AGENTS MÒBILS EN SITUACIONS D'EMERGÈNCIA. MOBILITAT D'ETIQUETES DE TRIATGE.

Memòria del projecte de final de carrera corresponent als estudis d'Enginyeria Superior en Informàtica presentat per Robert Sallent López i dirigit per Abraham Martín Campillo.

Bellaterra, setembre de 2009

El firmant, Abraham Martín Campillo, 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 Robert Sallent López

Bellaterra, setembre de 2009

Firmat: Abraham Martín Campillo

A tots els que estimo de debò.

Agraïments

Gràcies a tots aquells que heu estat al meu costat tot aquest temps, especialment als meus pares, amics, companys i professors.

Índex

Índex de figures

Capítol 1

Introducció

1.1 Introducció

Quan es produeix una situació d'emergència, amb un nombre elevat de víctimes i en una extensió de terreny considerable, una de les tasques més importants és la de classificar aquestes víctimes en base al seu estat. Això permet distribuir de forma més efectiva els recursos mèdics de que es disposa, que poden ser relativament escassos en emergències d'aquest tipus.

El personal encarregat de fer aquesta classificació pot no ser mèdic i els requisits principals que ha de complir són: conèixer el protocol d'actuació en cas d'emergència, saber reconèixer les persones que es trobin il·leses i ser capaç de localitzar i classificar els ferits i morts.

El procés de classificació i selecció és conegut com a triatge o *triage* [triage] i accelera l'actuació de metges i metgesses especialitzats presents al lloc del sinistre, que ja no utilitzen el seu temps per avaluar i classificar sinó que poden concentrar-se en la tasca d'estabilitzar pacients, de manera prioritzada segons la classificació.

Mitjançant uns criteris definits en el protocol de triatge utilitzat [dprotocols], cal mesurar les constants del pacient per a determinar el seu estat real. Els criteris normalment es basen en paràmetres com el ritme respiratori, el pols i l'estat de consciència de la víctima; mentre que l'estat del pacient es pot expressar en funció

de la gravetat o tenint en compte la celeritat amb la que cal tractar-lo.

A dia d'avui, l'estat de salut de la persona triada es mostra utilitzant una etiqueta física de cartró, que es col·loca al voltant del coll amb un cordill, i que disposa d'un codi de colors que mostra la gravetat del cas. Aquesta etiqueta s'anomena *Triage Tag* [triagetag] i és un element indispensable en totes les actuacions, independentment del protocol de triatge que es faci servir.

El mètode tradicional d'utilització de les etiquetes de triatge presenta una sèrie d'inconvenients, els més notables relacionats amb el fet que la informació roman a prop del pacient, sense arribar al centre de coordinació de l'emergència, fins que no hi torna la persona responsable del triatge per notificar la situació. D'aquesta manera, el personal mèdic no pot disposar d'informació de l'estat de cada víctima per avançat ni conèixer exactament la ubicació dels pacients.

En aquest projecte, juntament amb altres del grup SeNDA que en conformen un de major envergadura amb el nom *Providing Early Resource Allocation During Emergencies: The Mobile Triage Tag* [mtt], es proposa una automatització del procés de triatge basada en agents mòbils [agent], que s'encarregaran de transportar etiquetes de triatge electròniques des del punt on es troba la víctima fins al centre de coordinació i control, on es gestionarà i coordinarà l'actuació dels equips de rescat.

Pensem que en una zona que pot estar inundada, cremada o contaminada no disposarem d'una infraestructura de comunicació de gran abast com pot ser una xarxa de cobertura telefònica fixa o mòbil, fins i tot pot ser que no existeixi la possibilitat de comunicar-se per satèl·lit i que l'únic canal viable de comunicació hagi de ser establert per nosaltres mateixos. En aquest sentit, els nostres agents es mouran per una xarxa MANET [manet], que apareix al mateix temps que es realitza l'actuació de rescat gràcies als dispositius emprats per al triatge, que permetran establir aquest tipus de comunicació sense fils.

A més de millorar la comunicació entre la zona damnificada i el centre de coordinació, així com la disponibilitat de les dades de les pacients, s'amplia el volum d'informació continguda a les etiquetes amb coordenades GPS, que permeten ubicar amb precisió cadascuna de les víctimes.

Amb aquestes innovacions es vol aconseguir una actuació més ràpida i efectiva en entorns crítics, una millor atenció als ferits, més control dels recursos disponibles i una coordinació del personal més eficient. Es soluciona el problema de no poder conèixer el nombre de víctimes, s'evita la possibilitat que en quedi alguna triada però no atesa i, gràcies a informació GPS, es poden planificar rutes optimitzades per als vehicles d'assistència i equips de rescat.

Amb tot el que s'ha comentat en aquest punt, és evident que el personal encarregat de fer el triatge haurà de disposar de material electrònic de suport, tant per emmagatzemar digitalment la informació com per a generar la xarxa sense fils, no n'hi ha prou amb les etiquetes de cartró i el bolígraf emprat fins ara. D'altre banda, l'enviament de les dades de les víctimes ha de ser automàtic i transparent, delegant la responsabilitat de l'arribada a destí a l'agent mòbil, que decidirà el camí més adient per accedir al centre de control com més aviat millor.

1.2 Objectius

El projecte *Mobilitat d'etiquetes de triatge* és part del conjunt de projectes del grup SeNDA destinats a millorar la gestió d'emergències in situ amb l'etiqueta de triatge electrònica.

L'objectiu principal és aconseguir la mobilitat dels agents portadors d'etiquetes entre les plataformes presents a l'àrea d'actuació. Aquests agents amb les dades de la víctima, han de ser capaços de viatjar de plataforma en plataforma fent servir el canal inalàmbric fins arribar al centre de coordinació.

És molt probable que des de la ubicació en la que es trobi un pacient i es generi la corresponent etiqueta, no es disposi de suficient cobertura per a que l'agent arribi directament a destinació, en aquest cas, cal aconseguir que decideixi de forma autònoma si ha de viatjar a un altre plataforma dins la MANET, des de la que es pugui arribar abans al punt de coordinació, o si ha de romandre quiet esperant. Aquest fet crea la necessitat d'avaluar de nou la situació quan canviï la topologia de la xarxa.

Els salts i migracions d'agents en xarxes sense fils *ad hoc* com la nostra, po-

den portar problemes derivats de la desaparició sobtada de plataformes o el soroll. Els agents han de ser tolerants a migracions fallides, de tal manera que detectin aquestes situacions i puguin determinar si és convenient intentar de nou la migració, canviar de destinació o quedar-se a la plataforma actual.

Simultàniament a aquest projecte, s'estan desenvolupant dos més estretament relacionats. D'una banda una arquitectura de serveis en *Gestió Dinàmica de Serveis* [serveis] de la que farem ús per conèixer quines plataformes properes poden acollir els nostres agents, de l'altre, el mecanisme de detecció de dispositius vinculats a la nostre xarxa. El nostre sistema ha de ser capaç de comunicar-se correctament amb l'arquitectura de serveis i, en el futur, haurà d'estar plenament integrat en el projecte *The Mobile Triage Tag* [mtt].

A mode de resum, es pot dir que els objectius principals de *Mobilitat d'etiquetes de triatge* són:

- Aconseguir la mobilitat dels agents entre les plataformes presents a l'àrea d'actuació, de manera que arribin al punt de coordinació i entreguin l'etiqueta de triatge electrònica que contenen.

- Establir un mecanisme de decisió per als agents que els permeti decidir, de forma autònoma, si han de viatjar a un altre plataforma diferent a l'actual des de la que es pugui arribar abans al centre de control de l'emergència.

- Impedir que una migració fallida provoqui la pèrdua de l'agent o de la informació que conté.

- Fer ús de l'arquitectura de serveis desenvolupada en un projecte paral·lel per conèixer quines plataformes veïnes poden acollir els nostres agents.

1.3 Estructura de la memòria

En aquest punt s'explica la distribució i organització en apartats d'aquesta memòria, amb una breu introducció de cadascun d'ells.

1. **Capítol 1: introducció**. Presenta algunes de les deficiències existents en els mecanismes de triatge actuals i el context en el que es produeixen, la raó de ser d'aquest projecte i les seves motivacions i objectius.

2. **Capítol 2: estat de l'art**. En aquest punt s'explica cóm es realitza actualment el triatge i els intents que hi ha per automatitzar-lo, es descriu el concepte d'agent mòbil amb una breu descripció, el funcionament de la plataforma de JaDE que fem servir, es parla de cóm es comuniquen els agents entre ells i cóm es pot aconseguir la seva migració gràcies al servei de mobilitat inter-plataforma desenvolupat al dEIC [mobility].

3. **Capítol 3: anàlisi**. Estudi del problema, anàlisi de viabilitat, descripció dels requeriments funcionals i no funcionals i resum de l'estat de projectes relacionats amb el nostre.

4. **Capítol 4: disseny**. Aquest apartat conté la proposta de disseny que compleix amb els requeriments, assoleix els objectius i s'integra amb els altres projectes. Es parteix d'una visió general del sistema i es segueix amb una visió més concreta de cadascun dels punts clau.

5. **Capítol 5: implementació integració i prova**. Descripció de cóm s'ha implementat el disseny, detalls d'integració amb altres projectes del grup i de cóm s'ha provat el sistema desenvolupat. S'expliquen tant els mecanismes i les tecnologies emprades, com les proves realitzades de forma local i distribuïda. Cal tenir present que quan es combinen agents amb migració inter-plataforma les proves són complicades de realitzar, per aquest motiu es dedica un apartat específic a explicar en detall cóm s'han realitzat els tests.

6. **Capítol 6: conclusions i línies d'ampliació**. En l'apartat de conclusions d'aquest projecte s'explica si s'han pogut assolir els objectius marcats a l'inici. Es fa una valoració de la feina feta, del grau d'adequació del disseny escollit i implementat i dels problemes que han aparegut al llarg de les

etapes de disseny i desenvolupament. També es descriuen possibles línies d'ampliació i feina per fer de cara a la integració dins el projecte global.

7. **Capítol 7: acrònims**. Llista ordenada de les abreviatures i acrònims emprats al llarg d'aquesta memòria.

Capítol 2

Estat de l'art

2.1 Triatge

Els protocols d'actuació en grans emergències no són un tema nou, la reducció del temps necessari per atendre a les víctimes pot salvar vides.

El triatge és una de les millores introduïdes per optimitzar el temps del personal mèdic i concentrar el seu esforç en els casos que necessiten una atenció prioritària. Forma part de tots els protocols d'actuació en cas d'emergència i difícilment se'n pot deslligar si es vol obtenir el major grau d'èxit possible.

Com és de suposar, hi ha un gran nombre d'estudis centrats en perfeccionar, millorar i automatitzar els mecanismes de triatge. En aquest sentit, cal diferenciar entre els projectes destinats a l'estudi i creació de nous mecanismes i els projectes de millora i automatització dels mateixos, com és el nostre cas. El que volem aconseguir no és dissenyar un nou mètode de triatge sinó millorar el suport físic, el *Triage Tag*, del mecanisme existent al protocol START [start]. En la figura 2.1 es poden veure les dues cares d'una etiqueta de triatge actual que es fa servir en aquest protocol.

START (Simple Triage And Rapid Treatment) és un mètode simple, que especifica els criteris d'etiquetatge en funció de paràmetres com la respiració, el pols radial i l'estat neurològic de la víctima. Va ser desenvolupat l'any 1983 a l'hospital Hoah (Califòrnia) i actualitzat al 1996 pel centre mèdic Eisenhower.

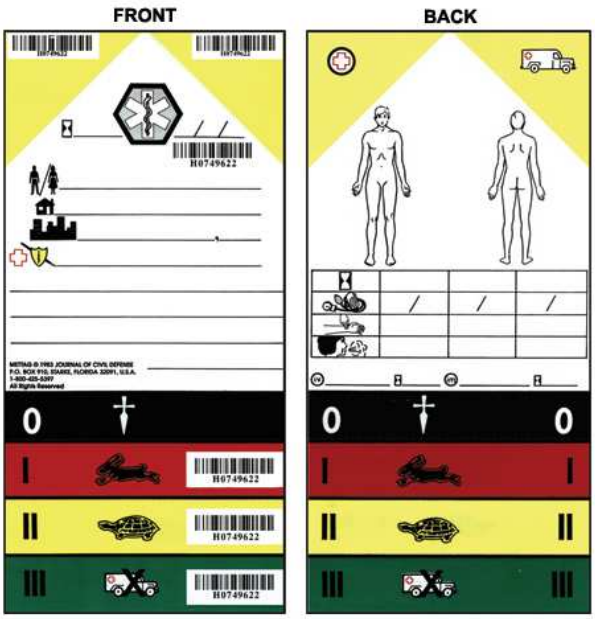

Figura 2.1: Etiqueta de triatge tradicional.

En el projecte *Disseny i implementació d'interfícies del protocol START* [dham], desenvolupat al dEIC a 2008 i que forma part també de *The Mobile Triage Tag* [mtt], es va realitzar una implementació d'aquest protocol i una interfície gràfica en JAVA.

START és simple, efectiu, i és conegut i utilitzat a gairebé la totalitat del nostre planeta, a excepció del Regne Unit i Alemanya.

2.2 Automatització dels mecanismes de triatge

L'automatització dels mecanismes de triatge és una mesura que ha estat proposada freqüentment per resoldre els problemes derivats de l'etiquetatge manual de les víctimes. Les propostes van des de l'etiquetatge amb codi de barres o radiofreqüència (RFID) fins a etiquetes que monitoritzen el seu estat.

Moltes de les propostes fetes fins ara no han aconseguir proporcionar una resposta factible i de baix cost al problema, sovint donen per suposada l'existència

de xarxes de comunicacions telefòniques o satèl·lit al lloc sinistrat i, de vegades, es basen en la utilització de codis de barres, propensos a patir errors de lectura en el procés de vincular les etiquetes electròniques amb els pacients.

A continuació hi ha una llista de les propostes més rellevants pel que fa a l'automatització dels *Triage Tags* amb una petita descripció:

a AMBULANCE proposa enviar senyals vitals i imatges des d'una unitat mòbil al centre de control utilitzant GSM, satèl·lit o xarxes fixes.

b TETRA (TErrestrial Trunked RAdio) transmet veu i dades mitjançant ràdio digital en situacions d'emergència, fent servir comunicació directa o *ad hoc*.

c IMPROVISA es centra en els aspectes comunicatius de l'àrea d'emergència, especialment amb l'ús de MANETS i utilitza agents intel·ligents per a la gestió de la xarxa.

d WIISARD (Wireless Internet Information for Medical Response in Disasters) fa servir tecnologia sense fils per a la coordinació, donant suport per a transmissió a temps real de dades mèdiques de l'estat de la víctima.

e TacMedCS es basa en l'ús d'etiquetes radiofreqüència per identificar les víctimes i emmagatzemar les dades. També fa servir dispositius handheld i GPS.

f MASCAL integra hardware i software per a millorar la gestió dels recursos en hospitals quan s'ha produït un desastre. Fa servir el sistema de triatge TacMedCS.

g The Descentralized Electronic Triage System és una etiqueta electrònica de baix consum que monitoritza l'estat de la víctima amb sensors i permet mostrar l'estat amb leds de colors.

h ARTEMIS planteja monitoritzar les víctimes de forma remota a través d'una PDA amb sensors que transmet informació mitjançant missatges en xarxes ad hoc sense fils.

i Traceability System for Sick and Injured in Event Major Disasters permet registrar l'escriptura a mà amb les dades de la víctima i, mitjançant un lloc web dedicat, accedir a les dades dels pacients.

j EMTrack també és un sistema de seguiment electrònic de víctimes amb PDA i comunicacions via telèfon mòbil o satèl·lit.

2.3 Agents

Un agent és un sistema computacional que habita en un entorn dinàmic complex, percebent i actuant de forma autònoma per tal d'assolir l'objectiu per al que va ser dissenyat. Aquest concepte té una llarga llista d'aplicacions possibles en camps com els processos de fabricació, control, comerç electrònic, xarxes, sistemes de transport, sanitat i computació científica.

La computació basada en agents s'ha convertit en una potent tecnologia per al desenvolupament de sistemes de software distribuïts i complexes des dels anys 90, fins al punt que hi ha investigadors que creuen que representa el paradigma més important des que es va idear el disseny orientat a objecte.

Un agent és l'actor fonamental en una plataforma d'agents, que aporta la infraestructura física en la que poden ser desplegats i consisteix en la màquina, el sistema operatiu, el software de suport per agents, els components d'administració d'agents FIPA [fipa] i els mateixos agents.

FIPA es una organització internacional fundada al 1996 dedicada a promoure la industria dels agents intel·ligents, desenvolupant obertament especificacions que suporten la interacció entre ells i amb les seves aplicacions. Les seves especificacions representen un conjunt d'estàndards que pretenen promoure la interacció d'agents heterogenis i els serveis que poden representar.

En 2002 FIPA va completar el procés d'estandardització en un subconjunt de 25 especificacions de diferents categories: comunicació entre agents, transport, manipulació, arquitectura i aplicacions. De totes aquestes categories, la comunicació entre agents és el centre del model de sistema multi agent FIPA.

La majoria de les plataformes d'agents existents són per JAVA i gairebé totes compleixen les especificacions FIPA, alguns exemples són: Mobile-C, SECMAP, W-map, JMAP, JaDE, Aglets o Voyager.

Aquest projecte empra JaDE (Java Agent Development Framework) [jade], que ens proporciona eines per l'execució, migració i comunicació entre agents. El seu objectiu es fer més senzill el desenvolupament d'aplicacions respectant les especificacions FIPA.

2.4 JaDE

JaDE es el middleware desenvolupat per *Telecom Italia Lab* (TILAB) per al desenvolupament d'aplicacions distribuïdes multi agent basades en l'arquitectura de comunicació peer to peer.

Tant la intel·ligència, la iniciativa, la informació, els recursos i el control poden ser totalment distribuïts, en terminals mòbils o en ordinadors d'una xarxa fixa. L'entorn por evolucionar dinàmicament amb agents, que es comuniquen de forma totalment simètrica i poden iniciar la conversa o respondre independentment del tipus de xarxa.

JaDE està totalment desenvolupat en JAVA i té com a principis bàsics la interacció, uniformitat, portabilitat i la facilitat d'ús; compleix les especificacions FIPA i d'aquesta manera els agents JaDE poden interactuar amb altres agents que també les compleixin. A més a més, proveeix un conjunt homogeni de APIs que són independents de la xarxa que hi hagi per sota i de la versió de JAVA, amagant la complexitat del middleware i facilitant la feina als programadors.

Com es pot veure en la figura 2.2, JaDE també disposa d'una interfície gràfica des de la que mostren els agents actius localment, donant la possibilitat de crear-ne de nous, finalitzar la seva execució o enviar-los missatges.

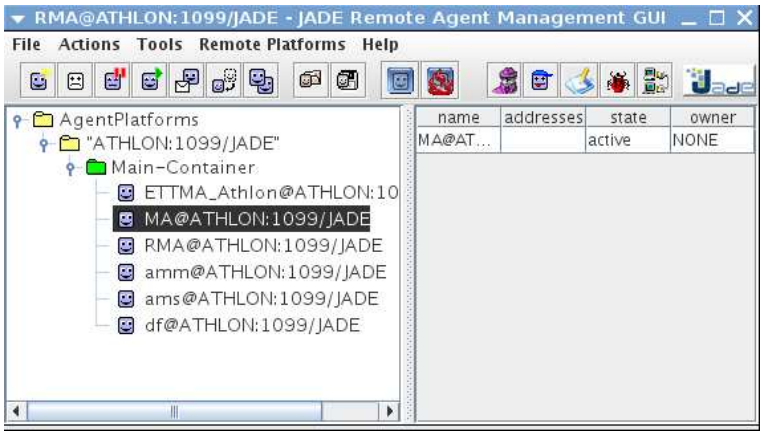

Figura 2.2: Interfície gràfica de la plataforma de JaDE

2.5 Comunicació entre agents

Quan un agent A es comunica amb un altre agent B, la informació és transferida de A a B en forma de **missatge ACL**. Dins d'un missatge d'aquest tipus les dades es representen com una expressió en un llenguatge específic i es codifiquen en el format adequat.

De vegades, guardar les dades en el missatge en forma de cadena (que és la forma més simple) no resulta pràctic a l'hora de tractar-les i és molt més útil fer-ho amb objectes JAVA de la classe específica que necessitem. Per poder fer això cal que l'agent A converteixi les dades en una expressió que pugui ser inserida en un missatge ACL, mentre que B ha de realitzar l'acció inversa i una sèrie de comprovacions semàntiques per assegurar que les dades rebudes tenen un significat complet.

Les conversions es duen a terme mitjançant un gestor de continguts disponible a tots els agents de JaDE, que ofereix les interfícies adequades per accedir a aquesta funcionalitat, però que realment delega la conversió i les tasques de comprovació a una ontologia i un llenguatge de continguts.

De forma més específica, l'ontologia valida la informació que s'ha de convertir des del punt de vista semàntic i un còdec realitza la traducció d'acord amb les normes sintàctiques del llenguatge de continguts.

2.5.1 Llenguatges de continguts a JaDE

Un còdec per a un llenguatge de continguts és un objecte JAVA capaç de manipular expressions escrites en aquest llenguatge. JaDE inclou directament còdecs per a dos llenguatges de continguts: **SL** i **LEAP**; en la major part dels casos, el programador només ha d'adoptar un d'aquests i fer us del còdec relacionat sense més esforç.

- El **llenguatge SL** és llegible pels humans ja que es troba codificat com a cadena de text i és un dels més utilitzats en la comunitat científica a l'hora de treballar amb agents. És especialment pràctic per depurar i provar les aplicacions.

- El **llenguatge LEAP** no és llegible ja que es troba codificat en forma de bytes. Una de les característiques més destacables és que la classe *LEAPCodec* és més lleugera que la *SLCodec*, cosa que fa que sigui el preferit quan tenim limitacions de memòria com pot ser en el cas de treballar amb terminals mòbils (telèfons o PDAs).

2.5.2 Ontologies

La ontologia valida la informació a convertir des del punt de vista semàntic. Per fer-ho cal classificar els possibles elements del domini de discussió d'acord amb les seves característiques semàntiques. A continuació es mostra la classificació d'elements derivada de la definició de llenguatge ACL proposada per FIPA:

- Els **predicats** són expressions que ens parlen del món que ens envolta i poden ser certes o falses, per exemple: *En Joan treballa molt*.

- Els **conceptes** són expressions que recullen entitats amb estructures complexes, per exemple: *Persona : nom Joan : edat 25*.

- Les **accions** són conceptes especials que indiquen accions que poden ser realitzades per alguns agents.

- Les **primitives** són expressions que indiquen entitats atòmiques, com una cadena o un sencer.

- Els **agregats** són expressions que indiquen entitats que formen grups d'altres entitats, per exemple: *seqüència (Persona : nom Joan : edat 25) (Persona : nom Pere : edat 22)*.

- Les **variables** són expressions que indiquen un element genèric no conegut a priori.

La ontologia per a un domini concret és un conjunt d'esquemes definint l'estructura dels predicats, accions i conceptes d'aquest domini.

2.6 Inter-Plattform Mobility Service

La mobilitat dels agents entre diferents plataformes no estava disponible en inici a JaDE, només es permetia mobilitat entre diferents contenidors a la mateixa plataforma. El servei de mobilitat inter-plataforma [mobility] ha estat desenvolupat al dEIC amb la intenció de permetre mobilitat plataforma a plataforma als agents de JaDE.

Està construït per ser transparent al programador d'agents, de forma que moure agents entre plataformes diferents és tan simple com moure els agents entre contenidors d'una mateixa plataforma.

En aquesta arquitectura de migració, quan un agent es vol moure envia un missatge ACL cap al destí amb una llista de protocols a fer servir i algunes restriccions de compatibilitat. El missatge de resposta retorna amb l'acord de migració acceptat o denegat. Actualment hi ha implementats quatre protocols diferents de transferència: *Push Cache Transfer Protocol, On Demand Transfer Protocol , Fragmented Transfer Protocol* i *REST Transfer Protocol*.

2.7 Xarxes MANET

Una MANET és una xarxa sense fils, sense infraestructura ni punts d'accés, en la que cada node actua simultàniament com a node final i com a router, cosa que es coneix com xarxa multi-salt. És un tipus de xarxa ad-hoc que canvia la seva topologia amb molta freqüència i és capaç de configurar-se *al vol*.

Les xarxes centralitzades no són presents en totes les situacions. En una operació de rescat amb un escenari en situació d'emergència com hem descrit en el capítol introductori d'aquesta memòria, segurament no tindrem cap altre opció que establir una comunicació dinàmica entre els dispositius, per tant necessitarem un tipus de connectivitat com el que ofereix una MANET.

Com que aquestes xarxes son mòbils, utilitzen sistemes de comunicació sense fils per connectar xarxes diferents entre sí, sistemes que poden ser connexions Wi-Fi estàndard o transmissions de telefonia mòbil o satèl·lit.

Una de les característiques de la MANET és la comunicació multi-salt de forma distribuïda, per establir-la tots els nodes han de poder actuar com a routers per als altres. Les rutes es configuren i mantenen amb un protocol de routing, que acostuma a ser força complex degut als canvis ràpids i constants que es produeixen en la topologia. El manteniment pot ser reactiu si els protocols estableixen les rutes sobre demanda, o proactiu si proven de tenir coneixement sempre de l'esquema complet de la xarxa.

Les MANETs, especialment amb manteniment proactiu, tenen un problema d'escalabilitat perquè l'increment del nombre de nodes en la xarxa produeix un augment del nombre de paquets intercambiats per al manteniment de rutes, consumint ample de banda i rebaixant el *throughput*.

Tot i això, està demostrat analíticament a *Ventajas de usar subredes en una red ad-hoc con nodos móviles* [manetsadv] que la càrrega generada pel protocol d'encaminament es pot reduir si es fa servir un protocol amb jerarquies (subxarxes).

Capítol 3

Anàlisi

3.1 Descripció de la proposta

L'escenari d'una intervenció el formen dues àrees: l'àrea d'emergència (EA) i el centre de coordinació d'emergències o **Emergency Coordination Centre** (ECC). El triatge de les víctimes el fa el personal encarregat d'aquesta tasca, els doctors estabilitzen els ferits i els equips de rescat els evacuen.

La proposta realitzada es centra en l'àrea on s'ha produït el sinistre i considera que no existeixen mitjans de comunicació in situ, a excepció dels que podem establir nosaltres mateixos mitjançant la utilització de la xarxa MANET, generada a partir dels dispositius de comunicació sense fils dels nostres terminals de triatge.

En aquest context, **es vol un sistema capaç de transportar etiquetes electròniques de triatge**, des dels terminals del personal encarregat de la tasca de classificació fins al centre de coordinació, el més ràpidament possible i fent servir la xarxa MANET.

El sistema ha d'estar basat en agents intel·ligents, capaços de decidir on i quan moure's en funció de l'estat de la xarxa. Així es pot aconseguir un manteniment flexible, en el que canviar el comportament del sistema només representa canviar el de l'agent i una comunicació asíncrona sense connexió, que permet migracions ràpides de l'agent cap a qualsevol plataforma a l'abast, sense necessitat d'establir canals de comunicació prèviament i en qualsevol instant de temps.

Disposarem de terminals amb comunicació sense fils, la plataforma de JaDE i l'anàlisi fet a *The Mobile Triage Tag* [mtt] en el que es determina que el criteri més encertat per migrar d'una plataforma a un altre és el concepte de **Time To Return** (TTR), comentat una mica més endavant en aquest mateix capítol.

The Mobile Triage Tag proposa per al nostre sistema una estructura de dos agents: l'agent de transport, que conté l'etiqueta de triatge d'una víctima i un agent de gestió, que l'informa de l'estat del món i porta control sobre els agents de transport presents. Així dons, existiran tants agents de transport com víctimes etiquetades però només un agent de gestió per plataforma. També es proposa que l'agent de gestió sigui l'encarregat de presentar la interfície gràfica per a interactuar amb l'usuari encarregat del triatge, i que generi els agents de transport amb les dades introduïdes en ella.

A mode de resum, es pot dir que volem un sistema amb dos tipus d'agents diferenciats: un que gestionarà la informació relativa a la situació del món, la subministrarà a l'agent de transport i mostrarà una interfície gràfica a l'usuari del terminal; i l'altre que ha de transportar una etiqueta de triatge des del punt on ha estat generat fins al centre de coordinació.

3.2 Anàlisi de requeriments

En el capítol d'*introducció* d'aquesta memòria s'han definit els objectius d'aquest projecte, a continuació es presenten els requisits que ha de complir per ajustar-se exactament a la descripció de la proposta.

3.2.1 Requeriments funcionals

Es demana que el software sigui capaç de:

- Transportar etiquetes de triatge des dels terminals del personal de triatge fins al centre de control el més ràpidament possible.
- Separar la gestió d'agents portadors de *Triage Tags* de les tasques de generació d'etiquetes.

- Fer servir el concepte de *Time To Return* per avaluar si l'agent ha de migrar a una nova plataforma o ha de romandre en l'actual.

- Recuperar el *Time To Return* de les plataformes veïnes i subministrar-lo a l'agent de transport per tal que aquest decideixi què ha de fer.

- Detectar l'arribada al centre de coordinació i realitzar l'entrega de les dades de la víctima.

- Controlar i gestionar la creació i migració d'agents amb informació dels pacients, evitant que aquests es perdin o puguin romandre inadvertits.

3.2.2 Requeriments no funcionals

- Fer servir JAVA, la plataforma de JaDE i el paradigma de programació amb agents.

- Aconseguir que el projecte pugui ser integrat amb projectes realitzats anys anteriors i amb els que es desenvolupen actualment en paral·lel. Sobre aquest punt hi ha més informació a la secció de *anàlisi del software* d'aquest mateix capítol.

- Minimitzar la utilització de la xarxa per afavorir una migració ràpida, que redueixi la possibilitat de migracions fallides, provocades pel moviment constant de les plataformes.

- Evitar un volum de computació excessiu, degut a la potència limitada dels terminals mòbils en els que s'ha d'executar.

- Fer ús de la capa de serveis que es trobarà per sota i es desenvolupa al projecte *Gestió Dinàmica de Serveis* [serveis].

- Cada agent de transport ha de dur només una etiqueta de classe *TriageTag*, creada al projecte *Disseny i implementació d'interfícies del protocol START* [dham].

- El nostre sistema ha de ser transparent al disseny de les etiquetes i a la elecció del protocol de triatge. Un canvi en el protocol no ha d'obligar a la modificació del sistema de transport, i un canvi en el disseny de l'etiqueta només ha de suposar canviar la classe on es troba definida la seva estructura.

3.3 TTR (Time To Return)

El concepte de TTR o Time To Return és de vital importància per a aquest projecte, ja que gran part del desenvolupament gira al seu voltant.

Quan una persona encarregada de la tasca de triatge surt del centre de control i comença el seu recorregut, ha d'indicar quant temps trigarà a tornar. El compliment d'aquest temps és indispensable per a garantir la seva seguretat (no oblidem que aquesta persona es mou en un terreny perillós) i poder detectar les complicacions sorgides mentre porta a terme la seva feina.

El temps de retorn s'introdueix al terminal mitjançant la interfície gràfica i s'ha de guardar al sistema. Assumint que s'acompleixen els TTRs indicats, es pot determinar que si no tenim un centre de coordinació a l'abast, però sí altres plataformes de triatge, la que tingui menor TTR probablement serà des de la que es pugi accedir abans al ECC.

En el context del nostre sistema, si diverses plataformes de triatge amb TTRs es troben en la mateixa MANET, però no existeix cap punt de coordinació, els agents de transport han de migrar cap a la de menor TTR.

3.4 Gestió d'etiquetes de triatge

La etiquetes no han de ser creades pel nostre sistema, donat que aquest només s'encarrega del seu transport. Tot i així és adient estudiar cóm s'han de gestionar des del moment de la seva creació fins a l'entrega al centre de control per aconseguir una idea global del problema.

De la mateixa manera que en el procés de triatge tradicional una víctima era identificada amb una etiqueta de cartró, ara aquesta persona disposarà també d'una

etiqueta electrònica, vinculades les dues mitjançant una RFID. En el procés de guardat de les dades d'un sol pacient, es genera un objecte TriageTag amb la seva informació dins un agent de transport.

Els agents de transport s'han de moure per la xarxa migrant en funció del TTR fins arribar al centre de coordinació, on entreguen la seva etiqueta per tal que la víctima sigui identificada, localitzada i pugui ser tractada el més aviat possible. Evidentment, en el moment que es disposi de un ECC a l'abast, es migra directament cap a ell sense importar els TTRs veïns.

3.5 Anàlisi de software

En el capítol anterior s'ha parlat de JaDE i del mòdul de migració, en aquesta secció es presenta l'anàlisi de projectes relacionats, a tenir en compte al llarg de les fases de disseny i implementació.

3.5.1 Projectes relacionats

Com ja hem dit anteriorment aquest projecte no és independent, es troba emmarcat en un conjunt de projectes que han de formar un de major. Per aquest motiu n'existeixen d'altres directament relacionats amb el nostre, alguns d'ells realitzats amb anterioritat i altres dissenyats en paral·lel. Tot això implica l'existència d'un gran nombre de restriccions per tal d'aconseguir la cohesió necessària, que faci que la unió d'aquest projecte i la resta funcioni com un tot.

Disseny i implementació d'interfícies del protocol START

El projecte, ja finalitzat, que es troba més estretament relacionat amb aquest és el treball *Disseny i implementació d'interfícies del protocol START* [dham] d'en Xavier Jurado, que va implementar la interfície gràfica a utilitzar i la creació d'agents de transport amb les *Triage Tags*. Abans d'endinsar-nos en el nostre disseny, cal fer un anàlisi d'aquest treball per determinar exactament el punt de partida i les restriccions que s'en deriven de la feina ja feta.

Aquest és un resum de les característiques d'aquest projecte:

- Disposa d'un assistent per al protocol de triatge START, que pregunta de forma interactiva les dades que necessita per assignar un estat, mitjançant una interfície gràfica.
- Implementa una segona interfície en la qual un usuari pot entrar de forma manual totes les dades relacionades amb el pacient.
- Crea una estructura capaç d'emmagatzemar les dades de cada pacient, independentment del mètode amb el que han estat recollides, i l'incorpora en un agent mòbil, facilitant una futura integració amb altres sistemes mèdics basats en el mateix paradigma.
- També disposa de gestió de les dades GPS i mètodes per identificar pacients mitjançant RFID.

Tot i que el quart punt no resulta interessant des del punt de vista de la migració d'agents que ens ocupa, ens pot interessar saber que les dades GPS i l'identificador de la RFID associada a la víctima viatjen en els nostres agents, dins de les etiquetes electròniques de triatge.

Amb tot això tenim que: en el nostre projecte no hi ha d'haver interacció directa amb l'usuari perquè es fa a través de la interfície gràfica, l'estructura de l'etiqueta de triatge està definida, hi ha una implementació del protocol START i es crea un agent de transport per pacient triat.

Projecte paral·lel: Gestió Dinàmica de Serveis

La coordinació amb els projectes paral·lels és complicada perquè els canvis en aquests projectes al llarg de la fase de disseny, poden crear la necessitat d'introduir també canvis en el nostre. Cal arribar a un consens de disseny global que garanteixi la independència funcional entre treballs desenvolupats simultàniament.

El projecte desenvolupat de forma paral·lela amb el que ha d'haver una relació més estreta és *Gestió Dinàmica de Serveis* [serveis], en el que es dissenya i crea

una infraestructura de serveis. Els nostre agent de gestió ha de fer ús d'aquesta infraestructura per demanar informació de TTR de les plataformes properes, així com dels centres de control a l'abast.

La comunicació és el punt clau en la interacció entre ambdós sistemes i el primer que cal determinar és si ha de ser entre agents, donat que aquest paradigma de programació estableix sistemes de comunicació molt específics, amb la utilització d'ontologies i missatges ACL.

3.6 Anàlisi de Hardware

El hardware real sobre el que ha de funcionar el sistema un cop finalitzat, són terminals mòbils de Nokia model N810 com el que es pot veure a la figura 3.1, però per al desenvolupament d'aquest projecte no cal fer servir aquestes màquines. Com que disposen d'un sistema operatiu basat en Linux i s'han preparat expressament amb les plataformes de JAVA i JaDE adients, podem fer servir un ordinador convencional que disposi del mateix software. Gràcies a la portabilitat de JAVA tenim la garantia que funcionarà al dispositiu escollit encara que finalment es faci servir un altre diferent del de Nokia.

Realitzar proves resulta complicat, degut al fet que s'ha de fer migrar agents i comprovar si el seu comportament és correcte a les plataformes a les que viatgen. Necessitarem una petita xarxa amb diversos terminals que disposin de la plataforma de JaDE, tant se val si són màquines reals o virtuals. Amb un sol ordinador i diverses màquines virtuals es pot simular el que podria ser una situació real sobre el terreny amb l'emergència.

3.7 Estudi de viabilitat

3.7.1 Viabilitat tècnica

L'equip físic necessari per al desenvolupament és mínim. Com s'acaba d'explicar, només ens calen un parell d'ordinadors o un PC amb algunes màquines virtuals.

Figura 3.1: Nokia N810.

Pel que fa al software, cal un entorn de desenvolupament JAVA i la plataforma de JaDE, amb el servei de migració interplataforma desenvolupat a la UAB.

Els agents mòbils ja han demostrat que, amb el mòdul *Inter-Platform Mobility Service* [mobility], poden saltar de plataforma i en aquest sentit no ha d'existir cap problema, però pel que fa a l'aplicació real del sistema a l'entorn d'emergència, existeixen algunes dificultats tècniques que analitzem a continuació:

- **Tolerància a fallades**. En aquest punt podem de diferenciar entre les fallades en la migració dels agents i les pròpies del dispositiu de triatge. Una migració fallida no representa una amenaça perquè l'agent no s'elimina de l'origen fins que no es rep la confirmació del destí informant que tot ha funcionat correctament. Si el terminal de triatge falla per motius no controlables des del nostre punt de vista, per exemple si l'operatiu es penja, es reseteja o bloca, existeix la possibilitat que els agents amb les etiquetes de triatge es perdin si no estaven guardats en un sistema d'emmagatzemament estable. El grup SeNDA també treballa actualment en un projecte de tolerància a fallades que prova d'eradicar les pèrdues d'agents degudes a fallades del terminal, mitjançant redundància d'informació i emmagatzemament estable.

- **Connexions lentes i desconnexions**. En les xarxes MANET de la zona de l'actuació, els dispositius són visibles els uns amb els altres en funció del moviment del personal de triatge, la distància i els obstacles del terreny. Cal comprovar que el temps en que dos terminals de triatge es veuen és normalment superior al temps que triguen els agents en migrar d'un a l'altre. S'ha de tenir en compte que, abans que es faci la migració de l'agent de transport, s'han de detectar els dispositius, connectar-se i intercanviar les dades necessàries per a que aquest prengui la seva decisió. En l'anàlisi fet a *The Mobile Triage Tag* [mtt] s'ha avaluat el temps emprat en migrar per l'agent de transport amb els mecanismes escollits, i s'ha determinat que aquest oscil·la entre els 3 i 7 segons. També s'ha analitzat el temps de visibilitat entre dos dispositius en funció de la distància i la velocitat d'allunyament i, per al pitjor cas estudiat amb dos persones que es troben a 50 metres i es creuen a 6 km/h en sentits oposats, és de 30 segons.

Sembla que teòricament els problemes mencionats no han de representar un perill per a la viabilitat tècnica del projecte, tot i això, no es pot parlar amb un cent per cent de seguretat fins que no es realitzin simulacions reals del sistema complet.

3.7.2 Viabilitat operativa

La migració d'agents entre plataformes mitjançant una xarxa sense fils ad-hoc, és una bona forma de transportar informació on no hi ha altres possibilitats, com xarxes telefòniques fixes o mòbils. Transportant els *Triage Tags* dins un agent portador no podem garantir una arribada ràpida de les dades al centre de control, cosa que depèn de si en tenim algun a l'abast, si veiem altres plataformes amb temps de retorn baixos... però sí assegurem més velocitat que amb el mètode manual.

És evident que altres punts forts de la gestió electrònica de les etiquetes de triatge són: que no passa víctimes per alt perquè es troben totes localitzades gràcies a les dades GPS, permet la planificació de rutes i s'informa amb exactitud de la

situació als equips de rescat.

Des del punt de vista de la persona encarregada del triatge, que serà l'usuari final, el sistema de transport és totalment transparent i només ha d'interactuar amb la interfície que ja havia estat realitzada en el passat. Cóm es faci la migració no és depenent de l'usuari, ni les seves accions o decisions es veuran afectades en absolut per la utilització del nostre sistema.

Estem parlant d'un sistema operativament viable, però com ja passava amb la viabilitat tècnica, que aquest projecte concret sigui viable operativament no és motiu suficient per a garantir la viabilitat del projecte de caire superior. Aquest estudi, més complex i que contempla molts altres projectes, no forma part del nostre treball.

3.7.3 Viabilitat econòmica

El cost de desenvolupament és baix pel que fa a recursos materials i programari. D'una banda tenim que el software de desenvolupament és gratuït (JAVA, JaDE i un entorn de programació com pot ser Eclipse) i de l'altre tenim que es pot treballar només en un PC.

Per a realitzar proves de migració entre diferents plataformes es pot recórrer a la virtualització, fent servir també software gratuït com VMWARE i màquines virtuals funcionant amb sistemes operatius de la família Linux.

La dedicació en hores de treball pot ser suficient amb el temps marcat per a la dedicació a projectes, sempre i quan no es trobin limitacions o problemes seriosos pel camí i es disposi de un cert grau de coneixement en programació d'agents mòbils i la seva comunicació.

Si pensem en el cost d'aplicació en actuacions d'emergència reals, ens cal equipar cada membre de l'equip de triatge amb un terminal i altres recursos addicionals que no tractarem. Donem per fet que aquest cost és assumit pel projecte que ens engloba i no és necessari avaluar-lo de nou en un sistema que només està destinat al transport de les etiquetes de triatge. Tanmateix, no cal cap mena de formació específica de més per al personal encarregat d'etiquetar les víctimes.

3.7.4 Viabilitat legal

Un problema legal que es planteja és el de la privacitat de les dades que viatgen en els agents. Aquestes dades contenen informació de les víctimes i poden arribar a contenir informació sobre malalties infeccioses dels pacients, que són rellevants de cara al seu tractament (com pot passar amb la SIDA o l'hepatitis per exemple).

Només el personal específicament dedicat al rescat portarà terminals amb el software i la configuració necessaris per a que l'agent de transport s'hi pugui moure. Difícilment podrà arribar una etiqueta de triatge a mans no desitjades, així que l'encriptació o codificació de les dades emmagatzemades als agents no es contempla en aquest projecte i no sembla, a priori, necessària.

3.8 Planificació temporal

La fase prèvia, consistent en la recerca d'informació per a dur a terme aquest projecte, va començar el mes de novembre de 2009 i es va basar en l'estudi de programació amb agents. La data d'inici correspon al dia vint de febrer de 2009, tot just després dels exàmens del primer semestre a la nostre facultat.

La planificació inicial es pot veure en el diagrama de Gantt de la figura 3.2 i les etapes de desenvolupament proposades es poden resumir de la següent manera:

1 **Anàlisi**: estudi del problema, del hardware i software necessari, dels projectes relacionats, de viabilitat i descripció de requeriments.

2 **Disseny**: disseny del sistema d'agents mòbils i de relacions amb l'entorn, ubicació a l'esquema de capes i generació de casos d'ús.

3 **Desenvolupament**: implementació del sistema definit a la fase de disseny.

4 **Prova**: tests de verificació funcional.

5 **Aplicació i integració**: unificació del projecte amb d'altres relacionats, proves d'integració i estudi de millores.

6 **Redacció de la memòria.**

7 **Creació de la presentació** per a l'exposició oral.

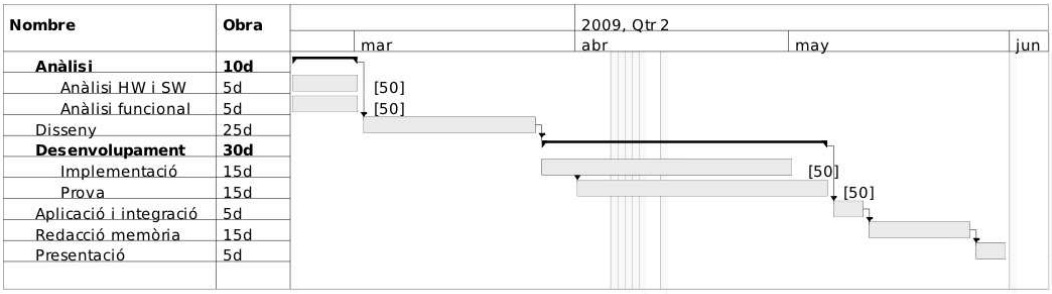

Figura 3.2: Planificació temporal inicial.

Tot i que s'indiquen temps molt concrets per cadascuna d'aquestes fases del treball al diagrama de Gantt de la figura 3.2, les etapes d'anàlisi i disseny són dependents de projectes paral·lels del grup SeNDA i, per aquest motiu, resultava complicat determinar exactament la seva durada abans de començar.

La planificació temporal final ha estat diferent a la inicial, allargant-se notablement la fase de disseny com a conseqüència de les dificultats derivades de la dependència entre els projectes del grup. En el capítol de *conclusions* d'aquesta memòria s'explica amb més detall la planificació que s'ha dut a terme finalment i els motius pels que s'ha decidir retardar l'entrega del projecte al setembre.

Capítol 4

Disseny

En aquest capítol de la memòria explicarem cóm s'ha definit el sistema en funció de l'anàlisi realitzat, començant per un enfocament global i fins arribar al detall.

El projecte *The Mobile Triage Tag* [mtt] es troba dividit en nombrosos projectes més petits dels que ens interessen especialment els de: detecció de dispositius presents a la xarxa MANET, creació de la infraestructura de serveis, mobilitat d'etiquetes de triatge (el nostre) i creació de la interfície gràfica i implementació del protocol START.

Tots aquests subprojectes dibuxen un esquema de capes que disposa de capa de xarxa, capa de serveis i capa d'aplicació. A més a més, com que parlem d'un sistema d'agents, necessitem una capa més que ens ofereixi aquesta infraestructura: la capa de JaDE.

Mobilitat d'Etiquetes de Triatge és un projecte a desenvolupar en la capa d'aplicació, cosa que implica la no existència d'una capa superior amb la que hagi de existir una relació directa, però per sota sí ens cal la capa de serveis i la de JaDE. En la figura 4.1 podem veure l'esquema de capes, quedant per sota la capa de xarxa amb la que no interactuarem directament.

En la capa d'aplicació també es troba el projecte *Disseny i implementació d'interfícies del protocol START* [dham], per fer una unió satisfactòria d'ambdós treballs cal determinar cóm podem incorporar la interfície gràfica i cóm podem aportar mobilitat i capacitat de decisió als agents de transport, generats amb les

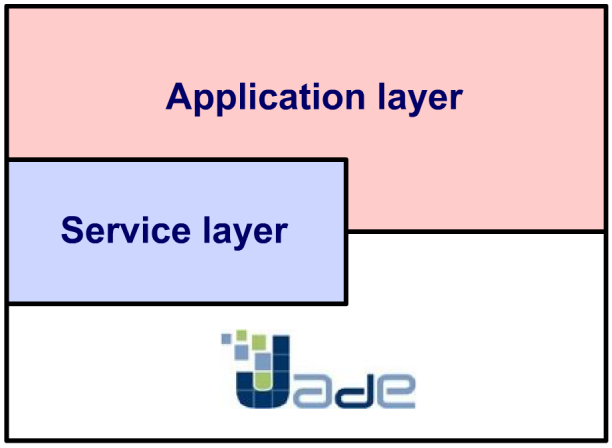

Figura 4.1: Estructura de capes del sistema.

dades que s'introdueixen des de la *GUI*.

La infraestructura de serveis ens proveirà d'alguns imprescindibles, com és el cas del servei de TTR amb el que podem obtenir els temps de retorn de les plataformes veïnes i el ECC per conèixer els centres de coordinació disponibles.

4.1 Agents i comportaments

Hem de tenir agents de transport d'etiquetes de triatge que arribin a l'*Emergency Coordination Centre* de forma autònoma, que reben el nom d'**Electronic Triage Tag Mobile Agent** (ETTMA) i tenen un comportament que els aporta l'autonomia per escollir el seu destí i entregar les dades de la víctima en arribar a un ECC. Com ja sabem, en una mateixa plataforma hi poden conviure diversos agents ETTMA, cadascun amb les dades d'un pacient.

Per coordinar i tenir control sobre els ETTMAs, necessitem l'agent anomenat **Manager Agent** (MA), del que només en pot existir un a cada plataforma. Una de les tasques més importants d'aquest agent és la d'informar als ETTMAs dels ECCs diponibles o dels temps de retorn de les plataformes properes, tenint en compte les variacions degudes a l'aparició o desaparició de dispositius de triatge a la xarxa, i les modificacions introduïdes pels usuaris fent ús de la *GUI*. Tota la

informació procedeix de la infraestructura de serveis, el MA s'ha de comunicar amb el **Service Manager Agent** (SMA), que és l'agent encarregat d'oferir serveis, per aconseguir-la. L'agent de gestió també ha de disposar de la interfície gràfica del sistema i serà el creador d'agents ETTMA.

En la figura 4.2 es mostra cóm es situen els diferents agents a les capes del sistema i els canals de comunicació que s'estableixen entre ells.

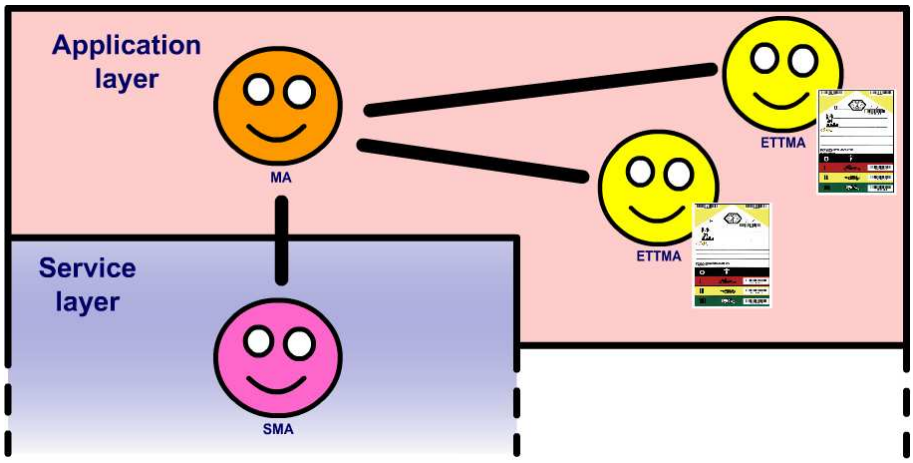

Figura 4.2: Canals de comunicació entre agents.

En vista del que estem descrivint, pot semblar més simple fer que els agents ETTMA consultin directament al SMA en comptes de haver de passar per l'agent de gestió MA, però aquesta no és una bona opció per diversos motius:

a El SMA concentrarà un conjunt ampli de serveis, no només els de TTR i ECC, i haurà d'interactuar amb molts altres agents de sistemes diferents. Fer això faria créixer el nombre d'agents que ha d'atendre, empitjorant el seu temps mitjà de resposta.

b Augmentarien els missatges de petició d'informació al SMA i amb això creix també la mida de la cua d'entrada de missatges i el temps de procés dedicat a la seva atenció.

c El MA allibera de la gestió dels agents de transport al SMA i és el que garanteix la separació entre les capes de servei i aplicació, aconseguint que

si el nostre sistema ha de ser modificat o desactivat, no calgui canviar res a la capa de servei.

4.1.1 Electronic Triage Tag Mobile Agent (ETTMA)

Els **agents de transport** ETTMAs porten etiquetes de triatge fins al centre de coordinació i s'en genera un cada vegada que es tria una víctima. El comportament que hem dissenyat per aquests agents té cinc estats, com es pot veure a la figura 4.3.

a **Estat inicial** *(Init State)*: l'agent identifica la plataforma en la que es troba i pregunta al Manager Agent per la situació de la xarxa, és a dir, si existeixen ECCs a l'abast i quin és el TTR dels veïns. El MA sempre respondrà a la pregunta amb un missatge que conté informació sobre els centres de control disponibles si n'hi ha, altrament ho fa amb dades relatives als temps de retorn de les plataformes properes.

b **Espera resposta** *(Wait for MA Info)*: en aquest estat roman en espera d'un missatge de resposta per part del MA, quan el rep, n'agafa la informació i passa a l'estat de decisió de destí. Un agent ETTMA pot arribar a aquest estat per tres motius diferents: ha enviat una sol·licitud d'informació al MA i espera resposta, ha provat de migrar n vegades sense èxit o bé la plataforma escollida a l'estat de decisió és l'actual. En qualsevol d'aquests casos ha d'esperar per nous missatges que informin dels canvis a la xarxa.

c **Estat de decisió** *(Decision State)*: es determina el destí de la migració. Si s'havia rebut informació sobre centres de coordinació ECC s'agafarà com a proper destí un dels presents a la llista, mentre que si s'havien rebut dades de temps de retorn, es decidirà saltar a la plataforma amb menor TTR.

d **Estat de migració** *(Migration State*): en aquest estat es salta a la plataforma escollida i es retorna a l'estat inicial, de fet, després d'un intent de migració es reseteja el comportament de l'agent, per tant aquest canvi d'estat és au-

tomàtic. Si no s'aconsegueix migrar cal tornar a aquest estat per intentar-ho de nou fins a n vegades.

e **Entrega de dades** *(Data Delivery State)*: si acabem d'arribar a un centre de control, cosa que podem saber mirant quin ha estat el criteri de decisió que ha seguit l'agent per moure's (en aquest cas ECC), passem a l'estat d'entrega de l'etiqueta.

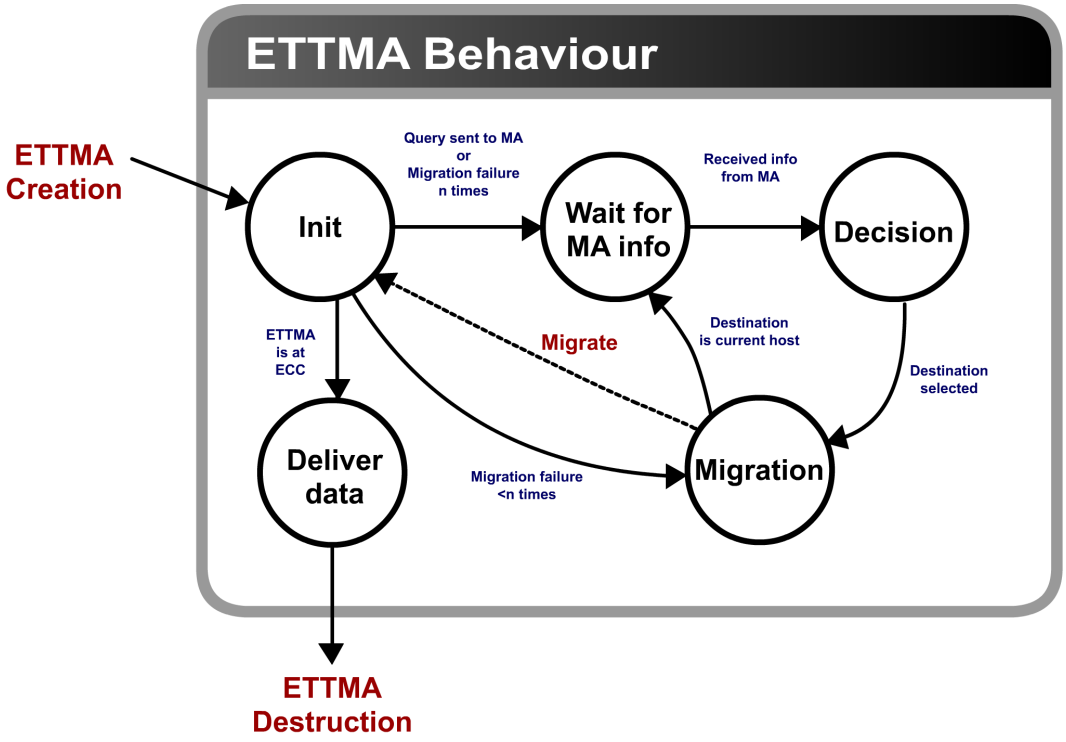

Figura 4.3: Comportament de l'agent ETTMA.

Elements de l'agent ETTMA

L'agent ETTMA ha de disposar d'una etiqueta de triatge electrònica amb les dades de la víctima, juntament amb una llista de centres de coordinació a l'abast i una amb el temps de retorn dels dispositius que l'envolten. Les llistes de TTR i ECCs

s'omplen amb les dades rebudes de l'agent MA i serveixen per decidir el destí de la migració.

L'etiqueta de triatge es trobarà present a l'agent des del moment de la seva creació. Inicialment es va pensar que podria ser modificada des de la interfície gràfica, però el tema de les migracions dificulta aquesta possibilitat. Si l'agent encara resta a la plataforma de creació es podria modificar, però si aquest ha migrat l'etiqueta ja no es troba present. Evidentment, modificar l'etiqueta d'un agent una vegada ha migrat, en una plataforma en la que no ha estat generada, és incorrecte perquè un membre del personal de triatge no ha de poder modificar dades d'una víctima que no ha vist.

Noms per als agents ETTMA

Una de les restriccions importants de JaDE, és que els identificadors dels agents han de ser únics en una mateixa plataforma, d'aquesta manera s'eviten ambigüitats a l'hora d'identificar els que s'hi troben presents.

Si dos agents ETTMA tenen el mateix nom, poden existir en plataformes diferents però no en la mateixa, per tant, qualsevol migració d'un agent cap a una plataforma on ja existeixi un amb el seu nom, serà fallida.

La solució per aquest problema passa per donar noms únics als agents en el moment de la seva creació, per exemple a partir del nom de la plataforma on s'ha creat més un comptador, que s'anirà incrementant cada vegada que es generi un nou ETTMA. El problema que ens trobarem ara és que, al treballar en xarxes ad-hoc amb topologia variable en el temps, existeix també la possibilitat de que puguin coincidir dues plataformes amb el mateix nom en la xarxa.

Els tema de noms de plataforma duplicats no es troba contemplat en aquest treball, ja que és estudiat i tractat en la capa de xarxa del nostre model. Aquí suposarem que aquests noms son únics en tot el recinte on es produeix l'actuació.

Una altra possibilitat per als noms dels ETTMAs és la de fer servir l'identificador de la RFID associada a la víctima, que es troba dins de l'etiqueta de triatge electrònica i és únic al món.

4.1.2 Manager Agent (MA)

L'agent de gestió MA té diversos rols. No només informa als ETTMAs de la situació de la xarxa sinó que també ha de portar el control dels que resten a la seva plataforma, ha de rebre informació de TTRs i ECCs comunicant-se amb l'agent SMA, ha de mostrar la interfície gràfica i ha de crear nous agents ETTMA a partir de les dades introduïdes en la mateixa.

Una altre tasca que realitza el MA és notificar al SMA el TTR local, que introdueix l'usuari mitjançant la *GUI*, per a que aquest el posi en comú amb els altres SMA de plataformes remotes. Un detall important a tenir en compte és que, en canviar el TTR, es notifica al SMA i aquest ens envia el missatge amb el nou estat de la xarxa, encara que només hagi variat el nostre TTR local. El MA no envia dades als ETTMAs que no procedeixin de la capa de serveis.

L'agent de gestió MA demana pels serveis de TTR i ECC al de serveis, indicant que vol ser notificat sempre que es produeixin variacions. A efectes pràctics l'agent de gestió queda subscrit als serveis de TTR i ECC, que ofereix la infraestructura de serveis mitjançant l'agent SMA.

En el nostre disseny el *Manager Agent* té tres comportaments diferents, que són executats en paral·lel, per acomplir amb les seves tasques.

Primer comportament: atendre les peticions dels ETTMAs.

Quan es crea un agent ETTMA o quan aquest prové d'una migració, pregunta al MA per les dades ECC i TTR. Per aquest motiu, s'ha d'estar pendent de les consultes d'aquest tipus i s'ha de poder respondre.

Es pot pensar en passar aquesta informació als agents de transport generats localment en el mateix moment de la seva creació, però és preferible fer-ho sempre amb consultes per tractar exactament igual els agents de nova creació i els que procedeixen d'una migració des d'un altre plataforma.

Segon comportament: rebre noves dades des de la infraestructura de servei i informar als ETTMAs.

El segon comportament interactua amb l'agent SMA de la capa de serveis de diverses formes: enviant el TTR cada vegada que sigui modificat per l'usuari, cosa que no serà habitual i es produirà normalment cada vegada que la persona encarregada del triatge surti del centre de coordinació; subscrivint-se al servei de TTR i ECC del SMA i rebent les dades procedents del mateix.

Cal estar alerta de les notificacions d'actualització de TTR i ECC i informar als ETTMAs que es trobin encara presents en la plataforma. Evidentment, hem de tenir un control dels agents ETTMA per evitar enviaments cap a els que hagin marxat.

Aquest comportament té quatre estats, com es mostra a la figura 4.4.

a **Enviament de TTR** *(TTR Notification State)*: l'agent de gestió ha d'informar al de serveis del TTR de la plataforma local.

b **Estat de subscripció al SMA** *(SMA Subscription State)*: el MA es subscriu al servei de notificació de TTR i ECC que ofereix la capa de serveis mitjançant el seu agent SMA.

c **Estat de recepció de dades** *(Listening State)*: el MA roman a l'espera de nova informació procedent de la capa de serveis, quan la rep actualitza la informació de TTR i ECC de que disposa i passa a l'estat en el que informa als ETTMAs.

d **Estat d'enviament d'informació als ETTMAs** *(Send Info State)*: s'informa als agents ETTMA que encara no han migrat a un altre dispositiu de la nova situació de la xarxa, després es torna a l'estat de recepció de dades.

Si l'usuari introdueix un nou TTR des de la interfície gràfica, independentment de l'estat en que ens trobem, es torna a l'estat d'enviament de TTR. Podem resetejar el comportament de manera immediata perquè no és rellevant si es perd algun missatge procedent del SMA o les dades que contingui, donat que aquestes

ja no es trobaran actualitzades. Després de que el SMA hagi processat i compartit la nova informació, ens enviarà la nova situació de la xarxa tenint en compte el nou TTR.

L'agent MA pot recordar si s'havia subscrit als serveis per evitar passar pel segon estat d'aquest comportament cada vegada que es canviï el TTR.

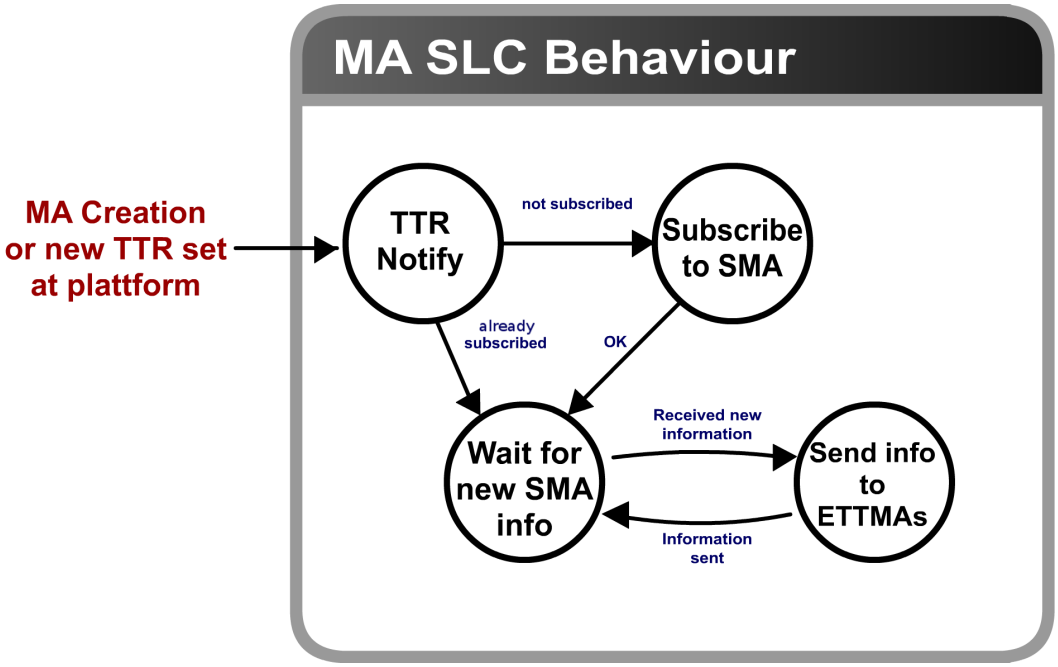

Figura 4.4: Segon comportament de l'agent MA.

Tercer comportament: mostrar la interfície gràfica i generar nous agents.

La creació d'agents de tipus ETTMA i la interfície gràfica ja es troben implementades, en el nostre disseny la *GUI* només s'ha d'afegir a l'agent dins un comportament. Els detalls d'implementació i integració es poden trobar en el capítol següent de la memòria.

Seqüències d'execució

En la figura 4.5 es veu la seqüència d'events que s'ha de produir quan s'introdueix un nou TTR, en aquest cas es tracta de la primera vegada que es notifica el TTR al sistema fent ús de la interfície gràfica, recordem que tota persona dedicada a la tasca de triatge ha de notificar el seu temps de retorn obligatòriament abans de sortir del centre de coordinació.

Si no és la primera vegada que es notifica el TTR (potser és la segona vegada que l'usuari surt del centre de coordinació) la subscripció al servei del SMA que es mostra no és necessària.

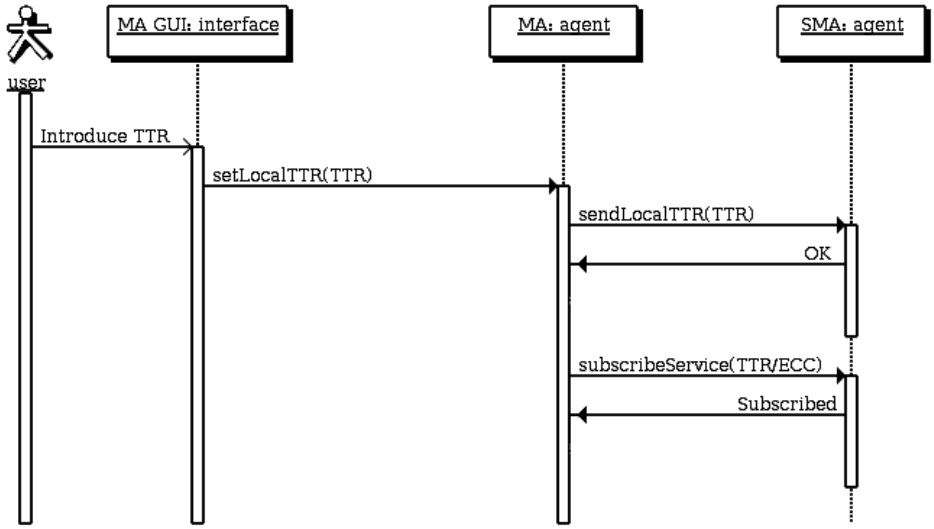

Figura 4.5: Diagrama de seqüència, nou TTR.

En la figura 4.6 es veu la seqüència de creació d'un agent ETTMA amb l'etiqueta de triatge associada a una víctima. L'usuari interactua amb la interfície gràfica, que delega la responsabilitat de la creació de l'agent i el *Triage Tag* al MA. Una vegada creat aquest agent ETTMA, consulta l'estat de la xarxa i decideix si ha de migrar o no en funció de la informació rebuda. Quan un agent ETTMA arriba a una plataforma nova que no és centre de coordinació, repeteix els passos de demanar informació sobre l'estat de la xarxa i decidir si ha de migrar.

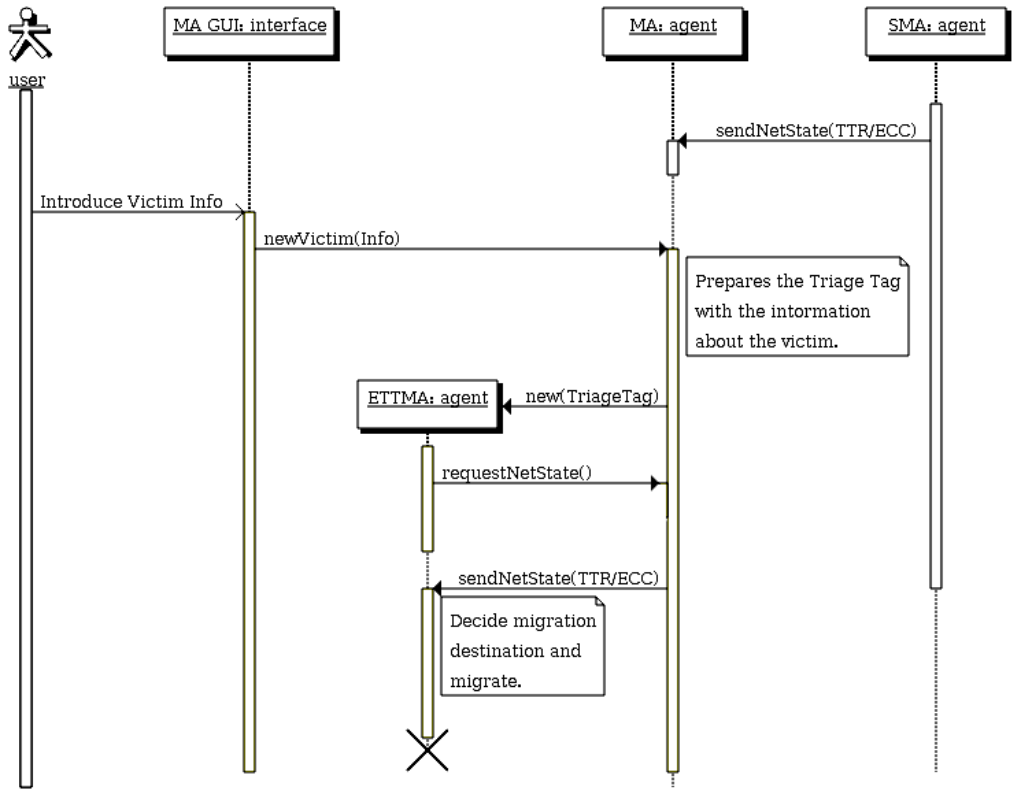

Figura 4.6: Diagrama de seqüència, introducció de dades d'una víctima.

Control d'agents presents a la plataforma

Imaginem que un agent ETTMA decideix no migrar i roman a la espera de noves informacions a la plataforma en la que es troba, independentment de si és la plataforma on ha estat creat o hi ha arribat des de una diferent.

Quan canviï la topologia de la xarxa, MA serà informat i ha de notificar les variacions produïdes a tots els agents de transport presents per a que puguin avaluar la nova situació. Per fer això ha de conèixer els agents de transport presents, per exemple fent servir una llista amb els identificadors d'agents coneguts. Cada cop que un agent ETTMA demana informació, MA guarda el seu identificador.

Recordar els agents presents és una tasca simple si pensem en un entorn estàtic però quan es poden produir migracions la cosa es complica per diversos motius:

a **L'agent de transport és lliure de decidir si migra o no**. Si decideix migrar podria informar al MA de la seva decisió per fer que aquests l'esborrés de la llista.

b **L'agent de transport pot no aconseguir una migració satisfactòria**. Tot i que l'agent estava decidit a migrar, no ho aconsegueix per motius diversos, com per exemple una fallida en la comunicació. Si aquest agent havia notificat que marxava haurà estat esborrat de la llista però, si no ho ha aconseguit, restarà inadvertit en el futur. Com es pot veure amb aquest exemple, la sol·lució del punt anterior no és bona.

És millor que els agents ETTMA no siguin els que informin directament de si marxen o no. JaDE disposa d'un servei de pàgines blanques en l'agent **Agent Management System** (AMS), on es registren tots els agents actius a la plataforma. Es pot aprofitar aquest servei per preguntar si els agents de la llista encara existeixen localment abans d'enviar informació.

Reducció de TTRs

Quan l'agent SMA de la infraestructura de serveis envia un missatge cap a l'agent de gestió MA amb nova informació de ECCs i TTRs, aquest la guarda i la torna a enviar cap als agents ETTMA presents.

Suposem que no tenim cap centre de coordinació a la vista però sí altres dispositius amb temps de retorn diferents. Passats uns instants des de l'arribada del missatge provinent del SMA, l'usuari que està fent les tasques de triatge, introdueix les dades d'un nou pacient i es genera una nova etiqueta i un nou agent ETTMA. Si l'agent de gestió MA li passa les dades que tenia d'abans, els temps de retorn ja no es trobaran actualitzats.

Per evitar aquest problema podem plantejar tres opcions:

a El MA ha de guardar l'instant de temps en el que es troba quan rep nova informació des del SMA i, abans de notificar a un agent ETTMA qualsevol, ha de calcular la diferència entre aquell instant i l'actual, restant-lo als temps

de retorn guardats. Això evita una notificació incorrecta de TTRs i rep el nom de **reducció de TTRs**.

b La segona opció és la de no guardar el TTR com a temps que resta per tornar al centre de coordinació sinó com la data i hora en la que s'arribarà en forma de **temps absolut**. D'aquesta manera, no cal aplicar la reducció de TTRs però com a contrapartida tenim la necessitat de sincronitzar l'hora a tots els terminals de triatge.

c La tercera i més simple és la de **no fer res**. Com que no hi ha noves notificacions des del SMA l'estat de la xarxa no ha canviat, aleshores la diferència que hi havia entre els TTRs encara és la mateixa (el més petit encara ho és). Aquesta opció té el problema que l'agent ETTMA no és conscient del temps real de tornada al centre de coordinació, només de l'ordre d'arribada de les plataformes i tampoc ens permetrà detectar TTRs anòmals.

Si un TTR de la nostre xarxa arriba a zero, o si l'hora d'arribada ja ha passat i no hi ha cap ECC a l'abast, tenim un **TTR anòmal**. Aquest fet implica que la persona encarregada d'aquell terminal no ha respectat el seu temps de retorn, cosa que pot ser conseqüència involuntària d'un accident o una dificultat. Podem intentar detectar aquestes situacions i implementar una alarma que avisi de membres de l'equip amb problemes.

Elements de l'agent MA

L'agent ha de disposar de tres elements principals, en forma de llista, per dur a terme la seva tasca en relació amb la mobilitat dels agents de transport.

- Una llista amb els **centres de coordinació disponibles** (ECC list), que només tindrà entrades si hi ha ECCs a la nostre xarxa i serà enviada als ETTMAs com a primera opció.

- Una amb els **temps de retorn dels dispositius disponibles a la xarxa** (TTR list) per si no es pot accedir directament al punt de coordinació però hi ha

altres plataformes amb TTR. Aquesta llista s'enviarà als ETTMAs si no hi ha dades ECC, per a que puguin avaluar si és més convenient marxar a una plataforma diferent.

- Una amb els **identificadors dels agents de transport coneguts** (ETTMA list) que es troben a la plataforma actual esperant informació i que han de ser notificats en rebre noves dades des del SMA.

Evidentment, el MA també ha de disposar de la interfície gràfica del sistema però no entrarem en detalls del disseny d'aquesta part, que ja es troba implementada i no ha de ser modificada en el nostre projecte.

4.2 Disseny per la fase de proves: el Dummy SMA

El disseny de **l'agent de gestió de serveis SMA** no forma part d'aquest projecte, però com que sí hem d'interactuar amb ell, serà necessària fer una implementació parcial per a poder realitzar proves de funcionament del sistema.

En aquest punt **no parlarem d'un agent SMA complet**, sinó d'un que anomenarem *Dummy SMA*, que només disposarà de les funcions bàsiques que necessitarem per a la relització dels tests.

Comportament de l'agent Dummy SMA.

Com ja sabem, les relacions entre l'agent MA i l'agent SMA s'estableixen en tres punts: el MA li notifica el TTR de la plataforma en la que ambdós es troben, el MA li demana pel servei de notificació de TTRs i ECCs i el SMA envia la informació sol·licitada cada vegada que hi hagi canvis.

Per aquest motiu, *Dummy SMA* ha d'incorporar un comportament de tres estats, un per cadascun dels punts descrits. Cal dir, però, que el comportament mostrat a continuació no funcionarà pel SMA real, ja que és una simplificació al mínim necessari per a poder fer proves.

a **Espera notificació de TTR** *(Wait TTR info State)*: l'agent espera rebre la notificació del TTR local, quan rep aquesta informació la incorpora a la seva

llista de TTRs i passa a l'estat següent. En l'SMA real, sempre s'ha d'estar escoltant per noves notificacions de TTR des del MA, no només una vegada com es fa en aquest cas, i també s'ha de passar la informació als SMA de les altres plataformes de la xarxa.

b **Espera subscripció del MA** *(Wait MA Subscription State)*: abans d'enviar cap mena d'informació cap a l'agent de gestió, s'espera que aquest demani el servei.

c **Envia informació** *(Info Sending State)*: una vegada es té constància que existeix un MA que vol informació de TTRs i ECCs, SMA envia informació periòdicament fent algunes modificacions per forçar els ETTMAs a moure's. En el SMA real només es realitzen els enviaments de dades quan hi ha canvis, no pas de forma periòdica.

Elements de l'agent Dummy SMA

L'agent Dummy SMA que haurem d'implementar només ha de disposar d'una llista de TTRs i una d'ECCs. L'actualització d'aquestes dues llistes es realitzarà de forma simulada, amb una modificació dels seus valors en funció del temps a l'estat *Info Sending State* del comportament.

4.3 Estructura de les llistes de TTR i ECC

Al llarg d'aquest capítol, hem parlat de la necessitat de guardar informació sobre les plataformes que ens envolten i els seus temps de retorn, així com els centres de coordinació a l'abast. Hem mencionat també que aquestes dades es guardaran en forma de dues llistes diferents.

Resulta evident que per a cada entrada de la **llista de TTRs**, que es correspon amb una plataforma, necessitarem com a mínim les dades per a poder migrar i el temps que li resta per a tornar al ECC. Així dons, una entrada a la llista TTR ha de tenir el nom de la plataforma a l'abast, la IP corresponent i el temps de retorn al centre de coordinació.

En el cas de la **llista de ECCs** no ens cal temps de retorn, donat que el centre de coordinació és el destí final de l'agent de transport i hi saltarà en el mateix moment que tingui coneixement de la seva existència. La informació mínima necessària per a identificar un ECC és el seu nom de plataforma i la IP associada.

4.4 Missatges i ontologia

Per establir comunicació entre els diferents agents que formen part del sistema calen missatges ACL simples com són els de sol·licitud d'informació, la notificació de TTR i la subscripció al SMA, però també alguns de més complexos per a transmetre l'estat de les plataformes veïnes mitjançant les llistes de TTRs i ECCs.

A JaDE, la forma més simple d'enviar objectes és utilitzar cadenes de text per a representar el seu contingut en els missatges, cosa que implica la necessitat de parsejar les cadenes per a poder extreure informació dels objectes enviats i pot arribar a ser molt complicat depenent del nombre i complexitat dels camps.

Per enviar estructures complexes com les llistes TTR i ECC, necessitarem definir els objectes a transmetre amb una ontologia. En el capítol d'*estat de l'art* d'aquesta memòria hem parlat de les ontologies en JaDE com a mitjà de comunicació entre agents, i hem explicat cadascun dels elements de que es composen. Sense entrar ara en detalls d'implementació, veiem cóm podem crear l'ontologia per a enviar els objectes amb les nostres llistes de TTRs i ECCs.

Necessitarem els **conceptes** següents: entrada de la llista de TTR amb els seus atributs, llista de TTR, entrada de la llista ECC amb els seus atributs i llista ECC. Els conceptes representen estructures amb diversos atributs i no poden aparèixer directament als missatges, per tant cal que estiguin inclosos en altres elements com ara predicats. Farem servir un **predicat** amb el nom genèric de *informació* que inclourà els conceptes mencionats per a poder passar llistes de TTRs i ECCs entre els agents.

Capítol 5

Implementació, integració i prova

A continuació es mostra cóm s'ha dut a terme les fases d'implementació, integració i prova del disseny explicat en el capítol anterior. Primer realitzem una introducció mostrant l'entorn de programació escollit i els motius, juntament amb l'explicació de tota una sèrie de conceptes importants en la programació d'agents; després veurem la implementació realitzada del sistema començant des d'un punt de vista global i centrant progressivament l'atenció en els detalls, algunes recomanacions per a la integració i, finalment, cóm s'han realitzat les proves de funcionament.

5.1 Introducció

5.1.1 Entorn de programació

Abans de començar a codificar, es va estudiar quin és l'entorn de desenvolupament més adient i còmode per a generar el nostre codi de la forma més fàcil, ràpida i segura possible. Disposant de les eines adequades des de bon principi ens podem estalviar maldecaps i pèrdues de temps, especialment les degudes a incompatibilitats o manca d'alguna de les utilitats necessàries per a les fases de creació i prova del sistema.

Des de bon començament, l'entorn d'**Eclipse** ha estat el candidat principal, tenint en compte que la programació es realitza en JAVA i desestimant l'opció

d'escriure el codi amb un editor de text convencional, que és lenta i més propensa a l'aparició d'errors.

La principal avantatge de fer servir l'entorn Eclipse és que ja l'he fet servir amb anterioritat i estic familiaritzat amb la disposició i utilització dels seus elements, a més a més, aquests son també altres motius pels quals l'ús d'Eclipse és adequat per al desenvolupament del nostre projecte:

- És pensat i dissenyat per a treballar amb JAVA.
- Facilita la feina del programador amb la supervisió de codi, la notificació d'errors i variables mortes i amb les diferents vistes que presenta.
- Permet la configuració de repositoris per al nostre software.
- Podem guardar els paràmetres de diferents execucions per a realitzar proves sobre el nostre sistema sense dificultat i de forma gairebé immediata (*Run as...*).
- Existeix una gran quantitat de plugins compatibles, com per exemple eUML2 de Soyatec que hem fet servir per a generar els diagrames de classe.

Resulta realment útil el fet de poder configurar plenament l'entorn en el que treballarem des de l'inici del desenvolupament, evitant tasques repetitives cada vegada que ens posem a treballar, com poden ser configurar el *Classpath* o recordar quins serveis de JaDE havíem d'arrencar exactament en iniciar la nostre plataforma per a fer proves.

5.1.2 Versions del software

En el desenvolupament d'aquest projecte s'han fet servir eines com Eclipse, eUML2 o Quick Sequence Diagram Editor, però la base software fonamental la conformen JAVA, JaDE i l'afegit de mobilitat entre plataformes. Les versions utilitzades han estat les següents:

- *JAVA 6* del paquet *java-6-sun-1.6.0.14*

- *JaDE 3.6.1*
- *Inter-Platform Mobility Service: IPMS-v.1.104 for JaDE 3.5 and 3.6*

5.1.3 Creació i execució d'agents

Tots els agents JaDE deriven de la classe **jade.core.Agent**. Per a dissenyar un agent propi, que faci allò que desitgem, només cal crear la classe per al nou agent derivada de la superclasse *Agent* i afegir els comportaments adients. En aquest projecte necessitem tres classes per a tres agents diferents: ETTMA, MA i SMA.

En cada plataforma JaDE, existeixen alguns agents especials que no són creats pel programador, dels quals ens interessarà especialment el **Agent Management System** (AMS), que exerceix un control d'accés i utilització de la plataforma i ens proveeix del servei de pàgines blanques. Tots els agents locals, ja estiguin actius o en espera, són coneguts i registrats a la plataforma pel AMS.

En el procés de creació d'un agent nou es produeixen automàticament una sèrie de tasques: es crida al seu constructor, es crea un identificador anomenat **JaDE Agent Identifier** (AID), es registra en el AMS i es comença a executar el seu mètode *setup()*. En el codi del mètode podem afegir les tasques i els comportaments o *behaviours* que executarà l'agent al llarg de la seva vida amb els dos mètodes *addBehaviour(Behaviour)* i *removeBehaviour(Behaviour)*.

Per a executar un agent, ho podem fer de tres maneres diferents:

- Des de la línia de comandes amb **java jade.Boot -container <nomAgent>:<classe>**.
- Aprofitant la **interfície gràfica de JaDE** que es pot veure a la figura 2.2.
- Creant un nou agent des d'un altre amb el mètode **createNewAgent(nickName, className, args)** del contenidor d'agents.

En el nostre projecte la inicialització dels agents MA i SMA es fa des de la línia de comandes en el mateix instant que es posa en marxa la plataforma, mentre que els ETTMAs s'inicien tant des de línia de comandes com des de la interfície gràfica de JaDE. Una vegada es produeixi la integració amb la resta de

projectes, la creació de ETTMAs es farà amb el tercer mètode mencionat, des del comportament del MA que ha de disposar de la interfície gràfica.

5.1.4 Comportaments

Els comportaments o *behaviours* fan referència a una funcionalitat que incorpora un agent amb les tasques o serveis que realitza. Cada tasca de l'agent serà una instància d'una classe que ha d'heretar de **jade.core.behaviours.Behaviour**.

Per a poder programar agents basats en comportaments hem d'estudiar què ha de fer el nostre agent, associar cada tasca amb un comportament i escollir quin tipus de *behaviour* és el més adient d'entre els que hi ha disponibles. De la planificació temporal de cadascun dels comportaments a la CPU durant l'execució s'encarrega JaDE, amb una política round-robin sobre una cua *FIFO* de *behaviours* actius.

Tot comportament propi, que hereta de *Behaviour*, ha d'implementar els mètodes:

- **action():** inclou el codi de les accions a realitzar en el comportament i s'executa de manera atòmica.
- **done():** invocat al finalitzar el mètode *action()*, determina si el comportament ha estat completat o no.

El paquet *jade.core.behaviours* conté les classes que s'utilitzen per a implementar comportaments, entre elles hi trobem les que ens serveixen per implementar els següents:

- **SimpleBehaviour**: comportament simple que s'executa de forma atòmica.
- **OneShotBehaviour**: només s'executa una vegada i sense interrupció.
- **CyclicBehaviour**: s'executa de manera cíclica, estant actiu tant de temps com ho estigui l'agent. S'ha de evitar que s'apoderi de la CPU per complet i no necessita el mètode *done()*.

- **Comportaments compostos**: SequentialBehaviour, ParallelBehaviour i FSMBehaviour.

- **Comportaments temporals**: TickerBehaviour i WakerBehaviour permeten executar operacions en determinats instants de temps.

- **Comportaments personalitzats**: podem crear comportaments personalitzats fent servir una variable d'estat i un *switch*, de manera que només executarem la porció de codi del *behaviour* corresponent a l'estat en el que ens trobem. Això permet simular una màquina d'estats de manera més simple que amb el *FSMBehaviour*, on cada estat és un comportament.

De tots aquests, n'hi ha uns quants que ens resulten d'especial interès: **ParallelBehaviour** permet dur a terme paral·lelament dos o més comportaments (recordem que el nostre agent MA ha de gestionar tres tasques diferents alhora), **CyclicBehaviour** ens permet anar repetint les mateixes accions de forma cíclica, **SimpleBehaviour** és adient per als ETTMAs i, també farem servir l'esquema de comportament personalitzat.

Es va fer una primera implementació dels *behaviours* per a tots els agents del nostre sistema en classes separades, dins el paquet *mabett.behaviours*, però posteriorment es va apreciar que la interacció entre agent i comportament és tan forta pel que fa a variables i estructures comunes, que resultava molt més eficient i pràctic escriure els comportaments dins la classe de l'agent, així que finalment es va codificar aquesta opció.

En la fase de disseny ja hem determinat les accions que han de realitzar els nostres agents i hem separat els comportaments segons les tasques a dur a terme. Més endavant s'explica amb detall cóm queden codificats, el tipus de comportament escollit en cada cas i els motius d'aquesta decisió.

5.1.5 Missatges

La comunicació entre agents a JaDE es realitza mitjançant el llenguatge **Agent Communication Language** (ACL) seguint les especificacions FIPA, que permet

transmetre coneixements entre agents expressats amb un llenguatge de continguts. Tots els missatges ACL són instàncies de la classe **jade.lang.acl.ACLMessage**.

En un missatge ACL trobem diversos camps a omplir abans de poder realitzar la transferència d'informació entre dos agents, evidentment els més obvis són els de receptor (*receiver*) i contingut (*content*). No comentarem ara tots els camps que hi podem trobar però sí mencionarem els que necessitarem en l'àmbit d'aquest treball, com són: origen (*sender*), tipus d'acte comunicatiu (*performative*) que és obligatori i indica quina classe de missatge estem enviant, llenguatge de comunicació utilitzat (*language*), ontologia (*ontology*), protocol (*protocol*), identificador de conversa (*conversation-id*) i tot un seguit de camps per identificar i crear respostes (*reply-to, reply-with, reply-by*). La classe ACLMessage també ens proveeix d'un conjunt de mètodes per modificar i accedir als valors d'aquests camps.

Un agent pot enviar un missatge amb el mètode **send(ACLMessage)** i per rebre'l pot fer servir **receive()**, que bloca el comportament receptor després de finalitzar el mètode *action()*, o **blockingReceive()** que bloca instantàniament tots els comportaments fins a l'arribada d'un missatge. Per crear respostes es pot fer servir *createReply()*.

Per determinar des d'un comportament si el missatge que arriba és el que s'espera es fa servir la classe **MessageTemplate**, que permet definir filtres per cadascun dels atributs i es pot fer servir com a paràmetre en **receive(MessageTemplate)** i **blockingReceive(MessageTemplate)**. Si pensem en el MA, que rep missatges del SMA i dels ETTMAs, els filtres ens serveixen per identificar l'origen i determinar cóm s'ha de tractar la informació rebuda.

Per a poder enviar objectes com la llista de TTRs i la llista d'ECCs necessitem una ontologia que validi semànticament el contingut, per això s'ha implementat l'**Emergency Area Ontology** (EAOntology) detallada més endavant.

Tots els missatges que fem servir en el nostre sistema s'han codificat en classes derivades de ACLMessage dins el paquet **mabett.messages**. Com a exemple, quan un ETTMA vol enviar un missatge demanant dades al MA, crea una instància de **ETTMAQuery** passant al constructor la seva identificació AID i el códec a

utilitzar. El mateix constructor de *ETTMAQuery* omple tots els camps necessaris, per tant des del codi del comportament només cal crear l'objecte i enviar-lo, sense preocupar-se de res més. Això facilita creació i enviament dels missatges, evita repetició de codi, facilita la fase d'integració i permet un manteniment senzill en les comunicacions entre agents del sistema.

5.1.6 Servei de pàgines blanques de l'agent AMS

Cada agent que vol operar en una plataforma, ha de registrar-se amb el AMS per obtenir un identificador AID vàlid. Aquesta operació es realitza automàticament amb l'agent AMS per defecte de la plataforma en qüestió.

El AMS de la plataforma manté una llista de les AIDs de tots els agents registrats i aporta el servei de pàgines blanques. Per aprofitar-lo, hem d'importar la classe **AMSService** i fer servir el seu mètode **search()** indicant el criteri de cerca.

Aquest servei és el que farem servir per mantenir la llista de ETTMAs de l'agent de gestió MA actualitzada.

5.1.7 Migració entre plataformes de JaDE

La migració d'agents entre plataformes s'aconsegueix gràcies al servei de mobilitat inter-plataforma desenvolupat al dEIC. La idea bàsica és la de fer-los moure intercanviant missatges ACL entre uns agents especials, coneguts com agents de gestió de mobilitat o **Agent Mobility Manager** (AMM), presents totes les plataformes que en facin ús.

Per iniciar la plataforma amb el servei cal llençar-la de la següent manera:
java jade.Boot -services jade.core.mobility.AgentMobilityService;
jade.core.migration.InterPlatformMobilityService;

On s'indica que s'usaran els serveis de mobilitat i de migració entre plataformes. Fent això ja es pot fer migrar els agents entre les plataformes de la xarxa fent servir el mètode d'agent **doMove(PlatformID)**, on la identificació de la plataforma de destí *PlatformID* es crea a partir de l'identificador AID del seu agent AMM i la seva direcció s'afegeix amb el mètode **addAddresses(String)**, tal i com es

veu a continuació:
AID remoteAMM = new AID("amm@<remotePlatform>:1099/JADE", true);
remoteAMM.addAddresses("http://<remotePlatformAddress>:7778/acc");

De fet, actualment només cal el nom de plataforma remota i no pas afegir l'adreça per aconseguir una migració satisfactòria, però sí és important que les plataformes disponibles es trobin llistades al fitxer */etc/hosts*. La necessitat de conèixer els noms de les plataformes presents, en el context d'una xarxa MANET que es troba en evolució constant, és el que dóna sentit a l'arquitectura de serveis i de descobriment de plataformes disponibles.

5.2 Implementació

5.2.1 Vista general dels elements del sistema

Si observem aquest projecte des d'un punt de vista global, veiem que hi ha una sèrie d'elements indispensables per a dur-lo a terme amb èxit. Ens calen agents, un mecanisme de comunicació entre ells, comportaments amb estats i altres objectes amb informació relativa a la situació de tot allò que ens envolta: temps de retorn, centres de control, etiquetes de triatge...

Per aquest motiu, hem generat els *packages* descrits a continuació:

a **mabett.agents**: conté les classes amb els agents del sistema i els seus comportaments, concretament disposa del codi per l'agent de gestió MA, el de transport ETTMA i una implementació parcial d'un agent de gestió de serveis SMA útil per a fer proves de funcionament.

b **mabett.elements**: recull les classes necessàries per a generar les llistes de TTRs, ECCs i ETTMAs.

c **mabett.logger**: conté les que calen per al sistema de registre d'estat i incidències, utilitzat per a fer un millor seguiment dels agents i els seus comportaments.

d **mabett.messages**: classes derivades de ACLMessage amb la definició dels missatges emprats en cada situació pels nostres agents. Per exemple hi podem trobar el missatge de notificació de TTR des del MA cap al SMA, o la sol·licitud d'informació que l'agent ETTMA fa a l'agent MA.

e **mabett.net**: paquet amb les classes utilitzades per a guardar la informació dels hosts de la xarxa.

f **mabett.ontologies**: en el seu interior es troben les classes necessàries per definir l'estructura de l'ontologia que es fa servir en la comunicació entre els nostres agents.

g **mabett.triage**: disposa de les classes per a les etiquetes de triatge.

A continuació es presenta amb més detall la implementació del nostre sistema d'agents i els seus elements. Comencem explicant la forma d'implementar les llistes de TTR i ECC i la ontologia utilitzada, seguint per una descripció del funcionament dels tres agents que hem creat. L'esquema de classes dels agents ETTMA, MA i SMA i els seus comportaments es pot veure a la figura 5.5.

5.2.2 Estructura de les llistes TTR i ECC

Les llistes de TTR i ECC, on es guarden les dades relatives a l'entorn que ens envolta, es troben en els tres tipus d'agents i s'han de passar d'un a l'altre constantment mitjançant missatges ACL.

Una llista **TTRList** és formada per un conjunt d'elements de tipus **TTRInfo**, formats a la vegada per una tripleta d'elements: nom de plataforma, adreça de xarxa i temps de retorn. Una llista **ECCList** és un seguit d'elements **ECCInfo**, que només han de disposar dels atributs de nom i adreça.

La necessitat de passar els objectes de les classes *TTRList* i *ECCList* mitjançant missatges ACL i fent ús d'una ontologia, ens obliga a tenir en compte una sèrie de limitacions pròpies de la utilització d'ontologies i el llenguatge de continguts LEAP que fem servir.

Per començar, els camps de les nostres estructures han de ser d'un tipus primitiu o bé d'un tipus suportat pel llenguatge de continguts. Els atributs de *TTRInfo* i *ECCInfo* sí són de tipus primitiu, però per a poder enviar llistes d'aquests elements cal que aquestes siguin instàncies de la classe **jade.leap.ArrayList**.

El llenguatge LEAP no facilita la creació de vectors, donat que no són necessaris a JaDE amb la política de scheduling que implementa i que no permet l'accés concurrent. La opció de fer servir *ArrayLists* és bona.

En la figura 5.1 es pot apreciar l'esquema de classes d'aquestes dues taules, en el que existeix una superclasse **EAList** amb els mètodes comuns per a *TTRList* i *ECCList*.

5.2.3 Ontologia

Per a aconseguir que les llistes llistes TTR i ECC siguin encapsulades en missatges ACL i recuperades correctament en destinació, ens cal generar una estructura semàntica clara i definir-la amb una ontologia que serà la **Emergency Area Ontology** (EAOntology).

Com s'ha explicat en el capítol de disseny necessitem conceptes, que han de implementar la interfície **Concept** que es troba a *jade.content* i predicats, que implementen la interfície **Predicate**. Fem servir les classes TTRList, ECCList, TTRInfo i ECCInfo com conceptes i la **InfoPredicate** com a predicat en el que s'incorporen, tal i com es pot veure a la figura 5.2.

A continuació es mostra un exemple de cóm es crea l'esquema semàntic en el cas de la llista de TTRs. En ell es mostra clarament la forma en la que s'afegeixen les classes TTRList i TTRInfo amb tots els seus atributs, resultant d'especial interès veure la relació amb l'estructura de classes de la figura 5.1.

- Afegim a la ontologia les classes en les que es troben els nostres conceptes:
  add(new ConceptSchema(TTRLIST),mabett.elements.TTRList.class);
  add(new ConceptSchema(TTRELEMENT),mabett.elements.TTRInfo.class);

- Definim la forma del concepte llista de TTRs:
  ConceptSchema ttrListSc = (ConceptSchema) getSchema(TTRLIST);

Figura 5.1: Diagrama de classe de les llistes de TTR i ECC.

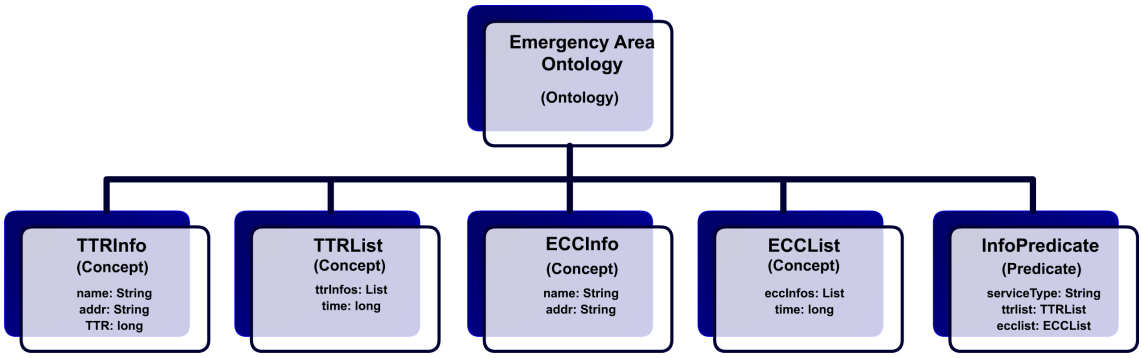

Figura 5.2: Esquema de la ontologia Emergency Area Ontology.

ttrListSc.add(LIST, new AggregateSchema(BasicOntology.SEQUENCE));
ttrListSc.add(TIME, (TermSchema) getSchema(BasicOntology.INTEGER));

- Definim la forma del concepte d'informació TTR:
  ConceptSchema ttrElementSc = (ConceptSchema) getSchema(TTRELEMENT);
  ttrElementSc.add(IP, (TermSchema) getSchema(BasicOntology.STRING), ObjectSchema.MANDATORY);
  ttrElementSc.add(NAME, (TermSchema) getSchema(BasicOntology.STRING), ObjectSchema.MANDATORY);
  ttrElementSc.add(TTR, (TermSchema) getSchema(BasicOntology.INTEGER), ObjectSchema.MANDATORY);

Cal detallar que el el tipus bàsic *BasicOntology.INTEGER* també és el que es fa servir per a *long* i que camps marcats com a *mandatory* no poden quedar en blanc.

5.2.4 Electronic Triage Tag Mobile Agent (ETTMA)

L'agent de transport d'etiquetes de triatge és l'únic agent del sistema que ha de disposar de mobilitat, per a poder migrar de plataforma en plataforma al llarg del seu camí cap al centre de coordinació.

Com ja s'ha dit, en el moment en que es crea i comença a executar un agent, es crida al seu mètode *setup()*, que en el cas dels ETTMAs afegirà el comportament

ETTMABehaviour, derivat de **SimpleBehaviour** i encarregat de la tasca de fer arribar l'agent al centre de coordinació. La classe **mabett.agents.ETTMA** conté tant la definició de l'agent ETTMA com la del seu comportament ETTMABehaviour a mode de classe interna.

Per detectar si un agent ETTMA es troba al centre de control, es fa servir el mètode **afterMove()**, que es posa en funcionament de forma automàtica després d'una migració. L'agent comprova si el criteri seguit per a la darrera migració va ser el de ECC i passa a l'estat d'entrega de dades si és així.

ETTMA Behaviour

El comportament de l'agent de transport es divideix en cinc estats, seguint el disseny plantejat al capítol de *disseny*.

- En **l'estat inicial** (*INIT STATE*) es comprova que no s'ha produït una migració fallida fent servir un atribut local del comportament de tipus sencer, que s'inicialitza a zero i augmenta de valor cada vegada que s'intenta migrar. Com que els *behaviours* es reinicien després d'una migració amb èxit, si el valor d'aquesta variable no és zero, hem detectat una migració fallida. Si no es detecten problemes en aquest punt, vol dir que l'agent s'acaba de crear o prové d'una migració des d'un altre terminal, per tant sol·licita informació al MA mitjançant un missatge ACL. Per facilitar les coses, aquest missatge es troba implementat directament en la classe **mabett.messages.ETTMAQuery**, així que només cal crear una instància ETTMAQuery i enviar-la amb un *send()*.

- En **l'estat d'espera de resposta** (*WAIT FOR MA INFO*) l'agent ETTMA espera rebre un missatge amb les dades des del MA i després n'ha d'extreure el contingut. Es fa servir un patró per evitar tenir en consideració missatges no desitjats i s'espera amb un *blockingReceive(MessageTemplate)*, bloquejant per complet l'agent fins a l'arribada de les dades. Amb el gestor de continguts i fent ús de l'ontologia **EAOntology** s'obté la informació continguda en el missatge amb **extractContent(ACLMessage)** i així l'agent ja

pot prendre una decisió respecte a si ha de migrar o no.

- En **l'estat de decisió de migració** (*MIGRATION DECISION*) s'avalua quina és la plataforma a la que l'agent ha de saltar per aconseguir arribar al centre de coordinació en funció de la política de migració a seguir. Si la decisió de migració s'ha de fer en base a informació de tipus TTR, per defecte es fa una crida al mètode **decideByMinTTR()** que escull la plataforma amb menor temps de retorn, tot i que es poden seleccionar altres polítiques diferents. Si la decisió s'ha de prendre en base a informació de tipus ECC, es crida al mètode **decideByRandomECC()** que té la característica de no enviar tots els agents d'una mateixa plataforma d'origen cap el mateix centre de coordinació si és que n'hi ha més d'un disponible, encara que també es troba implementat el **decideByFirstECC()**.

- En **l'estat de migració** (*MIGRATION STATE*) el ETTMA prova de migrar cap a la plataforma escollida amb el mètode d'agent **doMove(PlatformID)**. Si no ho aconsegueix, augmenta el valor la variable de control de migracions fallides i, si aquesta arriba a un nombre màxim d'intents (3 per defecte), torna a l'estat d'espera de resposta per part del MA.

- En **l'estat d'entrega de dades** (*DATA DELIVERY*) l'agent ha d'entregar les dades al centre de coordinació i finalitzar la seva execució. El procés d'entrega de dades no es troba implementat ja que encara no es disposa del disseny del ECC i es realitza de forma simulada mostrant un missatge per consola.

S'ha de dir també que existeixen quatre atributs configurables en el comportament de l'agent ETTMA: el primer es correspon amb el nombre d'intents de migració que es poden realitzar abans de desistir i tornar a l'estat d'espera, el segon i el tercer determinen les polítiques de decisió de destí a seguir en cas de migrar per ECC o per TTR i el quart permet activar el sistema de *log* del comportament. Amb el registre activat, es mostren missatges per consola sobre l'estat de l'agent i les operacions que realitza, guardant-los també en un fitxer de text.

Polítiques de decisió de destí

La decisió que es pren en el tercer estat del comportament s'implementa amb mètodes de la classe *ETTMABehaviour*. Existeixen diversos mètodes diferents per escollir el destí de l'agent ETTMA en funció de si es pot arribar directament a un ECC o s'ha de anar saltant mitjançant el TTR, però també segons la política de salts decidida per al sistema.

En el cas de TTR es troben implementades les polítiques de **salt per menor TTR** (utilitzada per defecte), salt a la primera plataforma de la llista i elecció aleatòria. Evidentment la millor opció i única que té sentit en el context en el que ens ocupa és la de menor TTR, tot i que es deixa la porta oberta a la implementació futura de noves polítiques.

Per a la decisió de migració cap a un ECC, només disposem de la opció de salt al primer centre de coordinació de la llista o **elecció aleatòria** (per defecte). En aquest cas no queda clar quina és la millor opció, tot i que sembla més adient l'aleatòria per evitar que tots els agents ETTMA existents a la MANET intentin migrar cap al mateix destí i puguin arribar a col·lapsar-lo.

5.2.5 Manager Agent (MA)

L'agent MA gestiona la informació de TTRs i ECCs que rep des del agent SMA i s'encarrega del seu enviament cap als agents ETTMA. A més a més, incorpora la interfície gràfica del sistema i genera els agents ETTMA amb les etiquetes de triatge al seu interior.

Per fer les dues primeres tasques disposa de:

- Una llista de classe **TTRList** on guarda les dades de les plataformes properes que ofereixen el servei de TTR.
- Una llista de classe **ECCList** per guardar noms i adreces dels centres de coordinació a l'abast.
- Una llista **ETTMAList** amb els noms dels agents de transport coneguts, de la que podem veure l'estructura de classes a la figura 5.3.

La interfície gràfica encara no es troba incorporada al sistema, cosa que s'ha de fer més endavant en la fase d'integració, però sí s'ha preparat el comportament en el que s'incorporarà per a que encaixi plenament.

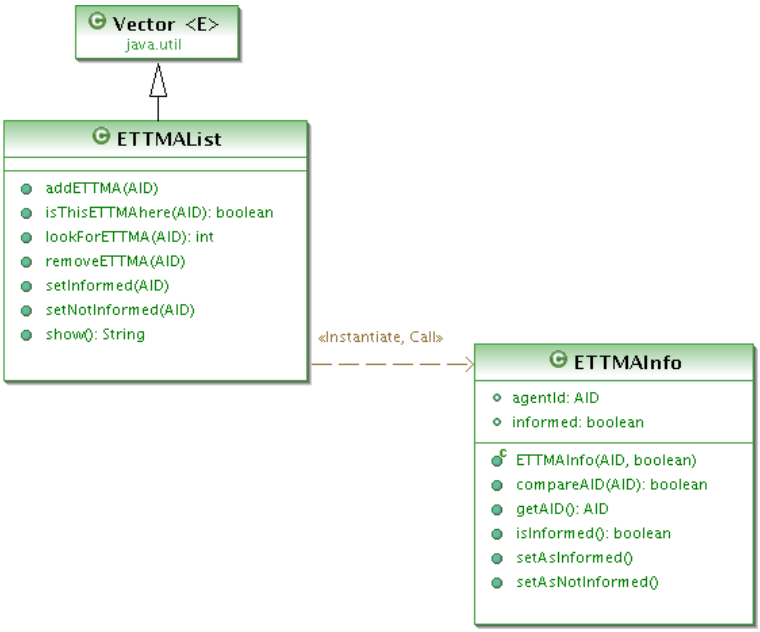

Figura 5.3: Diagrama de classes de la llista ETTMA.

El comportament de l'agent és un *behaviour* paral·lel de tres subcomporaments cíclics, cadascun per una de les tasques. Tot i que s'accedeix a les llistes de TTR i ECC des de dos d'ells (quan es faci la integració des dels tres) no ha calgut implementar cap mecanisme de sincronització. En JaDE cada agent és un thread i la planificació d'execució de comportaments que es fa servir, evita l'accés simultani a recursos fent que els mètodes *action()* dels *behaviours* s'executin de forma atòmica.

En la figura 5.4 es mostra la relació existent dels tres subcomportaments de l'agent de gestió amb la resta d'agents i la interfície gràfica del sistema.

MA First Shot Behaviour (MAFSBehaviour)

Aquest comportament cíclic escolta les sol·licituds d'informació des dels ETTMAs i les contesta sempre i quan es disposi de dades per fer-ho.

Mitjançant un *template* identifica els missatges dels ETTMAs i afegeix els remitents a la llista d'agents coneguts fent ús del mètode *addETTMA(AID)* que ofereix la classe *ETTMAList*. Després envia en un missatge de tipus **InformationMessage** al ETTMA amb la informació sol·licitada i el marca com a informat amb *setInformed(AID)*. Si no és possible enviar informació no s'envia, deixant que l'agent ETTMA quedi a l'espera i delegant la responsabilitat d'enviar la informació al *behaviour MASLCBehaviour*, que ho farà quan es produeixin novetats a la xarxa. En aquest cas, l'agent de transport queda marcat com a *not informed* a la llista *ETTMAList*.

Cal tenir en compte que abans d'enviar informació als ETTMAs, ha de descomptar als temps de retorn el temps transcorregut des de que es va rebre la informació, en la operació de **reducció de TTR**.

MA Sevice Layer Communicator Behaviour (MASLCBehaviour)

El *behaviour* de comunicació amb la capa de serveis s'executa contínuament seguint l'esquema descrit a la figura 4.4. Els seus quatre estats es troben implementats de la següent forma:

- En l'estat d'**enviament de TTR** crea una instància del missatge **MATTRNotification** a partir del TTR local i l'envia cap al SMA. El codi no és el definitiu en espera de la unificació amb el projecte de gestió de serveis, quan es farà servir l'ontologia de serveis entre aquests dos agents i no pas la *EAOntology* com ara.

- En el de **subscripció al SMA** es crea missatge **SMASubscription** i s'envia cap al agent de gestió de serveis. També caldrà completar el codi en la posada en comú del codi d'ambdós projectes.

- Per a l'estat de **recepció de dades** cal un patró que detecti els missatges de

notificació del SMA i fer ús del gestor de continguts de l'agent per extreure la informació. Amb *extractContent(ACLMessage)* s'obtenen les dades TTR o ECC i posteriorment es guarden a les llistes **TTRList** o **ECCList**.

- Finalment, en l'estat d'**enviament cap als ETTMAs** s'envien missatges **InformationMessage** a tots els ETTMAs encara presents en la plataforma. Es recorre la llista **ETTMAList**, enviant els missatges a tots els agents marcats com a no informats i, per als que han estat informats alguna vegada anteriorment, es comprova si encara existeixen a la plataforma actual consultant al AMS. Això ho fem amb el mètode *search(Agent, AMSAgentDescription, SearchConstraints)* de la classe *AMSService*. Els paràmetres fan referència a l'agent que fa la cerca, la descripció de l'agent AMS (en el nostre cas el AMS per defecte) i les característiques de cerca (per exemple el nombre de resultats i la profunditat).

MA GUI Behaviour

En aquest comportament es trobarà la interfície del sistema una vegada es produeixi la integració, actualment només s'hi pot trobar al codi l'esquelet de la classe.

5.2.6 Dummy Service Manager Agent (Dummy SMA)

En **mabett.agents.SMA** es troba implementada una versió parcial de l'agent de gestió de serveis SMA, que anomenarem **Dummy SMA**. El disseny i implementació d'aquest agent no pertany a aquest projecte però és necessari per a poder fer proves de funcionament i per això ha estat codificat.

Dummy SMA Behaviour

Dins el comportament de l'agent SMA per proves, trobem tres estats diferents, un per cada tasca que el relaciona directament amb el nostre sistema d'agents i que ha estat simulada:

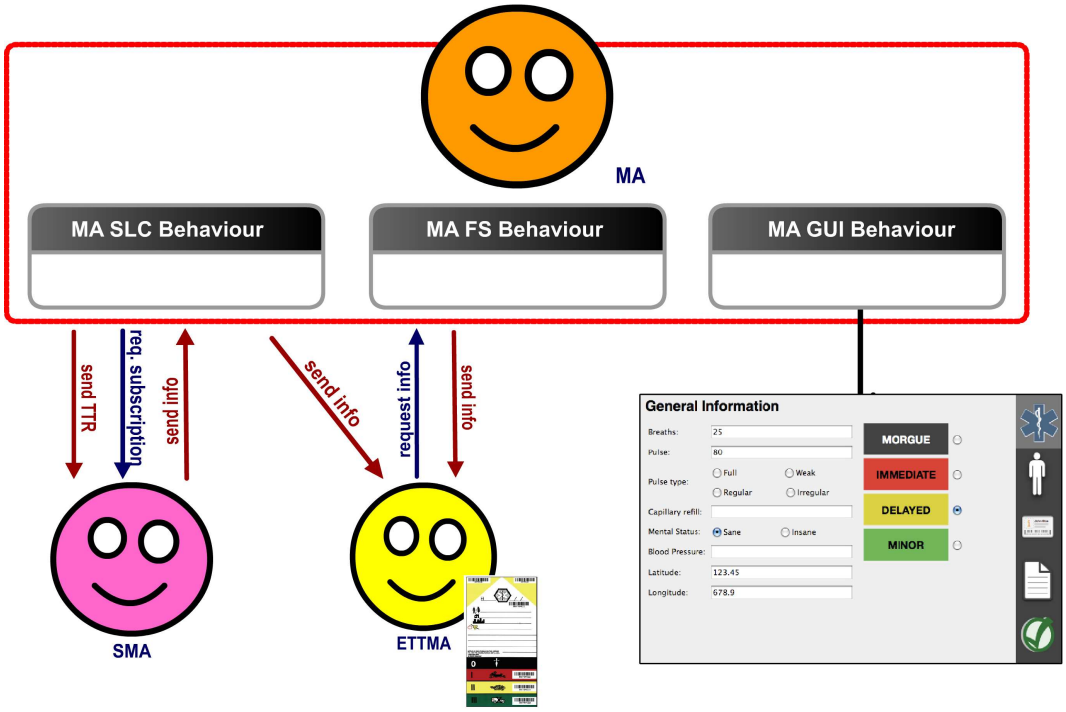

Figura 5.4: Comportaments de l'agent MA amb les seves relacions.

- El MA ha envia informació de temps de retorn local al SMA, per tant s'ha fet una implementació del procés de rebuda d'aquest missatge i de l'operació de guardat a la llista de TTR interna del SMA, fent servir el mètode *updateTTR(<platformName>,<TTR>)* de la classe *TTRList*.

- El SMA subscriu al servei de TTR o de ECC els agents MA que ho sol·liciten. Aleshores, ha calgut implementar també la rebuda del missatge de subscripció.

- El SMA proveeix les dades de TTR i ECC a l'agent MA cada vegada que hi ha canvis a la topologia de la xarxa o modificacions en algun dels TTRs. Per a poder simular això fem servir un comportament **CyclicBehavior** que envia dades repetidament a partir de uns valors inicials que són introduïts manualment al codi del *Dummy SMA*, i que es van modificant a mesura

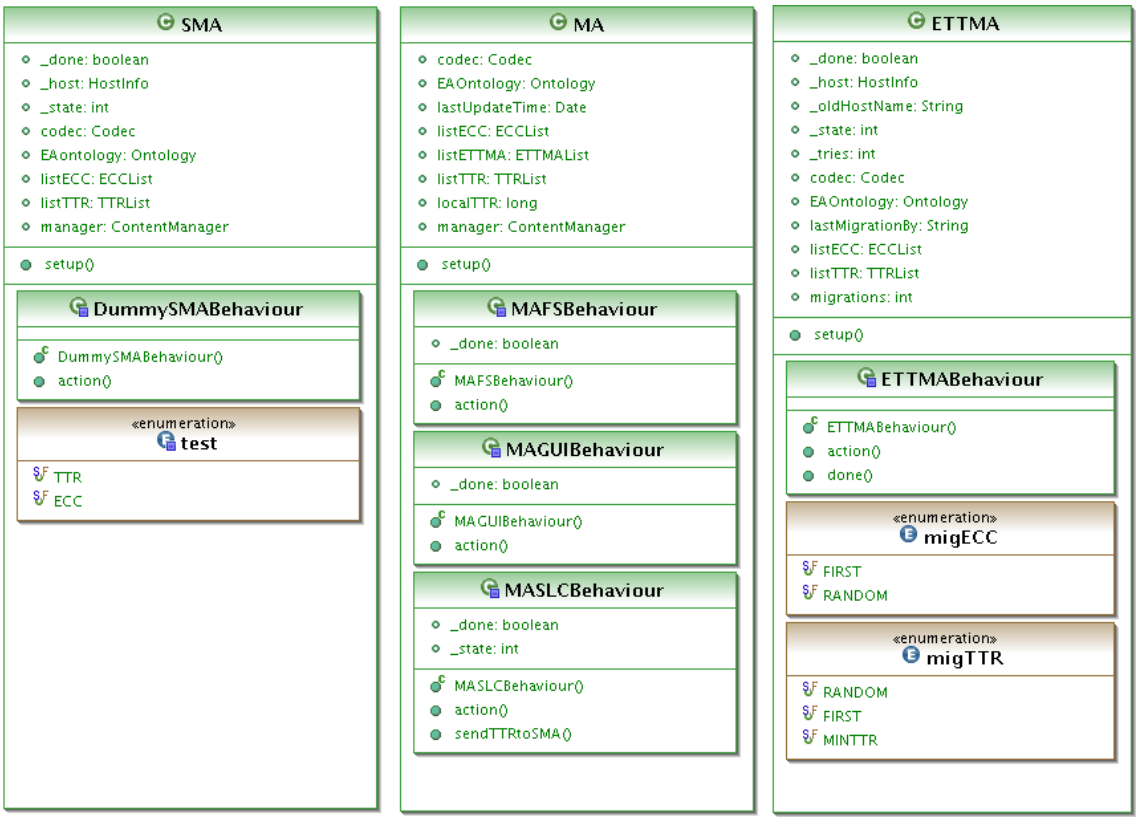

Figura 5.5: Diagrama de classe dels agents del sistema.

que avança el temps amb els mètodes *incTTRbyName(String <plattformName>, long <increment>)* i *decTTRbyName(String <plattformName>, long <decrement>)* de la classe TTRList.

Els dos primers estats avancen seqüencialment mentre que el tercer s'executa continuament una vegada s'hi ha arribat. En la implementació real d'aquest agent s'ha d'estar atent a noves notificacions de TTR provinents del MA i a les demandes de subscripció. Tot això implica que s'haurà de fer servir més d'un comportament, però per a realitzar les nostres proves ens és suficient amb la implementació realitzada.

En l'agent SMA per proves es permet configurar si es desitja enviar missatges de tipus ECC o TTR cap al MA, el temps que ha de passar entre dos enviaments consecutius d'informació i si es desitja fer ús del sistema de registre de missatges o *logging*.

5.3 Integració

La fase d'integració amb altres projectes no s'ha pogut dur a terme per manca de temps. Tanmateix, sí s'ha realitzar un petit anàlisi d'integració que aporta detalls útils per a propers projectistes del grup SeNDA que s'encarreguin de seguir en aquesta línia.

En aquesta secció aportem algunes idees sobre cóm realitzar la integració amb els projectes relacionats *Gestió Dinàmica de Serveis* [serveis] i *Disseny i implementació d'interfícies del protocol START* [dham].

5.3.1 Integració amb el projecte: Gestió Dinàmica de Serveis

La interacció amb el primer treball es realitza amb una comunicació entre agents, concretament el MA i SMA, per tant integrar els dos projectes és simplement posar en comú la ontologia de missatges entre aquests dos agents.

En la nostre implementació hem creat un *Dummy SMA* que no fa servir la ontologia de serveis sinó la *EAOntology*, el que cal fer és afegir l'ontologia de serveis

al comportament *MASLCBehaviour* i modificar el codi per extreure la informació del missatge que arriba des del SMA, tenint en compte la nova estructura.

5.3.2 Integració amb el projecte: Disseny i implementació d'interfícies del protocol START

Integrar el segon projecte amb el nostre és més complicat ja que en realitat han de compartir codi. En aquesta llista es mostren un seguit de relacions entre projectes que s'han de tenir en compte per aconseguir una integració satisfactòria:

- L'estructura de paquets és idèntica en ambdós treballs, de forma que es permet la unió directa de classes i paquets del codi.
- El nostre agent MA es correspon amb l'agent *BootAgent*, que ja disposa del seu propi comportament amb la interfície gràfica. El codi d'aquest comportament, o tot ell, poden ser traslladats directament al nostre *MAGUIBehaviour* actualitzant els noms de classes i les referències necessàries.
- Els nostres agents ETTMA es corresponen amb els agents *TriageTagAgent*, que no disposaven de mobilitat i només contenien les dades del pacient sense fer cap acció.
- La classe *mabett.core.Controller* es la responsable de crear els agents de transport, donant un nom derivat de l'hora de triatge a cadascun d'ells. Ara, els agents generats no hauran de ser de classe *TriageTagAgent* sinó *ETTMA* i la generació de noms també hauria de ser modificada, fent servir alguna de les recomanacions donades respecte aquest tema en el capítol d'*anàlisi*.
- Per guardar un nou valor de TTR introduït des de la interfície gràfica, només caldrà modificar l'atribut *localTTR* de la classe *MA* i resetejar el comportament *MASLCBehaviour* amb el mètode *reset()*.

Evidentment, aquests punts no contenen tota la informació necessària per a aconseguir posar en comú completament ambdós projectes, segurament en el procés apareixeran noves dificultats i reptes a superar per propers projectistes.

5.4 Prova

El treball amb agents requereix una fase de prova més complexa que quan es treballa amb altres paradigmes de programació tradicionals. Per començar ens cal una xarxa sobre la que poder fer migrar els nostres agents i un mecanisme de control sobre els mateixos.

La fase de prova ha estat paral·lela en tot moment a la de implementació, i molt important per a corregir errors difícils de detectar a mesura que s'anava avançant. En el primer annex d'aquesta memòria s'explica cóm obrir el codi del nostre projecte des d'Eclipse i cóm configurar execucions amb els agents que desitgem, en el segon es mostren alguns exemples d'execucions, juntament amb els resultats obtinguts.

5.4.1 Xarxa de proves

Per a la realització de proves hem configurat una xarxa amb quatre ordinadors, dos convencionals i dues màquines virtuals amb VMWare tal i com es mostra a la figura 5.6. Cadascun d'aquests ordinadors simula un dispositiu de triatge, amb la plataforma JaDE inicialitzada, un agent de gestió MA i un de serveis *Dummy* SMA, un valor de TTR propi i alguns agents ETTMA.

Amb aquesta topologia de xarxa, tenim que els fitxers /etc/hosts dels equips implicats tenen les següents entrades:
127.0.0.1 localhost
192.168.32.4 robsportatil
192.168.32.2 athlon
192.168.32.5 debian
192.168.32.6 ubuntu

5.4.2 Configuració de les proves

En el codi del *Dummy SMA*, es troben els atributs configurables necessaris per a poder realitzar diferents tipus de proves. Com que el nostre SMA de test no

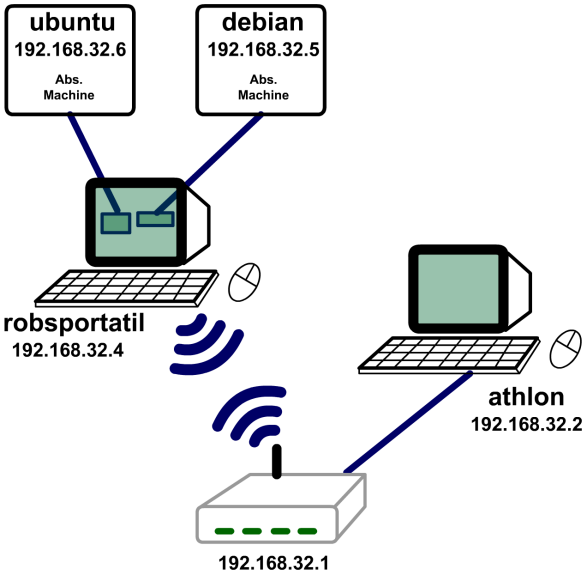

Figura 5.6: Topologia de la xarxa de proves.

es comunica amb els altres SMA de plataformes remotes i no pot aconseguir la informació de la xarxa per si mateix, les dades de que disposa i que envia cap a l'agent MA es troben introduïdes directament al seu codi, on trobem una llista de TTRs i una d'ECCs amb informació dels equips de presents a la xarxa. Un atribut de tipus *test* que pot tenir valor TTR o ECC, com pot veure a la figura 5.5, ens permet configurar si s'enviarà informació corresponent als temps de retorn o als centres de coordinació.

El comportament del *Dummy SMA* també disposa de paràmetres configurables, com el temps que ha de passar abans de realitzar el primer enviament de dades al MA i el temps entre dos enviaments consecutius, recordem que aquest agent provisional envia la informació de manera periòdica.

5.4.3 Registre (*log*)

Comprovar que els agents fan allò que volem quan hi ha migracions pel mig, resulta força complicat si no es disposa de cap mecanisme de control.

Per supervisar les accions, l'estat, l'arribada de missatges i la presa de decisi-

ons dels agents, s'ha afegit un sistema de registre o *log* a cadascun dels comportaments. Mitjançant l'atribut configurable *debug*, que es troba en cadascun dels *behaviours* implementats, es pot activar o desactivar el registre de successos per aquell comportament concret.

Per a fer el registre, s'ha creat la classe **mabett.logger.AgentLogger** derivada de **java.util.logging.Logger** i el formatejador **MinimalFormatter**.

Amb el registre de missatges actiu, es registren els events i errors detectats en temps d'execució, guardant-los en arxius al sistema de fitxers i mostrant-los per consola. Es guarda un arxiu diferent per agent, que disposa del mateix nom de l'agent del qual es registren les incidències.

En el cas dels ETTMAs, com hi ha salts entre plataformes i la classe *AgentLogger* no és serialitzable, requisit necessari per a que formi part d'un agent que es mou en una xarxa, ha calgut fer servir el modificador **transient** per indicar que l'objecte *logger* no ha de ser salvat ni recuperat pel mecanisme de serialització. En arribar a la nova plataforma, el ETTMA crea un nou objecte logger que comença a registrar els events a partir del moment d'arribada.

Capítol 6

Conclusions

A continuació es presenta la valoració de la feina realitzada al llarg del temps dedicat a la realització d'aquest projecte, el resum de la feina feta, la planificació temporal real i les línies d'ampliació proposades.

6.1 Objectius acomplerts

Si recordem el principal objectiu d'aquest projecte, que és aconseguir la mobilitat dels agents portadors de les etiquetes de triatge des de la plataforma de creació fins al centre de coordinació ECC, podem determinar que sí hem aconseguit el nostre objectiu.

Els agents ETTMA migren en funció dels temps de retorn de plataforma en plataforma fins arribar al centre de coordinació portant la informació de la víctima amb ells. Un agent de gestió aporta la informació necessària per a que els ETTMAs coneguin els terminals de la xarxa als que poden saltar. Cada vegada que un agent de transport demana dades o quan es produeixen canvis en la topologia, el MA envia missatges als ETTMAs que depenen d'ell, fent possible que aquests prenguin la decisió més adient.

També hem implementat mecanismes per evitar que les migracions fallides provoquin la no arribada dels agents de transport a destinació, fent que l'agent MA mantingui un control constant dels agents ETTMA presents a la plataforma i

no en pugui restar cap inadvertit.

Es realitza una comunicació amb l'agent SMA de la infraestructura de serveis que garanteix la independència entre la capa d'aplicació i la de serveis, tot i que encara es tracti d'una implementació no definitiva. També es segueix una estructura de paquets idèntica a la utilitzada en el projecte on s'implementa la interfície del protocol START, per garantir que el sistema permetrà una integració fàcil amb aquest treball.

6.2 Resum de la feina feta

La feina en aquest projecte ha seguit un ordre clar i ben estructurat, avançant sempre a la següent fase només després de poder fer-ho amb totes les garanties d'èxit. Aquest és el resum de tots els passos seguits per a dur-lo a terme:

a **Estudi de la programació amb agents amb JADE**, aconseguint documentació des de les fonts oficials de TILAB, universitats, fòrums de debat i altres recursos web diversos que es llisten a la bibliografia.

b **Anàlisi del projecte anterior** *Disseny i implementació d'interfícies del protocol START* [dham], estudiant què s'havia implementat fins al moment i fent proves de funcionament per comprendre millor el context en el que ens mouríem.

c **Anàlisi de requisits del sistema**, avaluant els requisits funcionals i no funcionals i estudiant les eines i el material disponible.

d **Disseny del sistema** tenint en compte els requisits i les restriccions, assolint els objectius i aconseguint independència respecte als treballs desenvolupats en paral·lel.

e **Implementació** del sistema d'agents amb mecanismes afegits, com el registre de successos, la protecció contra migracions fallides i la possibilitat de configurar diferents paràmetres per ajustar el comportament, pensant especialment en la fase de proves.

f **Conjunt de proves** amb dos ordinadors reals i dues màquines virtuals, estudiant el comportament del sistema d'agents mòbils en diferents circumstàncies i simulant entorns diversos.

g **Creació de la documentació**, especialment aquesta memòria amb els seus annexos i la presentació final.

6.3 Planificació temporal final

Si mirem la figura 3.2 amb la planificació temporal inicial d'aquest projecte, es pot veure que s'ha produït un retard en l'entrega respecte a la data proposada inicialment.

Els motius del retard tenen a veure en gran mesura amb la fase de disseny, en la que les dificultats per separar les tasques a realitzar en cadascun dels projectes relacionats del grup, van obligar a redissenyar per complet el nostre projecte en diverses ocasions.

La separació en capa de serveis i d'aplicació, que són les dues capes que ens afecten, es va decidir posant en comú idees de projectistes i professors del dEIC, però una vegada establerta aquesta primera divisió calia determinar les tasques associades a cada capa i l'estructura final d'agents. Mitjançant reunions setmanals de grup, es comentaven els dissenys dels projectistes, en els que sovint apareixien dificultats per mantenir la independència funcional entre els projectes relacionats.

El disseny final va trigar aproximadament un mes més de l'esperat, retallant completament el marge de temps disponible entre la finalització i l'entrega previst a la planificació inicial. Davant de la possibilitat de no poder entregar a temps en cas de trobar noves dificultats en la implementació, o no poder obtenir el nivell de qualitat desitjat, vaig optar personalment per ajornar l'entrega fins al setembre.

Com es pot veure en la figura 6.1, la fase de disseny ha ocupat més dies, en els que no s'hi ha pogut dedicar tot el temps previst en inici. La dependència existent entre treballs creava situacions en les que, per a poder avançar correctament, calia conèixer detalls de cóm es dissenyarien alguns aspectes del projecte de gestió de serveis i cóm s'integraria tot en *The Mobile Triage Tag* [mtt]. Fins a les reunions

setmanals no coincidíem tots els projectistes i professors implicats per proposar i debatre solucions.

La decisió d'entregar al setembre i fer una nova planificació m'ha permès dedicar tres setmanes del mes de juny exclusivament a la preparació i realització d'exàmens, allargant la durada del projecte fins a finals de juliol.

6.4 Línies d'ampliació

Resten encara línies obertes per a completar el projecte *The Mobile Triage Tag* [mtt], especialment pel que fa a la gestió que s'ha de realitzar amb les etiquetes de triatge al centre de coordinació i la planificació de rutes per a l'actuació dels serveis d'emergència.

A nivell més local, en el nostre projecte encara s'ha de treballar en la integració amb la resta de projectes del grup, es poden realitzar algunes millores i afegir funcionalitats addicionals que han estat desestimades per manca de temps. Aquestes són algunes de les línies d'ampliació proposades:

a Integració total amb el projecte *Dispositius Handheld per a Aplicacions Mèdiques* [dham], fusionant el codi dels dos i integrant la interfície gràfica en el comportament *MAGUIBehaviour* de l'agent MA. També cal integrar-lo amb el projecte *Gestió Dinàmica de Serveis* [serveis], adaptant el comportament *MASLCBehaviour* per a que es comuniqui amb el SMA real fent servir l'ontologia de serveis, no pas l'*EAOntology*.

b Implementació de noves polítiques de migració en funció del TTR si s'escau. Per exemple, en el cas que es pensés en una opció més eficient que la de menor TTR, o per evitar col·lapses deguts a migracions massives d'agents cap al mateix destí.

c Implementar el procés d'entrega de les etiquetes de triatge al centre de coordinació, tot i que abans cal decidir el disseny final d'aquest. També s'ha de dissenyar l'ontologia adient per validar semànticament els missatges amb els *Triage Tags* que s'entreguen.

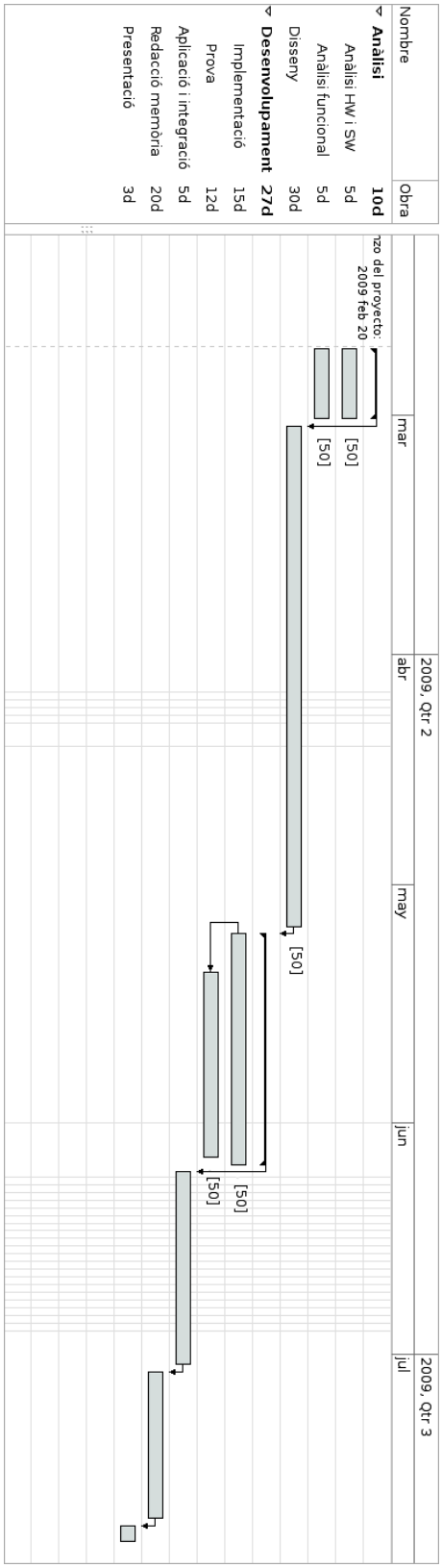

Figura 6.1: Diagrama de Gantt amb la planificació temporal resultant.

d Es pot implementar una alarma de detecció de TTRs anòmals per avisar que una persona encarregada del triatge de víctimes ha tingut complicacions.

e En el nostre sistema existeixen atributs que poden ser configurats per determinar alguns paràmetres del comportament dels agents, com per exemple la política de salts que s'ha de fer servir. Aquests atributs es modifiquen canviant el valor directament sobre el codi, així que pot resultar molt útil dissenyar una interfície gràfica de configuració avançada del sistema (o afegir-la a la ja existent) i permetre que els agents acceptin la introducció dels valors per paràmetre.

Aquestes són propostes per a propers projectes del grup, algunes d'elles ja es poden dur a terme i altres s'han d'anar realitzant a mesura que quedin clars els rols dels elements que resten per dissenyar.

Capítol 7

Acrònims

- **TT** (*Triage Tag*): etiqueta de triatge.
- **EA** (*Emergency Area*): zona que es troba en estat d'emergència.
- **ECC** (*Emergency Coordination Centre*): centre de coordinació de l'emergència.
- **ETTMA** (*Electronic Triage Tag Mobile Agent*): agent mòbil per a etiquetes electròniques de triatge o agent de transport.
- **MA** (*Manager Agent*): agent de gestió.
- **SMA** (*Service Manager Agent*): agent de gestió de serveis.
- **AMS** (*Agent Management System*): agent encarregat de supervisar i controlar l'accés i ús de la plataforma d'agents.
- **AMM** (*Agent Mobility Manager*): agent de gestió de mobilitat present a les plataformes que tenen actiu el servei de mobilitat inter-plataforma.
- **DF** (*Directory Facilitator*): agent que proveeix del servei de pàgines grogues a la plataforma.
- **TTR** (*Time To Return*): temps de retorn.

- **AID** (*JADE Agent Identifier*): identificador de l'agent JaDE.
- **ACL** (*Agent Communication Language*): llenguatge de comunicació entre agents a JaDE.

Bibliografia

[agent] Jim White, Mobile Agents White Paper, 1996.

[agentlink] European Co-ordination Action for Agent-Based Computing [online].
<http://www.agentlink.org/>

[ambulance] S. Pavlopoulos, E. Kyriacou, A. Berler, S. Dembeyiotis and D. Koutsouris, A novel emergency telemedicine system based on wireless communication technology-AMBULANCE, in: IEEE Transactions on Information Technology in Biomedicine, 1998.

[artemis] S. McGrath, E. Grigg, S. Wendelken, G. Blike, M. D. Rosa, A. Fiske and R. Gray, ARTEMIS: A Vision for Remote Triage and Emergency Management Information Integration, in: Dartmouth University, Dartmouth University, 2003.

[dham] X. Jurado, Dispositius handheld per a aplicacions mèdiques. Disseny i implementació d'interfícies del protocol START, Department of Information and Communications Engineering, Universitat Autònoma de Barcelona, 2008.

[dprotocols] Peoria Area EMS System, Prehospital Disaster Protocols, Illinois, 2005.
<http://www.paems.org/>

[emtrack] EMSystems, LLC., EMTrack Patient and Evacuee Tracking System, EMSystems, LLC. USA.

[fipa] Foundation of Intelligent Physical Agents (FIPA) [online]. <http://www.fipa.org/>.

[fipaACL] Foundation of Intelligent Physical Agents (FIPA), FIPA ACL Message Structure Specification, August 2001.

[fipaONTO] Foundation of Intelligent Physical Agents (FIPA), FIPA Ontology Service Specification, March 2002.

[goodidea] D. Chess, C. G. Harrison, and A. Kershenbaum, Mobile Agents: Are they a good idea?, IBM Research Report, RC 19887, October 1994.

[improvisa] J. R. Velasco, M. A. López-Carmona, M. Sedano, M. Garijo, D. Larrabeiti and M. Calderon, Role of multi-agent system on minimalist infrastructure for service provisioning in ad-hoc networks for emergencies, in: Proceedings of AAMAS'06, 2006, pp. 151-152.

[jade] JADE - Java Agent DEvelopment framework: JADE online documentation [online].
<http://jade.tilab.com/>

[jadesem] G. Aranda Corral, Seminario de JADE: Curso de Programación de Agentes, May 2005.

[java6] Sun Microsystems Inc.: Java Platform Standard Ed. 6 API Specification [online], 2008.
<http://java.sun.com/javase/6/docs/api/>

[jwpaper] F. Bellifemine, G. Caire, A. Poggi and G. Rimassa, JADE - A White Paper, september 2003.

[jguide] F. Bellifemine, G. Caire, T. Trucco and G. Rimassa, JADE Programmer's Guide, Telecom Italia, June 2007.

[manet] D. K. Kim, A New Mobile Environment: Mobile Ad Hoc Networks (MANET), IEEE Vehic. Tech. Soc. News, August 2003

[manetsadv] J. López, J. M. Barceló and J. García-Vidal, Ventajas de usar subredes en una red ad-hoc con nodos móviles, Departamento de Arquitectura de Computadores, Universidad Politécnica de Cataluña (UPC).

[mascal] E. A. Fry and L. Lenert, Mascal: Rfid tracking of patients, staff and equipment to enhance hospital response to mass casualty events, in: AMIA Annual Symposium Proceedings, 2005, pp. 261-265.

[mobility] J. Cucurull, Inter-Platform Mobility Service Documentation for the JADE Platform, SeNDA Group (UAB), v.1.104.

[mtt] R. Martí, S. Robles, A. Martín-Campillo and J. Cucurull, Providing Early Resource Allocation During Emergencies: The Mobile Triage Tag, Department of Information and Communications Engineering, Universitat Autònoma de Barcelona, December 2008.

[multiagent] L. Berdun and A. Monteserin, Taller de Sistemas Multiagentes, Universidad Nacional del Centro de la Provincia de Buenos Aires, noviembre 2007.
<http://www.exa.unicen.edu.ar/catedras/tmultiag/>

[onto04] G. Caire and D. Cabanillas, Jade Tutorial: Application-defined Content Languages and Ontologies, TILab S.p.A., 2004.

[serveis] D. Morcillo, Agents Mòbils en Situacions d'Emergència. Gestió Dinàmica de Serveis, Department of Information and Communications Engineering, Universitat Autònoma de Barcelona, Juny 2009.

[start] START Triage, The Race Against Time [online], february 2009.
<http://www.start-triage.com/>

[tacmedcs] U. Navy, Tactical Medical Coordination System (TacMedCS).
<http://www. namrl.navy.mil/clinical/projects/tacmedcs.htm>.

[tetra] J. Dunlop, D. Girma and J. Irvine, Digital Mobile Communications and the TETRA System, John Wiley and Sons, 1999.

[triage] Wikipedia. Triage [online], february 2009. <http://en.wikipedia.org/wiki/Triage>

[triagetag] METTAG Medical Emergency Triage Tag [online]. <http://www.mettag.com/>.

[tssiemd] DPN, T. Komura, Catena Corporation, Traceability System for Sick and Injured, Dai Nippon Printing Co. Ltd. (DNP); T. Komura, Fuji Tokoha University, Shizuoka city; Catena Corporation.

[uml05] UML Unified Modeling Language, june 2005. <http://www.uml.org/>

[wiisard] L. Lenert, T. C. Chan, W. Griswold, J. Killeen, D. Palmer, D. Kirsh, R. Mishra and R. Rao, Wireless Internet Information System for Medical Response in Disasters (WIISARD), in: AMIA Annual Symposium Proceedings, 2006, pp. 429-433.

[wikimanual] Wikimanual de los fundamentos de la programación de agentes, Escuela Superior de Ingeniería Informática de la Universidad de Vigo, 2009. <http://programacionjade.wikispaces.com/>

Apèndix A

Posada en marxa des d'Eclipse

En aquest annex s'explica cóm obrir amb Eclipse el codi font del nostre projecte per a poder fer proves, modificacions o afegir noves funcionalitats, especialment pensant en l'etapa de integració amb els projectes relacionats. Aquest tutorial facilitarà la feina als futurs projectistes que treballin en la mateixa línia d'investigació els propers anys.

A.1 Obtenció i instal·lació de la plataforma de JaDE i l'afegit de mobilitat

En aquest punt donem per fet que es disposa de l'entorn de desenvolupament d'Eclipse, que pot ser descarregat directament des de la majoria de repositoris de les distribucions Linux, i de la JRE de JAVA 6.

Podem descarregar la plataforma de JaDE des de <*http://jade.cselt.it/*> i el mòdul de migració des de <*https://tao.uab.cat/ipmp/*>, tenim la opció de descarregar els binaris ja compilats o baixar el codi font i realitzar la compilació al nostre equip. Les versions utilitzades en el desenvolupament del nostre projecte han estat la 3.6.1 de JaDE i la 1.104 pel mòdul de migració.

Una vegada tenim tot el necessari, col·loquem la carpeta de JaDE en una ubicació coneguda i situem en el seu interior el *plugin* de mobilitat, fent servir l'esquema de directoris existent dins el fitxer comprimit descarregat (*/add-*

ons/migration/). Aquest esquema de directoris no és obligatori però sí resulta recomanable.

A.2 Apertura del projecte i configuració de les llibreries a Eclipse

Des de Eclipse, establim l'espai de treball i creem un nou projecte indicant la carpeta amb el nostre codi, com es pot veure a la figura A.1. Eclipse ja ens detecta automàticament l'estructura de paquets existent.

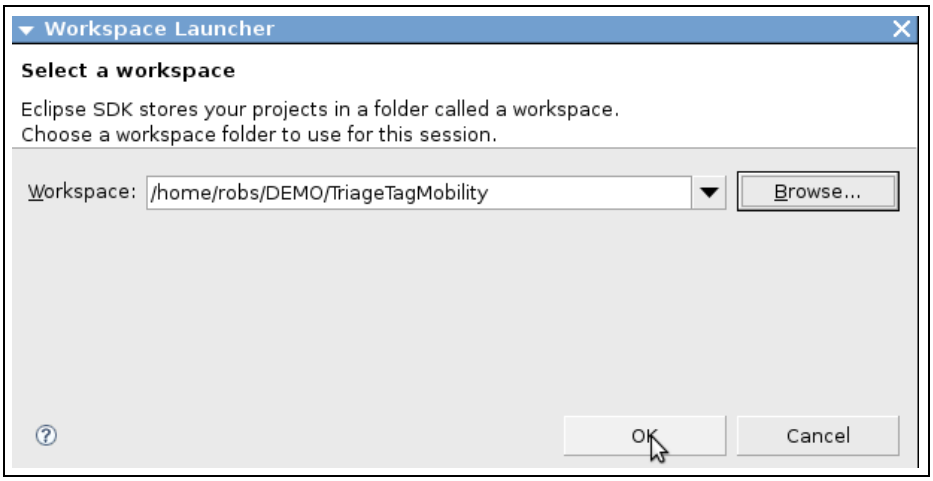

Figura A.1: Creació del projecte.

A continuació indiquem la ubicació de les llibreries JaDE com es veu a la figura A.2, fent ús de la opció *add external JARs* a la pestanya *Libraries*. Concretament hem d'afegir les que es troben a:

- *jade/lib*
- *jade/lib/common-codecs*
- *jade/add-ons/migration/lib*

També a la pestanya *Libraries* afegim la JRE de JAVA 6 fent ús de la opció *Add Library* com es mostra a la figura A.3. En el cas que no es trobi instal·lada

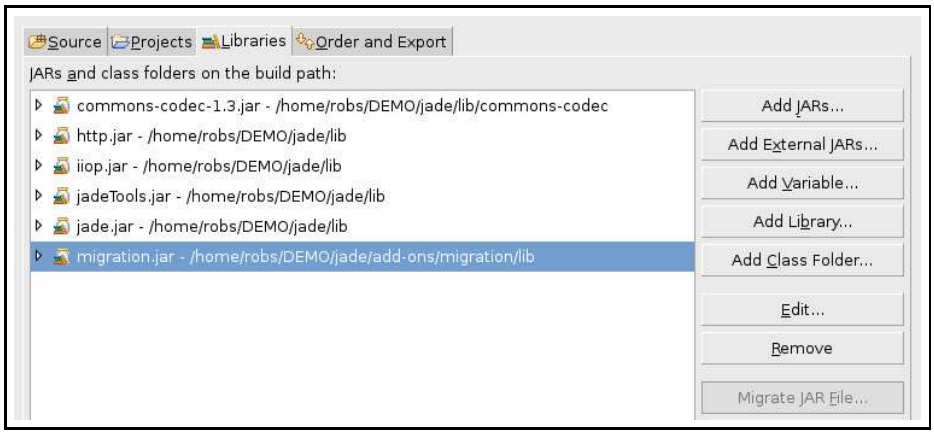

Figura A.2: Afegint les llibreries.

al nostre equip podem fer servir una versió anterior o bé descarregar-la amb el paquet *java-6-sun-1.6.0.14*, indicant a Eclipse la seva localització (habitualment */usr/lib/jvm/java-6-sun*).

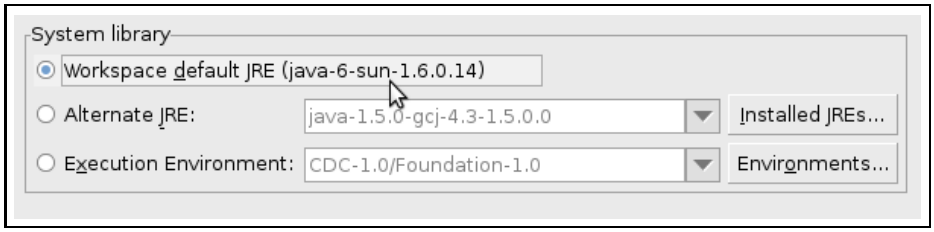

Figura A.3: Selecció de la JRE de JAVA.

Amb això ja tenim configurat el projecte en Eclipse i podem començar a treballar amb ell.

A.3 Configuració d'una prova de funcionament

Fent ús del quadre de diàleg *Run* crearem una nova configuració d'execució com es mostra a la figura A.4. Cal indicar que la *Main Class* és *jade.Boot* i també especificar els arguments com es veu a la figura A.5, que són els següents:
-gui -services

jade.core.mobility.AgentMobilityService;
jade.core.migration.InterPlatformMobilityService;
jade.core.event.NotificationService;
SMA:mabett.agents.SMA MA:mabett.agents.MA ETTMA:mabett.agents.ETTMA

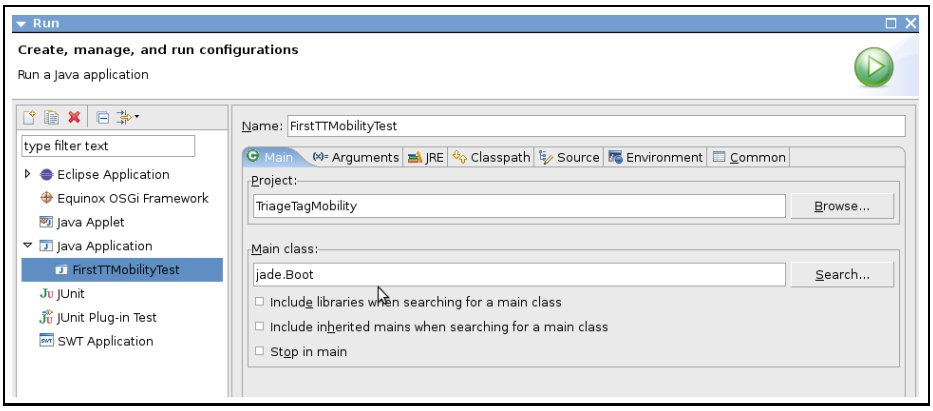

Figura A.4: Configuració d'execució.

Aquests paràmetres tenen el següent significat:

- **-gui**: obre la interfície gràfica de JaDE, que ens serveix per crear i eliminar agents així com enviar missatges amb la plataforma en funcionament.
- **-services <serviceName>**: indiquem els serveis que farem servir, en el nostre cas els serveis de mobilitat i notificació d'events. No hi ha d'haver cap espai de separació entre serveis, només el punt i coma, com es veu a la figura A.5.
- **<agentName>:<className>**: podem afegir una llista d'agents a generar en inicialitzar la plataforma posant-los com a paràmetres separats per un espai en blanc. Per aquesta prova crearem un SMA, un MA i un ETTMA.

El resultat és el que es pot veure a la figura A.6, amb els tres agents funcionant, comunicant-se i actuant segons els seus comportaments dins la plataforma local. En el següent annex es mostren exemples més específics de funcionament i els *logs* que deixen els agents a les diferents plataformes per les que passen.

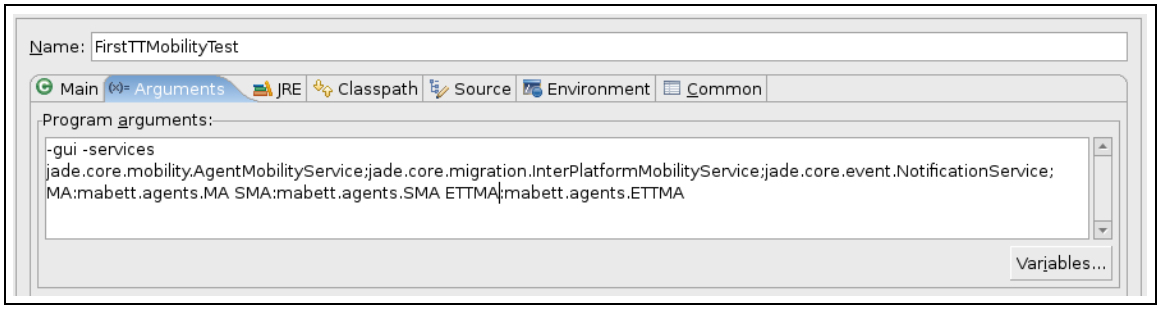

Figura A.5: Exemple d'arguments per a la execució.

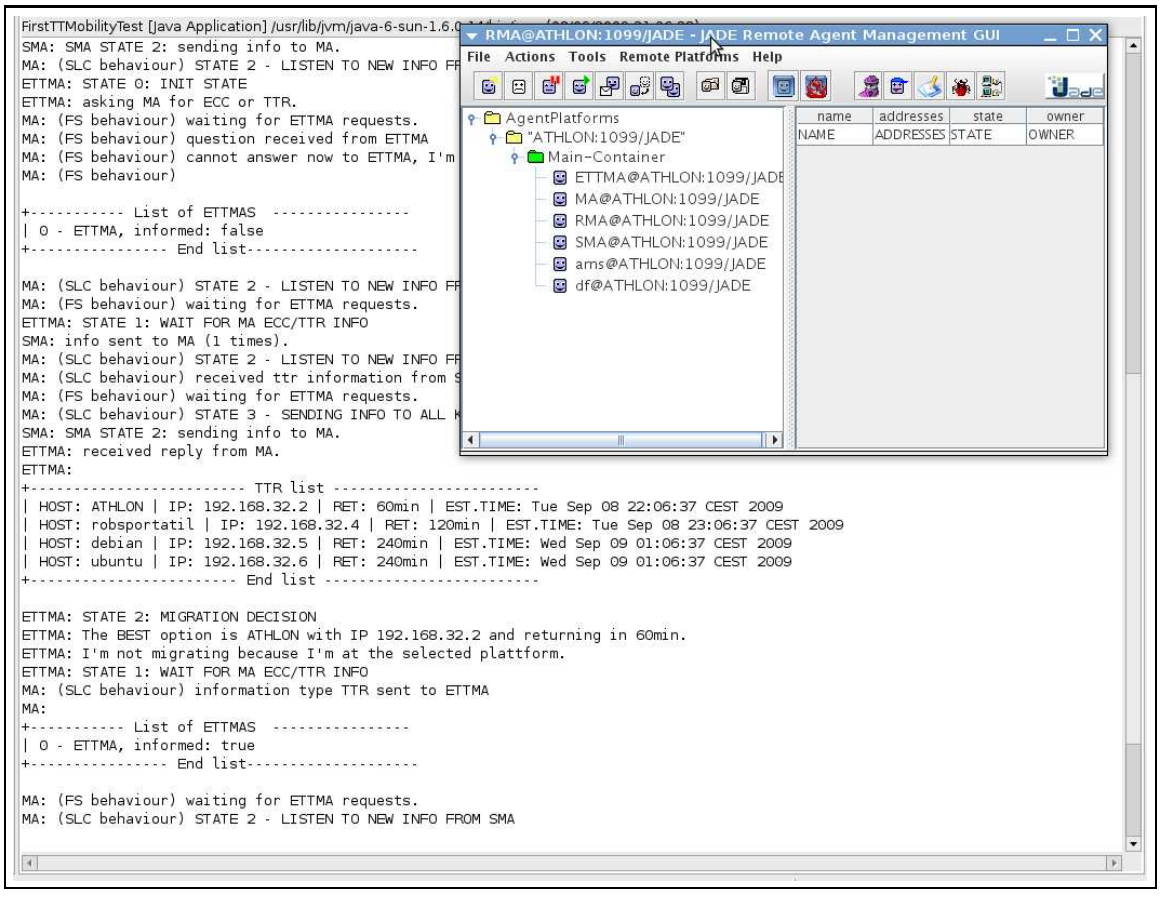

Figura A.6: Execució en marxa.

Apèndix B

Proves de funcionament

En aquest annex es mostren tres proves de funcionament que verifiquen una bona part part del sistema generat en aquest projecte.

B.1 Primera prova: migració per TTR

En la primera prova es fan migrar dos agents ETTMA amb els noms *ETTMA_1* i *ETTMA_2* seguint el criteri de menor TTR. Aquesta és la més complexa de les tres i cal estudiar amb cura el comportament de cadascun dels agents.

- El primer agent ETTMA es genera abans que el *Dummy* SMA realitzi l'enviament de dades al MA. Com que l'agent de gestió no disposa d'informació per respondre la primera pregunta del de transport, el deixa en espera i no contesta fins que rep la informació de l'estat de la xarxa.
- El segon ETTMA es crea posteriorment, des de la interfície gràfica de de JaDE com es pot veure a la figura B.1, simulant la creació a partir de la interfície gràfica. Com que ara l'agent MA ja té informació respon instantàniament des del comportament *MAFSBehaviour*.

Els ETTMAS són creats des de l'equip que té nom *robsportatil* i els valors de TTR han estat preparats per que decideixin migrar a *athlon* i quedar-se allà, a continuació es mostren alguns fragments dels registres que han deixat aquests agents

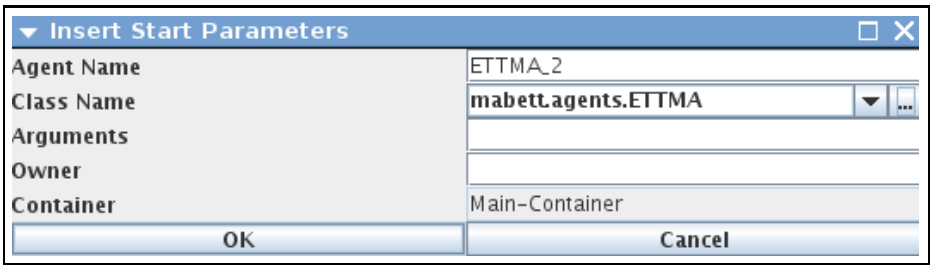

Figura B.1: Creació d'un ETTMA des de la interfície de JaDE.

en els equips. S'ha de tenir en compte que no existeix compartició d'informació entre els SMA de les dues plataformes i els temps de retorn varien uns segons entre elles, per la diferència de temps en iniciar la prova a una i altra màquina.

MA (robsportatil)

NEW EXECUTION: Sat Sep 12 20:14:51 CEST 2009

———————————

Sat Sep 12 20:14:51 CEST 2009 FINE - MA: (SLC behaviour) STATE 0 - SEND LOCAL TTR INFO TO SMA
Sat Sep 12 20:14:51 CEST 2009 FINE - MA: (FS behaviour) waiting for ETTMA requests.
Sat Sep 12 20:14:51 CEST 2009 FINE - MA: (SLC behaviour) STATE 1 - SUBSCRIBE TO SMA
Sat Sep 12 20:14:51 CEST 2009 FINE - MA: (SLC behaviour) STATE 2 - LISTEN TO NEW INFO FROM SMA
Sat Sep 12 20:14:51 CEST 2009 FINE - MA: (FS behaviour) question received from ETTMA_1
Sat Sep 12 20:14:51 CEST 2009 FINE - MA: (FS behaviour) cannot answer now to ETTMA_1, I'm waiting for info from SMA.
Sat Sep 12 20:14:51 CEST 2009 FINE - MA: (FS behaviour)
+——— List of ETTMAS ———-
| 0 - ETTMA_1, informed: false
+———- End list———
Sat Sep 12 20:14:56 CEST 2009 FINE - MA: (SLC behaviour) received ttr information from SMA
Sat Sep 12 20:14:56 CEST 2009 FINE - MA: (SLC behaviour) STATE 3 - SENDING INFO TO ALL KNOWN ETTMAs
Sat Sep 12 20:14:56 CEST 2009 FINE - MA: (SLC behaviour) information type TTR sent to ETTMA_1
Sat Sep 12 20:14:56 CEST 2009 FINE - MA: (SLC behaviour)
+——— List of ETTMAS ———-
| 0 - ETTMA_1, informed: true
+———- End list———
Sat Sep 12 20:15:11 CEST 2009 FINE - MA: (FS behaviour) question received from ETTMA_2
Sat Sep 12 20:15:11 CEST 2009 FINE - MA: (FS behaviour) information TTR sent to ETTMA_2.
Sat Sep 12 20:15:11 CEST 2009 FINE - MA: (FS behaviour)
+——— List of ETTMAS ———-
| 0 - ETTMA_1, informed: true
| 1 - ETTMA_2, informed: true
+———- End list———

Sat Sep 12 20:15:26 CEST 2009 FINE - MA: (SLC behaviour) received ttr information from SMA
Sat Sep 12 20:15:26 CEST 2009 FINE - MA: (SLC behaviour) STATE 3 - SENDING INFO TO ALL KNOWN ETTMAs
Sat Sep 12 20:15:26 CEST 2009 FINE - MA: (SLC behaviour) I'm looking for ETTMA ETTMA_1.
Sat Sep 12 20:15:26 CEST 2009 FINE - MA: (SLC behaviour) agent ETTMA ETTMA_1 not present in local plattform. Removed from ETTMA list.
Sat Sep 12 20:15:26 CEST 2009 FINE - MA: (SLC behaviour) I'm looking for ETTMA ETTMA_2.
Sat Sep 12 20:15:26 CEST 2009 FINE - MA: (SLC behaviour) agent ETTMA ETTMA_2 not present in local plattform. Removed from ETTMA list.
+——— List of ETTMAS ————-
+————- End list————————

MA (athlon)

NEW EXECUTION: Sat Sep 12 20:14:47 CEST 2009
————————————————
Sat Sep 12 20:14:47 CEST 2009 FINE - MA: (SLC behaviour) STATE 0 - SEND LOCAL TTR INFO TO SMA
Sat Sep 12 20:14:47 CEST 2009 FINE - MA: (FS behaviour) waiting for ETTMA requests.
Sat Sep 12 20:14:47 CEST 2009 FINE - MA: (SLC behaviour) STATE 1 - SUBSCRIBE TO SMA
Sat Sep 12 20:14:47 CEST 2009 FINE - MA: (SLC behaviour) STATE 2 - LISTEN TO NEW INFO FROM SMA
Sat Sep 12 20:14:52 CEST 2009 FINE - MA: (SLC behaviour) received ttr information from SMA
Sat Sep 12 20:14:52 CEST 2009 FINE - MA: (SLC behaviour) STATE 3 - SENDING INFO TO ALL KNOWN ETTMAs
Sat Sep 12 20:14:52 CEST 2009 FINE - MA: (SLC behaviour) no ETTMAs present at local plattform.
Sat Sep 12 20:14:52 CEST 2009 FINE - MA: (SLC behaviour)
+——— List of ETTMAS ————-
+————- End list————————
Sat Sep 12 20:14:52 CEST 2009 FINE - MA: (SLC behaviour) STATE 2 - LISTEN TO NEW INFO FROM SMA
Sat Sep 12 20:14:59 CEST 2009 FINE - MA: (FS behaviour) question received from ETTMA_1
Sat Sep 12 20:14:59 CEST 2009 FINE - MA: (FS behaviour) information TTR sent to ETTMA_1.
Sat Sep 12 20:14:59 CEST 2009 FINE - MA: (FS behaviour)
+——— List of ETTMAS ————-
| 0 - ETTMA_1, informed: true
+————- End list————————
Sat Sep 12 20:15:11 CEST 2009 FINE - MA: (FS behaviour) question received from ETTMA_2
Sat Sep 12 20:15:11 CEST 2009 FINE - MA: (FS behaviour) information TTR sent to ETTMA_2.
Sat Sep 12 20:15:11 CEST 2009 FINE - MA: (FS behaviour)
+——— List of ETTMAS ————-
| 0 - ETTMA_1, informed: true
| 1 - ETTMA_2, informed: true
+————- End list————————
Sat Sep 12 20:15:22 CEST 2009 FINE - MA: (SLC behaviour) received ttr information from SMA
Sat Sep 12 20:15:22 CEST 2009 FINE - MA: (SLC behaviour) STATE 3 - SENDING INFO TO ALL KNOWN ETTMAs
Sat Sep 12 20:15:22 CEST 2009 FINE - MA: (SLC behaviour) I'm looking for ETTMA ETTMA_1.
Sat Sep 12 20:15:22 CEST 2009 FINE - MA: (SLC behaviour) agent ETTMA ETTMA_1 found!
Sat Sep 12 20:15:22 CEST 2009 FINE - MA: (SLC behaviour) information type TTR sent to ETTMA_1
Sat Sep 12 20:15:22 CEST 2009 FINE - MA: (SLC behaviour) I'm looking for ETTMA ETTMA_2.
Sat Sep 12 20:15:22 CEST 2009 FINE - MA: (SLC behaviour) agent ETTMA ETTMA_2 found!
Sat Sep 12 20:15:22 CEST 2009 FINE - MA: (SLC behaviour) information type TTR sent to ETTMA_2

ETTMA_1 (robsportatil)

NEW EXECUTION: Sat Sep 12 20:14:51 CEST 2009

———————————————

Sat Sep 12 20:14:51 CEST 2009 FINE - ETTMA_1: STATE 0: INIT STATE
Sat Sep 12 20:14:51 CEST 2009 FINE - ETTMA_1: asking MA for ECC or TTR.
Sat Sep 12 20:14:51 CEST 2009 FINE - ETTMA_1: STATE 1: WAIT FOR MA ECC/TTR INFO
Sat Sep 12 20:14:56 CEST 2009 FINE - ETTMA_1: received reply from MA.
Sat Sep 12 20:14:56 CEST 2009 FINE - ETTMA_1:
+————————- TTR list ————————
| HOST: athlon | IP: 192.168.32.2 | RET: 30min | EST.TIME: Sat Sep 12 20:44:56 CEST 2009
| HOST: robsportatil | IP: 192.168.32.4 | RET: 60min | EST.TIME: Sat Sep 12 21:14:56 CEST 2009
| HOST: debian | IP: 192.168.32.5 | RET: 240min | EST.TIME: Sun Sep 13 00:14:56 CEST 2009
| HOST: ubuntu | IP: 192.168.32.6 | RET: 240min | EST.TIME: Sun Sep 13 00:14:56 CEST 2009
+————————— End list ————————-
Sat Sep 12 20:14:56 CEST 2009 FINE - ETTMA_1: STATE 2: MIGRATION DECISION
Sat Sep 12 20:14:56 CEST 2009 FINE - ETTMA_1: The BEST option is athlon with IP 192.168.32.2 and returning in 30min.
Sat Sep 12 20:14:56 CEST 2009 FINE - ETTMA_1: STATE 3: MIGRATION STATE
Sat Sep 12 20:14:56 CEST 2009 FINE - ETTMA_1: migrating FROM robsportatil TO athlon with IP: 192.168.32.2
———————————————

END OF EXECUTION

ETTMA_1 (athlon)

NEW EXECUTION: Sat Sep 12 20:14:59 CEST 2009

———————————————

Sat Sep 12 20:14:59 CEST 2009 FINE - ETTMA_1: STATE 0: INIT STATE
Sat Sep 12 20:14:59 CEST 2009 FINE - ETTMA_1: I'm coming from robsportatil and I have migrated 1 times
Sat Sep 12 20:14:59 CEST 2009 FINE - ETTMA_1: asking MA for ECC or TTR.
Sat Sep 12 20:14:59 CEST 2009 FINE - ETTMA_1: STATE 1: WAIT FOR MA ECC/TTR INFO
Sat Sep 12 20:14:59 CEST 2009 FINE - ETTMA_1: received reply from MA.
Sat Sep 12 20:14:59 CEST 2009 FINE - ETTMA_1:
+————————- TTR list ————————
| HOST: athlon | IP: 192.168.32.2 | RET: 29min | EST.TIME: Sat Sep 12 20:44:44 CEST 2009
| HOST: robsportatil | IP: 192.168.32.4 | RET: 59min | EST.TIME: Sat Sep 12 21:14:44 CEST 2009
| HOST: debian | IP: 192.168.32.5 | RET: 239min | EST.TIME: Sun Sep 13 00:14:44 CEST 2009
| HOST: ubuntu | IP: 192.168.32.6 | RET: 239min | EST.TIME: Sun Sep 13 00:14:44 CEST 2009
+————————— End list ————————-
Sat Sep 12 20:14:59 CEST 2009 FINE - ETTMA_1: STATE 2: MIGRATION DECISION
Sat Sep 12 20:14:59 CEST 2009 FINE - ETTMA_1: The BEST option is athlon with IP 192.168.32.2 and returning in 29min.
Sat Sep 12 20:14:59 CEST 2009 FINE - ETTMA_1: I'm not migrating because I'm at the selected plattform.
Sat Sep 12 20:14:59 CEST 2009 FINE - ETTMA_1: STATE 1: WAIT FOR MA ECC/TTR INFO

ETTMA_2 (robsportatil)

NEW EXECUTION: Sat Sep 12 20:15:11 CEST 2009
———————————————
Sat Sep 12 20:15:11 CEST 2009 FINE - ETTMA_2: STATE 0: INIT STATE
Sat Sep 12 20:15:11 CEST 2009 FINE - ETTMA_2: asking MA for ECC or TTR.
Sat Sep 12 20:15:11 CEST 2009 FINE - ETTMA_2: STATE 1: WAIT FOR MA ECC/TTR INFO
Sat Sep 12 20:15:11 CEST 2009 FINE - ETTMA_2: received reply from MA.
Sat Sep 12 20:15:11 CEST 2009 FINE - ETTMA_2:
+————————————- TTR list ————————————
| HOST: athlon | IP: 192.168.32.2 | RET: 29min | EST.TIME: Sat Sep 12 20:44:56 CEST 2009
| HOST: robsportatil | IP: 192.168.32.4 | RET: 59min | EST.TIME: Sat Sep 12 21:14:56 CEST 2009
| HOST: debian | IP: 192.168.32.5 | RET: 239min | EST.TIME: Sun Sep 13 00:14:56 CEST 2009
| HOST: ubuntu | IP: 192.168.32.6 | RET: 239min | EST.TIME: Sun Sep 13 00:14:56 CEST 2009
+————————————— End list ————————————-
Sat Sep 12 20:15:11 CEST 2009 FINE - ETTMA_2: STATE 2: MIGRATION DECISION
Sat Sep 12 20:15:11 CEST 2009 FINE - ETTMA_2: The BEST option is athlon with IP 192.168.32.2 and returning in 29min.
Sat Sep 12 20:15:11 CEST 2009 FINE - ETTMA_2: STATE 3: MIGRATION STATE
Sat Sep 12 20:15:11 CEST 2009 FINE - ETTMA_2: migrating FROM robsportatil TO athlon with IP: 192.168.32.2
———————————————
END OF EXECUTION

ETTMA_2 (athlon)

NEW EXECUTION: Sat Sep 12 20:15:11 CEST 2009
———————————————
Sat Sep 12 20:15:11 CEST 2009 FINE - ETTMA_2: STATE 0: INIT STATE
Sat Sep 12 20:15:11 CEST 2009 FINE - ETTMA_2: I'm coming from robsportatil and I have migrated 1 times
Sat Sep 12 20:15:11 CEST 2009 FINE - ETTMA_2: asking MA for ECC or TTR.
Sat Sep 12 20:15:11 CEST 2009 FINE - ETTMA_2: STATE 1: WAIT FOR MA ECC/TTR INFO
...

B.2 Segona prova: migració per ECC

En aquesta prova migrem per ECC fent servir la política *Random*, comprovant que en arribar al centre de coordinació l'agent entrega les dades (encara de forma simulada) i es destrueix.

Mostrem el *log* només de un agent ETTMA, ja que el del MA és pràcticament el mateix que en el cas de TTR perquè actua de la mateixa manera.

ETTMA_1 (robsportatil)

NEW EXECUTION: Sat Sep 12 20:20:51 CEST 2009

——————————————

Sat Sep 12 20:20:51 CEST 2009 FINE - ETTMA_1: STATE 0: INIT STATE
Sat Sep 12 20:20:51 CEST 2009 FINE - ETTMA_1: asking MA for ECC or TTR.
Sat Sep 12 20:20:51 CEST 2009 FINE - ETTMA_1: STATE 1: WAIT FOR MA ECC/TTR INFO
Sat Sep 12 20:20:56 CEST 2009 FINE - ETTMA_1: received reply from MA.
Sat Sep 12 20:20:56 CEST 2009 FINE - ETTMA_1:
+——— ECC list ———
| athlon - 192.168.32.2
| debian - 192.168.32.6
+——— End list ———-
Sat Sep 12 20:20:56 CEST 2009 FINE - ETTMA_1: STATE 2: MIGRATION DECISION
Sat Sep 12 20:20:56 CEST 2009 FINE - ETTMA_1: The RANDOM ECC chosen is athlon with IP 192.168.32.2.
Sat Sep 12 20:20:56 CEST 2009 FINE - ETTMA_1: STATE 3: MIGRATION STATE
Sat Sep 12 20:20:56 CEST 2009 FINE - ETTMA_1: migrating FROM robsportatil TO athlon with IP: 192.168.32.2

——————————————

END OF EXECUTION

ETTMA_1 (athlon)

NEW EXECUTION: Sat Sep 12 20:20:58 CEST 2009

——————————————

Sat Sep 12 20:20:58 CEST 2009 FINE - ETTMA_1: STATE 4: DATA DELIVERY STATE
Sat Sep 12 20:20:58 CEST 2009 FINE - ETTMA_1: delivering data to ECC after 1 migrations. (Not implemented yet!)
Sat Sep 12 20:20:58 CEST 2009 FINE - ETTMA_1: deleting agent ETTMA_1

B.3 Tercera prova: migració fallida

En la tercera i darrera prova que mostrarem en aquest annex, es prova a fer migrar un agent de la màquina *robsportatil* a *athlon* per TTR, indicant un màxim de tres intents davant migracions fallides.

L'agent no aconsegueix migrar en els tres intents per problemes en la connexió i ha de tornar a posar-se en espera, quan es recupera la connexió i l'agent SMA torna a notificar la situació de la xarxa, el ETTMA sí aconsegueix migrar. La simulació d'aquesta situació es realitza tancant temporalment la plataforma JaDE en el terminal de destí *athlon* i tornant-la a arrencar despés que l'agent ETTMA_1 hagi decidit esperar.

ETTMA_1 (robsportatil)

NEW EXECUTION: Sat Sep 12 20:23:44 CEST 2009

———————————————

Sat Sep 12 20:23:44 CEST 2009 FINE - ETTMA_1: STATE 0: INIT STATE
Sat Sep 12 20:23:44 CEST 2009 FINE - ETTMA_1: asking MA for ECC or TTR.
Sat Sep 12 20:23:44 CEST 2009 FINE - ETTMA_1: STATE 1: WAIT FOR MA ECC/TTR INFO
Sat Sep 12 20:23:49 CEST 2009 FINE - ETTMA_1: received reply from MA.
Sat Sep 12 20:23:49 CEST 2009 FINE - ETTMA_1:
+————————————- TTR list ————————————
| HOST: athlon | IP: 192.168.32.2 | RET: 30min | EST.TIME: Sat Sep 12 20:53:49 CEST 2009
| HOST: robsportatil | IP: 192.168.32.4 | RET: 60min | EST.TIME: Sat Sep 12 21:23:49 CEST 2009
| HOST: debian | IP: 192.168.32.5 | RET: 240min | EST.TIME: Sun Sep 13 00:23:49 CEST 2009
| HOST: ubuntu | IP: 192.168.32.6 | RET: 240min | EST.TIME: Sun Sep 13 00:23:49 CEST 2009
+————————————— End list —————————————-
Sat Sep 12 20:23:49 CEST 2009 FINE - ETTMA_1: STATE 2: MIGRATION DECISION
Sat Sep 12 20:23:49 CEST 2009 FINE - ETTMA_1: The BEST option is athlon with IP 192.168.32.2 and returning in 30min.
Sat Sep 12 20:23:49 CEST 2009 FINE - ETTMA_1: STATE 3: MIGRATION STATE
Sat Sep 12 20:23:49 CEST 2009 FINE - ETTMA_1: migrating FROM robsportatil TO athlon with IP: 192.168.32.2
Sat Sep 12 20:23:50 CEST 2009 FINE - ETTMA_1: STATE 0: INIT STATE
Sat Sep 12 20:23:50 CEST 2009 FINE - ETTMA_1: migration failure detected. I'm at robsportatil instead of athlon, trying again.
Sat Sep 12 20:23:57 CEST 2009 FINE - ETTMA_1: STATE 3: MIGRATION STATE
Sat Sep 12 20:23:57 CEST 2009 FINE - ETTMA_1: migrating FROM robsportatil TO athlon with IP: 192.168.32.2
Sat Sep 12 20:23:57 CEST 2009 FINE - ETTMA_1: STATE 0: INIT STATE
Sat Sep 12 20:23:57 CEST 2009 FINE - ETTMA_1: migration failure detected. I'm at robsportatil instead of athlon, trying again.
Sat Sep 12 20:23:57 CEST 2009 FINE - ETTMA_1: STATE 3: MIGRATION STATE
Sat Sep 12 20:23:57 CEST 2009 FINE - ETTMA_1: migrating FROM robsportatil TO athlon with IP: 192.168.32.2
Sat Sep 12 20:23:57 CEST 2009 FINE - ETTMA_1: STATE 0: INIT STATE
Sat Sep 12 20:23:57 CEST 2009 FINE - ETTMA_1: cannot migrate, returning to wait state.
Sat Sep 12 20:23:57 CEST 2009 FINE - ETTMA_1: STATE 1: WAIT FOR MA ECC/TTR INFO
Sat Sep 12 20:24:19 CEST 2009 FINE - ETTMA_1: received reply from MA.
Sat Sep 12 20:24:19 CEST 2009 FINE - ETTMA_1:
+————————————- TTR list ————————————
| HOST: athlon | IP: 192.168.32.2 | RET: 29min | EST.TIME: Sat Sep 12 20:53:49 CEST 2009
| HOST: robsportatil | IP: 192.168.32.4 | RET: 59min | EST.TIME: Sat Sep 12 21:23:49 CEST 2009
| HOST: debian | IP: 192.168.32.5 | RET: 239min | EST.TIME: Sun Sep 13 00:23:49 CEST 2009
| HOST: ubuntu | IP: 192.168.32.6 | RET: 239min | EST.TIME: Sun Sep 13 00:23:49 CEST 2009
+————————————— End list —————————————-
Sat Sep 12 20:24:19 CEST 2009 FINE - ETTMA_1: STATE 2: MIGRATION DECISION
Sat Sep 12 20:24:19 CEST 2009 FINE - ETTMA_1: The BEST option is athlon with IP 192.168.32.2 and returning in 29min.
Sat Sep 12 20:24:19 CEST 2009 FINE - ETTMA_1: STATE 3: MIGRATION STATE
Sat Sep 12 20:24:19 CEST 2009 FINE - ETTMA_1: migrating FROM robsportatil TO athlon with IP: 192.168.32.2

———————————————

END OF EXECUTION

ETTMA_1 (athlon)

NEW EXECUTION: Sat Sep 12 20:24:22 CEST 2009

Sat Sep 12 20:24:22 CEST 2009 FINE - ETTMA_1: STATE 0: INIT STATE

Sat Sep 12 20:24:22 CEST 2009 FINE - ETTMA_1: I'm coming from robsportatil and I have migrated 1 times

Sat Sep 12 20:24:22 CEST 2009 FINE - ETTMA_1: asking MA for ECC or TTR.

Sat Sep 12 20:24:22 CEST 2009 FINE - ETTMA_1: STATE 1: WAIT FOR MA ECC/TTR INFO

Firmat: Robert Sallent Lopez

Bellaterra, setembre de 2009

Resum

En les actuacions de rescat en emergències amb un gran nombre de víctimes, una de les primeres tasques a realitzar pels serveis sanitaris és la de triar-les en funció del seu estat i la urgència amb que han de ser tractades. En el grup SeNDA es treballa en un sistema d'agents mòbils que automatitza el mecanisme manual de triatge, fent ús de dispositius electrònics i una xarxa MANET com a canal de comunicació. En aquest projecte es proposa un sistema de gestió de mobilitat pels agents que transporten dades de víctimes fins al centre de coordinació, juntament amb un mecanisme de control d'aquests agents totalment transparent a l'usuari.

Resumen

En las actuaciones de rescate en emergencias con un gran número de víctimas, una de las primeras tareas a realizar por los servicios sanitarios es la de seleccionarlas en función de su estado y la urgencia con la que deben ser tratadas. En el grupo SeNDA se trabaja en un sistema de agentes móviles que automatiza el mecanismo manual de selección, haciendo uso de dispositivos electrónicos y una red MANET como canal de comunicación. En este proyecto se propone un sistema de gestión de movilidad para los agentes que transportan los datos de las víctimas hasta el centro de coordinación, junto con un mecanismo de control de estos agentes totalmente transparente al usuario.

Abstract

In rescue proceedings for emergencies with a large number of victims, one of the first tasks to be performed by health services, is to classify victims on the basis of their status and the urgency with which must be treated. The SeNDA group is working on a mobile agent system that automates the manual selection mechanism, using electronic devices and a MANET network as communication channel. This project proposes a mobility management system for agents transporting victim information to the emergency coordination center, and a control mechanism of these agents fully transparent to the final user.