taula de resultats (ms)

7.4 Resultats

Després de llargues hores de proves, els resultats obtinguts han estat positius. Si ens fixem amb la Taula 7.1, i fem un cop d'ull general als temps obtinguts en les diferents proves, es veu que excepte amb els agents amb JAR 3 i 10 kb, en la resta el protocol fragmentat obté millors temps en les diferents variacions que el protocol clàssic.

Anem a observar més detalladament els resultats obtinguts, i a analitzar-los segons el tamany del JAR. En la Figura 7.1, es poden veure els temps de migració dels agents de 3 kb fins a 50 kb. Si ens fixem amb els agents de 3 kb surt més a compte enviar tot el codi en un mateix missatge ACL, en canvi en agents de 10 kb ja no. Encara que la diferència de temps no sembla gaire significativa, enviant el codi en fragments de 3 i 5 kb s'obté una petita millora temporal que si l'enviem tot a l'hora.

Tot això canvia quan l'agent té un pes igual o superior a 25 kb. Observant els dos darrers conjunts de barres de la Figura 7.1, ens comencem a adonar del que pot venir per a agents amb codi molt gran. El temps que s'obté amb el protocol clàssic comença a ser molt superior a la resta. Ens fixem també que amb fragments de 1 kb el nombre de missatges ACL que s'envien és prou alt com per no obtenir el millor rendiment del protocol. El mateix passa amb fragments de 25 kb, encara que el nombre de missatges es molt menor, el tamany és massa gran i el temps empitjora. Per tant, sembla que per agents entre 25 i 50 kb, el millor és escollir un

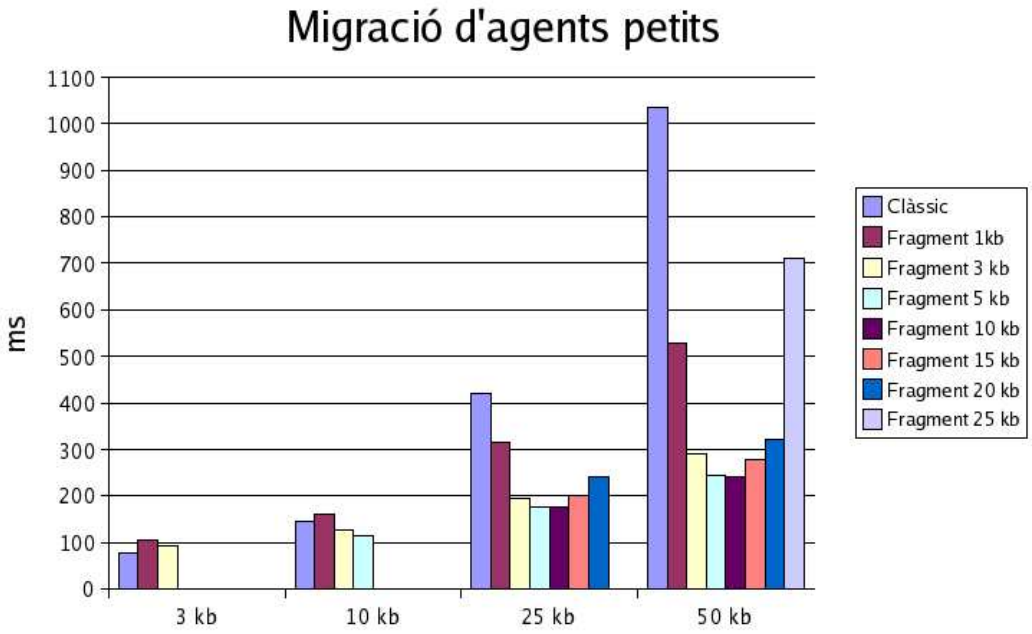

Figura 7.1: Migració d'agents petits

tamany de fragment entre 3 i 20 kb.

En el diagrama de la Figura 7.2, no hi hem inclòs els resultats corresponents a la migració amb el protocol clàssic, ja que al ser molt més grans emmascaraven la resta. Tot i això, si es miren aquests resultats a la Taula 7.1, es pot veure que són notablement superiors a la resta. Per això es miraran més les diferents respostes temporals del protocol fragmentat, i s'analitzarà en cada cas quin tamany de fragment és més adequat.

Si es dóna un cop d'ull general a la Figura 7.2, es veu que en l'elecció de tamanys de fragment 1 kb i 25 kb, passa el mateix que en la migració de JARs de 25 kb i 50 kb. Per tant, es pot assegurar que en la migració fragmentada per agents amb una mida de codi igual o superior a 25 kb, no s'obtenen els millors resultats possibles.

També es pot observar que conforme el tamany de JAR augmenta, la migració amb fragments de 20 kb s'observa cada cop més llunyana de l'òptima. Per tant, el rang de tamany de fragment òptim queda entre 3 i 15 kb per als diferents tamanys

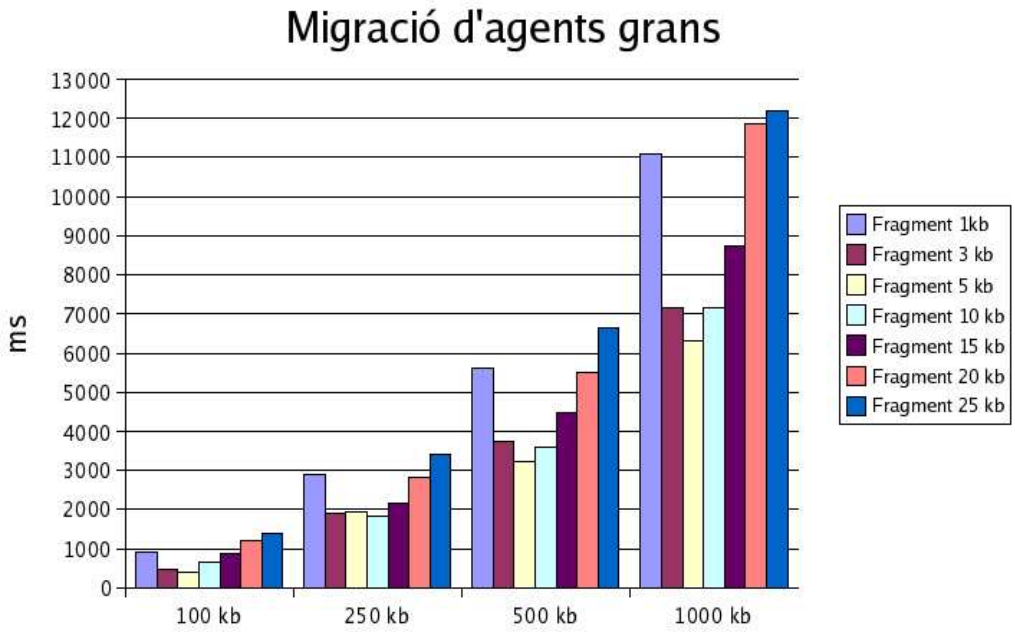

Figura 7.2: Migració d'agents grans

de JAR que hem escollit.

En la Figura 7.2 es pot observar, fent una ullada per sobre, quin pot ser el tamany de fragments òptim per migrar agents amb codi gran. Tot i això, mirant la Figura 7.3 es pot veure l'evolució del protocol fragmentat amb l'ús de mides de fragment diferents i com és la seva resposta temporal pels diferents tamanys de codi. Aquí es pot veure clarament que amb conforme incrementa el tamany de JAR, el tamany de fragment que dóna més bon rendiment és el de 5 kb.

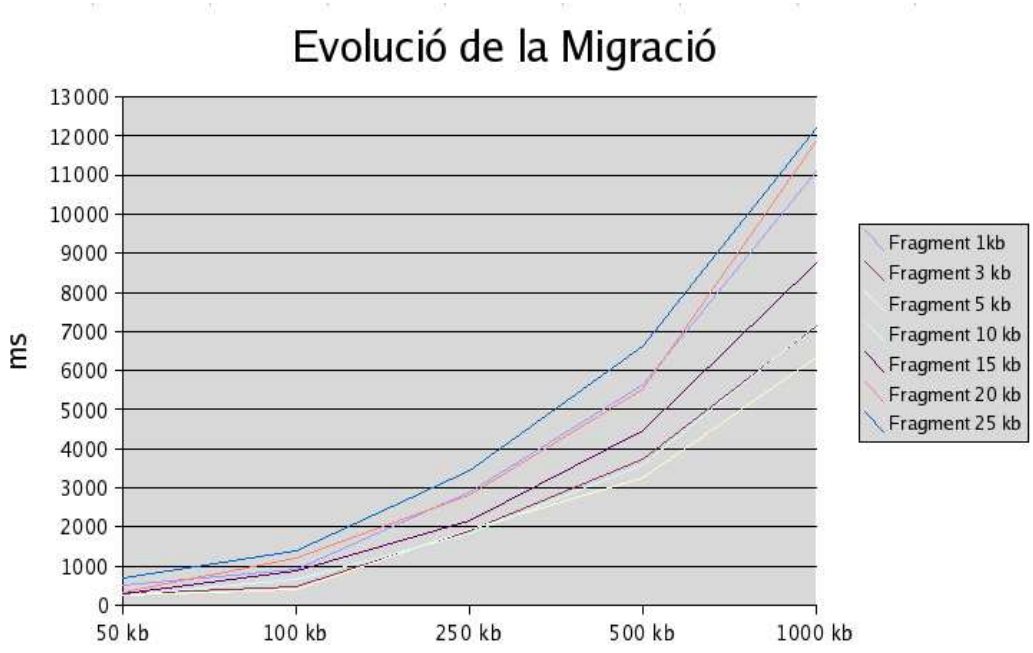

Figura 7.3: Evolució de la migració fragmentada

Capítol 8

Conclusions

El projecte que s'ha desenvolupat, i que s'explica en aquesta memòria, tracta sobre la migració fragmentada d'agents de JADE. El que s'ha volgut fer és un estudi sobre com es podia dur a terme aquest tipus de mobilitat, i proposar una implementació d'un protocol que s'adaptés a l'arquitectura multi-protocols del servei de mobilitat inter-plataforma (IPMS).

La planificació que s'ha fet a l'inici del projecte ha estat encertada en quan a les tasques que s'havien de realitzar, però els temps previstos no han estat exactament els que s'havien marcat.

Els estudis prèvis s'han dut a terme en el temps indicat, tot i que durant la realització de les posteriors tasques s'han hagut de revisar alguns punts que no havien quedat clars. Les tasques d'anàlisi i disseny s'han dut a terme amb un temps inferior al previst. Això ha provocat que la implementació es pugués començar i acabar abans de la data prevista. La part de proves de rendiment s'ha fusionat bastant amb l'escriptura de la memòria, ja que després de la configuració dels tests s'ha pogut començar a escriure la present memòria mentre aquests s'anaven realitzant. El procés de redacció d'aquesta memòria ha estat més complicada del previst inicialment i ha allargat el temps reservat per la seva elaboració.

A continuació s'explicarà com s'han desenvolupat els diversos objectius que s'han plantejat a l'iniciar el projecte. Es comentarà en cada cas quines han estat les dificultats trobades i com s'han solventat.

El primer objectiu plantejat, ha format part de l'estudi de l'estat de l'art, i pretenia realitzar-lo sobre el servei de mobilitat IPMS que desenvolupa el grup de recerca SENDA. Per iniciar l'elaboració del projecte s'ha necessitat conèixer el funcionament del servei i l'arquitectura de protocols que implementa. Per això s'ha volgut fer un estudi sobre el disseny inicial de la mobilitat del servei i conèixer quines han estat les raons per la seva evolució a un conjunt de protocols.

Com a recolzament per aquests coneixements s'ha hagut de fer un estudi sobre com la plataforma JADE implementa els seus agents i l'estructura de serveis. Encara que aquest últim no ha estat rellevant pel projecte ha ajudat a entendre com està implementat el servei de mobilitat inter-plataforma.

El segon objectiu consistia en fer un anàlisi sobre la migració fragmentada. En aquesta part s'ha determinat un conjunt de requisits que havia de complir el protocol. Aquests requisits han abastat aspectes tant del disseny del funcionament del protocol, com de la implementació dins el servei de mobilitat. Per tant, ha significat un punt de partida pel desenvolupament de la migració fragmentada.

Com a part d'aquest segon objectiu, s'ha plantejat la realització del disseny del protocol. En aquest protocol s'han hagut de prendre un conjunt de decisions sobre aspectes reflexats en l'anàlisi. Un dels més significatius ha estat que el protocol havia de fer ús dels estàndards de FIPA. Això és bàsic, ja que el projecte IPMS, usa les especificacions de FIPA pel seu desenvolupament.

El tercer objectiu s'ha basat en dur a terme la implementació del protocol. Aquesta implementació ha estat marcada pel requisit d'haver-lo d'adaptar a l'estructura de protocols que defineix el servei de mobilitat IPMS. Això vol dir que una part de la implementació ha estat lligada al funcionament del servei que s'ha de respectar. La resta, s'ha basat en el disseny que s'havia realitzat prèviament.

Un aspecte important de la implementació ha estat la necessitat que havia de soportar el protocol en el seu behaviour participant per rebre més d'un agent concurrentment. En les proves de rendiment que s'han produit posteriorment s'ha pogut comprovar que això funcionava correctament.

El quart objectiu s'ha basat en fer proves de rendiment del protocol implementat comparant-lo amb el que ja hi havia, anomenat protocol clàssic, el qual envia

l'agent en un sol missatge. Per fer això s'ha dissenyat un conjunt de tests per evaluar la implementació del protocol que s'ha implementat. Amb l'ús de l'agent que implementava els tests, hem configurat les diverses proves de manera ràpida i fàcil. Això ha ajudat a poder dedicar més temps a d'altres aspectes del projecte, com és la redacció d'aquesta memòria. Els resultats han estat força satisfactoris, obtenint valors temporals en el protocol fragmentat fins a trenta vegades millor que en el clàssic.

L'últim objectiu plantejat, ha consistit en optimitzar el tamany de fragment. S'ha realitzat un petit estudi per comprovar quin és el millor tamany de fragment que es pot escollir. S'han observat els resultats obtinguts, i aquests han donat com a guanyador el fragment de 5 kb per a migracions d'agents grans.

Per tant, fent una avaluació general sobre els objectius que s'havien plantejat, es podria dir que aquests s'han resolt satisfactòriament, encara que s'haguessin pogut millorar si s'hagués disposat de més temps.

Com a línies de futur, es poden realitzar una sèrie de millores i prestacions que ajudarien al protocol a obtenir millors resultats. A continuació s'en comentaran unes quantes que són molt interessants.

En primer lloc, el disseny es podria simplificar utilizant una ontologia en comptes de dos com s'ha implementat en aquest projecte. Es creu que amb l'ús de dos conceptes i tres accions dins una sola ontologia seria suficient per estructurar la informació que s'han d'intercanviar les plataformes. Això no s'ha fet així per simplificar la implementació del tractament dels missatges.

Un aspecte, que ja s'ha comentat a la part d'anàlisi, sobre el funcionament del protocol, és l'enviament de confirmació de fragments. En aquesta primera aproximació a la migració fragmentada s'envia un ACK per cada fragment rebut i el següent fragment no s'envia fins que no es rep l'ACK del fragment actual. Aquest aspecte té múltiples millores diferents que es podrien aplicar. Potser la més atractiva seria la de l'ús d'algun mecanisme tipus finestres lliscants de TCP, per aprofitar més el throughput de la xarxa. En aquest mecanisme es controlen els fragments que no han estat confirmats i si el nombre supera el tamany de la finestra, no s'en envien més.

Una altra millora que es va descartar per simplificar la implementació, té a veure amb la retransmissió de fragments. En aquesta implementació quan es produeix algun tipus d'error, automàticament se suspèn la migració de l'agent informant a les dues plataformes implicades. Però seria més interessant, aplicar-hi alguna política de retransmissió en cas de que es perdés algun fragment. En les proves que s'han realitzat no ens hem trobat amb aquesta situació ja que s'ha treballat sempre a nivell de xarxa local, sense passar per cap router.

Una altra línia de futur, és respecte l'elecció del tamany de fragment, seria que aquesta es podria realitzar automàticament de forma intel·ligent a mida que es van enviant fragments. La idea seria calcular els temps de transferència dels fragments durant una migració i anar adaptant-los per obtenir millors resultats.

Per acabar, una última línia de futur consistiria amb la possibilitat de fragmentar la instància. En l'anàlisi del projecte, es va descartar aquesta possibilitat, perquè es va pensar més amb la migració d'agents amb tamany de codi gran i instància petita. Però el gran abast de les aplicacions basades en agents, com ara aquelles on aquests recorren les plataformes recopilant un conjunt de dades, fa pensar que en algunes ocasions valdria la pena fragmentar també la instància quan aquesta fos gran.

A nivell personal aquest projecte m'ha servit per poder comprovar com es desenvolupa un projecte d'enginyeria. M'ha donat la possibilitat de posar en pràctica tots aquells coneixements que havia adquirit durant la carrera. A més, el paradigma dels agents mòbils és un tema que m'interessava molt i he tingut l'oportunitat de conèixer-lo més a fons i treballar en una eina, IPMS, que pot servir per altres persones en un futur.

Bibliografia

[JADE] Java Agent DEvelopment Framework.
<http://jade.tilab.com/>

[FIPA23] FIPA Agent Management Specification, 2004.
<http://www.fipa.org/specs/fipa00023/index.html>

[FIPA67] FIPA Agent Message Transport Service Specification, Desembre 2002.
<http://www.fipa.org/specs/fipa00067/index.html>

[FIPA36] FIPA Propose Interaction Protocol Specification, Desembre 2002.
<http://www.fipa.org/specs/fipa00036/index.html>

[FIPA26] FIPA Request Interaction Protocol Specification, Desembre 2002.
<http://www.fipa.org/specs/fipa00026/index.html>

[GC04] G. Caire. JADE: the new kernel and last developments, 2004.
<jade.tilab.com/papers/Jade-the-services-architecture.pdf>

[DMA07] F. Bellifemine, G. Caire, D. Greenwood. Developing Multi-Agent Systems with JADE, 2007.

[IPMS] JADE Inter-Platform Mobility Service. Source Project, 2007.
<http://sourceforge.net/projects/jipms>

[MPA] J. Cucurull, R. Martí, G. Navarro, J. Borrel and S. Robles. A Multi-Protocol Architecture for Agent Migration. Technical Report.

[CMS] J. Cucurull. Contribució de la Mobilitat i Seguretat dels Agents Software, Maig 2006.
<https://tao.uab.cat/ipmp/files/jcj-master-thesis.pdf>

[JMT04] J. Cucurull. JADE MTP-TFTP, Juny 2004. Projecte Final de Carrera.

[JTO04] G. Caire and D. Cabanillas. Jade Tutorial: Application-defined content languages and ontologies. TILab S.p.A. Novembre 2004
<http://sharon.cselt.it/projects/jade/doc/CLOntoSupport.pdf>

Firmat: Víctor Sales Barberà

Bellaterra, Juny de 2007

Resum

En aquest projecte s'ha realitzat l'anàlisi, disseny i implementació d'un protocol de migració d'agents software basat en l'enviament del codi dels agents fragmentat en múltiples missatges. Aquest protocol es troba dins d'un arquitectura de migració multi-protocol per la mobilitat d'agents entre plataformes JADE. Finalment, s'ha realitzat un estudi comparant el rendiment assolit pel protocol i les prestacions que aporta.

Resumen

En este proyecto se ha realizado el análisis, diseño e implementación de un protocolo de migración fragmentada de agentes software basado en el envío del código de los agentes fragmentado en múltiples mensajes. Este protocolo se encuentra dentro de un arquitectura de migración multi-protocolo para la movilidad de agentes entre plataformas JADE. Finalmente, se ha realizado un estudio comparando el rendimiento logrado por el protocolo y las prestaciones que aporta.

Abstract

In this project it has been made the analysis, design and implementation of a software agents migration protocol based on the shipment of multiple messages with the fragmented code of the agents. This protocol is implemented within a multi-protocol migration architecture for the mobility of agents between JADE platforms. Finally, a study has been made comparing the performance obtained by the protocol and the benefits of the contribution.