DESENVOLUPAMENT D'UN COMPONENT DE TOLERÀNCIA A FALLADES PER A UN SISTEMA D'AGENTS MÒBILS SOBRE PLATAFORMES LLEUGERES

Memòria del projecte de final de carrera corresponent als estudis d'Enginyeria Superior en Informàtica presentat per Xavier Piñol Torné i dirigit per Joan Borrell Viader.

Bellaterra, maig de 2008

El firmant, Joan Borrell Viader, 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 Xavier Piñol Torné

Bellaterra, maig de 2008

Firmat: Joan Borrell Viader

A vosaltres, que heu estat al meu costat

Agraïments

Voldria aprofitar aquesta secció per agrair a en Joan Borrell i a en Jordi Cucurull la seva paciència i la seva inesgotable font de saber. Al departament per donar-me aquesta oportunitat. A la resta de companys del SENDA amb els quals hem passat estones inoblidables: Àlex, David, Gerard, Oriols, Xavi (*en el fondo 15" no está tan mal*). Als llunàtics del CCG que segueixen creient que MAGMA serà OpenSource algun dia. A la resta de companys i companyes de la carrera que tantes estones hem viscut i tants cafès hem pres. També voldria agrair a la Raquel per ser al meu costat sempre i ajudar-me a tirar endavant. Als meus pares, que m'aguanten cada dia i només per això ja es mereixen un doctorat. A l'Anna, el Fran i la petita Alba, una petita estrella que té molta llum per donar. A tots, i als que em deixo, que sé que són molts...gràcies!
...i al meu *Acer* que després d'això es mereix unes vacances!

Índex

Índex de figures

Capítol 1

Introducció

En els temps en que vivim, les noves tecnologies i els avenços en el camp de la informació han fet que el volum de les dades amb les que tractem sigui immens.

El concepte de *mar de dades* defineix una nova manera d'estructurar la informació, deixant de tenir les dades centralitzades en un únic lloc en favor de la distribució de recursos i repartiment de tasques. D'aquesta manera se'ns presenta la necessitat de gestionar la informació, i d'aquí sorgeix la idea dels agents mòbils. Aquests són entitats autònomes amb uns objectius definits i amb la capacitat de copiar el seu codi a través de la xarxa entre màquines per continuar executant-se.

L'hàbitat dels agents rep el nom de plataforma o agència i s'ha de troba en execució en les màquines on es vol permetre la presència d'un o més agents. Amb aquesta situació que acabem de definir neix la idea de l'agent que va a buscar les dades i no que les dades van cap a l'agent.

En les situacions en les que hem de tractar grans volums d'informació, entorns on les connexions són extremadament lentes o on les dades son sensibles a algun terme legal, com poden ser els expedients mèdics, són casos típics on una implementació acurada de la gestió de serveis és important. En aquest marc, les aplicacions basades en agents mòbils poden tenir un pes important. Gràcies a la versatilitat, la capacitat d'extendre les seves funcionalitats i adaptar-se a les diverses situacions amb les quals es va trobant, ofereixen una alternativa als sistemes distribuïts convencionals, on no prima l'alt rendiment de resultats sinó que la tasca

s'acompleixi satisfactòriament.

Pensem ara en què passaria si una tasca que porta hores o dies recopilant informació, interaccionant amb l'entorn i canviant l'estat d'aquest, per algun motiu fallés i tota la feina de la qual en depenia es perdés i no es pogués recuperar. Seria frustrant per la persona responsable. Una solució seria anar emmagatzemant les dades parcials del procés per poder recuperar el punt de fallada. Però si aquesta es fes en un entorn restrictiu on els permisos són molt limitats i no pogués accedir als recursos del disc dur? Aquest seria un cas típic en el que un agent mòbil es pot trobar.

Al llarg dels anys, s'han anat investigant diversos mecanismes per tractar de minimitzar el dany ocasionat per una fallada del sistema. Danys a nivell de temps d'inactivitat de servei o de quantitat de dades perdudes. L'execució en paral·lel de les mateixes tasques, o un element de suport que recolzi l'extracció de dades són possibles vies per trobar una solució a aquesta problemàtica. És el que coneixem amb el nom de tolerància a fallades.

El grup de recerca SENDA [1] del Departament d'Enginyeria de la Informació i de les Comunicacions porta ja uns quants anys treballant en el desenvolupament d'agents i serveis per la plataforma JADE [2], estenent la seva funcionalitat amb un servei de migració multi protocol entre plataformes. JADE és una bona solució en el paradigma d'agents mòbils, ja que compleix els estàndards de FIPA [3], organització dedicada a definir els protocols de comunicació i interacció entre agents. Els treballs de recerca sobre agents mòbils del grup SENDA que estan orientats a una aplicació mèdica [4] basada en aquesta arquitectura.

Un dels projectes desenvolupats pel departament ha sigut la construcció modular d'agents basats en components [5]. Seguint amb la filosofia d'aquest projecte, ens marcarem com a objectiu la construcció d'un component que ofereixi la funcionalitat de tolerància a fallades. Aquest component haurà de complir amb la premissa de les plataformes lleugeres definit quan es va proposar el servei de migració del departament. El codi de la plataforma ha de ser portable, per tant, no s'ha de modificar. Tota la feina ha de recaure en l'agent mòbil.

Establim, per tant, els objectius concrets que ha de tenir el projecte i que tractarem d'assolir:

- Analitzar els diversos protocols de tolerància a fallades que hi ha actualment per esquemes d'agents mòbils i extreure'n un que compleixi les expectatives.
- Definir els requisits que hem d'acomplir per oferir un servei de tolerància a fallades que doni robustesa i fiabilitat al nostre sistema d'agents mòbils.
- Ampliar la biblioteca de components disponibles pels agents mòbils amb un component de tolerància a fallades. Aquest ens ha d'oferir fiabilitat, no ens podem permetre perdre la feina feta per l'agent. El component ha de complir:
  - Ha de ser autònom, no ha de dependre d'altres components
  - La comunicació entre agents ha de complir amb els estàndards de FIPA.
  - Mantenir l'estructura de l'agent mòbil el més intacta possible, evitar modificar l'agent en la mesura del possible.
  - Garantir la propietat d'execució única, és a dir, les tasques només s'hauran d'executar un cop. Si falla un agent, quan es recuperi, no s'hauria de tornar a executar aquesta tasca.
  - Procurar que aquest component no repercuteixi gaire en el rendiment de l'agent sense penalitzar la seva funcionalitat.

Descrivim ara les parts en que s'ha distribuït la memòria, comentant el que el lector es trobarà en cada capítol:

Capítol 2: Estat de l'art. En aquest capítol fem una breu anàlisi de quins són els models de tolerància a fallades que hi ha en la actualitat i què en podem extreure d'ells. També introduïm els agents i les plataformes de JADE i fem un breu incís en els estàndards de FIPA.

Capítol 3: Anàlisi. Després d'introduir la situació actual, analitzem els requisits del projecte, tant funcionals com no funcionals. Comentem també la viabilitat del projecte, tant legal com econòmica i tècnica. En aquest anàlisi introduïm les eines que farem servir per aquest projecte.

Capítol 4: Disseny. Analitzats els requisits del projecte, ens encaminem cap a dissenyar l'esquema de funcionament i els elements que hi intervenen. Extraiem una visió de com es comportaran els actors del model adaptant-la als requisits extrets a la fase d'anàlisi.

Capítol 5: Implementació i proves. Tenim el disseny ben definit, entrem en la fase d'implementació i proves. En aquest capítol expliquem les característiques més peculiars de la implementació, mencionem les eines utilitzades en tot el procés del projecte i finalment comprovem que el codi funciona, i quins resultats ens dona.

Acabem la memòria amb el capítol de Conclusions.

Capítol 2

Estat de l'art

Al llarg d'aquest capítol anem a veure quin és el nostre punt de partida. Explicarem el paradigma d'agents mòbils i les diverses vies d'investigació que hi ha referents a la tolerància a fallades en agents mòbils.

2.1 Els agents mòbils

Definim els agents mòbils com entitats autònomes que executen una sèrie de tasques programades, amb la capacitat de poder transportar el seu codi, el seu estat i les seves dades entre plataformes. També tenen la capacitat de poder comunicar-se amb d'altres agents mitjançant protocols estàndards.

Definim l'itinerari d'un agent com les plataformes que ha de recórrer aquest per executar les seves tasques abans de poder finalitzar la seva execució.

Entrant una mica més en detall i investigant una mica el seu comportament, veiem que els agents segueixen els principis de la intel·ligència artificial. Són capaços de decidir quin és el codi que han d'executar en funció de la tasca a realitzar. S'ajusta al concepte d'agent racional autònom amb un objectiu per complir i que va decidint les accions a prendre segons la situació, *interactuant* amb l'entorn, adaptant-se a les situacions que s'esdevenen.

Això ho podem aplicar a un entorn distribuït amb milers de màquines que contenen informació. Per processar tota aquesta informació necessitem d'una aplicació en cada màquina per fer els càlculs pertinents i demanar dades a d'altres màquines cada cop que es necessitin, amb la càrrega de tràfic de xarxa que suposa.

En canvi, l'agent s'executa a cada màquina independentment, i si necessita dades localitzades en un altre lloc, quan passi per allà ja les demanarà. La idea que se n'extreu és que tenim un algorisme que es va movent a la recerca de les dades i quan les troba les processa. Tot això sense que necessiti de la nostra supervisió per fer efectius els resultats.

Figura 2.1: Agent

Un agent està format pel codi, les dades que ha anat generant o recollint i l'estat en que es troba (figura 2.1). Com si fos una màquina d'estats, l'agent anirà executant el seu codi considerant l'estat en que es trobi, utilitzant la plataforma on es trobi com a entorn d'execució.

2.2 Les plataformes

També reben el nom d'agències. Aquestes són l'entorn d'execució dels agents, el medi sobre el qual treballaran. Les agències donen tots els serveis necessaris per que els agents puguin exercir les seves tasques. També s'encarreguen de controlar les peticions de l'agent als recursos de la màquina, impedint qualsevol acció fora del normal.

Aquestes han d'estar distribuïdes sobre totes les màquines a les que es vol

accedir amb un agent, ja que ofereixen l'espai de treball d'aquest, per tant, haurien de ser genèriques, sense codis personalitzats, per facilitar la seva distribució. Un dels objectius del grup de recerca SENDA és precisament aquest, orientar la feina cap a l'agent, deixant que les plataformes siguin el màxim lleugeres possible.

2.3 JADE

JADE és un *software* de distribució gratuïta regulat sota llicència LGPL (Lesser General Public License Version 2).

L'entorn de desenvolupament JADE esta íntegrament desenvolupat en JAVA. Serà l'eina que utilitzarem per al desenvolupament d'agents.

El fet que estigui implementat en JAVA ens permet desenvolupar aplicacions multiplataforma. JADE ens permet el desenvolupar agents de forma senzilla, oferint serveis bàsics tals com enviament de missatges, programació de tasques, etcètera. Per estendre aquestes funcionalitats, es poden afegir comportaments (*behaviours*), permetent realitzar tasques més complexes als agents. A la figura 2.2 podem veure l'estructura d'un agent de JADE.

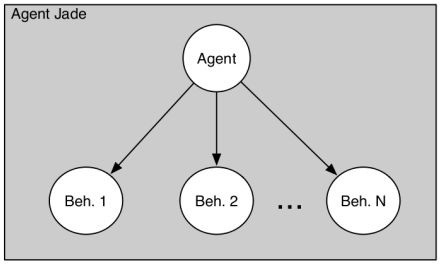


Figura 2.2: Agent amb behaviours

JADE ofereix per la gestió de les seves plataformes dues entitats, l'AMS (Agent Management Service) i el DF (Directory Facilitator).

L'**AMS** és l'encarregat de gestionar els agents registrats en una plataforma. Quan es llença un agent, o es destrueix un agent, etcètera, l'AMS és l'encarregat de tramitar aquestes peticions i mantenir en un registre la correspondència d'agents actius. Tot agent que es trobi en una plataforma estarà registrat per l'AMS.

Posem que la equivalència és que l'AMS seria les pàgines blanques dels agents.

El **DF** és als serveis el que l'AMS és als agents. Els serveis que ofereix una plataforma estaran registrats al DF. Aquest serà l'encarregat de llistar els serveis que tingui registrats quan un agent faci una petició. L'equivalent a les pàgines grogues de les plataformes.

La comunicació entre agents i/o serveis de plataforma es fa seguint els estàndards de FIPA, cosa que simplifica bastant la implementació d'agents. Anem a parlar d'aquests estàndards

2.4 FIPA

FIPA és una organització de la *IEEE Computer Society* que promou les tecnologies basades en agents i la interoperabilitat dels seus estàndards amb d'altres tecnologies.

Aquesta organització ha basat es seus esforços en la recerca d'estàndards per definir com s'executaran els agents (a través d'agències) i com es comunicaran aquests agents entre ells.

Remarcables són els estàndards per la comunicació entre agents FIPA-ACL (*Agent Communication Language*) i les ontologies per administració d'agents (*Agent Management*).

2.5 Models de tolerància a fallades

En la actualitat existeixen diversos models per gestionar el tractament de les fallades en sistemes distribuïts, que podem emmarcar en dues vessants.

La primera pretén emmascarar les fallades de l'agent mòbil o de la plataforma on es trobi aquest, duplicant la feina amb diversos agents com es proposa a [9] o replicant les dades com a [8].

La segona pretén detectar l'error i tractar de solventar-lo. És aquí on entra el concepte d'agent de suport o de rereguarda per fer el seguiment de la activitat de l'agent mòbil. En l'esquema proposat a [10] un agent de recolzament segueix

l'activitat de l'agent mòbil. Aquest agent mòbil va registrant tota la seva activitat en la màquina host on es troba per les possibles posteriors tasques de recuperació, mentre que l'agent de suport serveix de còpia backup de les dades que va recollint l'agent que es troba al front.

Anem a comentar amb detall aquests models,basant-nos en exemples concrets.

FATOMAS ([8])

La línia que segueix aquest model és la de tractar d'evitar el bloqueig de l'agent quan una plataforma falla en la seva execució. En aquest document es proposa tenir un model amb les plataformes duplicades, com un clúster de plataformes. Quan arriba un agent, es replica i s'envia una còpia a cada plataforma duplicada. Quan l'agent executi les seves tasques, tindrem un petit inconvenient, que l'esquema ha de complir amb la propietat d'execució única, ja que segons quines tasques no poden estar duplicades. Tenim dues opcions per solucionar això: o tirem enrere les accions fetes sobre totes les plataformes duplicades excepte en una (*rollback*), o fem que s'executi només un agent, i si l'execució d'aquest falla, aleshores entra en funcionament el segon, i així successivament.

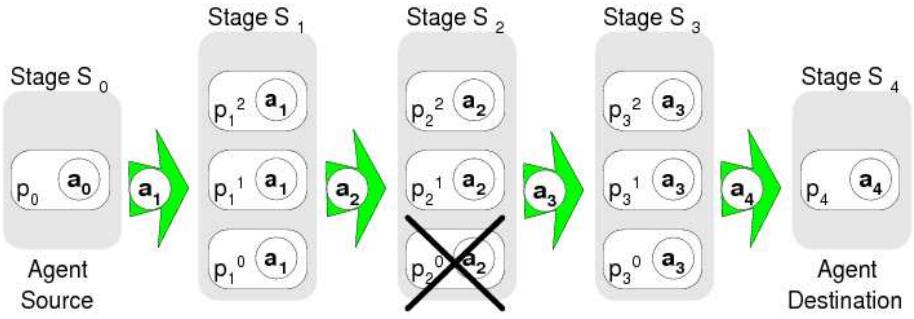


Figura 2.3: Esquema de funcionament del model FATOMAS amb replicació d'agents

JAMES ([9])

Aquest model es basa en tenir un agent estàtic en una plataforma que va rebent la informació que l'agent mòbil li va enviant, mantenint una sincronia en les dades. Té tres modes de funcionament.

Simple Mobile Agent Llança un agent cap a una plataforma, fa les tasques que hagi de fer i torna amb els resultats cap al JAMES Manager (la plataforma central). (fig. 2.4)

Migratory Agent Llança un agent que recorrerà tot un itinerari marcat, executant les tasques respectives en cada plataforma i tornarà al JAMES Manager.(fig. 2.5)

Master/Worker Model Tenim un agent master que crea tants agents com agències a les que vulguem anar, i els llença un a cadascuna d'elles, executant les tasques en paral·lel i després tornant a l'origen amb els resultats.(fig. 2.6)

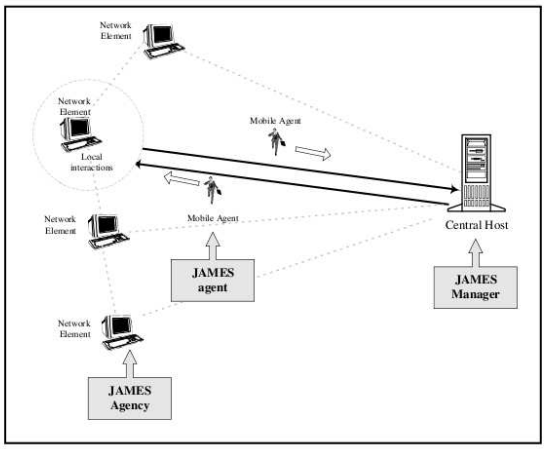


Figura 2.4: JAMES Simple Mobile Agent

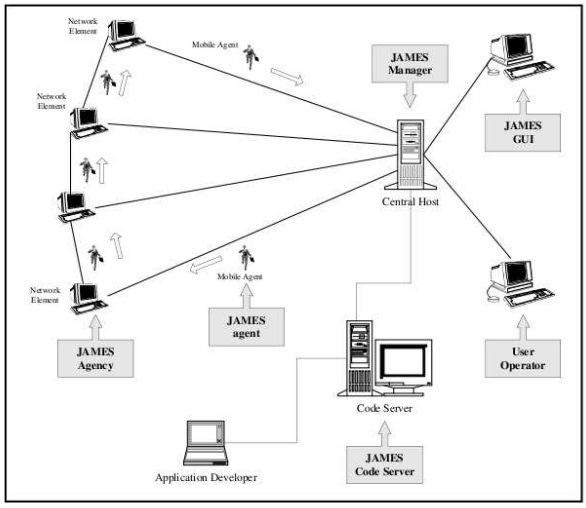


Figura 2.5: JAMES Migratory Agent

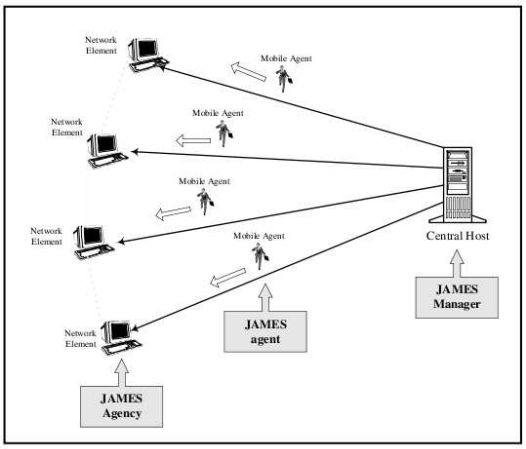


Figura 2.6: JAMES Master/Worker Model

Fault-Tolerant Mobile-Agent System ([10])

El sistema dissenyat per Lyu, Chen i Wong es basa en utilitzar dos agents mòbils que es van donant suport, mentre l'agent actiu va executant les tasques i generant logs i checkpoints en la plataforma actual, també va enviant missatges a l'agent de suport que està inactiu en la plataforma anterior. Quan s'acaben les tasques

de l'agent actiu, aquest passa a estar inactiu i el segon agent migra a la següent agència de l'itinerari comú. El primer agent passa a esperar que el segon li envii la informació recollida, i així successivament. (fig. 2.7)

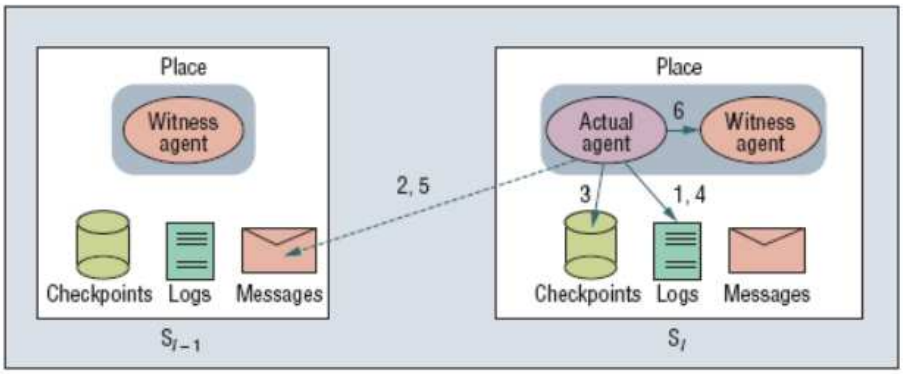


Figura 2.7: Fault-Tolerant Mobile-Agent System: Passos 1, 4: Registra log, passos 2, 5: Replica informació, pas 3: Checkpoint, pas 6: Desperta l'agent de suport

2.6 Conclusions

La implementació del model JAMES dóna una primera visió del que es pretén aconseguir, és a dir, un model d'agents amb dades duplicades i un seguiment de l'itinerari de l'agent mòbil. Però podem veure, que el model de Lyu, Chen i Wong, té una complexitat més elevada, i ofereix un xic més de robustesa, en el fet que els dos agents són itinerants, i no n'hi ha cap d'estàtic que depengui d'una sola plataforma. El model FATOMAS pretén assegurar la correcta execució de la tasca de l'agent duplicant la seva execució, cosa que veiem que és el menys òptim dels models esmentats. Les implementacions prèvies existents al grup SENDA condicionen la implementació del projecte, ja que aquest serà un component que haurà de seguir uns estàndards ja definits per a agents amb components. També haurà de seguir uns protocols per poder ser executat en plataformes lleugeres, adaptades pel grup SENDA per oferir mobilitat a l'arquitectura d'agents estàtics inicial de les llibreries de JADE (Java Development Framework).

Capítol 3

Anàlisi

En aquest capítol analitzem quins requisits, tant funcionals com no funcionals, són necessaris per marcar les pautes d'aquest projecte per ajudar-nos a emprendre les primeres decisions de disseny. Introduirem els objectius marcats pel departament que esdevenen requisits no funcionals, proposarem i argumentarem una solució i acabarem veient si aquest projecte és viable tècnicament, econòmica i legal.

3.1 Requisits inicials

Aquest projecte està emmarcat dins del paradigma d'agents mòbils sobre plataformes lleugeres. La idea general d'aquest és el de fer que les plataformes tinguin el mínim codi necessari per ser el màxim transportables possibles. D'aquesta manera, tota la feina recaurà sobre els agents, i aquests podran formar una comunitat de serveis i recursos per abastir-se els uns als altres. En el nostre cas, l'agent mòbil tindrà una sèrie de components que estendran les seves funcionalitat.

3.1.1 Requisits funcionals

Tenim com a fita assegurar que quan l'agent mòbil executi les seves funcions sobre les plataformes que hi ha en el seu itinerari, l'execució es faci correctament d'inici a fi. Per poder assolir aquest objectiu, haurem de procurar implementar un protocol d'enviament i confirmació de dades que ens permeti verificar, en tot

moment, que si qualsevol cosa li passés a l'agent mòbil, des de l'agent de suport es podrà recuperar la feina feta i continuar-la des d'on s'ha interromput.

Hem de garantir la propietat d'execució única, és a dir, que una tasca assignada a una plataforma i per un agent concret, només s'executi un cop. Posem per cas una transacció bancària, si a mitja transacció hi ha un problema, hem de procurar que la transacció s'acabi correctament, però que no es repeteixi. Hem de fer les operacions el més atòmiques possibles.

3.1.2 Requisits no funcionals

Per poder assolir els requisits funcionals ens haurem d'acotar a l'entorn de desenvolupament i les llibreries de JAVA, que com ja hem dit, és el llenguatge amb el qual està implementat el framework que ens ofereix JADE per permetre'ns treballar amb les plataformes i els agents mòbils. Aquest entorn de desenvolupament té l'avantatge que és multiplataforma i exportable a qualsevol arquitectura.

Aquest projecte forma part dels treballs de recerca sobre agents mòbils del grup SENDA que estan orientats a desenvolupar una aplicació mèdica [4] basada en l'arquitectura d'agents mòbils. Un dels clars objectius marcats és el de fer que les plataformes tinguin la mínima càrrega possible, traslladant la feina als propis agents.

Per poder treballar en un sistema de plataformes distribuït necessitarem d'un servei de migració. En el nostre cas, el departament SENDA ha desenvolupat un servei de migració per a plataformes lleugeres, InterPlatform Mobility Service [7], sobre el framework de JADE, que ens permetrà executar la migració d'un agent entre plataformes còmodament, seguint els estàndards de FIPA. Per efectuar la comunicació entre agents, seguirem també els estàndards de FIPA, ja que d'aquesta manera tenim unificada la comunicació entre els diversos agents que hi pugui haver en tot l'esquema distribuït.

Pretenem que l'agent mòbil sigui el més genèric possible, la seva implementació no s'ha de veure alterada pels components que s'hi afegeixin. Hem de modificar l'agent mòbil només quan necessitem algun servei que sigui comú a tots o gran part dels components que aquest contingui i estigui justificada la seva necessitat.

El mateix passa amb els atributs que contindrà l'agent. S'ha de procurar que només contingui les dades que poden ser utilitzades per tots els components. En el nostre cas les dades pertinents a l'agent de suport i a l'itinerari seran necessàries.

Som conscients, però, que aquest comportament seguirà de prop l'execució de la resta de components, per tant, necessàriament requerirem controlar l'agent a través del seu component de tolerància a fallades.

3.2 L'agent de suport

Presentem el concepte d'agent de suport introduït a [12]. En un sistema d'agents mòbils on certs components requereixen d'una part complementària per completar la seva funcionalitat, aquesta formarà part de l'agent de suport.

Entre l'agent mòbil i l'agent de suport del nostre projecte existeix una simbiosi latent, l'un no pot existir sense l'altre, ja que la voluntat d'atorgar una funcionalitat de Tolerància a Fallades a l'agent mòbil implica que aquest ha de tenir un agent de suport que atengui les seves peticions. Aquest agent de suport, per la seva banda, requereix de certes accions per part de l'agent mòbil.

L'agent de suport, com el seu nom indica, oferirà les tasques de suport de l'agent mòbil. En el marc concret en que ens trobem, aquest haurà d'oferir els serveis complementaris de tolerància a fallades que no pugui exercir l'agent mòbil directament.

3.3 Proposta de Model de tolerància a fallades

El nostre esquema serà una aproximació a l'esquema proposat a [10], on l'agent de suport o rereguarda estarà en una plataforma fixada, controlada per l'usuari. L'agent mòbil utilitzarà els mínims serveis de la plataforma que l'allotgi, ja que a priori no coneixerem l'entorn per on viatjarà l'agent i per tant no sabem si tindrem accés a certs recursos com el de disc en totes les plataformes per on passi.

Per poder implementar aquest esquema haurem de definir, per cada agent, dos comportaments aïllats però que treballaran conjuntament per donar robustesa al

sistema.

Aquests dos comportaments els podem separar en els iniciats per l'agent mòbil, i els iniciats per l'agent de suport.

- Els iniciats per l'agent mòbil han de ser els que ajuden al *tracking* i els que inicien l'acció de *checkpointing*. És a dir, els que permeten que l'agent de suport pugui recollir la posició de la seva contra part i els que acaben volcant la còpia de l'estat de l'agent mòbil cap a l'agent de suport per posteriorment emmagatzemar-lo, respectivament. Aquest comportament rebrà el nom de **Component de tolerància a fallades** o ***faulttolerance***.

- Els iniciats per l'agent de suport són els que forcen una reacció de l'agent mòbil per demostrar el seu estat de vida. Es forçarà cada cert temps una petició de *keepalive*. Aquest comportament rebrà el nom de **Component de prova de vida** o ***keepalive***.

3.3.1 Component de tolerància a fallades

Aquest component, rep l'adjectiu de *component compost* perquè té la seva funcionalitat repartida entre l'agent mòbil i l'agent de suport. Per tant, definim el component mòbil com la part del component que s'executarà en l'agent mòbil i el component de suport o estàtic a la part del component que s'executa en l'agent de suport.

És molt important pel funcionament del model que estem desenvolupant, que coneguem en tot moment la posició de l'agent mòbil. Per tant hem de tenir uns mecanismes per a complir amb aquest requisit. Ho farem a través d'aquest component.

En primer lloc, el component mòbil serà l'iniciador de les comunicacions en aquest component. Els tipus de comunicació que haurà de poder implementar haurien de ser la petició de migració a una altra plataforma, la confirmació d'arribada a la plataforma destí i la petició de *checkpoint*.

A través d'aquests missatges, el component de suport podrà recuperar la informació de localització de l'agent mòbil. Comentem amb una mica més de detall

quines són les característiques del component.

Part activa

La part activa d'aquest component serà la de l'agent mòbil.

En primera instància, per poder iniciar la part mòbil del component haurem de recuperar la localització de l'agent de suport. Aquesta informació haurà de ser prefixada. D'aquesta manera sabrem a on aniran destinats els missatges d'aquest. Per tant, l'agent mòbil serà l'iniciador de les comunicacions.

També haurem de saber a priori si es farà petició de *checkpoint* de l'estat de l'agent en l'execució del component. Quan hi hagi una petició d'aquest tipus, l'agent es comportarà com si es fes una migració normal però el destinatari serà l'agent de suport que s'encarregarà d'atendre aquesta petició.

Un paràmetre important que marcarà el rendiment del component és el temps de *timeout*. Aquest definirà els temps d'espera fins tornar a intentar la comunicació en cas que no hi hagi resposta de la contrapart. Aquest paràmetre és vàlid tant pel component mòbil com pel de suport.

Part passiva

La part passiva d'aquest component serà la de l'agent de suport.

El component de suport haurà d'estar escoltant les peticions del component mòbil, ja siguin de *tracking* o de *checkpointing*. Qualsevol petició i qualsevol missatge rebuts del component mòbil haurà d'ésser registrat com a event. Aquesta part del component, com ja hem dit, no farà cap acció d'iniciar comunicacions.

Aquest model de tolerància a fallades, com es pot veure, basa el seu funcionament en la constant comunicació entre el component mòbil i el de suport. S'ha de mantenir l'agent de suport sincronitzat, per això seria recomanable que just abans i just després de cada migració, el component mòbil informés el component estàtic de les seves intencions. Aquestes notificacions haurien d'estar confirmades per poder-se dur a terme per d'aquesta manera garantir que l'agent de suport està actiu i en servei, ja que sinó, el model podria patir inconsistències.

3.3.2 Component de prova de vida

Aquest component també serà compost. Si el component de tolerància a fallades ha de fer les funcions d'enviament de informació referent a l'agent mòbil, i l'actor principal és precisament aquest, en el component que anem a explicar tenim la situació inversa. L'actor principal d'aquest component és l'agent de suport. Les funcions del component de prova de vida, han de ser el certificar que l'agent mòbil està viu i operatiu. Per a això ha de saber en tot moment on es troba l'agent mòbil, i d'això s'encarregarà el component de tolerància a fallades.

El component de prova de vida té una altra tasca que ha de complir. Si fallés la comunicació entre el component mòbil i el de suport, degut a que l'agent mòbil ha mort, pel motiu que sigui, aleshores el component de suport hauria d'iniciar les tasques de recuperació, intentant que tota la feina que s'ha fet fins aleshores es pugui continuar.

Anem a explicar amb més detall cadascuna de les parts que composa aquest component.

Part activa

La part activa d'aquest component serà la de l'agent de suport.

Aquest component no podrà iniciar les seves tasques fins que no rebem la informació de localització de l'agent mòbil. El component de tolerància a fallades obrirà el pas a aquest component. Quan tinguem aquesta informació podrem començar el funcionament del component.

Haurem de definir també els paràmetres de tolerància a temps d'espera i nombre d'intents per establir una comunicació. Aquests marcaran la heurística de l'algorisme, podent-la definir com a permissiva, amb molts intents i molt espaiats, que donaran uns temps de reacció molt elevats, o una de més restrictiva que ens donarà uns temps de reacció força més baixos però amb la possibilitat de tenir falsos positius. Serà important tenir aquests paràmetres ben definits.

Però, que passa quan es perd el contacte amb l'agent mòbil?

Aquest component a més ha d'oferir resposta a fallades. El component de prova de vida de l'agent de suport ha de ser capaç de poder contactar amb la

plataforma destí i poder gestionar un nou arranc de l'agent des de la última tasca que estigués fent o des d'un estat consistent.

Part passiva

La part passiva d'aquest component serà la de l'agent mòbil.

Aquest component tindrà una única tasca, que serà la d'estar esperant a rebre la petició de prova de vida i quan la rebi, enviarà una confirmació. El funcionament d'aquest protocol equival al d'un ping amb resposta.

Anem a comentar ara l'arquitectura del protocol de comunicació dels dos components amb les seves parts de suport i mòbils. Definirem pas a pas cadascuna de les accions que faran, veient com interactuaran conjuntament.

3.3.3 Arquitectura del protocol

L'esquema de funcionament del protocol serà el següent: Tenim tres hosts amb una plataforma cadascun, la plataforma local, on existirà l'agent de suport i dues plataformes remotes per on passarà l'agent mòbil. Tots els components de tolerància a fallades i prova de vida funcionant. (figura 3.1)

1. Missatge del Component de FaultTolerance: Em moc a <plataforma> [1]
2. Mètode de migració doMove() on el destinatari és l'Agent de Control. [1]
   (a) AMS de la plataforma remota gestiona la migració de l'Agent Mòbil.
   (b) S'envia l'agent serialitzat a l'Agent de Control a través del component CAgentResponder.
   (c) L'Agent de Control per mitjà d'aquest behaviour, emmagatzemarà en disc la instància de l'agent i el jar.
3. Migració de l'agent mòbil normal.

[1] Iniciat per l'agent mòbil, requereix confirmació de l'agent de suport, es farà checkpoint

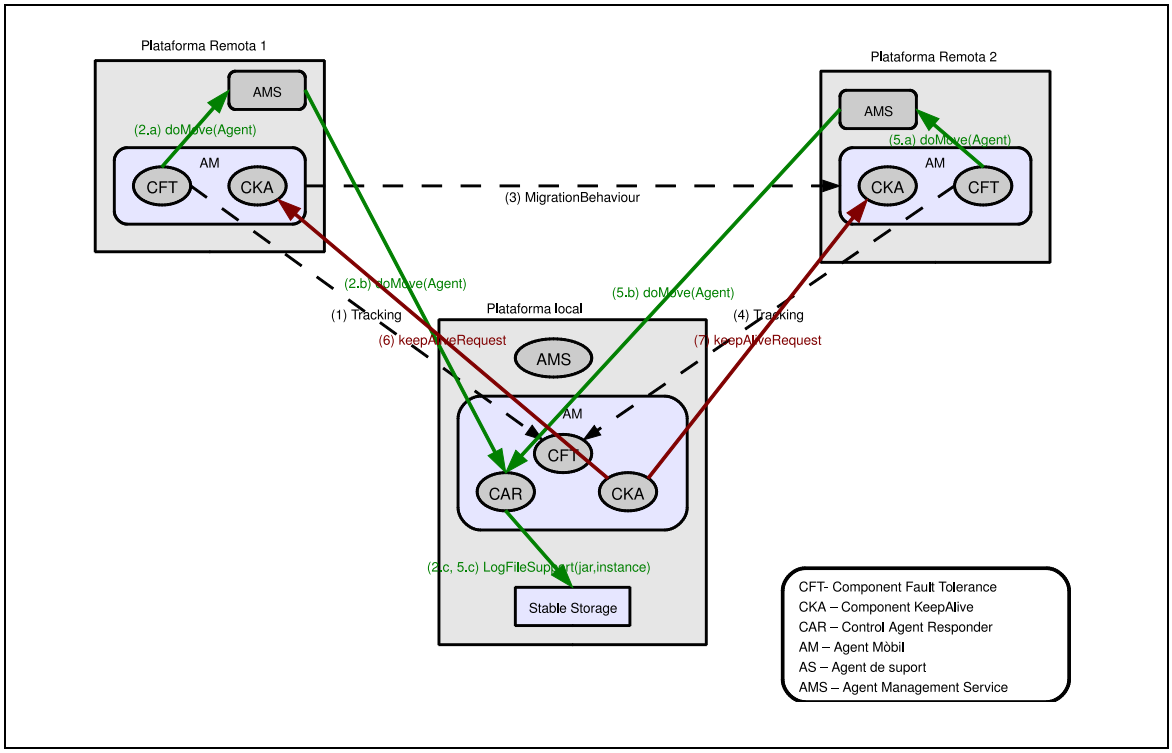


Figura 3.1: Arquitectura del sistema i disseny del protocol

4. Missatge del Component de FaultTolerance: Em moc a <plataforma> [1]

5. Mètode de migració doMove() on el destinatari és l'Agent de Control. [1]

   (a) AMS de la plataforma remota gestiona la congelació i serialització de l'Agent Mòbil

   (b) S'envia l'agent serialitzat a l'Agent de Control que capturarà el behaviour CAgentResponder.

   (c) L'Agent de Control per mitjà d'aquest behaviour, emmagatzemarà en disc la instància de l'agent i el jar.

6. Periòdicament, i en un behaviour concurrent, es procurarà contactar amb l'Agent Mòbil enviant un missatge REQUEST response [2]

[2]Iniciat per l'agent de suport, requereix confirmació

3.3.4 Detecció de fallades i recuperació

Ja introduït l'esquema de funcionament normal del model, amb els dos components treballant en paral·lel, anem a veure quin serà el comportament desitjat quan hi hagi algun problema, com hem de reaccionar. En la figura 3.2 veiem l'esquema de funcionament d'una recuperació a una fallada.

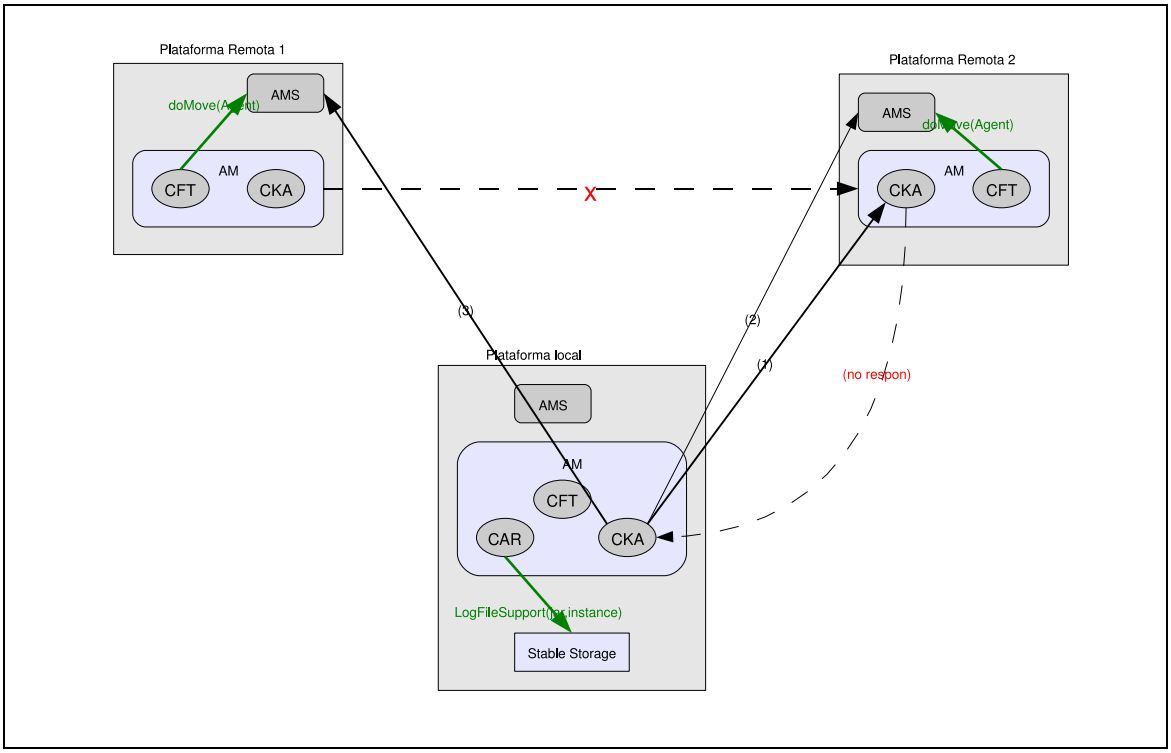


Figura 3.2: Esquema de funcionament d'una resposta a una fallada

1. L'agent de Control cada cert temps envia una petició de KeepAlive

   (a) Si l'Agent Mòbil no respon a aquesta petició, l'Agent de Control farà la petició nou cops més.

2. En cas de no rebre resposta, tractarà de contactar amb AMS per demanar si l'agent existeix.

   (a) Si no hi ha resposta de l'AMS donarem la plataforma remota per desapareguda.

i. Enviar missatge a l'administrador de la plataforma remota avisant que no funciona.

ii. Tornar a (1) i esperar que la plataforma contesti.

(b) Si hi ha resposta d'AMS, mitjançant FIPAManagementOntology consultarem a l'AMS remot si existeix l'Agent Mòbil.

(a) Si existeix donem l'ordre de matar l'agent que no contesta i recuperar el darrer agent copiat (Checkpoint): Migrar normalment

(b) Si no existeix, fem la consulta a l'AMS de la plataforma anterior per on ha passat l'agent. (Es pot recupertar gràcies al registre d'events o guardant les dues últimes plataformes visitades)

3.4 Viabilitat

En aquesta secció analitzarem la viabilitat del projecte. Determinarem si serà factible o no guiar a bon port aquesta aventura. Hem de veure si amb els recursos que tenim a disposició podem concloure'l amb èxit.

Viabilitat tècnica

L'arquitectura d'agents mòbils amb la qual tenim pensat fer el projecte i amb la qual s'està elaborant el macro-projecte [4] està desenvolupada utilitzant les llibreries de JAVA. Per tant, necessitarem el seu kit de desenvolupament de software (SDK). També haurem d'importar les eines per desenvolupar agents mòbils, el paquet de classes de JADE (Java Agent Development Framework), el component de mobilitat InterPlatform Mobility Project desenvolupat al departament DEIC (Departament d'Enginyeria de la Informació i les Comunicacions) i un entorn de desenvolupament del llenguatge Java com podria ser eclipse [13].

Viabilitat econòmica

Les eines de codi lliure actualment estan en estat de gràcia, amb força iniciatives i projectes que pretenen donar una ferma alternativa a les eines regides per

llicències i copyright.

Tant JAVA, com JADE són un conjunt de llibreries i eines desenvolupades amb aquest objectiu, on l'usuari té un entorn de treball complert per poder desenvolupar *software* a alt nivell.

També treballarem amb l'eina CAD eclipse, un entorn de treball amb interfície d'usuari intuïtiva i fàcil d'usar que ens permetrà desenvolupar sense problemes el codi font de la nostra aplicació.

Com que totes aquestes eines es regeixen per les normes del codi lliure, no suposaran un cost addicional al nostre projecte.

Viabilitat legal

El software utilitzat és de domini públic, per tant no tenim restriccions d'ús en cap cas. El desenvolupament del projecte es farà sota un entorn de desenvolupament tancat, per tant, tampoc haurà d'estar sotmès sota la llei de protecció de dades. No en farem cap ús comercial ni amb intencions de benefici econòmic, només és un projecte per completar el cicle acadèmic d'un servidor.

3.5 Conclusions

Hem analitzat els requisits que s'emmarquen en el desenvolupament d'un component de tolerància a fallades per a agents mòbils. Basant-nos en aquests requisits i abstraient algunes idees dels models comentats al capítol 2, hem dissenyat una possible solució a la problemàtica que se'ns presenta. Finalment hem fet un anàlisi de la viabilitat del mateix. Entrem ara en la fase de disseny del component.

Capítol 4

Disseny

En aquest capítol anirem dissenyant els diversos mòduls que necessitarem per poder dur a terme el projecte. D'aquesta manera aniran apareixent certes característiques i restriccions que aniran donant forma a la posterior implementació fins a obtenir una solució a la problemàtica que se'ns planteja.

També explicarem el funcionament del protocol que implementarem i que ens servirà de model per un sistema de tolerància a fallades real sobre un esquema d'agents mòbils.

Començarem la explicació a partir dels protagonistes d'aquest model, els agents. Farem una breu introducció a la ontologia de comunicació entre components *ComponentOnto*. Continuarem amb una descripció dels components, veient la interacció que hi ha entre ells i com serà la seva comunicació per entendre globalment com funciona l'esquema. Acabarem amb una breu explicació de l'estàndard d'administració de plataformes de FIPA Agent Management.

4.1 Agent mòbil

Aquest és l'actor principal del model. Està predestinat a migrar de plataforma en plataforma executant les tasques que té programades. Però perquè funcioni el model, s'haurà de comunicar amb l'agent de suport per anar informant de la seva activitat. La comunicació entre l'agent mòbil i el de suport és vital per tenir

un registre de l'activitat del primer guardada en emmagatzemament estable (sigui disc dur o qualsevol suport nomenat "segur").

Estructura:

Volem que l'agent s'executi sobre les plataformes de l'entorn de JADE, per tant, aquests hauran d'extendre la classe *Agent*. Per fer això haurem de sobreescriure el mètode setup d'aquesta classe que fa referència a la inicialització de l'agent.

El mateix passarà amb els components, aquests són comportaments que s'executaran amb un agent com a propietari, per tant, hauran d'extendre algun dels *Behaviours* que tenim a disposició. Haurem de sobreescriure el mètode action del component, per descriure el funcionament del mateix.

Hem comentat en la fase d'anàlisi que l'agent mòbil ha de contenir dades referents a l'agent de suport i a la seva ubicació. Les dades necessàries a l'agent mòbil seran:

L'agent necessitarà tenir la informació del destinatari dels missatges. Aquesta informació la obtindrà en la inicialització, abans del cicle d'execució de comportaments.

- *ownerAddress*: Aquest paràmetre és de tipus String i contindrà la informació d'on es troba l'agent de suport per poder contactar amb ell. Aquest paràmetre ha de ser obligatori per poder iniciar el protocol, ja que l'agent de suport no té altra manera de contactar amb l'agent mòbil si no és a partir del primer missatge que envii l'agent mòbil.

Com que estem especificant l'agent mòbil, aquest haurà de tenir una llista de plataformes que visitarà.

Nota: S'entén que un agent mòbil SEMPRE tindrà un component de migració per moure's a una altra agència.

- *itinerari*: Aquest paràmetre és de tipus vector de String i contindrà la llista de destins que visitarem. Serà necessari accedir a la llista cada cop que es cridi al component de migració. Aquesta llista la inicialitzarem en temps d'execució en el moment que inicialitzem l'agent.

Per definir les accions de l'agent mòbil utilitzarem una sèrie de comportaments. Aquests s'hauran d'executar en un ordre predeterminat, que especificarem utilitzant un CompositeBehaviour anomenat FSMBehaviour, que implementa una màquina d'estats finits on els subcomponents que hi afegim s'executaran en un ordre especificat. Els components que es vulguin afegir a més dels que implementen el protocol de tolerància a fallades seran decisió del programador.

S'ha de tenir em compte una excepció a la regla, que serà el component de prova de vida o *KeepAlive*. Aquest s'haurà d'executar a part de tots els altres components ja que no pot estar limitat al seu cicle d'execució. Per tant, l'haurem d'executar en un thread a part. Això ho farem amb un altre CompositeBehaviour anomenat ParallelBehaviour, que inicia un thread per cada subcomponent registrat. En resum, la situació quedarà de la següent manera:

Thread 1: Contindrà la màquina d'estats finits amb els components seqüencials.

Thread 2: Contindrà el component de prova de vida o *keepAlive*.

Comentem ara l'ordre d'execució dels components, ja que aquest serà important si volem que sigui útil el funcionament del protocol.

Un plantejament ideal de fiabilitat seria que just després de la execució de cada component hi hagués un emmagatzemament o *checkpoint* per assegurar que la última còpia de l'agent és la que conté la última tasca feta per l'agent, amb la penalització que això suposa. Però com a mínim s'hauria de planificar després de qualsevol tasca crítica que pugui executar l'agent (per exemple una transacció bancària o una compra), on es tingui constància de l'acció i no hi hagi inconsistència de dades. L'ideal seria fer una execució transaccional del cicle de comportaments de l'agent, però ja que ara per ara no està implementat, procurarem acostar-nos-hi el màxim possible fent que cada acció important o costosa per l'agent sigui guardada. De totes maneres això es deixarà com a decisió pel programador, ja que ha de ser ell el qui trii el millor esquema per preservar les dades contemplant el rendiment que li oferirà la seva configuració.

Posteriorment, l'*Agent Builder* s'encarregarà de muntar l'agent.

Aquests són els components que ha de tenir l'agent mòbil referents al model de tolerància a fallades.

- *FaultTolerance*: Aquest behaviour serà una instància de la classe FaultToleranceBeh. Aquest és l'encarregat de donar la funcionalitat de *tracking* i *checkpointing* a l'agent mòbil. El component en qüestió ha de rebre confirmació de les accions que demana per deixar continuar el cicle de vida de l'agent.

- *KeepAlive*: Aquest behaviour serà del tipus KeepAliveBehaviour. Aquest esperarà contínuament la petició del component de suport i informarà que aquest es troba actiu i funcionant.

Des del component de tolerància a fallades podem iniciar una migració de l'agent mòbil per fer un *checkpoint* cap a l'agent de suport i hem de saber on es troba aquest. Hem de tenir la manera de accedir a la informació referent a la posició.

- *getControlLocation*: Aquest mètode permet accedir a la informació referent a la plataforma on es troba l'agent de suport. Ens retornarà un valor de tipus String que podrem utilitzar per composar l'AID de l'agent destinatari del missatge ACL.

Prèviament a una migració, l'agent demanarà a l'agent de suport una petició de migració, informant de l'agència destí a la qual accedirà.

- *getNextHop*: Aquest mètode retorna la següent plataforma de l'itinerari, per tant, el valor de retorn serà de tipus String.

L'agent ha de permetre que tots els components que ho desitgin puguin enviar missatges cap a d'altres agents o components d'aquests. En el nostre cas, aquests destinataris seran l'agent de suport i AMS. Seguint el model proposat a [5] els propis components continguts en l'agent faran de filtre per saber si el missatge va destinat a ells o no. Per això hem d'habilitar un servei per poder enviar els

missatges i que pugui ésser recollits per cada component. Anem a parlar ara dels serveis que ha d'oferir l'agent per poder interactuar amb d'altres agents o amb els seus components.

- *sendMessage*: Aquest mètode rep un missatge ACL i el completa amb l'emissor i el receptor per despres enviar-lo. No retorna cap paràmetre.

Com hem comentat, l'ordre en que es trobin els components dins de la màquina d'estats finits és molt rellevant, ja que determina el correcte funcionament del protocol. El programador ha de conèixer quines tasques són realment importants en el cicle de l'agent i requereixen fer una còpia de l'estat del mateix.

Anem ara a parlar de l'agent de suport, l'encarregat de fer la feina complementària de l'agent mòbil. És de vital importància que aquest agent estigui operatiu i escoltant les peticions de l'agent mòbil, ja que altrament el primer no continuarà amb les seves tasques. Cal tenir en compte que el component de tolerància a fallades mòbil és bloquejant i demanarà confirmació per part del de suport per continuar l'execució dels següents components de l'agent.

4.2 Agent de suport

Aquest agent restarà en una plataforma concreta, sense migrar ni exercir diferents tasques com faria l'agent mòbil, per tant, només tindrà un cicle de comportaments que s'executaran concurrentment fins que la feina de l'agent mòbil hagi finalitzat.

La seva implementació estendrà la classe Agent. Aquí anirem afegint els components que requerim per completar les tasques de l'agent mòbil.

Executar l'agent de suport en una plataforma fixada i controlada per l'usuari, facilitarà el contacte de l'agent mòbil amb ell. A part, ens evitarem problemes de permisos quan tractem d'accedir a disc per fer l'emmagatzemament de les dades en lloc estable.

No volem limitar l'execució de l'agent mòbil a una plataforma predefinida. Per tant, la informació referent a aquest no haurà de ser inicialitzada sinó rebuda a través del primer missatge, amb el camp *sender* del missatge. És a dir, haurà

de ser l'agent mòbil qui doni aquesta informació per que l'agent de suport pugui començar les seves tasques actives.

Anem a parlar de les dades que ha de contenir aquest agent per poder oferir el servei de tolerància a fallades.

Estructura:

L'agent de suport ha d'oferir les funcionalitats que complementin els serveis que contingui l'agent mòbil. En el cas del component que estem desenvolupant, aquest haurà d'exercir dues funcions separades en el rol d'agent de suport. La primera serà passiva, esperant la recepció de la informació provinent de l'agent mòbil i enviant la confirmació de recepció. La segona serà activa, on l'agent de suport reclamarà contestació per part de l'agent itinerant per comprovar que encara està aquest operatiu.

Per poder implementar aquesta segona funcionalitat, l'agent de suport ha de conèixer en tot moment la posició de l'agent mòbil, i això ho aconseguirà a partir de la primera. Per tant, com que aquests dos comportaments es tractaran per separat, hauran de tenir un contenidor de la informació que serà la classe agent de suport. En aquesta classe tindrem la informació referent a l'agent mòbil, i a la plataforma on es troba aquest.

- *mobileAgentID*: Aquest atribut serà de tipus AID (*Agent IDentifier*) i contindrà la informació referent a l'agent mòbil i a la plataforma on es troba actualment. Aquest atribut ens permetrà tractar de contactar amb ell en el sondeig de prova de vida. Si l'agent mòbil deixa de respondre, tractarem de contactar amb l'AMS d'aquesta plataforma per comprovar que no s'ha quedat la instància en estat inconsistent, és a dir, que l'agent estigui registrat en la plataforma però sense que respongui a estímuls.

L'agent de suport ha d'oferir tasques de recuperació en cas que l'agent mòbil no respongui. Per tant, ha de emmagatzemar en un lloc estable i segur tota la informació que rebi de l'agent mòbil. Com que ens trobem en un entorn controlat, podrem disposar de serveis de disc o altres tipus de dispositius. Per tant, haurem

de guardar la ruta on s'emmagatzemarà la informació rebuda. Aquesta informació la tindrem pel servei de registre d'events de l'agent mòbil i per la informació que rebrem de l'agent quan es demani fer un *checkpoint*.

- *fitxerLog*: En aquest atribut hi trobarem la ruta relativa i el nom de l'arxiu on bolcarem els events de l'agent mòbil per tenir un registre de les accions del mateix. Aquest serà de tipus String.

- *fitxerAgent*: En aquest atribut hi trobarem la ruta relativa i el nom de l'arxiu on bolcarem el codi de l'agent. Aquest serà de tipus String.

- *prevAgentID*: Aquest camp serà de tipus AID i contindrà la informació referent a l'anterior plataforma visitada per l'agent mòbil. El component de prova de vida utilitzarà aquest atribut per tractar de contactar amb la plataforma anterior quan la plataforma marcada per l'atribut *mobileAgentID* no respongui. El contacte amb aquesta plataforma permetrà assegurar que l'agent no es troba en ella degut a una fallada en la migració

- *lastLog*: En aquest atribut de tipus AID tindrem la informació referent a la última plataforma en la qual hem fet un *checkpoint* de l'agent mòbil. Aquest serà el punt de recuperació més actualitzat que tindrem. La implementació del model permet només un checkpoint, emmagatzemant la última còpia bona en l'arxiu determinat per l'atribut *fitxerAgent*, per tant, aquesta còpia serà de la qual s'extraurà la recuperació de l'agent.

Per oferir un cert nivell de seguretat, utilitzarem un sistema de clau pública que ens permetrà autentificar l'agent de suport davant de l'agent mòbil, mitjançant la signatura del primer en els missatges del mateix. Aquest servei que ofereix l'agent és una primera aproximació a comunicacions segures entre aquest i l'agent mòbil.

- *keyPair*: Atribut contenidor d'un parell de claus que permetran autentificar l'agent de suport envers l'agent mòbil quan el primer es comuniqui.

A part dels atributs que necessitem mantenir, hem d'utilitzar una llista de comportaments o behaviours. Com que es tracta de components de l'agent de

suport, aquests s'hauran d'executar de forma concurrent per poder atendre independentment les peticions de l'agent mòbil. Per aconseguir això, utilitzarem un *ParallelBehaviour* que ens permetrà disparar un thread per cada subbehaviour que registrem. Aquests subbehaviours són els següents:

- *faultTolerance*: Aquest component instància la classe FaultToleranceCBeh, que ens permetrà atendre a les peticions del component mòbil.
- *keepAlive*: Aquest component del tipus KeepAliveCBehaviour ens donarà totes les funcionalitats necessàries per comprovar l'estat de l'agent mòbil i actuar en conseqüència.
- *controlAgentResponder*: Aquest component ens permetrà rebre el *checkpoint* de l'agent mòbil i guardar-lo en emmagatzemament estable.

L'agent de suport s'ha de comunicar amb l'agent mòbil i també pot iniciar les converses amb aquest, per tant, ha de tenir un servei d'enviament de missatges que reculli tota la informació referent a ell per poder-li enviar els missatges. Aquesta informació la recuperarà de les dades emmagatzemades pel component de tolerància a fallades que cada cop que rep informació, n'emmagatzema l'emissor del missatge.

- *sendMessage*: Aquest mètode rep un missatge ACL i el completa amb l'emissor i el receptor per despres enviar-lo. No retorna cap paràmetre.

Previ a parlar de l'esquema de tolerància a fallades, anem a comentar breument l'acte comunicatiu entre els components, d'aquesta manera entendrem millor el disseny dels mateixos.

4.3 La ontologia ComponentOnto

La comunicació entre agents està regulada pels estàndards de FIPA. Els missatges ACL (*Agent Communication Language*) permeten la comunicació entre els

agents de JADE i incorporen aquests estàndards. Els missatges ACL estan composats per *performatives*, és a dir, voluntats a l'hora d'enviar els missatges. Les ontologies, per la seva banda, es reparteixen en predicats, conceptes i accions. Aquestes definiran el tipus de comunicació entre agents amb la ontologia que s'estigui comunicant. Podriem dir que una ontologia defineix la estructura del missatge que s'enviarà, però és més que això. També podem demanar accions a agents remotament.

Aquesta ontologia, que es pot consultar a [5] pretén ser una ontologia genèrica que defineix un acte comunicatiu entre components. Aquesta ens anirà bé a l'hora d'implementar la comunicació entre agents ja que ens dóna els elements necessaris per a tal. D'ella utilitzarem el concepte *ObjectContainer*, que serà el contenidor de la informació signada de l'agent, amb una petita variació, afegirem un atribut on hi posarem la clau pública de l'agent de suport. Finalment la classe quedarà com en la figura 4.1.

També utilitzarem el concepte MessageContainer que ens servirà per transportar el missatges que contindran informació referent al nostre protocol, tals com l'acció que farà l'agent mòbil i la localització del mateix.

El concepte ComponentDescription ens definirà cap a quin component va destinat el missatge i quina versió de missatge estem enviant.

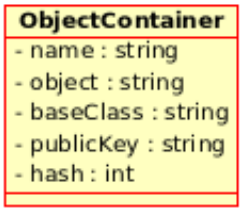


Figura 4.1: ObjectContainer

Aquesta ontologia ens definirà l'acte comunicatiu entre els components del nostre model, els components de tolerància a fallades i de prova de vida. Ara ja podem parlar del model de tolerància a fallades.

Anem a parlar del model de tolerància a fallades. Per parlar-ne haurem de parlar dels components que l'implementen, aquests són els que realment construiran l'esquema, i li donaran sentit.

Suposem que ja haurà quedat clar que tenim dos comportaments separats en aquest model. El primer és on l'iniciador de les comunicacions és l'agent mòbil, i composa totes les accions empreses pel component de tolerància a fallades. El segon és on l'iniciador és l'agent de suport, i les accions les empren el component de prova de vida. Els dos s'executaran en paral·lel, i compartiran informació.

Començarem parlant de les accions del component de tolerància a fallades. En aquestes accions prima la informació. Aquest component s'encarregarà de mantenir l'agent de suport informat perquè aquest ho comuniqui a l'usuari o perquè emmagatzemi aquesta informació en disc.

4.4 Component de tolerància a fallades

Aquest component és vital per un correcte funcionament de l'esquema de tolerància a fallades, ja que gràcies a ell l'agent de suport coneixerà la posició de l'agent mòbil. Per tant, aquest component s'haurà d'executar quan abans millor en arribar a una plataforma, sobretot si les tasques que ha de fer l'agent en aquesta requereixen força temps.

En la figura 4.2 podem veure l'esquema de comunicació del protocol, que a continuació explicarem en detall.

FaultToleranceBeh <-> FaultToleranceCBeh

Aquest component compost efectuarà un acte comunicatiu entre les dues parts. La primera part, com ja hem comentat, la iniciarà el component mòbil, el qual enviarà un missatge *INFORM* al component de suport (pas 3 de la figura 4.2), demanant migrar a una plataforma.

Aquest missatge serà rebut per la contra part de suport, que extraurà el missatge, capturarà la informació de l'emissor i registrarà un event amb aquesta informació.

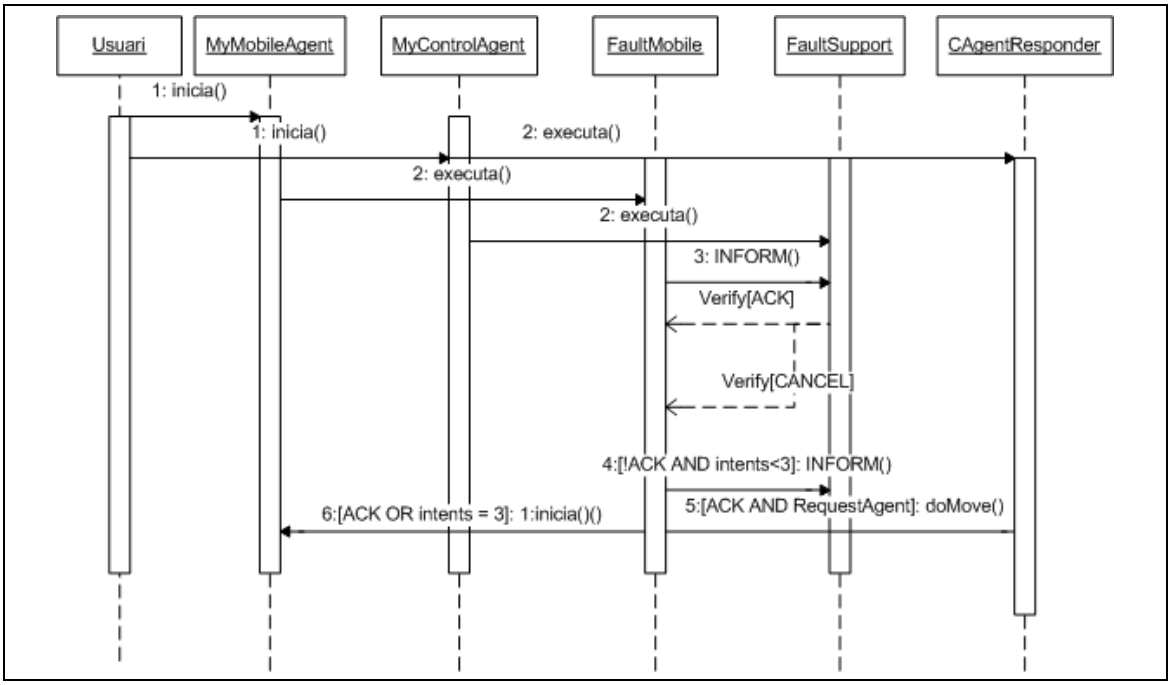


Figura 4.2: Diagrama de seqüència del component de tolerància a fallades

Acte seguit agafarà aquest missatge i el signarà amb la seva clau privada afegint la seva clau pública al missatge[1]. Crearem un missatge *INFORM* de resposta amb aquesta informació i li retornarem a l'agent mòbil.

Aquest rebrà la resposta i verificarà que l'emissor del missatge és l'agent de suport i que el missatge retornat és correcte. Quan hagi completat les verificacions, procedirà a finalitzar el component i passarà a la següent acció.

FaultToleranceBeh <-> CAgentResponder

Tenim també la possibilitat de fer un *checkpoint* de l'estat de l'agent (pas 5 de la figura 4.2). Aquesta opció serà executada si el component té activat l'atribut *agentRequest* a **cert**. Just després de les accions de tracking es comprovarà aquest valor, i si està habilitat, es procedirà al *checkpoint*. Aquest serà una migració normal amb destinatari l'agent de suport i no l'AMS de la plataforma destí com

[1]Som conscients que aquesta pràctica no és del tot segura, ja que no garanteix que sigui l'agent qui envia el missatge. Versions futures haurien d'implementar la seguretat amb un servidor criptogràfic

normalment. Això vol dir que la complexitat regeix en la part estàtica del model.

L'agent de suport tindrà dins de la llista de components en execució una modificació de la classe AMSResponder [11]: La classe CAgentResponder. Aquesta classe estén la classe SimpleAchieveREResponder, i implementa el protocol de migració Inter Plataforma [7] per poder tractar el *checkpoint* de l'agent mòbil com una migració. En aquest cas concret, el component CAgentResponder esperarà a que l'iniciador comenci la transferència de l'agent, quan ho faci, emmagatzemarem la instància i el codi de l'agent comprimit *jar* en disc. Per completar el *checkpoint* de l'agent mòbil haurem d'impedir que aquest es registri i s'inicii en la plataforma de l'agent de suport, evitant que es crei i que es faci l'arranc del mateix *(PowerUp)*.

Actualment el component de tolerància a fallades te habilitada la funció de *checkpoint* per fer una còpia de l'agent en emmagatzemament estable. Però com que per la part que inicia l'acció (l'agent mòbil), el *checkpoint* no és més que una migració cap a l'agent de suport, aquest es pot arribar a fer des de qualsevol component cridant el mètode *myAgent.doMove()* amb destí aquest agent.

4.5 Component de prova de vida

Aquesta part del model de tolerància a fallades que estem desenvolupant, fa referència a totes les accions iniciades per l'agent de suport, i que donen com a resultat la verificació de la disponibilitat de l'agent mòbil. Aquest també estarà dividit en dues parts: la mòbil i la de suport.

KeepAliveCBehaviour <-> KeepAliveBehaviour

La comunicació entre les dues parts del component de prova de vida es fa periòdicament mentre l'estat de la comunicació sigui satisfactori. En cas contrari, mirarem d'avaluar la situació i actuar en conseqüència.

En la figura 4.3 tenim la seqüència de passos del mateix.

Per comprovar l'estat de vida de l'agent mòbil, prèviament el component de tolerància a fallades haurà hagut de registrar la localització d'aquest. Quan tin-

guem aquesta informació disponible iniciarem una petició de *keep alive* des de l'agent de suport. Aquesta petició contindrà un missatge de tipus *REQUEST* cap a l'agent mòbil (pas 3 de la figura 4.3).

El component de prova de vida de l'agent mòbil s'executa en un thread diferent de la resta de components precisament perquè estarà escoltant la petició de la seva contra part de suport. Quan aquesta arribi, la capturarà, i crearà un missatge *AGREE* de resposta. Acte seguit l'agent de suport rebrà aquest missatge i procedirà al cicle normal de funcionament, és a dir, esperarem un temps prudencial i tornarem a fer la petició.

Iniciarem les tasques de recuperació de l'agent mòbil quan aquest no respongui als intents de *REQUEST* de l'agent de suport.

Els motius pels quals l'agent mòbil hagi pogut tenir problemes es descriuen a continuació:

- *Fallada de la plataforma*: Per motius de falta de memòria, parada de hardware o problema en temps d'execució degut a un bug.

- *Fallada de l'agent en la plataforma*: Per motius de falta de memòria, o errors en la programació de l'agent que fan que aquest no respongui i es quedi zombi o en un estat inconsistent però registrat en la agència.

- *Fallada de l'agent en la migració*: Falla la migració o es completa, però a l'hora de aixecar l'agent, és a dir, quan fem un *power up* aquest no respon satisfactòriament.

- *Fallada de component de xarxa*: Degut a la manca de servei d'un component de xarxa, els agents no poden contactar momentàniament.

El primer pas serà tractar de comunicar-se amb la plataforma (passos 5, 6, 7 i 8 de la figura 4.3) des d'on es va posar en contacte per últim cop l'agent. Aquesta plataforma pot o no estar viva, per tant, si rebem resposta haurem de mirar si l'agent està registrat. Si ho està mirarem de desregistrar-lo fent la petició a l'AMS de la plataforma destí. Si no rebem resposta mirarem de contactar amb la plataforma

anterior per veure si aquesta es troba disponible. Seguirem el mateix procediment amb aquesta, és a dir, tractarem de contactar amb l'AMS de la plataforma i si rebem resposta mirarem si l'agent mòbil està registrat. Quan ens hàgim assegurat que l'agent mòbil no està en circulació, mirarem si tenim un *checkpoint* i tractarem d'enviar-lo a la plataforma on està registrat l'últim backup de l'agent.

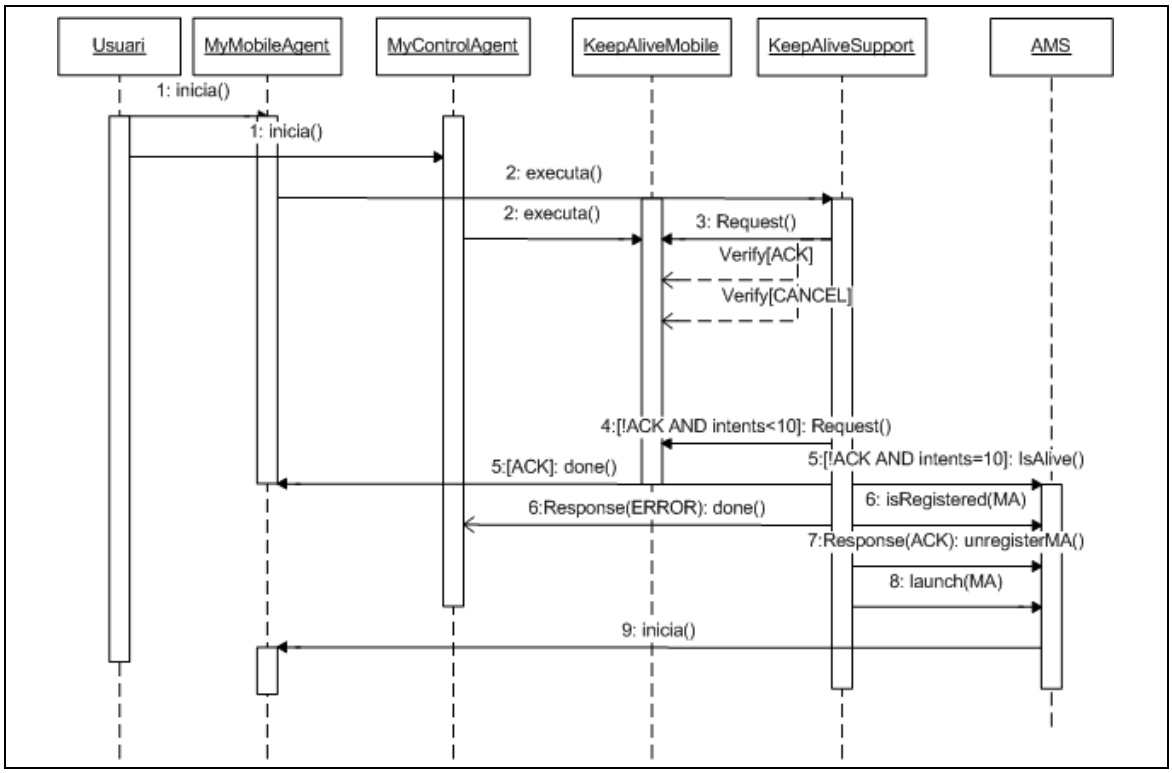


Figura 4.3: Diagrama de seqüència del component de prova de vida

En la figura 4.4 podem veure la màquina d'estats del component de prova de vida. Els estats en gris són els de l'agent de suport i els grocs són de l'agent mòbil. Podem veure dos comportaments separats, dels estats 0 al 3 són el funcionament normal quan l'agent mòbil està viu i contestant a les peticions de l'agent de suport. Els estats 5 a 9 son els estats de recuperació.

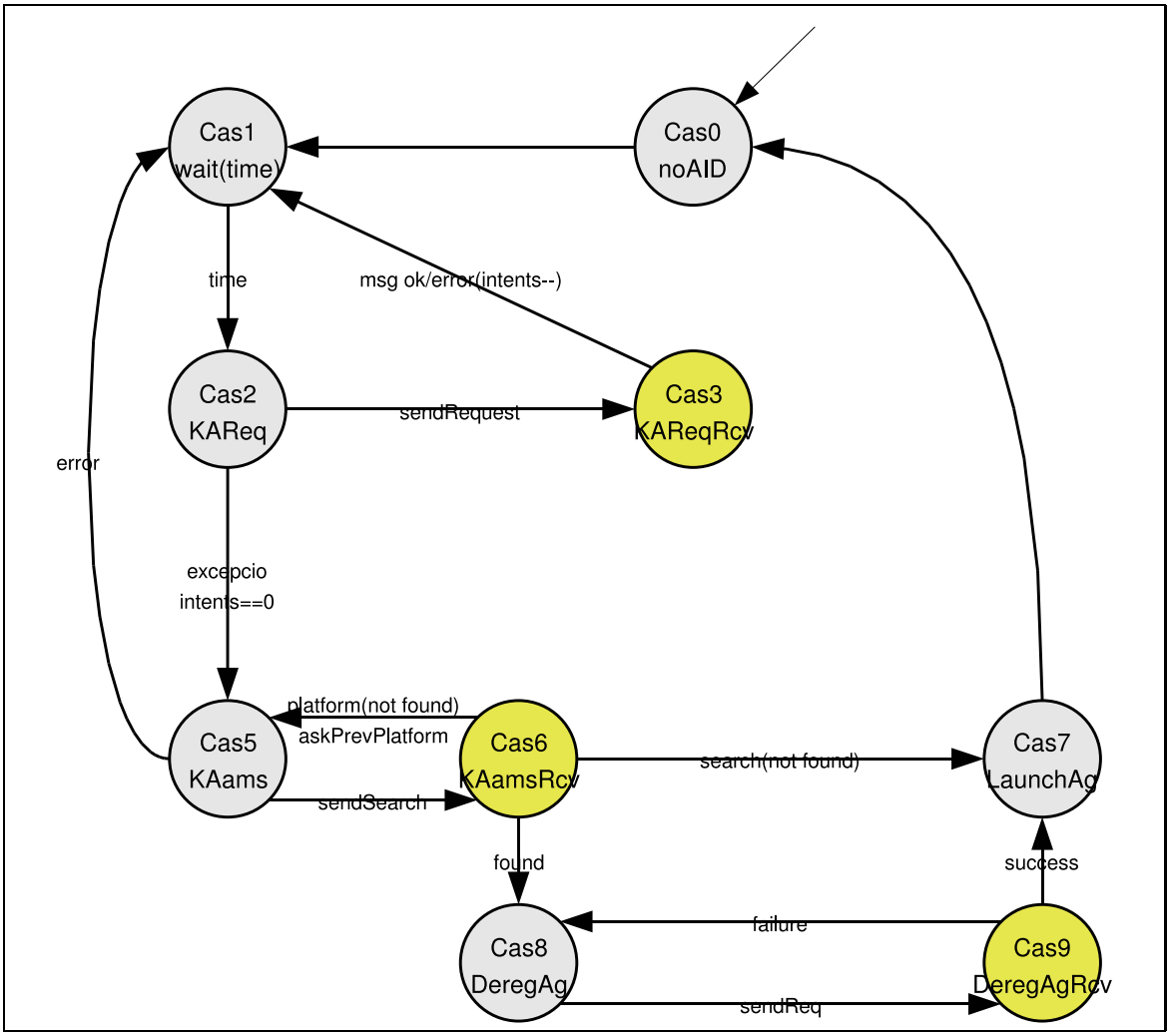


Figura 4.4: Màquina d'estats del component de prova de vida

Cas 0: Estat inicial, mentre no tinguem informació de la localització de l'agent mòbil estarem en aquest estat.

Cas 1: Esperem un temps per no bombardejar de peticions la xarxa.

Cas 2: Fem la petició de prova de vida i ens posem esperant resposta.

Cas 3: Rebem resposta i tornem a l'estat 1. Si no rebem resposta després d'un temps, decrementem el contador d'intents.

Cas 5: Si no rebem resposta en un número d'intents, iniciem tasques de recuperació.

Cas 6: Contacte amb l'AMS de la plataforma remota, si trobem l'agent registrat intentem desregistrar-lo, si no podem anem l'estat 7.

Cas 7: Llancem la còpia local que teníem de l'agent.

Cas 8: Cas que trobem l'agent registrat, llancem una petició de desregistrar l'agent.

Cas 9: Si conseguim desregistrar-lo anem a l'estat 7 per llançar la còpia de l'agent, sinó tornem a provar de desregistrar-lo.

Però, i com ho fem des de la nostra plataforma local per buscar i desregistrar un agent que es troba en una plataforma remota? No ho podem fer a través de mètodes de la classe AMSService, ja que aquests serveixen per accedir a un AMS que es troba en la mateixa plataforma que l'agent que està fent la petició. Però la idea és la mateixa, encara que ho farem mitjançant missatges ACL i:

La ontologia FIPAManagementOntology

Aquesta ontologia, definida dins del document FIPA Agent Management Specifications i implementada a JADE dins del paquet FIPAAgentManagement, conté les especificacions per interactuar remotament amb els serveis de plataforma, el DF (*Directory Facilitator*), que no entrarem a comentar, i l'AMS (*Agent Management System*) el qual necessitarem contactar per demanar accions sobre l'agent.

Aquesta ontologia modela una sèrie d'accions sobre l'AMS per poder registrar, desregistrar, localitzar, comunicar, migrar o crear agents.

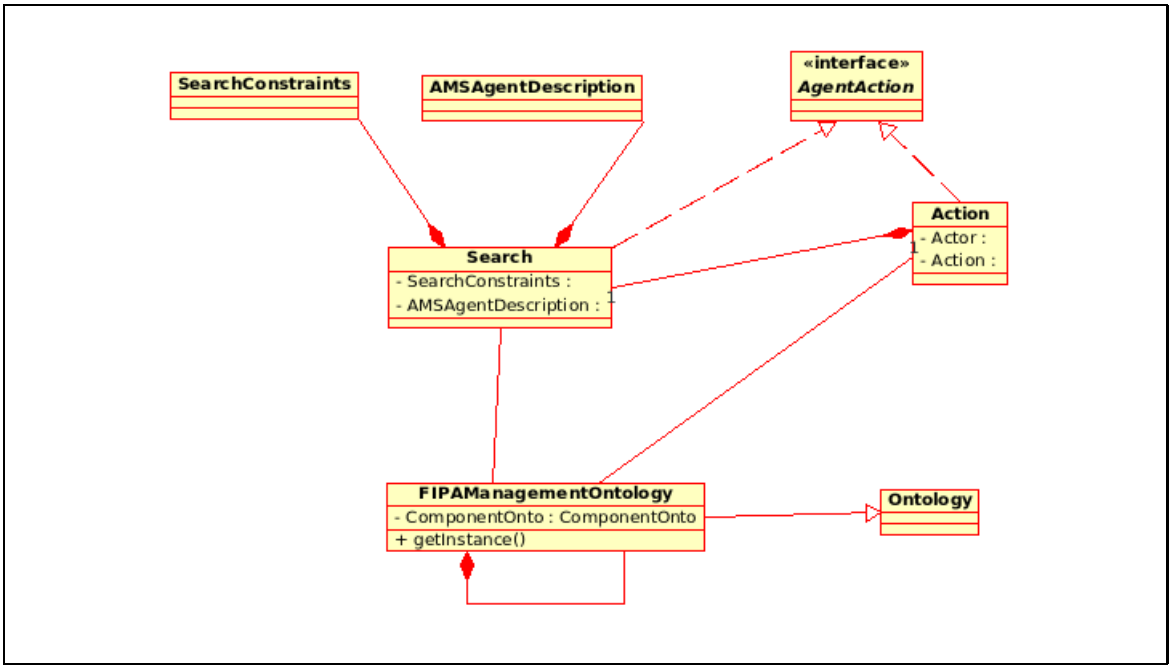


Figura 4.5: Diagrama de classes de l'ontologia FIPAManagementOntology amb l'Action Search

Per al nostre protocol necessitarem localitzar i desregistrar un agent en una plataforma remota, per tant, necessitarem utilitzar les accions *search* (figura 4.6) i *deregister* (figura 4.8).

El missatge ACL s'ha d'inicialitzar amb una sèrie de paràmetres, definits en les especificacions de FIPA, per configurar la comunicació utilitzant la ontologia *FIPAManagementOntology* que detallem a continuació.

En primer lloc, la performativa del missatge a enviar ha de ser un *REQUEST*. Com que aquesta ontologia està especificada seguint els estàndards de FIPA, el llenguatge de contingut serà *SL0* (*FIPA Semantic Language content language*) i el protocol de comunicació serà *FIPA REQUEST* (*fipa-request interaction protocol*).

En la figura 4.5 tenim l'esquema que farem servir per l'acció search.

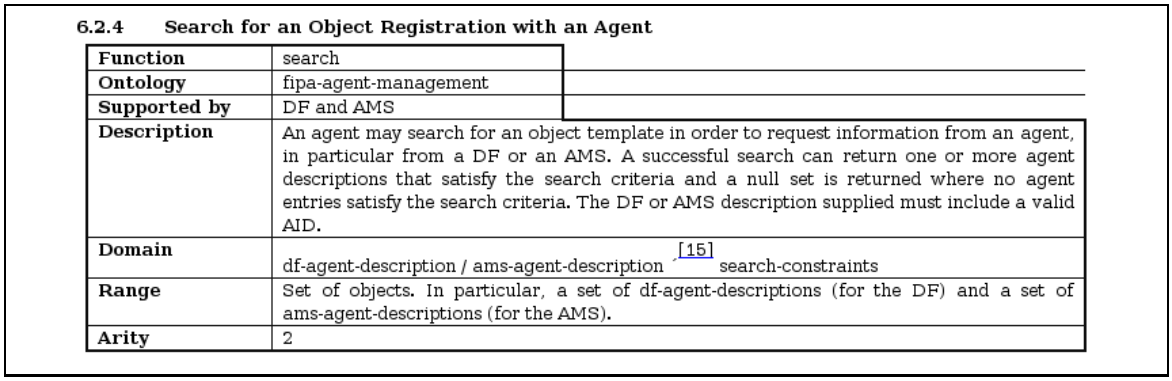
Dins del missatge haurem d'especificar la acció a fer, aquesta contindrà el propietari de la acció que serà l'agent de suport, i el tipus de la acció.

L'acció search anirà definida amb els SearchConstraints (regles de cerca) configurat per que retorni tots els resultats obtinguts amb la cerca. L'atribut AMSA-

gentDescription el deixarem buit, serà l'AMS remot qui ens retornarà una llista amb el resultat de la cerca en aquest objecte.

Després d'enviar el missatge rebrem la resposta. Si tot ha anat bé tindrem una llista d'AMSAgentDescription amb tots els agents registrats en la plataforma remota, sinó rebrem un missatge de *REFUSE*.

Després, per comprovar si l'agent està registrat en la plataforma en qüestió, haurem de consultar la llista i comprovar si hi és o no.



6.2.4 Search for an Object Registration with an Agent

Function	search
Ontology	fipa-agent-management
Supported by	DF and AMS
Description	An agent may search for an object template in order to request information from an agent, in particular from a DF or an AMS. A successful search can return one or more agent descriptions that satisfy the search criteria and a null set is returned where no agent entries satisfy the search criteria. The DF or AMS description supplied must include a valid AID.
Domain	df-agent-description / ams-agent-description [15] search-constraints
Range	Set of objects. In particular, a set of df-agent-descriptions (for the DF) and a set of ams-agent-descriptions (for the AMS).
Arity	2

Figura 4.6: Especificacions de l'acció Search

Si trobem l'agent mòbil registrat en aquesta llista i no ens respon a les peticions de prova de vida, possiblement s'hagi quedat en un estat de zombi, per tant el millor que podem fer és desregistrar-lo per que no es recuperarà. Després el tornarem a llançar des de la plataforma on tenim l'últim *checkpoint*.

Per inicialitzar el missatge ACL ho farem de la mateixa manera que ho hem fet amb l'acció search, el llenguatge serà *SL*, el protocol serà *FIPA REQUEST*, la performativa del missatge ACL serà *REQUEST*. Per desregistrar l'agent primer haurem d'introduir la AMSAgentDescription, és a dir, la descripció de l'agent que voldrem registrar. Recuperarem aquesta descripció fàcilment amb mètodes que ens ofereix l'agent de suport per recuperar la informació de l'agent mòbil. Un cop introduïda aquesta informació, només hem de construir l'acció Deregister amb aquesta descripció i posar-la dins d'un objecte Action. Aquest l'associarem al missatge ACL que té registrada la ontologia, el protocol i el llenguatge, i ho enviarà a l'AMS en qüestió. Com que ens estem comunicant mitjançant un *FI-*

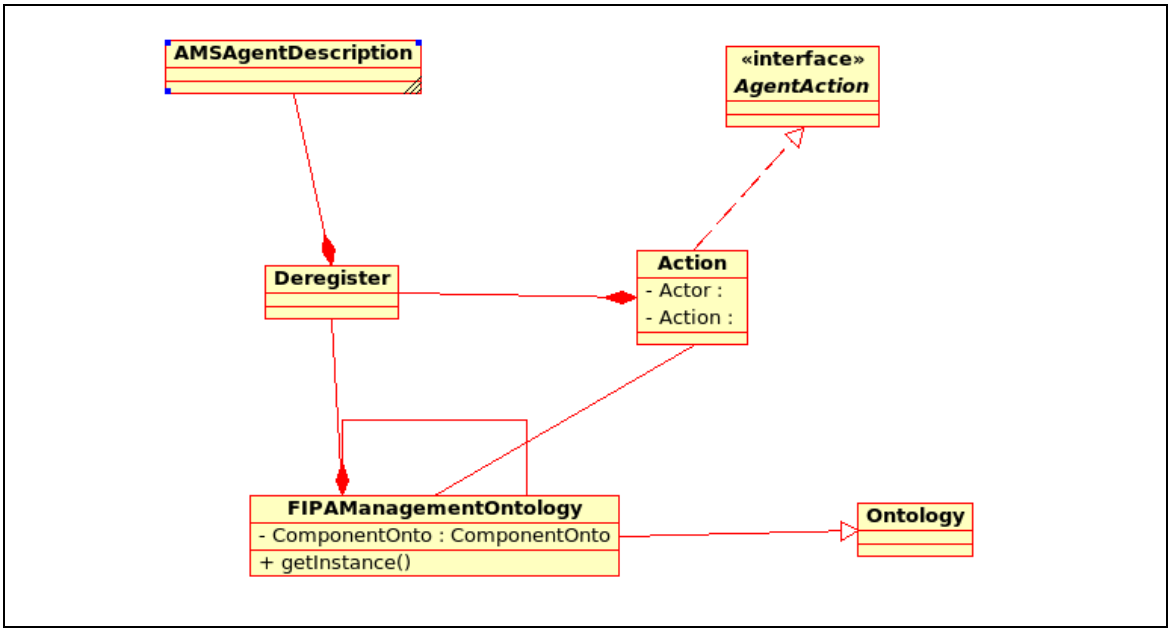


Figura 4.7: Diagrama de classes de l'ontologia FIPAManagementOntology amb l'Action Deregister

PA Request Interaction Protocol, la resposta que ens donarà l'AMS remot serà *AGREE* o *REFUSE*.

6.2.2 Deregistration of an Object with an Agent

Function	deregister
Ontology	fipa-agent-management
Supported by	DF and AMS
Description	An agent may deregister an object in order to remove all of its parameters from a directory. The DF or AMS description supplied must include a valid AID.
Domain	df-agent-description / ams-agent-description
Range	The execution of this function results in a change of the state, but it has no explicit result. Therefore there is no range set.
Arity	1

Figura 4.8: Especificacions de l'acció Deregister

4.6 Conclusions

Doncs fins aquí hem explicat el funcionament del model de tolerància a fallades, de prova de vida i hem detallat com s'ha solucionat el problema de tractar amb

un agent d'administració remotament. En el pròxim capítol us ensenyarem detalls d'implementació, els resultats obtinguts i sobre quin entorn s'han fet les proves.

Capítol 5

Implementació i proves

En aquest capítol explicarem l'entorn de treball sobre el que treballarem, quins elements són necessaris per executar els agents i com es comporten els elements desenvolupats.

5.1 Implementació

En aquesta secció comentarem alguns detalls d'implementació i certs problemes que mereixen ser mencionats.

El nostre agent de suport, per tramitar les migracions de l'agent mòbil cap a emmagatzemament estable hauria d'haver utilitzat la versió en desenvolupament 1.101 del servei de migració *IPMS*, però, degut a que la complexitat a l'hora d'adaptar-ho a les nostres necessitats ens estava ocupant massa temps, vam desistir, optant per la versió estable 1.2, molt més senzilla, de la qual, com ja hem comentat hem utilitzat per atendre les peticions de migració de l'agent mòbil.

La implementació inicial del model executava tots els components (el component de tolerància a fallades i el de prova de vida) en un mateix FSMBehaviour, però vam veure que sincronitzar l'enviament i rebuda dels missatges que s'enviaven era força difícil, més tenint el component de tolerància a fallades que és bloquejant en l'agent mòbil. Finalment vam decidir posar-los en threads separats.

La recepció bloquejant de missatges ACL (*myAgent.blockingReceive()*) atura completament l'execució de l'agent posant-lo en la llista de suspesos fins que transcorre el temps marcat (si n'hi hem posat) o es rep un missatge ACL. En un primer moment, el component de prova de vida de l'agent de suport estenia la classe OneShotBehaviour[1] i s'executava dins d'un TickerBehaviour (Behaviour que s'executa cada *timeout*). Aquest comportament no ens interessa perquè quan un dels components espera la resposta a una petició durant un cert temps, la resta de threads de l'agent s'aturen. Posem per exemple que el component de prova de vida de l'agent mòbil ha de rebre una petició, aquest bloquejarà tot el FSMBehaviour (Màquina d'estats finits de components amb transicions especificades) que conté la resta de components. La solució que hem implementat implica que tots els components estenen la classe SimpleBehaviour (Behaviour que s'executa cíclicament fins que *done()=false*) i tenen format de màquina d'estats finits. D'aquesta manera, en la execució del component, podem mantenir-nos en un estat concret d'espera de recepció de fins que no rebem el missatge ACL en qüestió.

Ens hem trobat un error de codificació en JADE. Un error que estava en la versió 3.5 i que perdura en la versió 3.6. Aquest error fa referència a l'enviament de missatges ACL entre dos agents. Posem pel cas que un agent mòbil i un agent de suport es creen en la mateixa plataforma, cadascun amb un nom diferent. El pas següent és que l'agent mòbil migri a una altra plataforma i just després envii un missatge ACL informant que ja ha arribat, esperant confirmació. Quan l'agent de suport rep el missatge li vol respondre amb un *acknowledge*. El problema ve a continuació: degut a que l'AID (Agent Identfier) de l'agent mòbil conté el mateix nom de plataforma que el remitent del missatge, l'AMS d'aquesta plataforma no trobarà l'agent mòbil perque no mirarà l'adreça de transport del receptor, sinó que només buscarà dins de la seva plataforma si hi ha algú registrat amb aquest nom. Com que l'agent mòbil ja no hi és, no el trobarà i donara error de temps d'espera esgotat. S'ha informat d'aquest error a Giovanni Caire encarregat del suport de JADE i han promès arreglar-ho en la pròxima versió.

[1] Behaviour que s'executa només un cop.

5.2 Eines de treball

Per desenvolupar aquest projecte hem utilitzat les eines de desenvolupament Eclipse SDK (*Software Development Kit*). Els diagrames de classe han estat dissenyats amb Umbrello UML Modeller i els diagrames de seqüència han estat fets amb Microsoft Visio 2007. Els esquemes de funcionament del protocol com la màquina d'estats finits del component de prova de vida han estat fets amb OpenOffice-Draw. La resta de imatges han estat extretes dels documents dels seus autors als quals es fa referència en la bibliografia.

5.3 Entorn de treball

Ens disposem a fer les proves pertinents de funcionament dels components desenvolupats, però abans, explicarem la situació real sobre la qual treballarem.

Les proves de funcionament s'han realitzat en un entorn tancat amb màquines virtuals on tenim:

Ubuntu Desktop: ENTAROADUN : Host vmware, 1Gb de RAM a 667MHz, processador Intel Core Duo 1.66GHz 2MB Cache, disc IDE 5200rpm, IP 192.168.31.1

Ubuntu Server: ubuntu-server : màquina virtual, 128Mb de RAM dedicada, 1 Processador, IP 192.168.31.129

Ubuntu Server: ubuSvr2 : màquina virtual, 128Mb de RAM dedicada, 1 Processador, IP 192.168.31.130

Per poder executar el component necessitem:

- un agent de suport que iniciarem en una agència fixe. (**ENTAROADUN**)
- un agent mòbil iniciat en una plataforma qualsevol (**ubuntu-server**). Aquest sabrà on es troba l'agent de suport per poder iniciar les comunicacions i tindrà un itinerari marcat en un arxiu de configuració.

- plataformes de l'itinerari (**ubuSvr2**), únicament tindran les agències arrancades amb un contenidor actiu (*Main-Container*)

5.4 Proves

Sobre aquestes màquines anem ara a executar les proves i veure si, després d'analitzar, dissenyar i implementar el model de tolerància a fallades, aquest realment funciona i es comporta com desitgem.

En la figura 5.1 podem veure la recepció de missatges de l'agent de control. Cada cop que rep un missatge de *FaultTolerance*, escriurà un event a l'arxiu de logs. El mateix farà el component de *KeepAlive* cada cop que fa una petició i rep la resposta. Per la seva banda el component de *KeepAlive* pot ser que en alguna situació no rebi resposta, degut a que el component llança la petició just quan l'agent mòbil està migrant o encara no hem rebut el *tracking* de la nova localització.

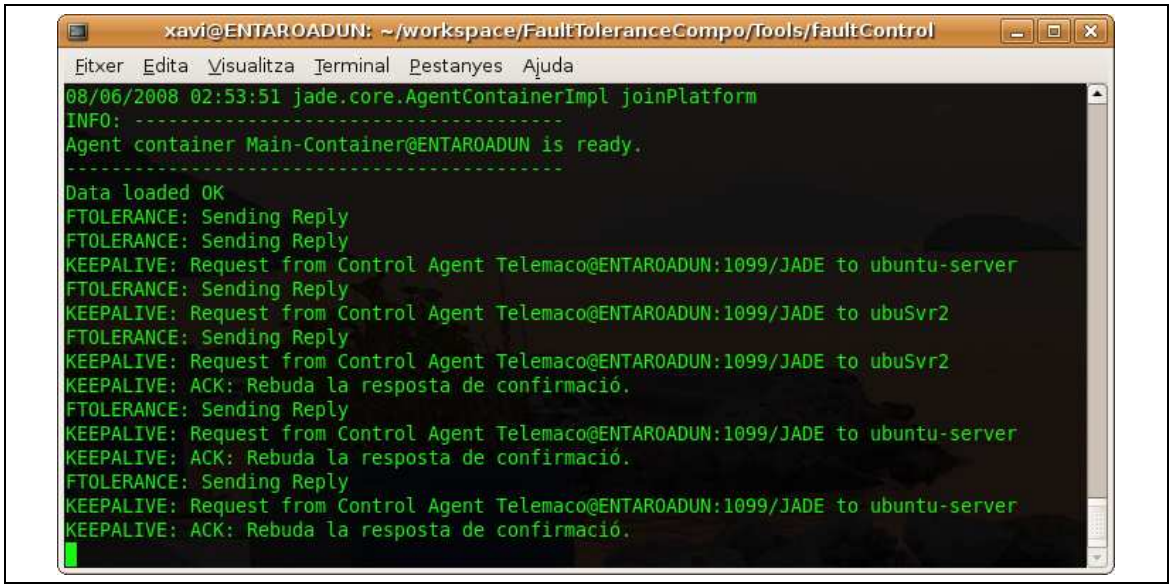


Figura 5.1: Consola de l'agent de suport

En la figura 5.2 podem veure com l'agent mòbil fa la petició de backup cap a l'agent de suport. Aquest li retornarà un *failure* perque l'agent no es destru-

eixi en aquesta plataforma i continuï el seu cicle de funcionament. El missatge `Agent Stored OK` ens informa que l'agent s'ha emmagatzemat en lloc segur correctament.

Figura 5.2: Consola de l'agent mòbil en un *checkpoint*

Aquí tenim els events registrats pels components de l'agent de suport. Per la seva banda el component *FTOLERANCE* registra cada petició de migració de l'agent mòbil. El component de *KEEPALIVE* registrarà tots els intents de petició de prova de vida i, si s'ha rebut resposta, acte seguit rebrem la contestació. Els missatges referents al CAGENTRESPONDER són els *checkpoints* completats de l'agent mòbil, és a dir, registrarem en l'arxiu de logs l'event de petició de *checkpoint* i, a més, grabarem en disc la migració de l'agent deguda a aquest *checkpoint*. Tot això ho farà el component CAgentResponder.

Missatges de l'arxiu d'events referents als components:

- `08/06/2008 03:03:02 - FTOLERANCE: Agent Ulises located in platform ubuntu-server moving to platform ubuSvr2`
- `08/06/2008 03:03:02 - KEEPALIVE: Request from Control Agent Telemaco@ENTAROADUN:1099/JADE to ubuntu-server`

- 08/06/2008 03:03:02 - KEEPALIVE: Acknowledge received.
- 08/06/2008 03:03:07 - FTOLERANCE: Agent Ulises located in platform ubuntu-server moving to platform ubuSvr2
- 08/06/2008 03:03:07 - KEEPALIVE: Request from Control Agent Telemaco@ENTAROADUN:1099/JADE to ubuntu-server
- 08/06/2008 03:03:07 - KEEPALIVE: Acknowledge received.
- 08/06/2008 03:03:12 - KEEPALIVE: Request from Control Agent Telemaco@ENTAROADUN:1099/JADE to ubuntu-server
- 08/06/2008 03:03:12 - CAGENTRESPONDER: Backup agent from (agent-identifier :name ams@ubuntu-server:1099/JADE :addresses (sequence http://ubuntu-server.localdomain:7778/acc))
- 08/06/2008 03:03:12 - KEEPALIVE: Acknowledge received.
- 08/06/2008 03:03:17 - FTOLERANCE: Agent Ulises located in platform ubuSvr2 moving to platform ubuntu-server
- 08/06/2008 03:03:17 - KEEPALIVE: Request from Control Agent Telemaco@ENTAROADUN:1099/JADE to ubuSvr2
- 08/06/2008 03:03:17 - KEEPALIVE: Acknowledge received.

Les proves que hem fet, han sigut executades sobre l'entorn de màquines abans comentat. Hem fet l'anàlisi de rendiment fent que un un agent mòbil faci 10 migracions, 8 migracions, 6 migracions i 4 migracions. Aquesta prova l'hem executat cinquanta vegades per cadascuna de les configuracions i n'hem extret la mitjana de temps en completar les migracions. Les figures 5.3 i 5.4 mostren els temps en segons que s'han extret d'aquestes proves fent *backup* de l'agent a cada migració i no fent-lo respectivament.

Les dades obtingudes a partir de les proves fetes són:

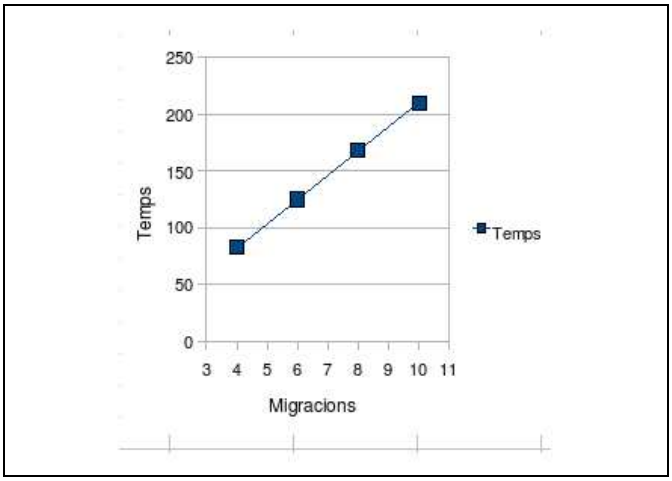


Figura 5.3: Temps de circulació fent backup de l'agent

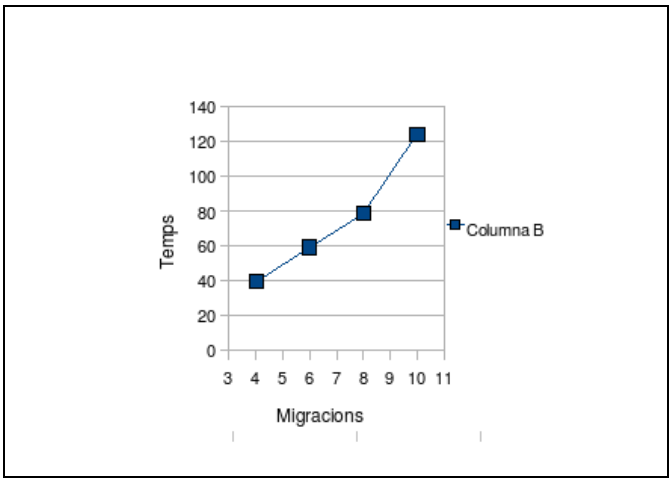


Figura 5.4: Temps de circulació sense fer backup de l'agent

- `Acum: 209826ms 10 MIGRACIONS 50 ITERACIONS BACKUP`
- `Acum: 168129ms 8 MIGRACIONS 50 ITERACIONS BACKUP`
- `Acum: 125199ms 6 MIGRACIONS 50 ITERACIONS BACKUP`
- `Acum: 82816ms 4 MIGRACIONS 50 ITERACIONS BACKUP`
- `Acum: 123998ms 10 MIGRACIONS 50 ITERACIONS NO BACKUP`
- `Acum: 78600ms 8 MIGRACIONS 50 ITERACIONS NO BACKUP`

- `Acum: 58993ms 6 MIGRACIONS 50 ITERACIONS NO BACKUP`
- `Acum: 39477ms 4 MIGRACIONS 50 ITERACIONS NO BACKUP`

Segons la formula $(Acum/1000)/MIGRACIONS = (Segons/MIGRACIO)$

La mitja de temps que un agent mòbil triga en fer una migració després de fer *checkpoint* és de **20,9** segons, mentre que quan ho fem sense demanar checkpoint, triga al voltant dels **10,5** segons. Aquests temps inclouen les peticions de migració i espera de resposta.

Prova de recuperació

Les proves de recuperació han sigut un èxit. Fent cicles de vuit migracions, en una de les migracions hem destruït l'agent des de l'eina GUI JADE Remote Agent Management (fig. 5.5). Com es pot veure en la traça de log a continuació, després de deu intents de *KEEPALIVE* s'intenta contactar amb la plataforma *ENTAROADUN*, al rebre resposta negativa, contactem amb la plataforma anterior, *ubuntu-server*, de la qual si que en tenim còpia, i procedim a llançar l'agent. Acte seguit comencem a rebre peticions de *FTOLERANCE* de l'agent mòbil i resposta a les peticions de *KEEPALIVE*.

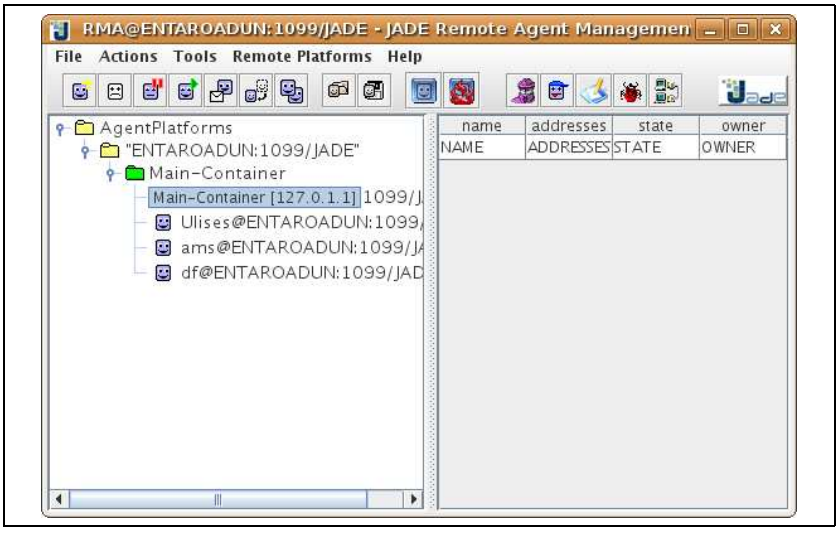


Figura 5.5: Consola Remote Agent Management amb l'agent Ulises actiu

- Jun 9, 2008 10:43:23 PM - KEEPALIVE: Acknowledge received.
- Jun 9, 2008 10:43:28 PM - KEEPALIVE: Request from Control Agent Telemaco@ubuSvr2:1099/JADE to ubuntu-server.localdomain
- Jun 9, 2008 10:43:33 PM - KEEPALIVE: Request from Control Agent Telemaco@ubuSvr2:1099/JADE to ubuntu-server.localdomain
- Jun 9, 2008 10:43:38 PM - KEEPALIVE: Request from Control Agent Telemaco@ubuSvr2:1099/JADE to ubuntu-server.localdomain
- Jun 9, 2008 10:43:43 PM - KEEPALIVE: Request from Control Agent Telemaco@ubuSvr2:1099/JADE to ubuntu-server.localdomain
- Jun 9, 2008 10:43:48 PM - KEEPALIVE: Request from Control Agent Telemaco@ubuSvr2:1099/JADE to ubuntu-server.localdomain
- Jun 9, 2008 10:43:53 PM - KEEPALIVE: Request from Control Agent Telemaco@ubuSvr2:1099/JADE to ubuntu-server.localdomain
- Jun 9, 2008 10:43:58 PM - KEEPALIVE: Request from Control Agent Telemaco@ubuSvr2:1099/JADE to ubuntu-server.localdomain
- Jun 9, 2008 10:44:03 PM - KEEPALIVE: Request from Control Agent Telemaco@ubuSvr2:1099/JADE to ubuntu-server.localdomain
- Jun 9, 2008 10:44:08 PM - KEEPALIVE: Request from Control Agent Telemaco@ubuSvr2:1099/JADE to ubuntu-server.localdomain
- Jun 9, 2008 10:44:13 PM - KEEPALIVE: Request from Control Agent Telemaco@ubuSvr2:1099/JADE to ubuntu-server.localdomain
- Jun 9, 2008 10:44:13 PM - KEEPALIVE: Try contacting ams@ENTAROADUN:1099/JADE
- Jun 9, 2008 10:44:13 PM - KEEPALIVE: Try contacting ams@ubuntu-server.localdomain:1099/JADE

- Jun 9, 2008 10:44:18 PM - KEEPALIVE: Launching stored Agent to (agent-identifier :name ams@ubuntu-server:1099/JADE :addresses (sequence http://ubuntu-server.localdomain:7778/acc))
- Jun 9, 2008 10:44:23 PM - KEEPALIVE: Request from Control Agent Telemaco@ubuSvr2:1099/JADE to ubuntu-server.localdomain
- Jun 9, 2008 10:44:28 PM - KEEPALIVE: Request from Control Agent Telemaco@ubuSvr2:1099/JADE to ubuntu-server.localdomain
- Jun 9, 2008 10:44:33 PM - FTOLERANCE: Agent Ulises located in platform ubuntu-server moving to platform ENTAROADUN
- Jun 9, 2008 10:44:33 PM - KEEPALIVE: Request from Control Agent Telemaco@ubuSvr2:1099/JADE to ubuntu-server.localdomain
- Jun 9, 2008 10:44:33 PM - FTOLERANCE: Agent Ulises located in platform ubuntu-server moving to platform ENTAROADUN
- Jun 9, 2008 10:44:33 PM - KEEPALIVE: Acknowledge received.

Fent una avaluació dels resultats i les proves, hem vist que quan l'agent mòbil no té càrrega de treball, és a dir, quan només té el component de migració i els dos components de tolerància a fallades (faultTolerance i keepAlive), l'agent de suport té dificultats a l'hora de contactar amb ell. Hem vist en el registre de logs que aquest ha arribat a fer cinc peticions abans de rebre resposta. Això pot ser degut a que quan es fa una migració, mentre l'AMS d'una plataforma negocia aquesta migració, l'agent mòbil estarà en estat de suspès. Si l'agent de suport fa la petició de prova de vida en aquestes circumstàncies, no rebrà resposta perquè l'agent mòbil estarà en procés de migració i quan torni a estar actiu serà en la nova plataforma, no a la que l'agent de suport ha preguntat.

Per ser permissius amb aquest comportament dels dos agents, hem configurat el component de prova de vida de l'agent de suport per que faci deu intents de petició amb cinc segons de marge entre peticions abans d'iniciar les tasques de

recuperació. D'aquesta manera, en uns **50 segons** podem tenir l'agent actiu de nou i funcionant correctament després d'una fallida.

Veiem que el component no és precisament ràpid, però el que és més important és que doni fiabilitat i la seguretat que la feina no es perdrà si a l'agent mòbil li passa quelcom.

Capítol 6

Conclusions

Quan parlem d'agents mòbils i veiem l'estructura modular dels mateixos, ens adonem de les possibilitats que això ens ofereix. Podem crear tants components com vulguem i sumar-los als components ja existents en l'agent. Tots aquests, formant un conjunt de tasques que definiran el comportament de l'agent.

Aquestes tasques serviran a un objectiu. La nostra fita ha sigut crear un component que asseguri que aquest objectiu s'acompleixi. Procurar que els problemes que tingui l'agent afectin el mínim possible aquest objectiu.

Per tant, fem un repàs dels objectius del projecte:

- Hem analitzat els diversos models existents i hem modelat un esquema de funcionament més ajustat la filosofia de desenvolupament d'agents mòbils. Hem vist la necessitat de crear un agent de suport localitzat en una plataforma familiar (*friendly*) per accedir a recursos que habitualment podrien necessitar permisos especials en plataformes alienes.

- Hem analitzat els requisits que ha de complir el model. Que l'agent sigui persistent. Encara que pateixi qualsevol incident, ha de poder continuar amb les seves tasques i acabar el seu cicle de vida. També ha de complir amb la propietat d'execució única. Procurar que si una tasca ja s'ha executat, no es torni a fer si es recupera l'agent que la conté.

- Hem dissenyat i després implementat un component que compleix amb les

especificacions de desenvolupament del departament, procurant que les plataformes no es modifiquin i els agents es mantinguin el màxim d'autònoms possibles. També hem seguit les especificacions de FIPA a l'hora de comunicar agents via missatges ACL i seguint els protocols estàndards.

- Hem implementat un component autònom, completament autosuficient, això si, amb dues parts que depenen l'una de l'altra per funcionar i que treballaran de forma concurrent sense interferir, a menys que sigui necessari, en el comportament dels altres components.

- Hem procurat que les modificacions en l'agent mòbil siguin mínimes perquè afegir-li aquest component sigui el més senzill possible. Realment, afegint els components de tolerància a fallades i de prova de vida i configurant-los per que el segon s'executi en un thread separat, ja permetrà que s'executi.

  L'agent de suport, per la seva banda, està dissenyat per treballar íntegrament com a component de tolerància a fallades, ja que necessita sincronia entre totes les parts per funcionar correctament.

- La propietat d'execució única la podrà garantir el programador d'agents quan els disseny, just després de cada tasca que consideri important podrà llançar un backup. Només li caldrà fer una migració cap a l'agent de suport i aquest s'encarregarà de fer un checkpoint que despres pot utilitzar per recuperar l'estat.

- Hem desenvolupat el component procurant que hi hagués un equilibri entre bon funcionament i rendiment. Actualment el sistema dóna bones respostes en quant a reacció davant de problemes. Els paràmetres de temps d'espera a resposta i de temps entre peticions són els que poden fer variar aquests llindar. El problema de sempre és que si volem un agent ràpid en el seu itinerari, no s'ha d'entretenir en fer tasques colaterals, però si passa quelcom, aleshores és quan venen els maldecaps. El cas contrari, assegurar que després de cada acció presa es faci una còpia de l'estat per no perdre res donarà uns temps molt pobres, omplirà de trànsit la xarxa, però ens assegurarem

que si passa res, l'agent es podrà recuperar des d'on ho ha deixat. Deixem a elecció del programador aquesta decisió un tant heurística.

Recapitulant en el que hem vist després d'haver estat una bona temporada convivint amb aquest projecte, caldria remarcar possibles línies futures cap a on encaminar les pròximes investigacions.

- Atorgar al component de tolerància a fallades i de prova de vida un grau de seguretat integrant-lo amb el servidor de claus públiques d'agents mòbils. Degut a la poca protecció de l'agent mòbil davant de plataformes malicioses, no és recomanable que aquest porti certificat, per tant, només podem oferir autenticació de l'agent de suport davant de l'agent mòbil. Integrar el servidor de claus públiques als quals poder accedir-hi seria un grau més de seguretat.

- La vida de l'agent de suport està lligada a la de l'agent mòbil. Continuant amb el desenvolupament de components per ampliar la biblioteca, la creació d'un component de finalització que s'executés quan acabés l'agent mòbil i que avises a l'agent de suport per que aquest es destruis quan ja no fos necessari. Aquest component seria un reclam per tots els components compostos que es desenvolupin, ja que van acompanyats del seu agent de suport.

- Integrar el component de tolerància a fallades amb les eines de desenvolupament d'agents SMARD (Secure Mobile Agent Rapid Development), i més concretament amb l'Agent Builder i el Component Selector. Aquest component, per la seva estructura en cinc subcomponents, no serà fàcil d'integrar.

- S'hauria d'analitzar si és millor tenir un agent de suport per cadascun dels agents mòbils als quals es vulgui atorgar la funcionalitat de tolerància a fallades o si és millor donar aquest servei a tots els agents mòbils que ho vulguin des d'un sol agent de suport. S'haurien de veure estadístiques de rendiment, de trànsit de xarxa, de temps de processador, etcètera.

- Actualment el model de tolerància a fallades comprova el bon funcionament de l'agent mòbil, però si és l'agent de suport el que té problemes, aquest no té cap mètode per recuperar-se i deixaria l'agent mòbil bloquejat en la plataforma on es trobés. Es quedaria en aquest estat fins que rebés el missatge de confirmació de l'agent de suport. Hauríem de tornar a aixecar l'agent de suport manualment i aleshores l'agent mòbil continuaria el seu itinerari.

- En el cas de caiguda d'una plataforma. És a dir, l'agent de control ha perdut el contacte amb l'agent mòbil degut a que la plataforma ha deixat de funcionar. L'acció que s'hauria de prendre seria la d'informar l'administrador d'aquella plataforma via missatge SMTP. Però per això s'hauria de mantenir una correspondència entre plataformes i persones de contacte, que haurien d'estar en un lloc públic, com un servei.

Aquestes són algunes propostes de millora o ampliació que apareixen a mesura que vas indagant en el marc de recerca d'agents mòbils. Esperem que aquesta memòria serveixi d'ajuda per desenvolupar nous components a futurs programadors.

Bibliografia

[1] Security of Networks and Distributed Applications. *http://senda.uab.cat*

[2] Java Agent DEvelopment Framework. *http://jade.tilab.com/*

[3] IEEE FIPA. Foundation for intelligent physical agents. *http://www.fipa.org*

[4] A. Martín. MedIGS, un sistema segur basat en agents per al descobriment i obtenció de dades mèdiques distribuïdes. Estudi i desenvolupament. *ETSE. Universitat Autònoma de Barcelona. Memòria projecte final de carrera.*, Jun. 2007.

[5] C. Martínez. Modular construction of component-based agents. Design and development. *ETSE. Universitat Autònoma de Barcelona. Memòria projecte final de carrera.*, Jun. 2007.

[6] Sun Microsystems, Inc.; JavaTM 2 Platform Standard Edition 5.0 API Specification. *http://java.sun.com/j2se/1.5.0/docs/api/*

[7] Inter-Platform Mobility Project *https://tao.uab.cat/ipmp/*

[8] S. Pleisch, A. Schiper. Approaches to Fault-Tolerant and Transactional Mobile Agent Execution. *ACM Computing Surveys*, 2004.

[9] L. M. Silva, V. Batista, J. G. Silva. Fault-Tolerant Execution of Mobile Agents. *International Conference on Dependable Systems and Networks.*, 2000.

[10] M. R. Lyu, X. Chen, T. Y. Wong. Design and Evaluation of a Fault-Tolerance Mobile-Agent System. *IEEE Intelligent Systems.*, Sep-Oct. 2004.

[11] J. Cucurull. Contribució a la mobilitat i seguretat dels agents software. *Treball de recerca pel progràma del doctorat en informàtica.* Maig 2006.

[12] A. Moratalla i S. Robles. Component-based development of secure mobile agents applications. *Lecture notes in Artificial Intelligence.*, Sep. 2007.

[13] Eclipse Foundation *http://www.eclipse.org/*

Firmat: Xavier Piñol Torné

Bellaterra, juny de 2008

Resum

A mesura que la complexitat de les tasques dels agents mòbils va creixent, és més important que aquestes no perdin el treball realitzat. Hem de saber en tot moment que la execució s'està desenvolupant favorablement. Aquest projecte tracta d'explicar el procés d'elaboració d'un component de tolerància a fallades des de la seva idea inicial fins a la seva implementació. Analitzarem la situació i dissenyarem una solució. Procurarem que el nostre component emmascari la fallada d'un agent, detectant-la i posteriorment recuperant l'execució des d'on s'ha interromput. Tot això procurant seguir la metodologia de disseny d'agents mòbils per a plataformes lleugeres.

Resumen

A medida que la complejidad de las tareas asignadas a los agentes móviles va creciendo, es más importante que no se pierda el trabajo realizado. Debemos saber en todo momento que la ejecución se está desarrollando favorablemente. Este proyecto trata de explicar el proceso de elaboración de un componente de tolerancia a fallos desde su idea inicial hasta su implementación. Analizaremos la situación y diseñaremos una solución. Trataremos de enmascarar los fallos que puedan suceder en un agente móvil, detectándolo y recuperando la ejecución desde donde se interrumpió. Todo esto tratando de seguir los pasos de la metodología de diseño de agentes móviles para plataformas ligeras.

Abstract

As the complexity of the tasks assigned to mobile agents grows, it is more important not to lose the job done. We must know at all times that execution is developing favourably. This project tries to explain the process of developing a component of fault tolerance from initial idea to its implementation. Analyse the situation and design a solution. Try to mask the failures that can happen in a mobile agent by detecting and recovering from where the execution was interrupted. All this trying to follow in the footsteps of the design methodology of mobile agents for light platforms.