DESCOBRIMENT DE SERVEIS EN XARXES AD-HOC: DIRECTORY FACILITATOR

Memòria del projecte de final de carrera corresponent als estudis d'Enginyeria Superior en Informàtica presentat per Sílvia Ezponda Mares i dirigit per Ramon Martí Escalé.

Bellaterra, Febrer de 2007

El firmant, Ramon Martí Escalé , 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 Sílvia Ezponda Mares

Bellaterra, Febrer de 2007

Firmat: Ramon Martí Escalé

Al Jordi, a la Marta, als meus pares
i als amics.

Agraïments

Gràcies al grup de recerca SENDA i al grup de projectistes per estar sempre a punt per donar-nos un cop de mà.
Gràcies als amics estar amb mi en els bons i mals moments, en especial a l'Andreu, al Javi, a l'Artur, al Miguel, al Toni i al Carlos.
I finalment gràcies a la família pel seu suport incondicional.

Índex

Índex de figures

Capítol 1

Introducció

1.1 Motivació

Els avenços en el desenvolupament de tecnologies sense fils han afavorit la comunicació entre petits dispositius amb comunicació i energia limitats. Les PDAs (*Personal Digital Assistants*) i els telèfons mòbils són els més visibles, però existeixen molts més dispositius d'aquest tipus al nostre voltant que passen desapercebuts. Per exemple els aparells quotidians que tenim a casa porten incorporats microprocessadors i cadascún d'aquest aparells ens proporciona un servei específic (rentadores, microones, etc.). En un futur tots aquest dispositius seran capaços de col·laborar els uns amb els altres per proporcionar-nos serveis més complexes.

Per poder aconseguir aquesta fita els dispositius han de poder descobrir-se dinàmicament i poder compartir serveis quan estiguin suficientment a prop, tal i com fan els nodes en xarxes *ad hoc*.

Una xarxa *ad hoc* és aquella que està formada per nodes mòbils els quals utilitzen interfícies sense fils per enviar i rebre dades sense cap infraestructura permanent i centralitzada. La topologia dinàmica de les xarxes *ad hoc* és un punt molt important a tenir en compte a l'hora de definir noves propostes ja que les limitacions dels dispositius mòbils són evidents (memòria, capacitat de processament i energia).

Si en un entorn tan inestable afegim un gran volum d'intercanvi d'informa-

ció veiem que el concepte d'agent mòbil adquireix una importància notable. Els agents es defineixen com entitats autònomes, que es comporten segons els objectius que volen assolir, reaccionen davant events externs i poden comunicar-se i col·laborar amb altres agents. A més si són mòbils poden migrar entre nodes d'una mateixa xarxa, cosa que fa que s'adaptin millor que les aplicacions client-servidor quan les xarxes tenen connectivitat intermitent com és el cas de les xarxes *ad hoc* [Cel04].

De tota manera l'ús de sistemes multi-agent en entorns dinàmics comporta reptes importants, i per tant, se'ns fa necessari adaptar el seu disseny per poder evitar haver de superar grans barreres. Un d'aquests reptes és el servei de pàgines grogues en plataformes d'agents per a xarxes *ad hoc*.

En aquest document presentarem una solució integrada amb les especificacions de la FIPA (*Foundation for Intelligent Physical Agents*) d'un servei de pàgines grogues per a entorns *ad hoc*. L'actual plataforma d'agents de JADE ja compta amb aquest servei per a entorns estàtics, a través del *Directory Facilitator*. Mantindrem aquesta funcionalitat en el nou servei i oferirem als agents una nova eina, incorporant extensions anomenades *Discovery Middleware* per interoperar amb diverses tecnologies *ad hoc* (JXTA, Bluetooth, IrDA, etc.).

1.2 Objectius

El nostre projecte titulat "**Descobriment de serveis en xarxes *Ad hoc*: Directory Facilitator**" es planteja els següents objectius:

1. Estudiar les especificacions proporcionades per la FIPA en relació amb el servei de pàgines grogues.

2. Estudiar les tecnologies proposades per la FIPA sobre descobriment de serveis en xarxes *ad hoc*.

3. Estudiar la plataforma d'agents de JADE, veient els modes d'execució i com treballa el DF dins d'aquesta.

4. Analitzar, dissenyar i desenvolupar un servei de pàgines grogues integrable amb l'actual plataforma d'agents de JADE i amb els mòduls de *Discovery Middleware*.

5. Dissenyar conjuntament amb el desenvolupador del *Discovery Middleware* de JXTA una interfície de comunicació entre el nou DF i els DM, que sigui compatible amb qualsevol implementació de DM.

6. Crear un *add-on/plug-in* per JADE que faciliti la incorporació d'un nou servei en una plataforma, el DF-DM per JXTA.

1.3 Contingut de la memòria

L'estructura de la memòria està basada en una sèrie de capítols on seguidament especifiquem una breu descripció de cadascun:

Capítol 2: Agents mòbils i Descobriment de serveis. En aquest capítol comentarem els elements principals de la plataforma d'agents de JADE (*Java Agent DEvelopment Framework*) que és sobre la qual treballa el servei de pàgines grogues que desenvoluparem i els conceptes de xarxa *ad hoc* i middleware.

Capítol 3: Anàlisi. Dins aquest capítol veurem els requisits de l'aplicació tant funcionals com operacionals, proporcionats per les característiques de l'entorn i per les especificacions de la FIPA. Comentarem les diferents alternatives que hem estudiat per utilitzar la xarxa d'una manera òptima i triarem la que servirà de base per a la nostra implementació.

Capítol 4: Disseny. En el capítol de disseny desenvoluparem a fons la opció que hem triat per implementar el *Ad hoc Directory Facilitator* amb l'ajuda dels diagrames UML i finalment veurem la interfície entre l'ADF i els DM.

Capítol 5: Implementació. Aquí veurem les entranyes de l'aplicació. Comentarem com hem implementat el nou DF, les classes que ens han fet falta i explicarem cadascun dels mètodes que han estat necessaris perquè tot funcioni correctament.

Capítol 6: Proves. En el capítol de proves i resultats veurem quines són les

proves que hem realitzat per saber si l'aplicació funciona de manera correcta, tractarem les proves de caixa blanca i les de caixa negra per elaborar un procediment a fons.

Capítol 7: Conclusions. Per finalitzar el document exposarem les conclusions que podem extreure de tot el treball realitzat. Revisarem els objectius assolits i comentarem les possibles línies de continuïtat del projecte.

Capítol 2

Agents mòbils i descobriment de serveis

Un cop vista la introducció al problema a desenvolupar, i abans de continuar, necessitem fer un incís en alguns conceptes generals per tal de descriure amb profunditat el projecte i els seus requisits.

L'objectiu d'aquest capítol és explicar el paradigma dels agents mòbils, les eines que utilitzarem per treballar sobre aquests agents i els estàndards que trobem. També introduirem els conceptes de *directory facilitator*, un tipus d'agent especial creat per JADE sobre el qual es basa el projecte, xarxa *ad hoc*, escenari sobre el que treballarem i per acabar la noció de *middleware*, eina necessària pel descobriment de nous serveis.

2.1 Agents mòbils i sistemes multi-Agent

Els agents software són entitats que poden actuar en nom d'una altra entitat i que tenen certes propietats, entre altres: autonomia, cooperació, intel·ligència i mobilitat. El paradigma d'agents està basat en l'abstracció d'un agent: un component software que és autònom, proactiu i social.

- Autònom: els agents tenen un grau de control sobre les seves accions, i sota algunes circumstàncies, són capaços de prendre decisions.

- Proactiu: els agents no només reaccionen en resposta a events externs, sinó que també presenten un comportament en la direcció dels seus objectius, i són capaços de prendre la iniciativa.

- Social: els agents són capaços i necessiten interactuar amb altres agents per complir certes tasques i aconseguir completar l'objectiu del sistema.

La mobilitat, però, no és una característica comuna a tots els agents. En aquest document trobarem dos tipus d'agents:

1. Agents estacionaris: aquests agents s'executen únicament al sistema on inicien la seva execució. Si necessiten informació que es troba en algun altre sistema fan servir mecanismes de comunicació remota (crida a mètodes remots).
   exemple: agents d'interfície (el RMA, el DF, el AMS,...)

2. Agents mòbils: no estan acotats pel sistema en el qual inicien la seva execució. Els agents mòbils són capaços de moure's per la xarxa, transporten el seu codi amb ells, creant d'aquesta manera un àmbit d'execució.
   exemple: agents d'aplicació

Bàsicament, un agent és una entitat de software que té un comportament autònom i un sistema multi-agent és un conjunt d'agents que tenen la capacitat d'interactuar en un entorn comú, per al qual necessiten disposar de les capacitats de comunicació, negociació i coordinació.

2.2 FIPA

La FIPA (Foundation for Intelligent Physical Agents) és una organització internacional que es dedica a promoure la indústria dels agents intel·ligents a través del desenvolupament obert d'especificacions que acrediten la interoperabilitat entre agents desenvolupats en marcs de treball els quals suporten transports heterogenis i aplicacions basades en agents. La col·lecció d'aquests serveis i la seva interfície estàndard, formen la normativa que permet a les societats d'agents existir i operar.

A l'actualitat pot ser considerat l'estàndard d'agents més àmpliament reconegut i extès internacionalment, i s'ha convertit en un referent per al desenvolupament de sistemes basats en agents.

2.3 JADE

Per la seva execució, els agents necessiten d'un software especial que els proveeixi d'una sèrie de serveis bàsics: identificació, mobilitat, comunicació, seguretat, etc. JADE (Java Agent DEvelopment Framework) és una plataforma de software desenvolupada al TILab (Itàlia) sota la filosofia Open Source per al desenvolupament d'aplicacions distribuïdes basades en agents, que simplifica la implementació de sistemes multi-agent i que segueix les especificacions de la FIPA.

JADE està totalment desenvolupat en Java. L'arquitectura de JADE està basada en contenidors en els quals resideixen els agents. El conjunt de contenidors formen una plataforma i així es proveeix d'una capa homogènia per als agents, és a dir, el desenvolupament de les aplicacions es faria sense importar hardware, sistema operatiu, tipus de xarxa o JVM (*Java Virtual Machine*).

JADE presenta les següents característiques:

- P2P: Arquitectura *peer-to-peer*, cada agent pot prendre la iniciativa en una comunicació o bé respondre a peticions que facin altres agents.
- Interoperabilitat: JADE compleix amb les especificacions de la FIPA, per les quals els agents desenvolupats en JADE poden interactuar amb altres agents que no tenen perquè estar desenvolupats amb JADE però si que han de complir les especificacions.
- Portabilitat: la API que proporciona JADE és independent de la xarxa sobre la qual es vol operar, així com de la versió de Java utilitzada.
- Intuïtiva: JADE s'ha desenvolupat per oferir una API fàcil d'aprendre i senzilla de manegar.

JADE proporciona aquestes i altres eines com poden ser:

- Administració remota d'agents: mitjançant una interfície gràfica.
- És configurable: permet fer servir només les característiques que es requereixin, disminuint la sobrecàrrega computacional.
- Descobriment dinàmic: els agents tenen noms únics

L'esquema bàsic d'una plataforma d'agents seria el següent:

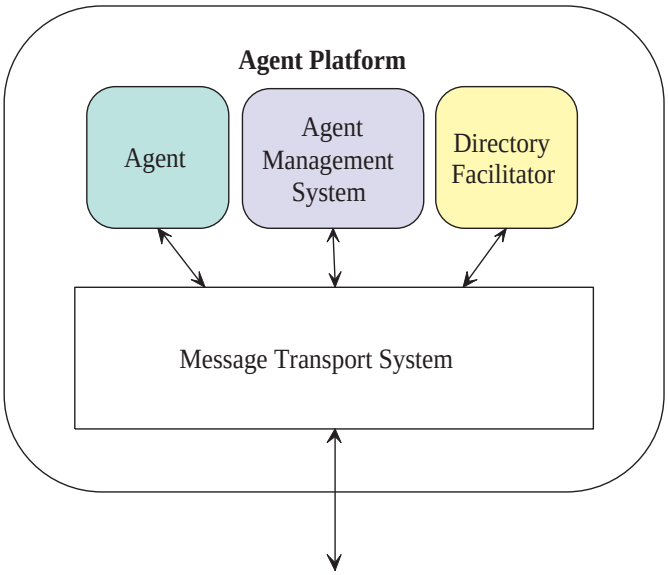

Figura 2.1: Arquitectura FIPA d'una Plataforma d'Agents

2.4 La plataforma de JADE

La plataforma d'agents proporciona una infraestructura física en la qual els agents poden ser desenvolupats. Una plataforma està formada per màquines, sistema operatiu, software de suport, components de gestió de la FIPA (DF, AMS i MTS) i agents. Com vèiem a la figura 2.1, segons la FIPA una plataforma d'agents conté els següents elements bàsics:

- **Agent Management System:** representa l'autoritat de la plataforma i s'encarrega de la supervisió i el control d'accés, així com l'ús que se'n fa de

la mateixa. Proveeix el servei de noms (s'encarrega que cada agent tingui un identificador vàlid i únic) i proporciona el servei de pàgines blanques i cicle de vida (mantenir un directori d'agents residents a la plataforma i el seu estat).

- **Agent Communication Channel:** és l'encarregat de controlar l'intercanvi de missatges entre agents. Si la comunicació es realitza entre diferents plataformes també necessitarem el *Message Transport Protocol* per poder establir la connexió.

- **Directory Facilitator:** el DF proveeix de pàgines grogues, és a dir, que un agent pot sol·licitar serveis que algun dels agents de la plataforma proveeix. Si un agent vol publicar els seus serveis ho haurà de notificar al DF i quan un agent sol·liciti un servei el DF li comunicarà quin o quins agents el proporcionen.

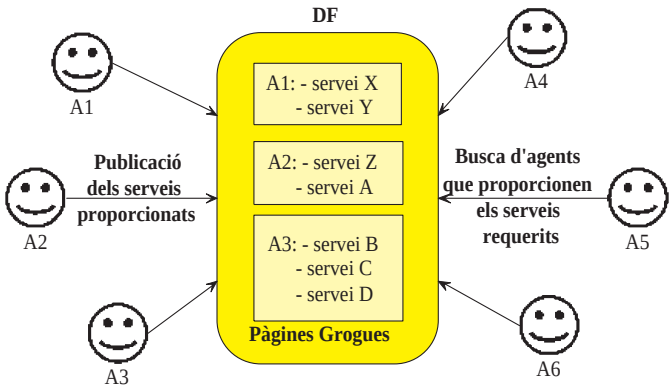

Figura 2.2: Esquema d'un Directory Facilitator

Per poder accedir al directori on es troben les descripcions dels agents administrades pel DF, cada DF ha de se capaç de realitzar les operacions de *registre, desregistre, modificació* i *cerca*. A més un DF també pot suportar el mecanisme de *subscripció*, mecanisme pel qual el DF ha d'implementar el protocol d'interacció de subscripció [SIPS02] per poder permetre als agents que es subscriguin per ser notificats sobre registres, desregistres i modificacions de certs agents.

Segons la FIPA, el DF és un component opcional d'una plataforma d'agents. Tanmateix una plataforma d'agents pot suportar qualsevol nombre de DFs i alhora, els DF compten amb un mecanisme, anomenat **federació**, pel qual es poden registrar els uns amb els altres, estiguin o no a la mateixa plataforma.

Amb la federació es reforça el mecanisme de cerca del DF. Quan un agent envia una cerca, aquesta cerca es realitza primer de manera local i després es determina si la cerca s'exten als DFs federats.

2.5 Xarxes *Ad hoc*

Una MANET (*Mobile Ad-hoc NETwork*) descriu un sistema de nodes mòbils sense fils que poden autoorganitzar-se de forma dinàmica i lliure dins d'unes topologies de xarxa temporals i arbitràries, permetent a persones i dispositius actuar conjuntament en àrees sense cap infraestructura prèvia de comunicació.

La connexió a una MANET s'estableix per una sessió de durada i no requereix estació base (o punt d'accés). En el seu lloc, els dispositius es descobreixen els uns als altres dins d'un radi per formar una xarxa per a aquests computadors. Els dispositius hauran de buscar els nodes que estiguin fora de l'abast inundant la xarxa amb *broadcasts* que alhora seran reenviats per cada node.

Les connexions son possibles a través de múltiples nodes (xarxa *ad hoc* multisalt). Aleshores, els protocols d'encaminament proveeixen connexions estables encara que els nodes canviïn de lloc.

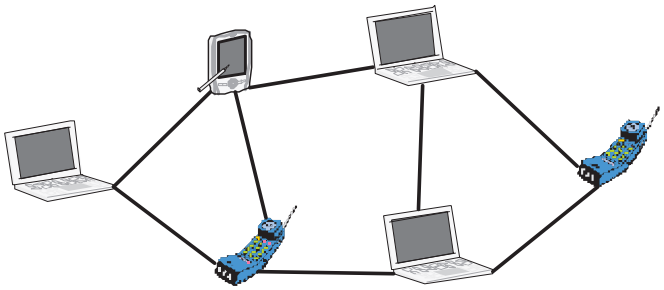

Figura 2.3: Exemple de Xarxa *Ad hoc*

2.6 Middleware

Software Intermediari, p *middleware* en anglès, és un software de connectivitat que permet oferir un conjunt de serveis que fan possible el funcionament d'aplicacions distribuïdes sobre plataformes heterogènies, és a dir, un software que permet que qualsevol aplicació pugui executar-se sobre qualsevol hardware o sistema operatiu.

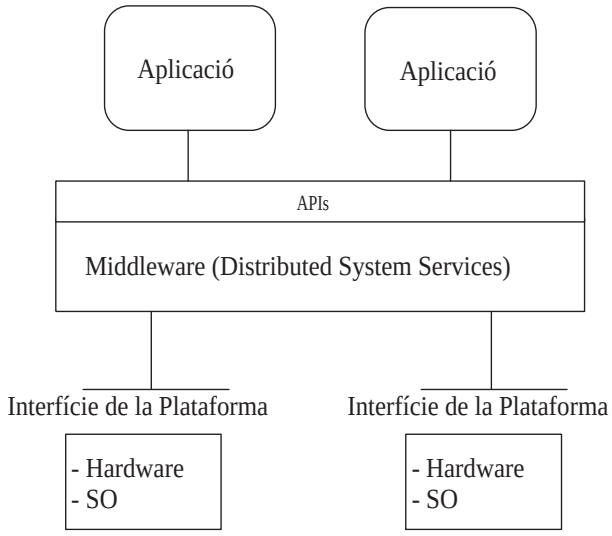

Figura 2.4: Middleware

Un *middleware* pot dur a termes les següents funcions:

- Amagar la distribució, per exemple, el fet que normalment algunes aplicacions estan formades per varies parts interconnectades que estan en funcionament en localitzacions distribuïdes.

- Amagar la heterogeneïtat de diferents components hardware, sistemes operatius i protocols de comunicació.

- Proveir als desenvolupadors i integradors d'aplicacions unes interfícies uniformes, estàndards i d'alt nivell, d'aquesta manera les aplicacions poden reutilitzar-se, ser portables i ser interoperables.

- Facilitar un conjunt de serveis comuns per proporcionar varies funcions de propòsit general, per evitar duplicar esforços i facilitar la col·laboració entre aplicacions.

JADE en molts sentits és considerat com un *middleware* ja que proporciona tant un entorn de desenvolupament com un entorn d'execució per a la realització i manteniment de sistemes multiagent.

2.7 Descobriment de serveis en xarxes *ad hoc*

La primera especificació de la FIPA relacionada amb entorns dinàmics va ser proposada al Novembre del 2003, la *Agent Discovery Service Specification* [ADSS03]. En aquesta especificació es defineix un nou agent, l'ADS (*Agent Discovery Service*), que substitueix la funcionalitat del DF en xarxes *ad hoc*, manté les funcions bàsiques: registre, desregistre i cerca, i n'introdueix dues noves: subscripció, que permet als agents rebre notificacions quan es descobreixen agents que ofereixen un determinat servei i desubscripció, que permet eliminar aquesta subscripció.

L'especificació també ens diu que l'ADS pot interaccionar amb un o varis mecanismes de descobriment, citats en el document *Discovery Middleware* (DM). Si a la plataforma ja existeix un DF, els agents han de fer servir l'ADS únicament per localitzar serveis remots en la xarxa *ad hoc*, els serveis locals es segueixen buscant a través del DF. A més, la FIPA ha desenvolupat una especificació anomenada *JXTA Discovery Middleware Specification*, que fa referència al sistema de descobriment JXTA, proposta desenvolupada en el projecte final de carrera [Ore06].

La nostra proposta amplia amb l'especificació de la FIPA en els següents aspectes:

- Substitució de l'ADS per un agent similar al DF pel que fa a la funcionalitat en la plataforma (registre, desregistre, modificació, cerca, subscripció, desubscripció i també federació).

- Independència amb el mecanisme de descobriment. El nou DF interactuarà

amb tots els DM de la mateixa manera, ja que són ells els que s'encarreguen de la tecnologia de descobriment.

- Interacció tant amb la xarxa fixa com amb la xarxa *ad hoc*, és un element independent al DF actual per tant pot viure a la plataforma sense ell.

- Vol ser un component únic i conegut pels agents, tant de la plataforma local com les remotes. Un agent podrà triar si requerir els serveis del DF tradicional o si per contra vol registrar-se en el nou DF, dependrà de les necessitats de cada agent.

2.8 Que és un *add-on?*

Un *add-on* és un mòdul opcional hardware o software que complementa o realça la unitat original a la qual s'afegeix.

JADE ho descriu com peces addicionals amb característiques que no són necessàries per fer funcionar la plataforma d'agents, però la proveeixen amb noves i interessants peces de funcionalitats.

A JADE els *add-on* tenen un estil uniforme per simplificar-ne el seu ús, compilació i l'accés a la documentació. En concret, un *add-on* és necessari que tingui una estructura de subdirectoris concreta per poder-ne facilitar la seva instal·lació.

Al finalitzar el projecte, aquest s'haurà de poder afegir com un *add-on* a la plataforma d'agents de JADE, juntament amb el projecte [Ore06].

2.9 Resum

En aquest capítol hem introduït els conceptes d'agent mòbil i sistemes multiagent i la proposta de JADE per al seu desenvolupament i execució, aquesta última realitzada sobre una plataforma desenvolupada basant-se en les especificacions de la FIPA. També hem introduït els components principals d'una plataforma d'agents, tenint un tracte especial amb el *Directory Facilitator*, el servei de pàgines grogues de la plataforma.

Hem comentat l'existència de la FIPA, model de referència dels agents. Les seves especificacions s'han convertit en un estàndard reconegut, ja que comprenen aspectes com la gestió dels agents o el flux lògic d'una conversa entre agents. Finalment hem descrit breument els termes *ad hoc* o xarxes dinàmiques i *middleware* o programari intermedi, i com aquests conceptes intervenen en el descobriment de serveis en entorns dinàmics.

En aquest punt podem passar a comentar els detalls de l'extensió del *directory facilitator*. En el capítol següent veurem l'anàlisi que hem realitzat per arribar a la conclusió de la creació d'un nou agent que faci les tasques de DF en entorns *ad hoc*.

Capítol 3

Anàlisi

Ara que ja coneixem la base del problema i els components bàsics de la plataforma de JADE, en concret el *Directory Facilitator*, mòdul en què es basarà el nostre projecte, podem començar amb el desenvolupament. La nostra finalitat serà crear un nou *Directory Facilitator* a la plataforma JADE que ens proporcioni, a través d'una xarxa *Peer to peer* dinàmica (en el nostre cas JXTA), la mateixa funcionalitat que ara ens dóna el DF per a un entorn estàtic.

En aquest capítol definirem l'anàlisi de requisits del nou DF a partir dels documents [AMS04] i [ADSS03] proporcionats per la FIPA, en què ens hem basat per considerar, revisar i estudiar varies possibilitats per trobar la més òptima. A partir d'ara a aquest nou mòdul l'anomenarem ADF (*Ad hoc Directory Facilitator*).

3.1 Anàlisi previ

Com ja hem vist, volem que l'ADF tingui noves característiques amb les premisses que, a nivell d'usabilitat, siguin similars a les que ja existeixen. Primer veurem els requisits funcionals, aquells que ens indiquen quines són les tasques que realitzarà la futura implementació de l'ADF. Finalment veurem els requisits no funcionals, o aquells requisits que provenen de les especificacions i fan que la implementació sigui portable, genèrica i extensible.

3.1.1 Requeriments funcionals

Per qualsevol agent el fet que en una plataforma coexisteixin DF i ADF ha de ser transparent, això vol dir que l'ADF ha de proporcionar la mateixa funcionalitat que el DF proporciona en una xarxa fixa però en una xarxa *ad hoc* o dinàmica.

Per començar farem un incís en les prestacions que ofereix l'actual servei de pàgines grogues (el DF) als agents. L'especificació estàndard [AMS04] ens diu que un DF ha d'oferir els següents serveis:

- Registre
- Desregistre
- Modificació
- Cerca
- Federació (que millora el mecanisme de cerca)

A més el DF pot suportar el següent mecanisme que estén la funcionalitat del directori:

- Subscripció, amb les conseqüents notificacions:
  - Notificació de registre
  - Notificació de desregistre
  - Notificació de modificació
- Desubscripció

Com ja hem esmentat l'ADF haurà de continuar proporcionant aquesta mateixa funcionalitat tant a nivell local com a nivell *ad hoc*. A més haurà d'interactuar amb els DM, el que implicarà també hagi d'incorporar la següent funcionalitat:

- Interfície de comunicació genèrica per comunicar ADF i varis DM de diferent naturalesa (JXTA, Bluetooth, etc).
- Coexistència de varis DM simultanis.

- Arrencada i aturada qualsevol DM en calent, és a dir, sense haver de reiniciar la plataforma.

- *Polling* de subscripcions remotes (aquest tema serà tractat amb més profunditat quan parlem de les subscripcions).

3.1.2 Requeriments no funcionals

Amb les noves tecnologies podem trobar-nos al mercat amb mecanismes amb limitacions de memòria, processament i autonomia. Els principals requisits operacionals de l'ADF haurien de ser:

- Adaptar-se a les plataformes d'agents de JADE perquè puguin executar-se en qualsevol dispositiu.

- Seguir les especificacions i els estàndards de la FIPA dels documents [AMS04] i [ADSS03], en la mida del possible, ja que són la base del descobriment de serveis en xarxes dinàmiques.

- L'impacte de la nova implementació sobre la plataforma existent de JADE ha de ser mínim. Els agents pràcticament no han de notar diferència alguna a l'hora de requerir, buscar o modificar serveis.

- Tot i que un ADF ha de suportar un nombre il·limitat de DM, en un dispositiu mòbil si que es podrà limitar el seu ús, ja que quan major sigui el nombre de DM també serà major el consum d'energia, el nombre de missatges transmesos, etc.

- En xarxes *ad hoc* és necessari que els dispositius continguin plataformes completes perquè puguin actuar de manera autònoma, ja que no sempre tindran accés a una xarxa amb infraestructura i podran accedir a una plataforma d'agents base i fixa.

- El treball definitiu ha de complir les característiques d'un *add-on* de JADE, perquè pugui ser fàcilment incorporable a una plataforma d'agents bàsica.

3.2 Esquema de l'ADF

Un cop tenim clara la idea del que ha de fer i proporcionar l'ADF podem mirar de triar un esquema o representació que més s'escaigui a les necessitats funcionals que hem descrit. El que presentem a continuació és una anàlisi de dues possibilitats que hem considerat.

La primera opció tracta d'extrendre la funcionalitat del DF actual i la segona crear un nou agent similar al DF però que compleixi els requisits i especificacions que hem exposat fins ara. Cadascuna de les opcions comporta una sèrie d'avantatges i inconvenients que descriurem tot seguit.

3.2.1 Extensió del DF actual

La possibilitat d'agafar el DF de la plataforma d'agents de JADE, i afegir-li nova funcionalitat perquè interaccioni també amb xarxes *ad hoc*, sembla un concepte força atractiu.

La idea és agafar el DF, mantenir la funcionalitat que dona fins ara i afergir-li la nova funcionalitat dins seu. L'ADF seria un submòdul del DF, heretaria la seva funcionalitat i incorporaria la nova.

Avantatges

- L'ADF manté la funcionalitat que ja tenia el DF.
- Manté el mateix identificador. Els agents el coneixen pel mateix nom que ja tenia el DF convencional. Per requerir els seus serveis no necessiten adquirir cap coneixement especial.
- La modificació dels serveis estàndards no afecta a l'ADF.

Inconvenients

- Encara que lleu, hauríem de tocar la plataforma JADE, ja que sino l'herència de serveis no podria fer-se de forma íntegra (per ex. tenim mètodes i variables privades del DF que són necessàries per extendre la funcionalitat).

- Com ja sabem, el DF és un element opcional en una plataforma, aleshores l'ADF com *add-on* perd el seu sentit. Si ens descarreguem un accessori per a un component que no existeix, l'accessori no serveix per a res. La dependència que tenen l'ADF i el DF entre ells, llavors, és molt forta.

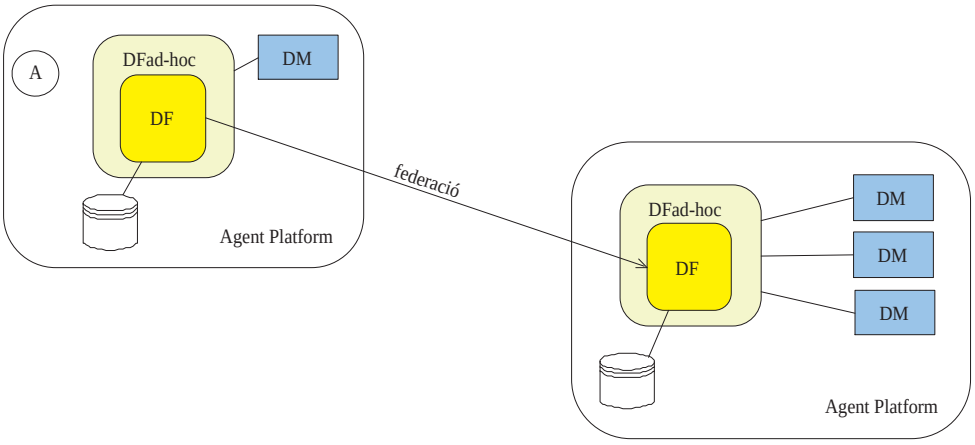

Figura 3.1: Extensió el DF actual afegint-hi funcionalitat

3.2.2 Creació d'un nou DF

La idea de crear un nou *Directory Facilitator* es basa en que xarxes fixes i xarxes *ad hoc* no treballen de forma conjunta, ja que el concepte de xarxa *ad hoc* es basa en aquesta premissa.

Recolzant-se en aquesta idea i pensant en el mínim impacte sobre la plataforma d'agents de JADE actual, el que es pretén és crear un nou agent DF que s'encarregui de la gestió del servei de pàgines grogues en entorns ubics. Estem davant dels següents casos:

1. *Existeix el DF a la plataforma*: En aquest cas el DF i l'ADF tenen la possibilitat de federar-se entre ells. El DF s'encarregarà del servei de pàgines grogues en l'entorn estàtic (cerques locals i federacions) i l'ADF serà el responsable del mateix servei en l'entorn dinàmic.

2. *No existeix el DF a la plataforma*: Aleshores la plataforma d'agents estarà provista únicament amb l'ADF que proporcionarà el servei de pàgines grogues en entorns *ad hoc* i a nivell local mantindrà la funcionalitat del DF.

Avantatges

- El DF i l'ADF no tenen cap tipus de dependència l'un de l'altre, poden coexistir en la mateixa plataforma d'agents i poden viure sols.
- La federació entre DF i ADF pot servir d'enllaç entre una xarxa convencional i una xarxa *ad hoc*, posa en contacte els dos entorns.
- Amb aquesta implementació l'impacte sobre la plataforma és pràcticament nul.

Inconvenients

- Els agents *a priori* no coneixen el nou agent DF. Necessiten informació extra per poder requerir els seus serveis.

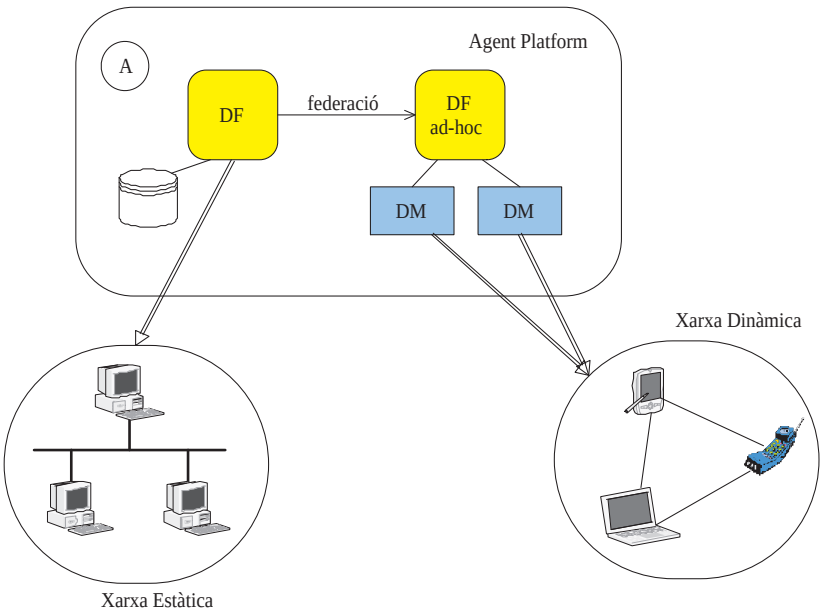

Figura 3.2: Creació d'un nou DF que faci la feina *ad hoc*

3.2.3 Conclusió

Veient els avantatges i inconvenients de cadascuna de les idees proposades arribem a la conclusió que el mes pràctic és afegir un nou mòdul independent que s'encarregui de la gestió de la xarxa dinàmica, reduint també d'aquesta manera l'impacte sobre la plataforma JADE. Alhora també evitem una dependència total de l'ADF sobre el DF, ja que actualment és un element opcional de la plataforma d'agents.

Llavors ja tenim definit l'ADF: serà un nou agent que proporcionarà el servei de pàgines grogues en un entorn dinàmic, i podrà interactuar tant amb el DF de la plataforma com amb els DM.

3.3 Intercanvi de missatges

La majoria dels missatges estan relacionats amb el mecanisme de subscripció, un dels paradigmes més importants a la plataforma d'agents de JADE. Alguns DF poden implementar el protocol d'interacció `fipa- subscribe` [SIPS02] que permet als agents subscriure's per ser notificats sobre registres, desregistres i modificacions sobre certes descripcions d'agents.

En la nostra implementació volem que l'ADF sigui capaç de implementar aquest mecanisme, i la base del servei de subscripcions és l'intercanvi de missatges. Per conèixer en profunditat el funcionament de l'intercanvi de missatges entre l'ADF i els DM primer descriurem l'estudi que vam realitzar per arribar a una solució el més òptima possible. Seguidament desglossarem les tres opcions més importants que hem estudiat.

3.3.1 Primera opció: Broadcast de serveis

La nostra primera alternativa va ser pensar en el procés més senzill: propagar qualsevol event a través de la xarxa JXTA. Això simplifica, i molt, el servei de subscripcions, l'ADF no ha de fer res nou, únicament s'haurà d'encarregar de servir les notificacions a nivell local.

Per veure aquesta proposta una mica més en detall desglossarem cadascun dels possibles events:

- **Registre**: Sempre que un agent registri un servei a l'ADF, s'afegirà una entrada en el registre de serveis locals i seguidament es propagarà cap a tots els DM actius en aquell moment. Un cop arriba aquest avís a un DM remot, aquest notifica al seu ADF i, en aquest ADF remot, també s'inclou una entrada, però aquest cop en el registre de serveis remots.

  Per tant en cada ADF mantenim un registre per als serveis locals i un registre per als serveis remots.

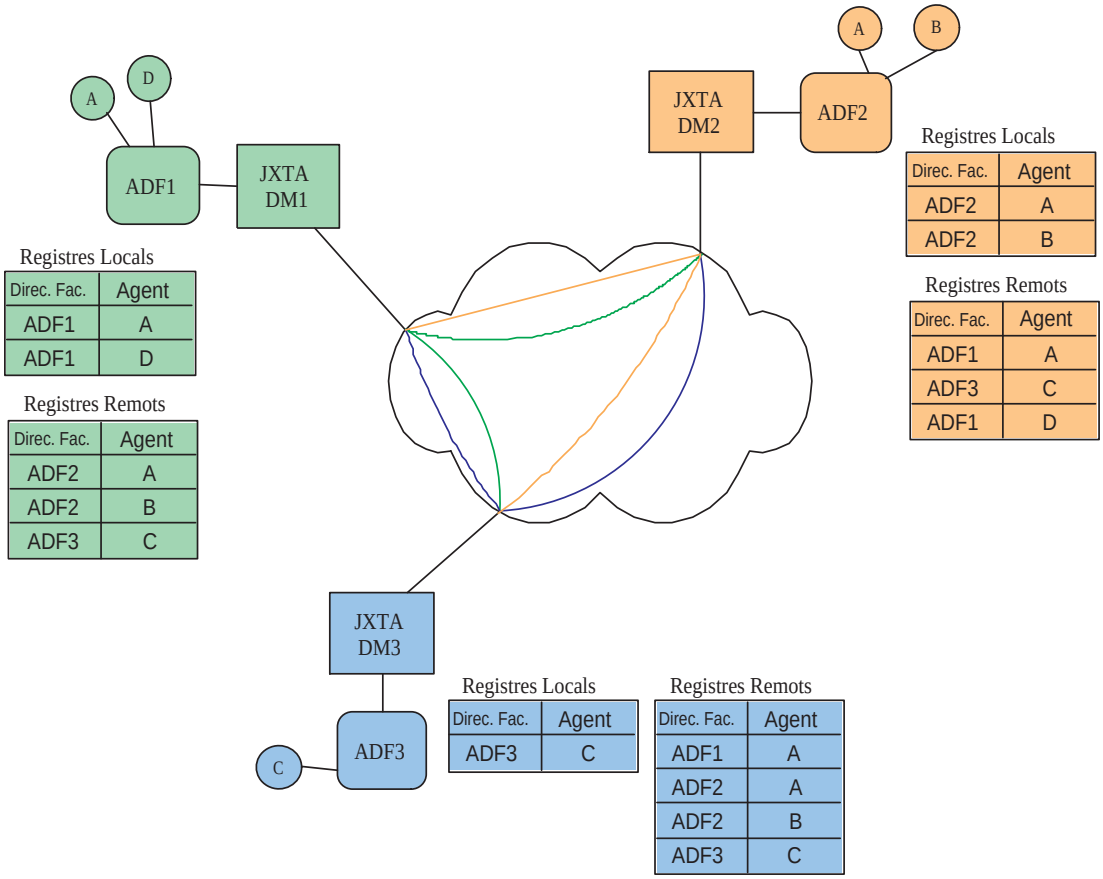

Figura 3.3: Taules de registres

- **Desregistre i Modificació**: Aquests dos events funcionen de manera similar al registre, es propaguen per la xarxa dinàmica en quant es produeixen. En arribar un avís a un ADF remot que tingui el servei registrat, modificaran o esborraran l'entrada en el registre de serveis remots segons convingui.

- **Cerca**: A diferència del que es pugui pensar, no hem estudiat la cerca en aquesta implementació com una cerca local encara que tinguem tots els serveis, locals i remots, en un mateix ADF. En una xarxa *ad-hoc* és molt fàcil perdre missatges (per la rapidesa amb què apareixen i desapareixen els nodes) per tant és factible la pèrdua de missatges importants al llarg de la transmissió.

  Llavors quan un agent faci una petició de cerca, aquesta serà propagada cap als DM i en arribar a una plataforma remota l'ADF remot cercarà en el seu registre de serveis i respondrà a l'origen amb els resultats trobats.

- **Subscripció**: Quan un agent es subscriu a l'ADF per ser notificat sobre algun servei en concret, en aquesta implementació no es propagarà cap avís de subscripció cap als DM. Com bé hem explicat abans, tots els events són propagats per la xarxa, per tant el servei de subscripció serà local.

  El mecanisme que executa l'ADF en quan es subscriu un agent és el següent:

  1. Un agent es subscriu a l'ADF per ser notificat sobre un tipus de servei en concret.
  2. L'ADF cerca al registre de serveis locals i remots qualsevol servei que coincideixi amb el patró de subscripció proporcionat.
  3. Si hi ha coincidència, l'ADF avisa a l'agent subscriptor dels serveis proporcionats i on es troben.
  4. Si no hi ha coincidència, l'ADF espera de forma activa a rebre qualsevol notificació provinent tant de la xarxa dinàmica (a través dels DM) com del registre local per avisar al subscriptor.
  5. Quan l'agent subscriptor vol deixar de rebre notificacions envia a l'ADF un missatge de desubscripció.

3.3.1.1 Anàlisi

La taula següent resumeix la nostra primera opció:

Acció	**Enviament**	**Recepció**
Registre	Broadcast	1. Avisar als subscrits 2. Guardar registre remot
Desregistre	Broadcast	1. Avisar als subscrits 2. Eliminar registre remot
Modificació	Broadcast	1. Avisar als subscrits 2. Modificar registre remot
Cerca	Broadcast	Resultats de cada DM
Subscripció	No s'envia avís	No es rep avís

Taula 3.1: Missatges intercanviats en Broadcast de serveis.

Com a avantatges d'aquesta alternativa trobem la senzillesa del servei de subscripcions, totalment local a la plataforma.

Per altra banda, el principal problema d'aquesta proposta són els registres de serveis remots. Les estructures de dades que continguin els registres remots a l'ADF creixeran d'una manera exponencial, ja que contindran els registres dels serveis de tota la xarxa *ad-hoc*, fet que fa que aquesta proposta no sigui escalable. En la part dels contres també trobem que a la xarxa dinàmica trobarem molta informació redundant, cosa que farà que les cerques puguin ser llargues i tedioses.

I per acabar, veiem que aquest sistema tot i fer les subscripcions molt senzilles les deixa inutilitzades, ja que tant si existeixen o no agents subscrits en plataformes remotes, propagarem tots els events (registres, desregistres i modificacions) inundant la xarxa amb tràfic inservible.

3.3.2 Segona opció: Propagació de Subscripcions

Aquesta proposta difereix de l'anterior en que no es propagarà cap servei a través de la xarxa dinàmica excepte les subscripcions, i d'una manera especial. Per tal de reduir el tràfic de la xarxa vam pensar en evitar la propagació dels events que

es podien donar amb una probabilitat mes alta: registres, desregistres i modificacions.

- **Registre**: Quan un agent registra un servei a l'ADF local aquest, com faria un DF comú, afegeix l'agent i el servei a la taula de serveis locals i no fa res més. Aquesta entrada servirà posteriorment per peticions de cerca i subscripció.

- **Desregistre i Modificació**: Igual que els registres, els desregistres i les modificacions només tindran efecte a nivell local. Tampoc sortiran a la xarxa subjacent.

- **Cerca**: La cerca seguirà el mateix mecanisme que l'anterior, un protocol de petició-resposta asíncron, és a dir, que quan l'ADF envia una cerca continua la seva feina sense bloquejar-se.

  La cerca està especificada en un document de la FIPA [ADSS03] el qual hem seguit per implementar-la.

- **Subscripció**: Com acabem de veure, el principal problema radica en que cap dels events susceptibles a la subscripció d'un agent a un determinat servei són propagats per la xarxa *ad hoc*. Com que aquesta informació no es rep a cap ADF remot, serà cadascun d'ells qui l'hagi d'anar a cercar.

  Per trobar la informació actual de la xarxa podem aprofitar el mecanisme de cerca, ja que és totalment compatible amb la subscripció. Ara bé, per a la notificació de nous registres, desregistres o modificacions en instants de temps posteriors a la subscripció, hem analitzat dues alternatives:

  - *Polling* de serveis: Cada cop que un agent faci una nova subscripció a l'ADF aquest la registrarà com a subscripció local, iniciarà una cerca extensiva per la xarxa *ad hoc* i avisarà de tots els registres que trobi.

    Ara bé, per notificar de desregistres i modificacions, cada cert temps iniciarà una nova cerca per veure si s'han produït canvis en la configuració de la xarxa.

– Propagació o *Pulling* de subscripcions: Quan un agent es subscriu a l'ADF, aquesta subscripció es propagada a través de la xarxa cap a tots els ADF remots. El procés que te lloc es descriu a continuació:

1. Un agent es subscriu a l'ADF per ser notificat sobre un tipus de servei en concret.
2. L'ADF guarda la subscripció local i propaga aquesta subscripció als ADF remots a través del seus DM.
3. Un ADF remot rep un avís de subscripció (a través d'algun dels seus DM) i registra la subscripció com a remota.
4. Quan en algun ADF connectat a la xarxa JXTA (en el nostre cas) es produeixi qualsevol event susceptible a una subscripció, l'ADF comprovarà totes les seves subscripcions tant locals com remotes, i notificarà als agents i plataformes corresponents.

3.3.2.1 Anàlisi

Com hem fet amb la primera opció, presentem una taula resum de la proposta:

Acció	**Enviament**	**Recepció**
Registre	No s'envia avís	No es rep avís
Desregistre	No s'envia avís	No es rep avís
Modificació	No s'envia avís	No es rep avís
Cerca	Broadcast	Resultats de cada DM
Subscripció	Polling o Pulling de serveis remots cada cert temps	Resultats de cada DM

Taula 3.2: Missatges intercanviats en Propagació de Subscripcions.

Com ja hem vist el pes de l'intercanvi de missatges en aquesta proposta es troba en les subscripcions. Cadascuna de les alternatives que hem proposat (el pulling i el polling) tenen els seus beneficis i els seus inconvenients.

El **polling** és l'opció mes desfavorable ja que té el gran desavantatge que inunda periòdicament la xarxa amb missatges de cerca per actualitzar cadascun dels registres de subscripcions dels ADF de tota la xarxa. A mes, a nivell d'ADF

individual, s'hauria de conservar un *estat anterior* per veure si realment hi han modificacions, i en el cas dels desregistres també es necessitaria una bona heurística per investigar si, quan no es troba un servei, es per un desregistre o el node s'ha caigut (`unreachable`).

L'alternativa del **pulling** presenta una mínima millora a l'hora d'inundar la xarxa cada cert temps, però també ho fa cada cop que hi ha una nova subscripció. També presenta el desavantatge del creixement exponencial de taules de subscripcions remotes, així com informació redundant per tota la xarxa. L'única millora que trobem, és que a l'hora d'avisar d'un event susceptible a una subscripció es fa de manera unicast.

Si tenim en compte que volem que aquesta tecnologia sigui aplicable a dispositius mòbils, quants mes missatges enviem, mes desgast d'energia tindrem, per tant no ens es factible.

3.3.3 Tercera opció: La conclusió

Com que cap de les altres alternatives ens va semblar eficient ni escalable vam optar per agafar el millor d'ambdues i intentar optimitzar alguna de les qualitats que no ens van acabar de convèncer. En intentar reduir el nombre de missatges vam optar per propagar en broadcast únicament els avisos de registre, de subscripció i cerca, i la resta de missatges enviar-los en mode unicast.

Tota aquesta proposta gira entorn a les subscripcions, així que cadascun dels events està pensat perquè els subscriptors siguin notificats de la manera més eficient possible sense col·lapsar la xarxa.

Continuació passem a descriure la proposta, simplificant l'explicació al fet de rebre o enviar notificacions d'events. Començarem amb les subscripcions perquè la resta d'accions no quedin confuses:

- **Subscripció**

    1. Enviar: Quan un agent fa una subscripció a un determinat servei en l'ADF local, aquest desa la subscripció com a local i propaga la subscripció en broadcast per la xarxa *ad hoc* mitjançant els DM actius.

2. Rebre: Quan un ADF remot rep un avís de subscripció busca entre els seus serveis locals. Si entre aquest es troba el servei sol·licitat es desa la subscripció com a remota (per a futures notificacions) i informa a la plataforma on es troba el subscriptor. Si per contra, l'ADF remot no conte cap servei com el que es requereix la subscripció, s'ignora.

- **Registre**

  1. Enviar: Quan es registra un nou servei a l'ADF local, aquest desa el registre i el propaga en broadcast per la xarxa.
  2. Rebre: En rebre un avís de registre un ADF remot avisarà a tots els seus subscriptors locals, ja que el registre arriba a tots els nodes de la xarxa *ad hoc*. Si un ADF remot, en rebre un avís de registre, té alguna subscripció per a aquell mateix servei, avisa a l'ADF emissor perquè guardi la seva subscripció com a remota.

- **Desregistre i Modificació**

  1. Enviar: Quan un agent desregistra o modifica un servei prèviament registrat a l'ADF, comprova si a les subscripcions remotes hi ha algun servei que coincideixi amb el que s'ha modificat o desregistrat, si és així se li envia un avís unicast cap a la plataforma a la qual pertany l'agent amb la subscripció remota.
  2. Rebre: Si rebem una notificació remota de desregistre o modificació, vol dir que tenim una subscripció remota en una altra plataforma, per tant, l'únic que hem de fer és avisar als subscriptors del servei.

- **Cerca**: La cerca roman igual que la primera alternativa; i continua sent un protocol de petició-resposta entre els ADF.

3.4 Objectius

Un cop estudiades les alternatives del servei de pàgines grogues pel descobriment de serveis en xarxes *ad hoc*, els objectius que ens plantegem per proporcionar una

solució al problema són els següents:

- Mantenir el DF com a servei de pàgines grogues a la xarxa estàtica, ja que la funcionalitat que proporciona és imprescindible tant en entorns fixes com en entorns *ad hoc*.
- Afegir l'ADF com a servei de pàgines grogues en un entorn ubic adaptant-lo a les característiques de l'entorn on volem aplicar.
- Suprimir la federació en entorns *ad hoc*. Interconnectar plataformes en entorns que poden canviar d'estat ràpidament no té massa sentit, per tant la federació entre els ADF no seria viable. Ara bé, la federació entre un DF tradicional i un ADF ha de ser possible, sobretot a la mateixa plataforma.

3.5 Resum

En aquest capítol hem donat forma a l'*Ad hoc Directory Facilitator*. Hem dut a terme, el que anomenaríem en Enginyeria del Software, la fase de captació de requeriments.

Hem realitzat un estudi sobre l'adaptació dels estàndars de la FIPA en entors *ad hoc*, centrant-nos en un servei de pàgines grogues alternatiu per al descobriment de serveis oferts per agents remots en aquest tipus d'entorns.

Capítol 4

Disseny: Ad hoc Directory Facilitator

Un cop finalitzada la fase d'anàlisi i ara que ja coneixem els requeriments funcionals i operacionals de l'ADF podem passar a veure el procés de disseny. En aquest capítol modelarem les especificacions que hem repassat anteriorment a través de diagrames UML (*Unified Modeling Language*) que ens ajudaran a visualitzar la funcionalitat real de cada element, i també volem donar cabuda al disseny de la interfície entre ADF i DM, element indispensable de comunicació entre aquests elements.

Per finalitzar veurem quines característiques de disseny ha de complir l'ADF (juntament amb el DM de jxta [Ore06], en el nostre cas) per convertir-se en un *add-on* segons les especificacions que trobem al web de JADE.

4.1 Diagràmes UML

Els actors principals de la nostra aplicació són l'ADF i el DM, els quals hem separat els respectius diagrames de casos d'ús per diferenciar quines accions corresponen a cadascun dels actors sempre relacionades amb el desenvolupament del nou *Directory Facilitator*.

4.1.1 Casos d'ús de l'ADF

En el diagrama corresponent a l'ADF (figura 4.1) podem veure la interacció que té amb els DM, els agents i el DF. Cadascun d'aquests elements corresponen a una part ben diferenciada de l'ADF, la que llinda amb la part estàtica i la que ho fa amb la part dinàmica. A continuació descriurem en profunditat cadascun dels casos d'ús i les seves característiques:

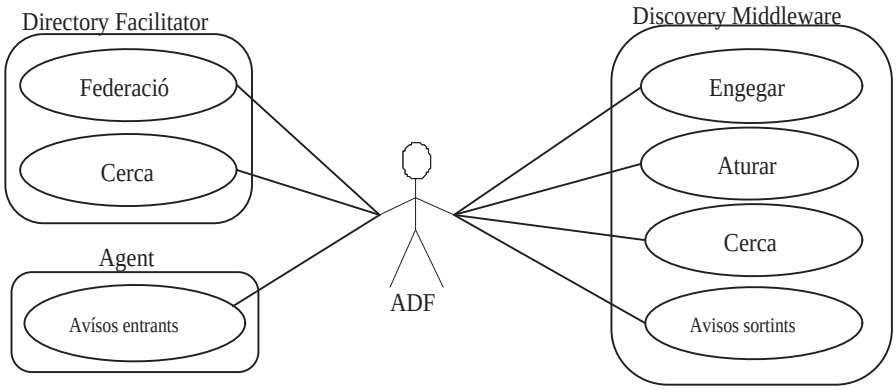

Figura 4.1: Diagrama de casos d'ús de l'ADF com actor.

4.1.1.1 Federació

L'ADF podrà federar-se amb el DF de la plataforma afegint-lo com a pare en el seu arbre de federacions (en la cerca veurem perquè no al revés). De manera opcional, un ADF pot federar-se amb el DF de la plataforma en el moment en què arrenca; només s'haurà d'especificar en l'arxiu de configuració que llegeix en inicialitzar-se. Aquesta federació és l'única que té permesa, ja que hem decidit que la federació en l'entorn *ad hoc* estigui prohibida.

4.1.1.2 Cerca

Com hem comentat en el capítol 2 (secció 2.3) la federació reforça el mecanisme de cerca en els DF. Aquest cas d'ús es refereix a la cerca en el moment en què l'ADF estigui federat amb el DF de la mateixa plataforma i hagi rebut una

cerca remota. La cerca entre DF federats està fixada per les especificacions, per tant segueix el que està descrit en el document [AMS04], on les dues premisses principals son:

- No canviar el valor del paràmetre `search-id` quan es propagui la cerca, ja que aquest valor és únic globalment (evita cicles de cerca).
- Abans de propagar la cerca hem de decrementar el paràmetre `max-depth` en 1.

El resultat de la cerca no es queda a la plataforma local, s'envia de retorn cap al DM remot que ha iniciat la petició.

4.1.1.3 Avisos entrants

Aquest cas d'ús fa referència a les notificacions provinents de la xarxa *ad hoc* que són enviades per plataformes remotes en dispositius mòbils. Segons el tipus d'avís l'ADF actuarà en conseqüència envers els seus agents locals:

- Avís de Registre: En rebre un avís de registre se'ns proporciona un `df-agent-description` (que a partir d'ara anomenarem DFAD) que conté la informació de l'agent remot registrat. L'ADF notifica a tots el seus subscriptors locals comparant les descripcions dels serveis. Si té agents locals subscrits al nou servei remot registrat envia un avís de subscripció **unicast** a l'ADF remot per formalitzar la subscripció remota en la plataforma on s'ha registrat l'agent de l'avís.
- Petició de Cerca: Amb una petició de cerca rebem els següents paràmetres: un patró de cerca (en forma de DFAD), nombre màxim de resultats, profunditat màxima de la cerca i un identificador únic de la cerca. Seguidament agafem aquests paràmetres i fem, en aquest ordre, cerca local, cerca en DF federats i cerca *ad hoc* mantenint el identificador de cerca i decrementem en 1 el valor de profunditat màxima cada vegada. Si en algún moment del procés s'assoleix el nombre màxim de resultats, s'atura la propagació de la cerca i es retornen les dades obtingudes.

- Petició de Subscripció: La petició de subscripció engloba més d'un tipus d'avís, exactament cinc: subscripció, desubscripció, modificació, desregistre i polling de subscripcions. En rebre una petició d'aquest tipus ens proporcionen les següents dades: dos DFAD, una cadena que ens indica el tipus d'avís, un identificador de DM remot i el mateix DM (en forma d'objecte). És un procés complexe, així que intentarem explicar-ho amb tots els detalls possibles.

  Quan rebem un avís d'aquest tipus pot ser per dos motius:

  1. Petició de subscripció real.
  2. Es produeix un event remot susceptible a la subscripció i tenim una subscripció remota en aquell punt.

En la figura següent podem veure un esquema general del diagrama d'activitats que presenta aquest mètode. Anirem desglossant cadascun dels diferents tipus de missatges en què pot derivar aquesta petició:

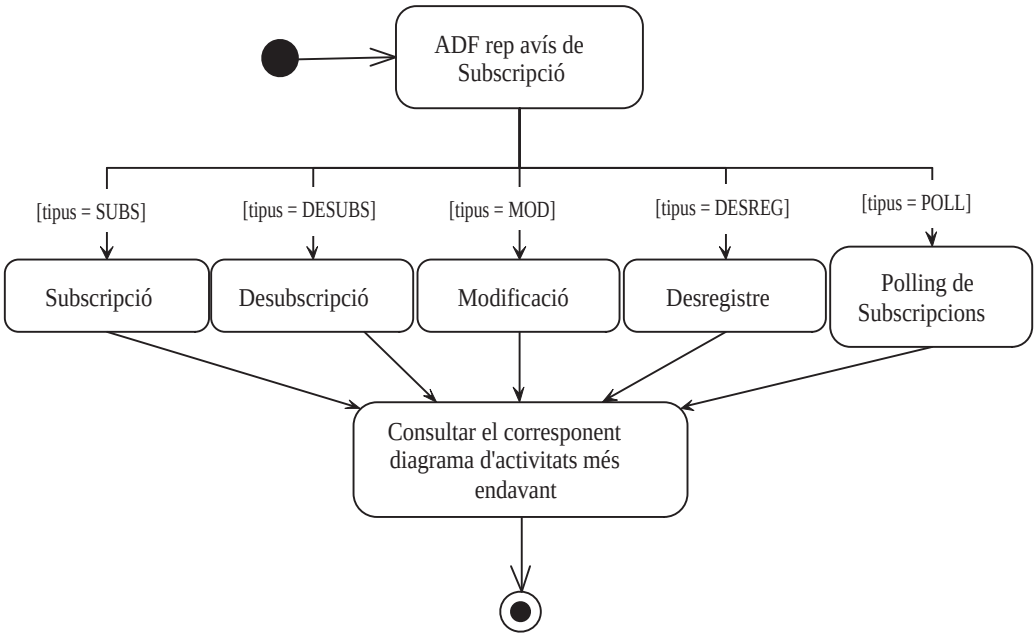

Figura 4.2: Diagrama d'activitats d'avís de subscripció genèric.

Començarem amb el diagrama d'activitats de la **petició de subscripció remota**. En cas de que a la plataforma local hi hagin agents que proporcionin el servei desitjat, afegirem la subscripció remota en una taula, desant totes les dades necessàries per posteriorment notificar dels events que es produeixin.

Si a la plataforma, en el moment que arriba la subscripció, no hi ha cap agent com el que demana el subscriptor s'ignora la petició.

Finalment per cada agent que proporcioni el servei desitjat que trobem enviarem una notificació al subscriptor remot.

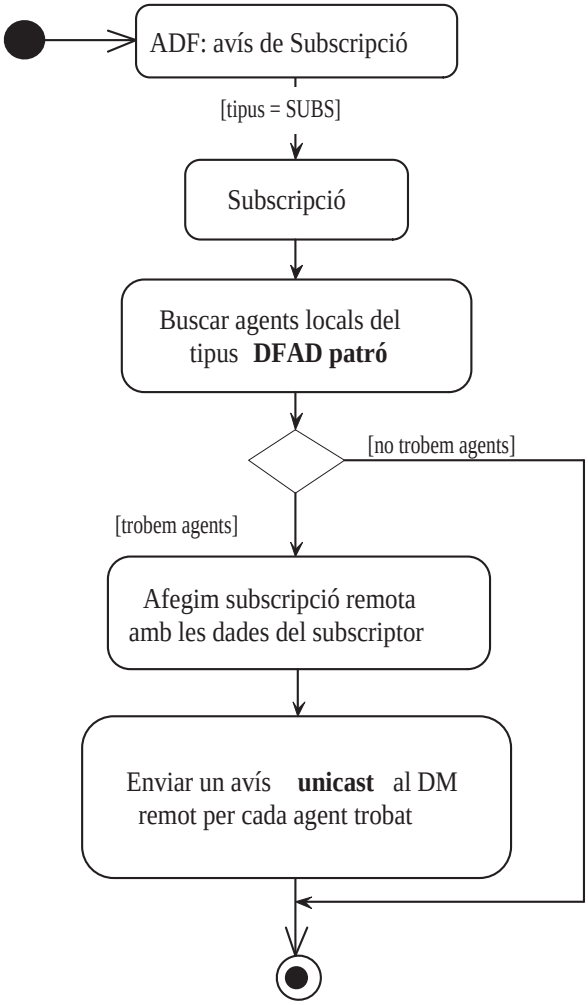

Figura 4.3: Diagrama d'activitats d'una subscripció remota.

En rebre un **avís de** *polling* haurem de decidir si renovem la subscripció remota o per contra ja no volem seguir sent notificats. Per prendre aquesta decisió tindrem en compte dos events:

1. La subscripció local està activa (es pot haver perdut l'avís de desubscripció).
2. L'agent encara és viu.

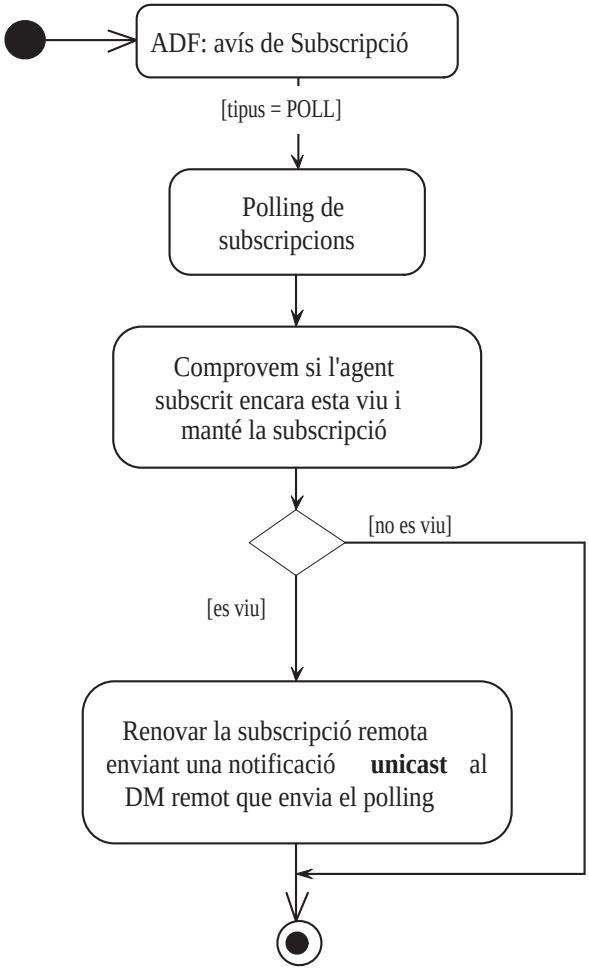

Figura 4.4: Diagrama d'activitats d'un avís de polling per a subscripcions remotes.

Si es compleixen aquestes dues condicions alhora llavors renovem la subscripció remota en la plataforma on es troben l'ADF i el DM que ens han enviat l'avís de polling.

Els **avísos de modificació i desubscripció remots** es tracten de manera similar. Si ens arriba un avís d'aquest tipus és que tenim subscripcions remotes a aquests serveis, per tant l'únic que hem de fer és avisar als agents locals corresponents dels canvis en la configuració de la xarxa.

I per acabar amb les notificacions que rebem des de la xarxa *ad hoc* veurem el diagrama d'activitats d'un **avís de desubscripció remota**. En rebre una desubscripció cercarem a la taula de subscripcions remotes i esborrem l'entrada corresponent al DFAD de l'agent remot que es vol desubscriure.

Per altra banda, si la subscripció remota ja no existia a la taula, ignorem l'avís de desubscripció.

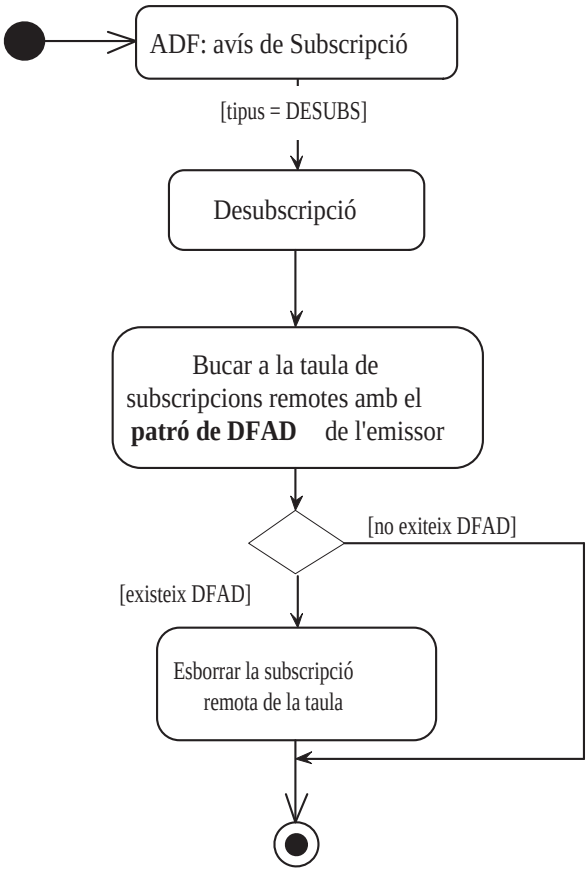

Figura 4.5: Diagrama d'activitats d'una desubscripció remota.

Fins aquí hem vist els casos d'ús que es refereixen als avisos entrants a l'ADF, és a dir, els que provenen de la xarxa JXTA, en el nostre cas, cap als ADF. També tenim una visió de la interacció de l'ADF amb la xarxa estàtica.

En les següents seccions tractarem els casos d'ús que fan referència al tractament de missatges entre la xarxa dinàmica i l'ADF.

4.1.1.4 Avisos sortints

Com acabem de comentar, aquest cas d'ús fa referència a les notificacions propagades cap a la xarxa *ad hoc*. Com en el cas dels avisos entrants, aquest engloba un conjunt d'avisos que comentarem tot seguit, els hem agrupat segons la mecànica d'enviament de l'avís:

- Propagar registre, subscripció o desubscripció: Quan un agent realitza qualsevol d'aquestes tres accions es crea un missatge amb el DFAD corresponent (en el cas del registre, el de l'agent registrat i en els altres dos casos, el DFAD que fa referència al patró de (de)subscripció) i es propaga a tots i cadascun dels DM actius en la plataforma local. Fem broadcast d'aquests events.

- Propagar modificació o desregistre: Per poder propagar un event de modificació o desregistre hem de tenir en l'ADF local alguna subscripció remota al servei en qüestió. Aleshores es recorre la taula de subscripcions remotes i es notifica únicament als agents remots que estiguin subscrits al servei modificat i/o desregistrat.

- Propagar *polling* de subscripcions: Per fer un polling necessitarem un temporitzador que salti cada cop que passi un cert temps. L'espera no haurà de ser massa curta ja que sinó inundarem la xarxa cada poc temps amb missatges de *polling*.

  Anirem recorrent la taula de subscripcions remotes de l'ADF. Per cada entrada de la taula enviarem un missatge unicast a l'ADF on es troba l'agent remot subscrit i seguidament esborrarem aquesta entrada de la taula. La

taula es tornarà a omplir a mida que els ADF remots vagin renovant (o no) les subscripcions remotes dels seus agents.

4.1.1.5 Cerca

La cerca és un dels processos més importants que realitza l'ADF, en aquest cas d'ús veurem la cerca de serveis a través de la xarxa *ad hoc*.

El procés global de cerca el desenvoluparem en detall en el diagrama de seqüència en la secció següent, però bàsicament consta aquests tres passos:

1. Cerca local
2. Cerca federada (opcional)
3. Cerca *ad hoc*

Quan arribem a la cerca cap a la xarxa JXTA hem de proporcionar als DM les següents dades:

- `df-agent-descriptor` (DFAD): un patró de cerca
- Nombre màxim de resultats: aquest valor es va actualitzant en temps real a mida que anem rebent resultats. És un disparador per aturar la cerca. Pot venir fixat per defecte o donat per l'agent que inicia la cerca.
- Temps d'espera màxim: a la xarxa JXTA els DM poden estar esperant infinitament fins que troben algun resultat. Amb aquest paràmetre limitarem el temps d'espera. Aquest paràmetre el fixa l'ADF.
- Profunditat màxima: paràmetre que ens servirà per saber fins on propagar la cerca federada en cada plataforma remota. Es decrementa en 1 cada cop que passa per un node.
- Identificador de cerca: amb aquest identificador evitarem bucles en la cerca, ja que és únic. Aquest ID es crea automàticament a l'ADF cada cop que s'inicia una nova cerca.

La cerca *ad hoc* s'iniciarà sempre que el nombre de resultats trobats en la cerca local i federada no superi el nombre màxim de resultats. Aleshores propagarem la cerca a través dels DM actius i a mida que ens vagin retornant els resultats els afegirem al resultat global. Com en els casos anteriors aturarem la cerca en assolir el nombre de resultats desitjats i proporcionarem les dades a l'agent que ha sol·licitat la cerca.

4.1.1.6 Arrencar DM

Quan un ADF inicia la seva execució posa en marxa una sèrie de mecanismes entre ells arrencar els DM que tingui especificats en un arxiu de configuració. En el moment en què arrenca el primer DM es posa en marxa el mecanisme, en el nostre escenari, que crearà un node per la xarxa JXTA i ens unirà a la mateixa (veure [Ore06] per conèixer el procés en detall). A partir d'aquest moment el dispositiu on es troba la plataforma de JADE (i en ella l'ADF i els DM) està disponible per comunicar-se i interactuar amb la xarxa JXTA.

A part de la configuració per defecte, l'ADF pot arrencar en qualsevol moment un DM específic a través d'una interfície senzilla que implementarem. Haurem de proporcionar la classe del DM i el seu nom.

4.1.1.7 Aturar DM

Com en el cas d'arrencar DM, la interfície que implementarem també ens permetrà aturar els DM en qualsevol moment de l'execució de la plataforma sense que li afecti (en calent).

En el moment en què s'aturi l'últim DM d'una plataforma es produirà l'efecte invers a quan s'arrenca el primer DM: abandonarem la xarxa JXTA.

4.1.2 Casos d'ús del DM

Els casos d'ús del DM han quedat pràcticament definits perquè l'ADF és qui interactua en ambdós costats de la xarxa amb els DM. Però perquè quedi una mica més clar veurem per sobre els casos d'ús del DM que afecten al nostre projecte:

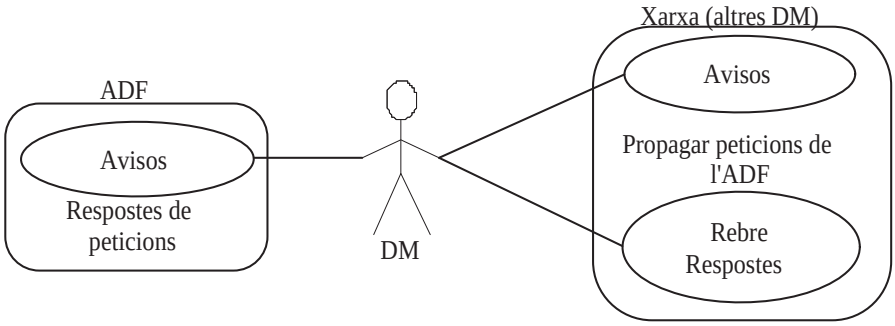

Figura 4.6: Diagrama de casos d'ús del DM com actor.

En la figura podem veure que la vista d'un DM des del punt de vista d'un ADF queda bastant simplificada. El DM propaga els avisos sortints cap a la xarxa *ad hoc* i espera rebre les respostes en casos de subscripcions, cerques o alguns registres. Quan rep un event de la xarxa dinàmica avisa a l'ADF a través del cas dt'ús d'avisos entrants tal i com ha quedat explicat en l'apartat anterior.

4.2 Diagrames de seqüència

Per veure amb una mica més de detall i aclarir alguns aspectes que havien quedat posposats dels casos d'ús més rellevants, com són cerca, subscripció i *polling*, a continuació mostrem els respectius diagrames de seqüència. D'aquesta manera podem fer-nos una idea dels objectes i classes que utilitzarem per implementar el nostre escenari.

4.2.1 Cerca

Ha quedat clar que la cerca té tres processos diferenciats. Passarem a descriure cadascun d'ells aturant-nos en el que ens interessa, la cerca *ad hoc*.

Una cerca s'inicia en el moment en què un agent demana a l'ADF, o el mateix ADF (ja que en si és un agent), que inicií un procés de cerca:

- **Cerca local**: busca entre els agents locals registrats en l'ADF.

- **Cerca recursiva**: aquesta cerca s'inicia si el nombre de resultats locals no és suficient i l'ADF està federat amb el DF. Llavors es buscarà a través de la xarxa estàtica (entre els DF federats).

  En aquest punt podem comentar que l'ADF es federa amb el DF i no al revés perquè la cerca federada es propaga de fills a pares segons ho especifica el document [AMS04].

- **Cerca** *ad hoc*: si entre els dos processos anteriors no han recopilat resultats suficients (aquest valor ve indicat pel paràmetre màxim nombre de resutlats) aleshores l'ADF propagarà la cerca cap a la xarxa JXTA enviant un avís cap al DM de JXTA.

  Un únic DM pot rebre respostes d'un nombre il·limitat de DM (tots els nodes que conformin la xarxa JXTA), però el DM local no respondrà al seu ADF amb el resultat fins que el procés de cerca hagi finalitzat.

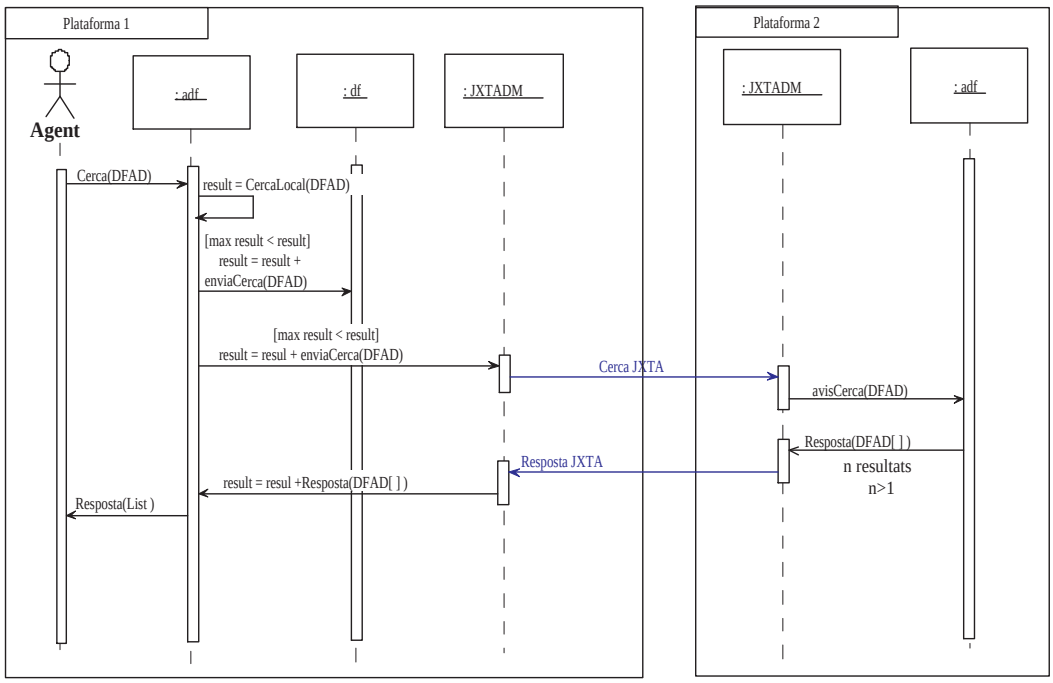

Figura 4.7: Diagrama de seqüència d'un procés de cerca.

4.2.2 Subscripció

La subscripció en entorns ubics no estava contemplada en cap document ni especificació, hem seguit un procés evolutiu per definir-la i hem obtingut el resultat següent:

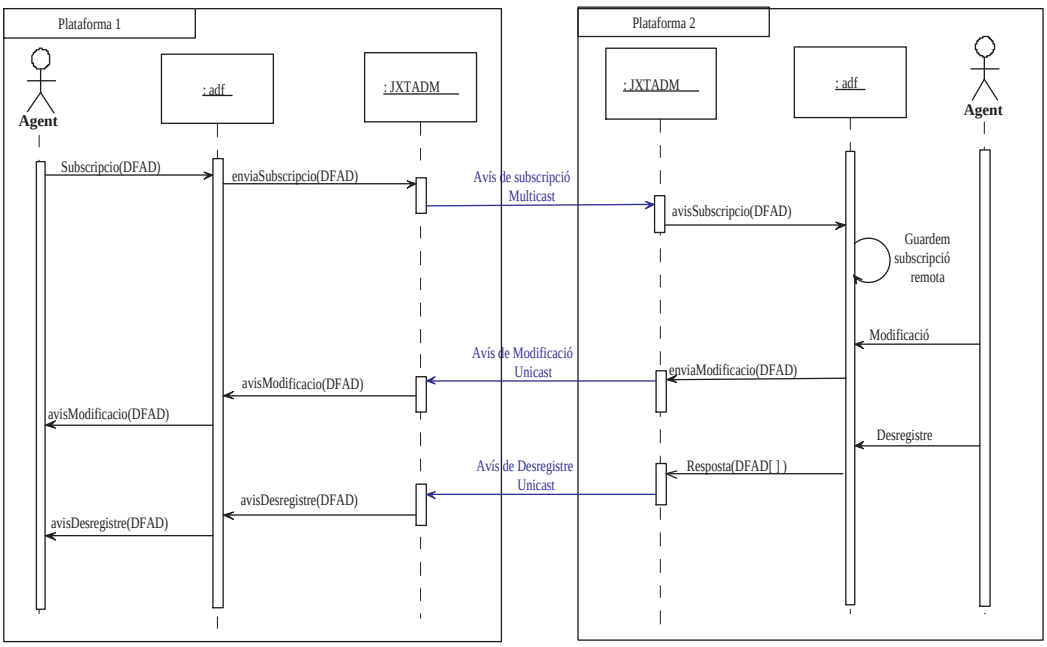

Figura 4.8: Diagrama de seqüència d'un procés de subscripció.

Ja hem explicat el procediment de subscripció en altres seccions, però aquí veiem com es propaguen els avisos en multicast per difondre la subscripció d'un agent i després en unicast per avisar de canvis en un servei.

4.2.3 *Polling*

El polling de subscripcions ha sorgit per la pròpia definició de subscripcions en la xarxa *ad hoc* que hem definit. Com que no ens es possible assegurar la recepció de tots els missatges en l'entorn ubic, periòdicament revisarem la taula de subscripcions remotes. D'aquesta manera si un dispositiu es cau o ha abandonat la xarxa de manera abrupta no haurem de mantenir la possible subscripció remota ni enviar els corresponents missatges.

En el procés de polling de subscripcions remotes tots els missatges s'envien de forma unicast, ja que quan guardem la subscripció remota també desem les dades especifiques de cada DM, ADF i agent remots. D'aquesta manera evitem inundar la xarxa amb cada procés de polling.

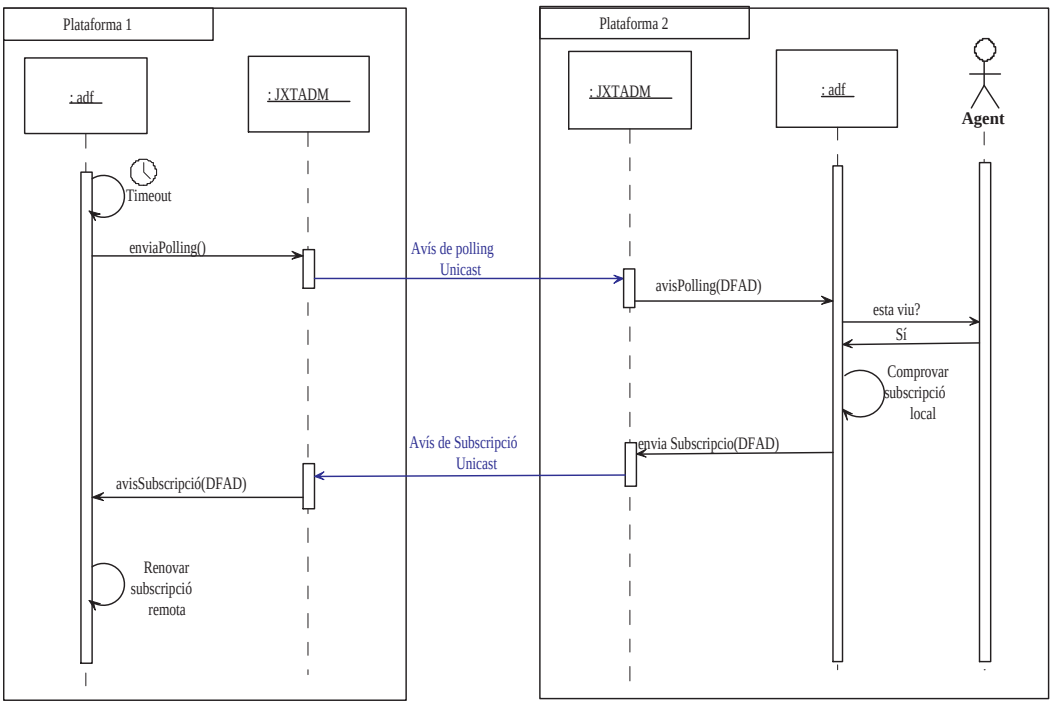

Figura 4.9: Diagrama de seqüència d'un procés de *polling*.

4.3 Interfície entre els DM i ADF

Per comunicar JADE amb JXTA és obvi que no podem traspassar els missatges ACL directament a través de la xarxa JXTA; necessitarem una interfície que comuniqui ambdues parts.

La interfície de comunicació entre ADF i DM ha de ser un element el més genèric possible, que pugui utilitzar indistintament qualsevol DM sigui quina sigui la tecnologia que utilitzi per sota i, a la inversa, que qualsevol DM pugui contactar amb l'ADF de la mateixa forma.

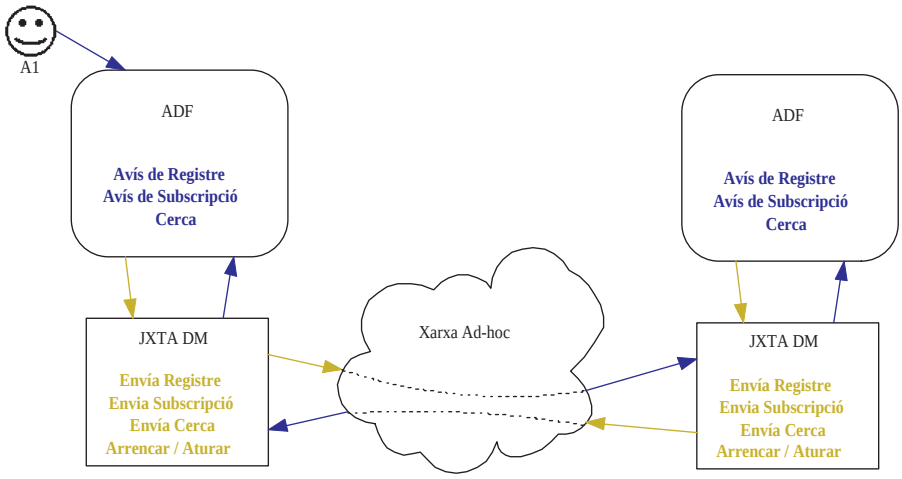

Figura 4.10: Idea d'interfície entre l'ADF i els DM.

Com veiem en la figura 4.10, la comunicació entre ADF i DM està clarament diferenciada en dos sentits: de l'ADF cap als DM i dels DM cap a l'ADF. Cadascun proporcionarà els seus mètodes i funcions per intercanviar-se informació.

- L'ADF es comunicarà amb els DM mitjançant una interfície comuna que contindrà els mètodes bàsics per propagar els missatges per la xarxa *ad hoc*. Aquests serien: arrencar DM, aturar DM, enviar anunci de registre, enviar cerca i enviar subscripció, que engloba: subscripció, desubscripció, modificació, desregistre, polling.

- Per altra banda els DM, per poder cridar els mètodes del seu ADF (al qual estan vinculats), hauran d'instanciar-lo. En crear un objecte del tipus DM en l'ADF li passarem una instància del mateix ADF. D'aquesta manera cada DM podrà utilitzar els mètodes: avís de registre, cerca, avís de subscripció, que engloba: subscripció, desubscripció, modificació, desregistre, polling.

4.4 Resum

En aquest capítol hem modelat el que serà la classe ADF per la seva posterior implementació i ho hem fet a través de diagrames UML. Amb els diagrames de

casos d'ús veiem les activitats que hauran de realitzar cadascun dels actors ADF i DM. Per entrar en una mica més de detall hem elaborat els diagrames de seqüència que ens mostren com interactuen els elements entre si.

Per acabar veiem que no obtenim tota la informació de les especificacions tècniques proporcionades per la FIPA, hem hagut de desenvolupar un protocol de subscripcions per entorns *ad hoc* basant-nos en l'actual protocol i seguint al màxim els estàndards.

Capítol 5

Implementació

Un cop finalitzades les etapes d'anàlisi i de disseny podem continuar amb la implementació de la nostra aplicació. Començarem analitzant en detall les classes principals que té l'aplicació una a una, així com els seus mètodes més rellevants. També comentarem com hem integrat els DM amb la nostra aplicació de l'ADF i descriurem la interfície i les seves característiques.

Per concloure el capítol descriurem l'escenari de proves que hem creat per testejar l'aplicació, comentarem el tipus de proves que hem realitzar per veure que tot funciona correctament i durem a terme un seguit de demostracions.

5.1 Interfície de comunicació

L'ADF desconeix la tecnologia que els DM utilitzen en les capes inferiors de l'aplicació (com poden ser JXTA, Bluetooth, etc.), per la qual cosa crearem un *interface* en java que contindrà els mètodes principals que cada DM haurà d'implementar perquè l'ADF pugui enviar-li missatges. L'hem anomenat `Discovery Middleware`, per donar-li un nom genèric, i conté els següents mètodes:

- `Start`

  Aquest és el mètode que arrenca i inicialitza un DM. En arrencar el primer DM de JXTA el mètode *Start* també és el que inicia el DM com a node dins la xarxa *ad hoc*.

- `Stop`

  Amb aquest mètode aturarem l'execució de qualsevol DM. La plataforma no serà accessible i deixarem de rebre els missatges provinents de la xarxa JXTA.

- `GetName`

  Ens proporciona en qualsevol moment el nom del DM. Aquest mètode únicament haurà de retornar el valor d'una variable global quan l'ADF li ho demani.

- `SetName`

  L'ADF cridarà a aquest mètode quan vulgui posar o canviar el nom a un DM. Li passarem una cadena amb el nou nom del DM i li assignarà aquest valor a una variable global.

- `SendRegAnnounce`

  Quan l'ADF necessiti enviar un avís de registre utilitzarà aquest mètode. Li proporcionarem el `df-agent-descriptor` de l'agent registrat.

- `SendSearch`

  Amb aquest mètode serà cridat quan l'ADF vulgui propagar una cerca cap a la xarxa *ad hoc*. Com a paràmetres necessita:

  1. `df-agent-descriptor`: com a patró de cerca.
  2. `maxResults`: nombre màxim de resultats.
  3. `maxDepth`: profunditat màxima de la cerca.
  4. `maxTime`: temps màxim d'espera.
  5. `searchId`: identificador únic de la cerca.

  Com a resultat de la cerca aquest mètode retorna una col·lecció de `df-agent-descriptor` que pot contenir cap, un o més resultats.

- `SendSubscribe`

  Tota la informació sobre subscripcions serà enviada cap a la xarxa *ad hoc* amb aquest mètode. Segons el tipus d'informació que l'ADF vulgui propagar cridarà a la funció amb uns paràmetres o altres:

  1. Propagar un avís broadcast de subscripció

     `sendSubscribe(dfadTemplate, dfadSender, "Subscribe", null)`

  2. Propagar un avís broadcast de desubscripció

     `sendSubscribe(null, dfadSender, "UnSubscribe", null)`

  3. Propagar un avís unicast de modificació d'un servei

     `sendSubscribe(dfad, oldDfad, "SubMod", remoteDM)`

  4. Propagar un avís unicast de desregistre d'un servei

     `sendSubscribe(dfad, oldDfad, "SubDes", remoteDM)`

  5. Propagar un avís unicast de polling de subscripcions

     `sendSubscribe(dfd, null, "Polling", remoteDM)`

5.2 Classes complementaries

Abans de comentar com hem desenvolupat la classe principal, `ADF`, necessitarem conèixer la implementació de varies classes i mètodes que son cridades des d'ella a l'hora d'inicialitzar i durant del procés d'execució.

5.2.1 Classe *ADFConfigurationReader*

La tasca que desenvolupa aquesta classe és llegir les dades d'un arxiu de configuració i ho fa mitjançant el tipus de dades `Properties`.

Un objecte `Properties` conté un conjunt parelles clau-valor. Aquestes parelles són com les entrades d'un diccionari: la clau és la paraula i el valor és la definició. Tant la clau com el valor són cadenes i el format d'un arxiu de propietats per la nostra aplicació té l'estructura que es mostra en la taula 5.1.

La classe `ADFConfigurationReader` llegirà aquestes parelles de propietats i les guardarà en diferents variables:

1. Un booleà per al cas de la variable de federació.

2. Una taula hash per al cas dels DM, que contindrà les parelles identificador-classe de cadascun dels DM descrits en el fitxer.

```
# Configuracio de l'ADF

# 1.Arrecar federat amb el DF per defecte
federate = false

# 2.Atributs dels DMs que arrencarem al iniciar
# dmID = dmCLASS
jxtadm1 = jade.domain.jxtadm.JXTADM
```

Taula 5.1: Arxiu de configuració de l'ADF, conté les propietats inicials.

Un cop llegides totes les propietats del fitxer ens mostrarà un petit *frame* amb tota la informació recopilada, per confirmar que el fitxer s'ha llegit i guardat correctament.

També li hem incorporat a aquesta classe els mètodes necessaris per iniciar la lectura i retornar les variables llegides, com són:

- `read`: inicia la lectura del fitxer de propietats, emmagatzema les dades en un objecte `Properties` i posteriorment les tracta i les guarda en les variables corresponents.

- `isFederated`: retorna un booleà amb el valor llegit per la clau `federate`.

- `getDMTable`: retorna una taula hash que conté els identificado
  dels DM llegits com a claus i les classes de cadascun
  com a valors.

5.2.1.1 Mètode ReadProperties

La descripció d'aquest mètode l'hem avançat a aquest punt perquè esta fortament relacionat amb la classe `ADFConfigurationReader`.

Aquest mètode es troba a la classe ADF i és l'encarregat d'inicialitzar variables globals i arrencar tots els DM especificats en el fitxer de configuració. El procediment és el següent:

1. Crea un objecte de la classe `ADFConfigurationReader`.
2. Cridem el seu mètode `read` per llegir les propietats.
3. Assignem el valor de la variable de federació a una variable global de la classe ADF.
4. Recuperem la taula hash que conté els DM que han de ser inicialitzats i per cadascun fem:
   - Creem un nou objecte `DiscoveryMiddleware` i li passem una instància de l'ADF.
   - Arranquem el DM amb el mètode `start`.
   - Afegim el DM arrencat en una llista de DM actius.

5.2.2 Classe *ADFKBSubscriptionManager*

Aquesta classe, com el seu nom indica, és la que administra les subscripcions. Registra i desregistra els agents que es subscriuen en una base de dades de subscripcions, i també és l'encarregada d'avisar als subscriptors quan algun canvi es produeix a través del mètode `handleChange`.

És una modificació de la classe `KBSubscriptionManager` que fa d'administrador de subscripcions del DF (per seguir la premissa de no modificar el codi existent).

El que hem fet ha sigut afegir la instanciació de l'ADF en el constructor de la classe, per poder cridar mètodes per propagar els events necessaris:

1. Quan es registra una nova subscripció es crida al mètode `propagateSubscription` de l'ADF.

2. Quan es desregistra una subscripció existent es crida al mètode `propagateDesubscription` de l'ADF.

Aleshores es passa el control de la propagació de la informació per la xarxa *ad hoc* a l'ADF.

5.2.3 Subclasse *SubscriptionData*

Aquesta és una subclasse de la classe ADF, i l'hem construït per crear un nou tipus de dades que ens permetés conservar tota la informació que necessitem guardar d'una subscripció remota i alhora mantenir un repositori de totes les subscripcions remotes.

El codi es tan simple que el podem mostrar a continuació: Llavors creant una

```
private  class SubscriptionData {
         DFAgentDescription sender = null;
         DFAgentDescription template = null;
         String remoteDM = "";
         DiscoveryMiddleware dm = null;
}
```

llista de elements del tipus `SubscriptionData` tindrem emmagatzemada tota la informació de les subscripcions remotes.

5.3 La classe ADF

En aquesta secció tractarem la implementació de la classe més important de l'aplicació i sobre la qual es basa tot el projecte. Hem dividit la seva implementació en vàries parts perquè quedi més clara i estructurada: inicialitzacions a la classe principal, mètodes per propagar i rebre missatges, mètodes per administrar els DM i finalment un conjunt d'utilitats per a la transformació de dades.

Si la idea proposada d'ADF en el capítol 2 era crear un nou agent que proporcionés el servei de pàgines grogues en entorns *ad hoc* però que alhora mantingués la funcionalitat bàsica d'un DF convencional, el que hem decidit és copiar i renombrar la classe DF de la plataforma de JADE, i afegir-li o canviar-li la funcionalitat per adaptar-la a les necessitats de la nostra aplicació.

Com ja havíem comentat això es va decidir fer d'aquesta manera per:

1. Minimitzar l'impacte sobre el codi de la plataforma existent.

2. Evitar una dependència forta de l'ADF sobre el DF de JADE.

Un cop copiada la classe DF i reanomenada a ADF hem de realitzar sobre aquesta una sèrie de canvis bàsics i tenir en compte algunes restriccions perquè pugui ser utilitzada com un altre *directory facilitator* en la mateixa plataforma:

- Ha de trobar-se localitzada al *package* `jade.domain` perquè utilitza classes que només tenen abast a nivell de paquet, és a dir, les classes del paquet `jade.domain` només poden utilitzar-se dins del mateix paquet.

- Alguns *behaviours*, o classes que implementen el comportament d'un agent, del DF necessiten una instància del mateix DF per cridar-li mètodes en determinades ocasions. Per al cas de l'ADF no podem aprofitar aquests comportaments que interactuen de forma directa amb el DF, necessitem que interactuin amb el nostre ADF.

  La nostra solució passa per crear els *behaviours* propis de l'ADF que seran exactament iguals al del DF però instanciaran el servei de pàgines grogues per entorns *ad hoc* (ADF). Els comportaments específics que necessiten instanciar l'ADF són:

  1. `ADFAppletManagementBehaviour`
  2. `ADFFipaAgentManagementBehaviour`
  3. `ADFIteratedSearchManagementBehaviour`
  4. `ADFJadeAgentManagementBehaviour`

- En arrencar un ADF ha d'autoregistrar-se a la plataforma. D'aquesta manera el posem a disposició de la xarxa *ad hoc*, és a dir, que qualsevol agent que vulgui trobar els nodes disponibles a la xarxa JXTA pot iniciar una cerca del servei ADF. Les dades bàsiques per a la cerca són les següents:

Nom de l'agent :	`adf@platformID`
Tipus de servei :	`fipa-df`
Nom del servei :	`df-adhoc-service`

Taula 5.2: Informació per al descobriment de serveis de pàgines grogues.

- Hem d'afegir-li un conjunt de variables globals que són necessàries per a l'administració dels DM i la propagació/recepció de missatges a través de la xarxa *ad hoc*. Aquestes són:

  1. `federate`: variable booleana que ens diu si hem de federar l'ADF amb el DF a l'inici de l'execució de l'ADF.
  2. `active_dm`: llista de DM que es troben actius en un moment de l'execució. Aquesta llista pot variar si arrenquem o aturem algun DM en temps d'execució. Conté objectes del tipus `DiscoveryMiddleware`.
  3. `remote_subscriptions`: llista amb les subscripcions remotes que conté cada ADF. Aquesta llista també variarà en temps d'execució. Conté objectes del tipus `SubscriptionData`.

5.3.1 El mètode `setup`

Si observem el flux d'execució d'un agent en la figura 5.1 veurem que el primer que s'executa és el mètode `setup`. Seguidament es comprova si l'agent està viu i després es selecciona el següent *behaviour* a executar del conjunt de *behaviours* que encara li queden per executar a l'agent.

Cada *behaviour* té un mètode `action` i un mètode `done` que van controlant la seva execució. I així fins a acabar el cicle de vida de l'agent (o mantenir-lo indefinidament amb comportaments cíclics).

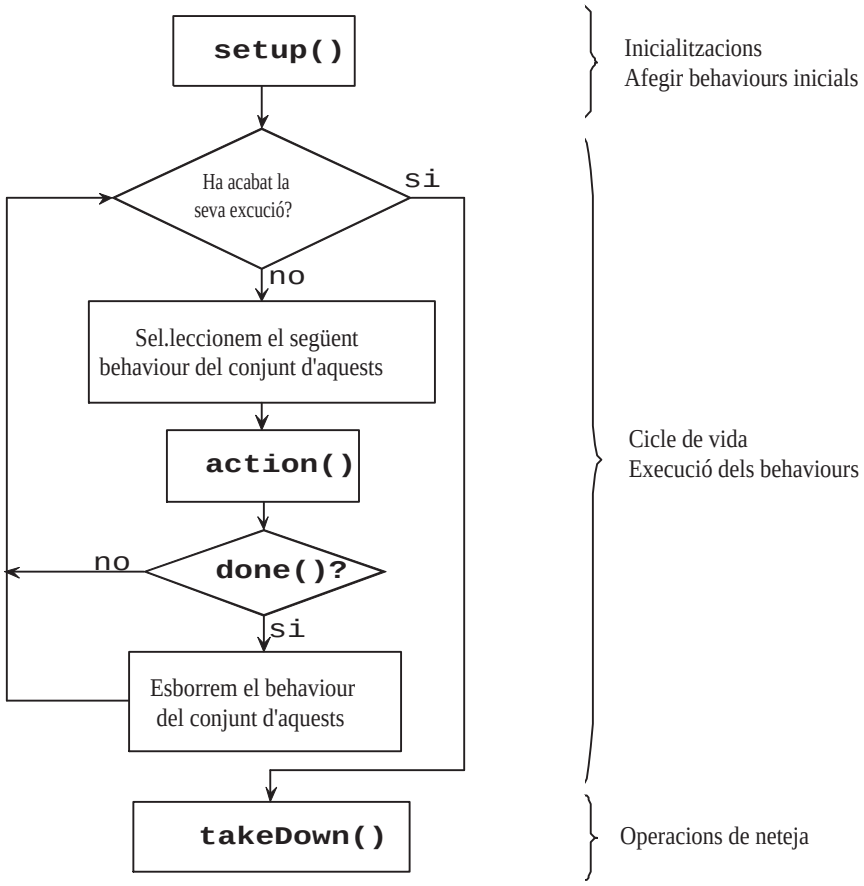

Figura 5.1: Model d'execució d'un agent.

Sabent que l'ADF és un tipus d'agent i, per tant, quan arrenca el primer mètode que executa és el `setup`, l'aprofitarem per fer les incialitzacions corresponents. Quan arranqui l'ADF farem:

1. Cridarem al mètode `ReadProperties` que donarà valor a la variable `federate`, arrencarà els DM (i en conseqüència posarà en marxa la xarxa JXTA) i crearà una llista de DM actius.

2. S'autoregistrarà a través del mètode `DFRegister`.

3. Si cal, es federarà amb el DF en dos pasos: registra el DF i l'afeigeix com a pare.

En la figura següent veiem un petit esquema de com interaccionen les classes que hem especificat anteriorment.

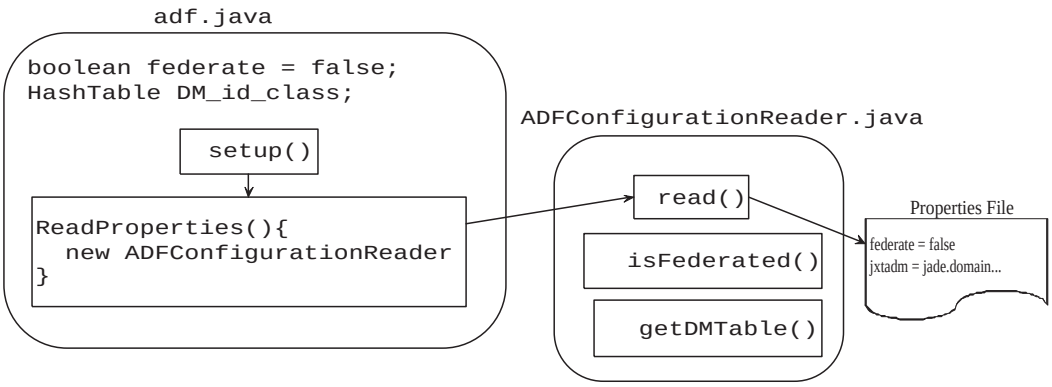

Figura 5.2: Esquema de la inicialització de l'ADF.

En aquest moment ja tenim la plataforma arrencada amb l'ADF en marxa, connectat a les xarxes *ad hoc* subjacents (en el nostre cas únicament a la xarxa JXTA) i a punt per rebre i enviar missatges.

5.3.2 El mètode `takeDown`

Com acabem de veure en el cicle d'execució d'un agent en la figura 5.1, igual que el mètode `setup` és el primer en executar-se quan arrenca un agent, el mètode `takeDown` és l'últim que s'executa abans que mori l'agent.

Aquest mètode realitza les tasques de neteja al morir l'agent, i l'utilitzarem per anar aturant tots els DM que quedin actius si per qualsevol motiu l'ADF finalitzes la seva execució.

Anirem recorrent la llista de DM actius i un per un anirem aturant-los a través del seus respectius mètodes `stop`. D'aquesta manera quan l'ADF mori, no deixarem un node inservible a la xarxa JXTA.

5.4 Mètodes de la classe ADF

Ara passarem a descriure un per un els mètodes que hem incorporat a la classe ADF. Els hem dividit en grups segons la funcionalitat que proporcionen per tenir un esquema més clar l'abast de la implementació.

5.4.1 Mètodes per propagar missatges

En aquest apartat parlarem dels mètodes que fa servir l'ADF per propagar missatges cap als diferents DM, veurem com es criden i quins paràmetres li fan falta i a més també especificarem en quin punt de l'execució de l'ADF són cridats.

`PropagateRegister`

- Funció: cada cop que es registri un agent a l'ADF aquest mètode recorrerà la llista de DM actius i propagarà en broadcast l'anunci del servei.
- Crida: és cridat des del mètode `DFRegister` que és on l'ADF registra als agents locals, i per tant és on podem aconseguir el DFAD de l'agent registrat.

```
DM.sendRegAnnounce(dfd)
```

- Paràmetres: únicament necessita el DFAD de l'agent que s'ha registrat.

`PropagateSubscription i PropagateDesubscription`

- Funció: quan un agent local es subscrigui/desubscrigui a un servei en l'ADF, el mètode en qüestió recorrerà la llista de DM actius i propagarà en broadcast l'anunci de la (de)subscripció per la xarxa *ad hoc*.
- Crida: aquest mètode és cridat des del *Subscription Manager* de l'ADF, és a dir, des de la classe `ADFKBSubscriptionManager` que és on es troba el mètode que registra/destregistra les subscripcions locals.

```
DM.sendSubscribe(dfdTemplate,dfdSender,"Subscribe",null)
DM.sendSubscribe(dfdTemplate,dfdSender,"UnSubscribe",null)
```

- Paràmetres: a aquest mètode li passem un DFAD que fa de patró de subscripcions, un DFAD que identifica l'agent que fa la (de)subscripció, una cadena que ens indica la funció de la crida, i l'últim paràmetre que correspon l'identificador del DM remot (per quan arribi l'avís a una altra plataforma) i que serà omplert per cada DM.

`PropagatePolling`

- Funció: quan passi un determinat temps (fixat per l'usuari o un temps per defecte), aquest mètode recorrerà la llista de subscripcions remotes i propagarà en unicast un avís de polling per a cada subscripció remota. Un cop fet això buidarà la llista.

- Crida: és cridat directament des de l'ADF, ja que mitjançant un *thread* controla la seva execució.
  `DM.sendSubscribe(dfdTemplate,dfdSender,"Polling",RemoteDM)`

- Paràmetres: els paràmetres els extraiem de cada subscripció remota, ja que hem guardat totes les dades necessàries. Són els mateixos que per als dos mètodes anteriors.

`PropatateDeregister i PropagateModify`

- Funció: quan un agent modifica o desregistra un servei propagarem aquests events en mode unicast cap a les plataformes remotes que tinguin algun agent subscrit. S'han de recórrer les subscripcions remotes i notificar només als agents subscrits al servei modificat/desregistrat i, a més, enviar un únic avís per plataforma.

- Crida: aquests mètodes es criden des dels respectius `DFModify` i `DFDeregister` mètodes de l'ADF que alhora són cridats quan l'agent es modifica o desregistra.

```
DM.sendSubscribe(dfd, oldDfd, "SubMod", remoteDM)
DM.sendSubscribe(dfd, oldDfd, "SubDes", remoteDM)
```

- Paràmetres: li haurem de proporcionar un DFAD de l'agent que s'ha modificat/desubscrit, un altre DFAD que ens indica l'estat anterior de l'agent, la cadena que indica la funció a realitzar i l'identificador del DM remot.

Propagar Cerca

- Funció: quan un agent inicia una cerca i aquesta ha de ser propagada per la xarxa *ad hoc* es recorren els DM actius i es propaga la cerca esperant resultats o esperant que s'esgoti el temps de cerca. Un cop rebuts els resultats de tots els DM els incorporem al resultat de la cerca local i esperem que l'ADF comuniqui el resultat a l'agent.

- Crida: per propagar la cerca no hem creat cap mètode específic, l'hem desenvolupat directament en el mètode `searchAction` perquè és el que gestiona la cerca recursiva de l'ADF.
  `DM.sendSearch(dfd,MaxResults,timeout,MaxDepth,SearchId)`

- Paràmetres: per realitzar una cerca necessitem un DFAD com a patró de cerca, un sencer que ens indiqui el nombre màxim de resultats, un valor de *timeout* per saber quan aturar la cerca, un sencer que ens proporcioni la profunditat màxima per al cas de la cerca recursiva i un identificador únic de la cerca generat per cada ADF.

5.4.2 Mètodes per rebre missatges

En aquesta secció parlarem dels mètodes que posa l'ADF a l'abast del DM per poder comunicar-se amb ell i per informar-lo dels missatges que provenen de la xarxa *ad hoc*. Veurem quins paràmetres ens passen i com es comporta l'ADF en cada situació.

`RegisterAdvice`

- Funció: en rebre un avís de registre el primer que farem serà avisar a tots els subscriptors locals sobre el nou servei. Si havia alguna subscripció local

al servei aleshores enviarem un avís de subscripció unicast per al servei registrat cap a l'ADF remot per ser notificats en el futur.

- Crida: `registerAdvice(dfd, RemoteID, dm)`
- Paràmetres: ens proporcionen un DFAD de l'agent que s'ha registrat en una plataforma remota, un identificador del DM remot i un objecte `DiscoveryMiddleware` que és el DM que s'ha comunicat amb l'ADF.

`Search`

- Funció: quan l'ADF rep un avís de cerca construeix un objecte `Search`, el qual conté les restriccions de la cerca, i crida al mètode de l'ADF `searchAction`, que ja vam explicar en el capítol de disseny com implementaba la cerca. Quan aquest mètode acabi de cercar localment, en DF federats i per la xarxa JXTA, retornarà el resultat al nostre mètode i aquest li propagarà al DM perquè contesti la cerca de l'agent remot.
- Crida: `search(dfd,maxResults,maxDepth,searchId)`
- Paràmetres: per a la cerca necessitem un DFAD patró de cerca, un sencer que ens indiqui el nombre màxim de resultats, un sencer per saber quina és la profunditat màxima per propagar la cerca i l'identificador de la cerca per evitar cicles.

`SubscriptionAdvice`

- Funció: Com ja hem comentat en altres seccions, aquest mètode engloba varies funcionalitats en si mateix, depenen de la cadena que ens indica el tipus de feina a fer:

  1. Subscripció: cerquem localment el servei al qual es vol subscriure l'agent remot. Si trobem algun resultat, guardem les dades en un objecte

`SubscriptionData` i afegim aquest objecte a la llista de subscripcions remotes. Posteriorment notifiquem en mode unicast a la plataforma remota dels serveis que hem trobat.

2. Desubscripció: si tenim la subscripció remota de l'agent remot que es vol desubscriure, l'únic que hem de fer es esborrar-la de la llista de subscripcions remotes.
3. Modificació i desregistre: avisem als subscriptors locals de l'event que s'ha produït remotament, mitjançant el `SubscriptionManager`.
4. *Polling*: busquem a les subscripcions locals si tenim la subscripció que, se'ns demana renovar i si la tenim i l'agent que s'ha subscrit encara es viu, renovem la subscripció cridant al mètode `sendSubscription` amb els paràmetres corresponents.

- Crida: `subscriptionAdvice(dfdTemplate,dfd,type,RemoteID,dm))`
- Paràmetres: se'ns proporcionarà un DFAD com a patró de subscripció i un altre DFAD com a subscriptor en el cas de la subscripció, desubscripció i polling.

  Per al cas de la modificació i el desregistre tenim un DFAD per conéixer l'agent que ha canviat i un altre DFAD per saber quin era el seu estat anterior.

  Finalment també tenim una cadena que ens indica el tipus d'operació, un identificador de DM remot i un objecte `DiscoveryMiddleware` que pertany al DM que ens ha enviat la notificació.

5.4.3 Mètodes per administrar DM

Com vam comentar al capítol anterior, l'ADF pot aturar o arrencar un DM sota demanda (de l'adminisrador de la plataforma, de l'usuari, etc.), aleshores necessitarà aquests mètodes que li permetin dur a terme les funcions desitjades. Hem desenvolupat una petita interfície gràfica que és la que crida aquests mètodes.

`StartDM`

- Funció: arrenca un DM mentre la plataforma i l'ADF estan en marxa. Cerca la classe del DM, si la troba crea un objecte i li passa una instància de l'ADF, l'arrenca i l'afegeix a la llista de DM actius.

- Crida: es crida des d'una petita aplicació gràfica que hem desenvolupat, on introduïm el nom del DM i la classe a la que pertany.

```
startDM(name, classe)
```

- Paràmetres: només necessitem dues cadenes, una pel nom del DM i una altra amb la ruta cap a la classe a la que pertany.

Figura 5.3: Interfície gràfica per arrencar o aturar DM.

`StopDM`

- Funció: atura l'execució d'un DM mentre la plataforma i l'ADF estan en marxa. Atura el DM indicat mitjançant el mètode `stop` de la interfície de comunicació, si abans havia estat arrencat, i seguidament l'esborra de la llista de DM actius.

- Crida: es crida des de la mateixa aplicació que hem desenvolupat per arrancar nous DM.

```
stopDM(name, classe)
```

- Paràmetres: necessitem dues cadenes, una pel nom del DM i una altra amb la ruta cap a la classe a la que pertany.

5.4.4 Utilitats

A més de tots els mètodes per enviar i rebre missatges, administrar la xarxa *ad hoc*, etc. També hem necessitat implementar una sèrie de funcions per transformar i comparar objectes específics de l'aplicació.

`List2DfdArray`

Aquesta funció transforma un element `List` en un *array* d'objectes `DFAgentDescriptor`.

L'hem necessitat perquè la cerca que tenia el DF implementada retorna una llista amb els resultats obtinguts i al DM li hem de proporcionar un array de resultats.

`AddDfdArray2List`

Per incloure un *array* de `DFAgentDescriptor` en una llista de resultats utilitzarem aquesta funció, que ens retornarà el resultat d'unir aquests dos objectes en una llista final.

Aquesta funció la utilitzem en la cerca a través de la xarxa *ad hoc* ja que volem afegir els resultats que ens proporciona als resultats locals.

`Contains`

Busca si una llista de subscripcions remotes (amb elements `SubscriptionData`) conté o no una determinada subscripció.

Cada cop que enviem un avís a una plataforma remota l'afegim a una llista per tal de no avisar a una plataforma tantes vegades com agents subscrits tingui.

5.5 Resum

En aquest capítol hem desenvolupat el procés d'implementació de la nostra aplicació. Hem vist com s'ha creat l'*Ad hoc Directory Facilitator* i els mètodes necessaris per a l'intercanvi de missatges amb la xarxa.

Capítol 6

Proves i Resultats

Ara que considerem que la aplicació ja està finalitzada, en aquest capítol descriurem la realització de les proves sistemàtiques per buscar errors a través de criteris específics: les proves de caixa negra i les de caixa blanca.

Tot i que al llarg de tot el procés de disseny i implementació el nostre projecte ha estat molt lligat al del DM [Ore06], el seu desenvolupament ha anat pràcticament en paral·lel, i hem anat realitzant petites proves de comunicació petició-respota, hem de comprovar que tots els objectius proposats es compleixen, i pet tant el conjunt de proves que realitzarem passaran de forma inevitable per aquest element, el *Discovery Middleware* de JXTA.

6.1 Proves durant el desenvolupament

Aquest tipus de proves són bàsicament proves de compilació i de funcionament. A mida que s'avançava en desenvolupant i la implementació de l'aplicació, mètode a mètode anem fent petites proves de funcionament amb un *Dummy DM* (petit mòdul que emula el funcionament d'un DM) per provar cada nou pas.

Amb el *Dummy DM* provem un a un els mètodes que es van finalitzant en el procés d'implementació. En finalitzar un mòdul, compilem i provem la funcionalitat bàsica sense intentar caure en errors (les proves exhaustives les realitzarem un cop finalitzada tota la aplicació).

Finalment, durant el desenvolupament, el tipus de proves que realitzem són les d'integració per verificar el funcionament conjunt de l'ADF i el DM real. Fem que s'enviïn missatges entre ells i veiem que s'ha establert la comunicació de forma correcta.

6.2 Proves després de la programació

Quan es considera que l'aplicació ja està finalitzada realitzarem les proves necessàries per intentar descobrir errors. Amb els tipus de proves que presentem a continuació decidirem si l'aplicació funciona correctament tant en el seu comportament com en la seva execució.

Com hem comentat en capítols anteriors un *Directory Facilitator* no garanteix la integritat de les dades proporcionades pels agents que es realitzen operacions sobre ell (registres, subscripcions, etc.); és únicament una eina que proporciona una funcionalitat determinada (pàgines grogues) als agents que ho requereixen.

6.2.1 Proves de caixa blanca

Les proves de caixa blanca estan pensades per comprovar el funcionament correcte de l'ADF a nivell intern davant d'entrades de valors diversos. Provarem els nous mètodes amb valors que puguin posar en perill la seva integritat per mirar de trobar errors i solucionar-los.

6.2.1.1 Mètodes cridats pel DM

- Crida al mètode `RegisterAdvice` amb valor de DFAD `null`.
  Funcionament desitjat: Missatge d'error: "DFAD null".
  Resultat: Funcionament correcte.

- Crida al mètode `Search` amb valor de DFAD n̂ull.
  Funcionament desitjat: Missatge d'error: "Paràmetres incorrectes".
  Resultat: Funcionament correcte.

- Crida al mètode `Search` amb nombre màxim de resultats menor que zero.
  Funcionament desitjat: Missatge d'error: "Paràmetres incorrectes".
  Resultat: Funcionament correcte.

- Crida al mètode `Search` amb profunditat màxima de cerca menor que zero.
  Funcionament desitjat: Missatge d'error: "Paràmetres incorrectes".
  Resultat: Funcionament correcte.

- Crida al mètode `SubscriptionAdvice` amb algun DFAD amb valor `null`.
  Funcionament desitjat: Missatge d'error: "DFAD null".
  Resultat: Funcionament correcte.

- Crida al mètode `SubscriptionAdvice` amb valor incorrecte a la cadena que ens indica el tipus d'acció a realitzar.
  Funcionament desitjat: No fa res.
  Resultat: Funcionament correcte.

També s'han realitzat proves del mateix tipus amb la resta de mètodes, comprovacions senzilles d'integritat dels paràmetres proporcionats. Seguint la mateixa mecànica que amb les proves acabades de veure, obtenim el mateix nombre d'encerts.

6.2.1.2 Mètodes per propagar missatges cap als DM

Si la comprovació de la integritat de les dades ja es fa en un costat de la comunicació, en el nostre cas en l'ADF receptor, per no fer comprovacions redundants, quan propaguem els missatges per la xarxa *ad hoc* ens refiarem de les dades proporcionades pels agents de la plataforma. Per tant no realitzarem cap comprovació que sigui redundant al punt anterior.

Hem decidit fer-les a l'ADF receptor per si per qualsevol motiu fallés la reconstrucció de les dades entre un DM i l'homòleg remot, ja que la xarxa *ad hoc* està subjecta a canvis constants en la seva configuració (els nodes poden existir i deixar d'existir molt ràpidament).

6.2.2 Proves de caixa negra

Les proves de caixa negra són les que tenen com a objectiu veure l'aplicació des de fora, és a dir sense entrar en el detall de la implementació ni en les dades que li passem. Ens imaginem l'aplicació precisament com una caixa negra a la qual li proporcionem una entrada i esperem una determinada sortida.

6.2.2.1 DM GUI

A continuació descriurem les diferents opcions que hem provat sobre la arrencada i aturada dinàmica de diferents DM.

Figura 6.1: Interfície gràfica per arrencar o aturar DM.

1. Arrencar un DM amb un nom no utilitzat i una classe correcta.
   Funcionament desitjat: Arrencar correctament el DM.
   Resultat: Funcionament correcte.

2. Arrencar un DM amb un nom utilitzat i una classe correcte.
   Funcionament desitjat: Missatge d'error "El DM ja existeix".
   Resultat: Funcionament correcte.

3. Arrencar un DM amb un nom no utilitzat i una classe incorrecte.
   Funcionament desitjat: Missatge d'error "La classe no existeix".
   Resultat: Funcionament correcte.

4. Aturar un DM amb un nom d'un DM existent i arrencat.
   Funcionament desitjat: Aturar correctament el DM.
   Resultat: Funcionament correcte.

5. Aturar un DM amb un nom d'un DM existent i no arrencat.
   Funcionament desitjat: Missatge d'error: "El DM no està arrencat".
   Resultat: Funcionament correcte.

6. Aturar un DM amb un nom d'un DM inexistent.
   Funcionament desitjat: Missatge d'error: "El DM no existeix".
   Resultat: Funcionament correcte.

6.2.2.2 Comunicació remota

En aquest últim joc de proves comprovarem la comunicació entre un ADF i dos ADF remots. Enviarem missatges i veurem que és el que esperem rebre a l'altra banda i el que rebem en realitat.

1. Proves realitzades amb el registre de serveis:

   - Registrar un agent que proporciona el servei del temps a l'ADF local.
     Funcionament desitjat: Els dos ADF remots reben un avís de registre.
     Resultat: Funcionament correcte.
   - Registrar un agent l'ADF local i un dels ADF remots té una subscripció al servei que proporciona l'agent local.
     Funcionament desitjat: Els dos ADF remots reben un avís de registre. L'ADF remot que té una subscripció al servei envia una petició de subscripció remota a l'ADF local. L'altre ADF remot no rep la subscripció.
     Resultat: Funcionament correcte.

2. Proves realitzades amb la cerca de serveis:

   - Tenim 5 agents que proporcionen el servei del temps. Iniciar una cerca amb màxim nombre de resultats 3.
     Funcionament desitjat: Rebo els tres primers resultats abans que s'exhaureixi el *timeout*.
     Resultat: Funcionament correcte.

- Tenim 5 agents que proporcionen el servei del temps. Iniciar una cerca amb màxim nombre de resultats 10.
  Funcionament desitjat: Rebo els 5 resultats un cop esgotat el *time-out*.
  Resultat: Funcionament correcte.

- Realitzem un esquema amb DF federats que formen un cicle. Iniciar una cerca.
  Funcionament desitjat: Rebo els resultats esperats abans del *timeout* i sense repetir cerques.
  Resultat: Funcionament correcte.

3. Proves realitzades amb les subscripcions:

   - Un agent se subscriu a un servei que cap plataforma proporciona.
     Funcionament desitjat: Els dos ADF remots reben un avís de subscripció.
     Resultat: Funcionament correcte.

   - Un agent se subscriu a un servei que proporciona un ADF remot.
     Funcionament desitjat: Els dos ADF remots reben un avís de subscripció. L'ADF remot que conté el servei notifica al subscriptor i es desa una subscripció remota per futures notificacions.
     Resultat: Funcionament correcte.

   - Un agent local modifica/desregistra un servei per al qual no tenim subscripcions remotes.
     Funcionament desitjat: No es fa res (a nivell *ad hoc*).
     Resultat: Funcionament correcte.

   - Un agent local modifica/desregistra un servei per al qual tenim subscripcions remotes.
     Funcionament desitjat: Únicament l'ADF que té la subscripció remota és notificat de la modificació.
     Resultat: Funcionament correcte.

- Un agent es desubscriu d'un servei.
  Funcionament desitjat: Tots els ADF reben la notificació i únicament aquells que tenien una subscripció remota l'esborren.
  Resultat: Funcionament correcte.
- Un ADF inicia un polling de subscripcions remotes.
  Funcionament desitjat: Els ADF que tenien subscripcions remotes en aquest ADF seran informats que han de renovar la subscripció si volen continuar rebent avisos. Els que estiguin en condicions de fer-ho les renovaran.
  Resultat: Funcionament correcte.

6.3 Resum

Com podem apreciar tots els casos de prova han tingut un funcionament correcte, això es perquè hem anat fent proves al llarg de tot el procés d'implementació i quan hem realitzat les proves d'integració i comunicació definitives només ens han calgut arreglar alguns detalls.

Veiem que en la funcionalitat bàsica l'aplicació funciona de la manera esperada.

Capítol 7

Conclusions

Per finalitzar el document exposarem les conclusions de tota l'elaboració del projecte.

Vam començar exposant la necessitat d'un nou mecanisme de pàgines grogues per la plataforma d'agents de JADE que abastés entorns dinàmics, tal com són les xarxes *ad hoc*. En una presentació al descobriment de serveis hem introduït els conceptes de *Middleware* i xarxes *ad hoc*, i posteriorment quina és la relació entre aquests elements.

Seguidament hem vist el procés d'anàlisi del nou DF, hem vist les diverses possibilitats i hem triat crear-ne un de nou que complementi la funcionalitat del que ja existeix. També hem tingut en compte per a l'anàlisi les característiques de la xarxa *ad hoc* i l'elaboració ha estat conjunta amb un altre projecte [Ore06] que complementa el nostre, el del *Discovery Middleware*.

En el procés de disseny veiem, a través de diagrames UML, quines funcionalitats recauran sobre cadascun dels mòduls i intentem ajustar-nos a les especificacions de la FIPA tant com a les restriccions que suposen els dispositius mòbils. Desenvolupem la interfície de comunicació entre DM - ADF i veiem alguns exemples de possibles execucions.

Finalment en el procés d'implementació observem la distribució del codi, les classes i els mètodes creats per donar la funcionalitat desitjada al nou *Directory Facilitator*.

7.1 Revisió dels objectius inicials

Seguidament repassarem els objectius que ens vam plantejar a l'inici del projecte per veure en quin grau els hem assolit.

El primer objectiu que ens vam proposar, i que hem assolit, va ser l'estudi de les especificacions de la FIPA en relació amb el *Directory Facilitator*, per poder entendre de forma teòrica el funcionament i les característiques del servei de pàgines grogues. Juntament amb aquest estava l'estudi superficial de les tecnologies *ad hoc* que tenien relació directa amb aquest projecte, en el nostre cas JXTA. I per tancar el bloc de documentació també ens havíem proposat entendre el comportament dels agents dins la plataforma de JADE.

Hem aconseguit entendre el mecanisme de xarxes *peer-to-peer* com a base per veure com funciona JXTA, alhora que compreníem i analitzàvem el funcionament dels diferents elements d'una plataforma d'agents, comprovant que complien totes les especificacions de la FIPA.

Un cop coneixem els requeriments de la FIPA sobre protocols de comunicació entre agents i serveis proporcionats pel DF, vam passar a l'anàlisi conjunt entre DM i ADF. Finalment hem aconseguit desenvolupar un *Directory Facilitator* per entorns *ad hoc* que compleixi les especificacions requerides, que proporcioni la funcionalitat desitjada i que interactui de forma transparent a l'usuari (i a l'agent) amb el *Discovery Middleware* de JXTA. Per tant, també hem assolit l'objectiu de crear una interfície entre ADF i DM independent de la tecnologia utilitzada per la xarxa *ad hoc*, ja que no utilitza cap informació específica de JXTA i que podrà ser utilitzada en futures implementacions de DM.

El nostre últim objectiu era convertir l'aplicació, juntament amb la del DM, en un *add-on* que complís les característiques especificades per JADE. Els passos per convertir les aplicacions en un *add-on* són senzills, ja que només ens cal seguir una distribució de carpetes especifiques i afegir-hi la documentació necessària per la configuració.

En aquest punt podem dir que hem assolit tots els objectius que ens havíem plantejat a l'inici del projecte, per tant considerem que els resultats obtinguts són satisfactoris.

7.2 Línies de continuïtat

Encara que l'aplicació que hem desenvolupat al llarg de tot el projecte funciona correctament encara queden aspectes que podem millorar i noves funcionalitats que poden millorar el servei. Les presentem a continuació:

- **Reconeixement del servei**: Si l'ADF reconeix el servei que requereix/proporciona un agent pot fer un ús intel·ligent dels DM que té a la seva disposició.
- **Gestió de DM prohibits per servei**: Quan un agent sol·liciti o proporcioni un servei determinat, l'ADF pot gestionar a quins DM pot o no pot notificar o demanar la informació.
- **Tolerància a fallades**: Mantenir a l'ADF una caché o una base de dades amb un estat de la xarxa en un determinat temps, ja que si un node es cau o es desconnecta, pugui mantenir la informació.
- **Millora de la interfície**: Millorar la interfície per carregar DM de forma dinàmica afegint recursos visuals.

Bibliografia

[uml05] UML Unified Modeling Language, gener 2007. <http://www.uml.org/>

[AAS02] FIPA Abstract Architecture Specification, decembre 2002. <http://www.fipa.org/specs/fipa00001/>

[AMS04] FIPA Agent Management Specification, març 2004. <http://www.fipa.org/specs/fipa00023/>

[SIPS02] FIPA Subscribe Interaction Protocol Specification, decembre 2002. <http://fipa.org/specs/fipa00035/>

[ADSS03] FIPA Agent Discovery Service Specification, novembre 2003. <http://www.fipa.org/specs/fipa00095/>

[FATC06] FIPA Ad Hoc Technical Comnittee, 2006. <http://www.fipa.org/activities/ad_hoc.html>

[Ore06] Orellana Quintilla, Oriol. JADE discovery middleware en JXTA: *Projecte Final de Carrera*. Bellaterra: Universitat Autònoma de Barcelona, 2006.

[Lac06] Lacasta Tomàs, Elisabet. Coffe shop roaming agent: textitProjecte Final de Carrera. Bellaterra: Universitat Autònoma Barcelona, 2006.

[Cel04] MłCeleste Campo Vázquez. Tecnologías middleware para el desarrollo de servicios en entornos de computación ubicua: *Tesis Doctoral*. Universidad Carlos III de Madrid, 2004.

Firmat: Sílvia Ezponda Mares

Bellaterra, Febrer de 2007

Resum

Aquest projecte presenta una solució intregrada amb les especificacions de la FIPA d'un servei de pàgines grogues per a la plataforma d'agents de JADE en entorns *ad hoc*. El servei de pàgines grogues interaccionarà a través d'un *middleware* amb diferents tecnologies P2P per proporcionar als agents un servei totalment fiable i transparent que els ajudarà a superar la limitació del sistema de federacions de FIPA.

Resumen

Este proyecto presenta una solución integrada con las especificaciones de FIPA de un servicio de páginas amarillas para la plataforma de agentes de JADE en entornos *ad hoc*. El servicio de páginas amarillas interaccionará a través de un *middleware* con distintas tecnologías P2P para proporcionar a los agentes un servicio fiable i transparente que los ayudará a superar la limitación que supone el sistema de federaciones de FIPA.

Abstract

This project presents an integrated solution, according to FIPA's specifications, of yellow pages service for JADE agent platform in ad hoc networks. The yellow pages service will interact through a middleware with several P2P technologies providing agents with a reliable and transparent service to solve the limitation that offers FIPA's federation.