UNIVERSITÀ DEGLI STUDI DI PADOVA

FACOLTÀ DI INGEGNERIA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA SPECIALISTICA IN INGEGNERIA INFORMATICA

Tesi di Laurea:

UN SISTEMA MULTIAGENTE PERVASIVO PER L'ASSISTENZA SANITARIA DOMESTICA

Relatore: Prof. Carlo Ferrari

Laureando: Francesco Fassina

ANNO ACCADEMICO 2009-2010

Alla mia famiglia

Indice

1. Introduzione

In questa tesi verrà esposto un progetto di un sistema distribuito pervasivo basato su una piattaforma ad agenti per l'assistenza sanitaria domestica.

Negli ultimi anni, con la rapida diffusione di dispositivi wireless (notebook, cellulari, ecc), gli utenti hanno modificato il loro modo di utilizzare i sistemi informatici, scoprendo la possibilità di poter accedere a tutte le informazioni che necessitano quando e dove desiderano.

In precedenza, le macchine di calcolo erano utilizzate per lavoro o per studio ed erano principalmente limitate a dispositivi fissi, spesso situati in uffici, case o centri che potevano offrire accesso ad internet. Con l'avvento delle tecnologie mobili gli utenti hanno iniziato ad utilizzare diversi dispositivi anche allo stesso tempo: telefoni cellulari in grado di accedere ad Internet, leggere ed inviare mail, oltre a notebook in grado di svolgere azioni che prima erano limitate ai soli computer fissi.

Le capacità di calcolo di questi dispositivi mobili, grazie al progredire sia dell'hardware sia del software, sono sempre più potenti arrivando spesso a sostituire gli stessi computer fissi.

Molte di queste macchine dispongono di tecnologie wired e wireless che permettono loro di poter essere collegate ad altri dispositivi simili, ma anche molto diversi da un punto di vista sia hardware

sia software (si pensi ad esempio al collegare un telefono cellulare ad un computer) creando molto spesso dei sistemi distribuiti.

L'obiettivo principale di questo tipo di sistemi è fornire accesso a delle risorse condivise: un tipico scenario è quello di una stampante in una rete di computer. Nella condivisione si deve tenere conto che più utenti possono accedere alla risorsa e che essa potrebbe non essere al momento disponibile o che potrebbe guastarsi. Inoltre dato che di solito i sistemi distribuiti sono generalmente costituiti da parti eterogenee e indipendenti, non si vuole dover modificare ogni singolo dispositivo quando il sistema viene modificato.

Pertanto nella progettazione di tali sistemi è importante tenere presenti aspetti come:

- Concorrenza: gestire tutta quelle serie di politiche che permettano a più utenti di usufruire, anche contemporaneamente, della stessa risorsa.
- Sicurezza: non si vuole che risorse private di un utente siano accessibili anche ad altri.
- Tolleranza ai guasti: se una risorsa viene meno non deve pregiudicare il funzionamento del sistema stesso
- Scalabilità: all'introduzione o alla rimozione di risorse, il sistema deve continuare a funzionare normalmente.

Dal punto di vista realizzativo spesso si vogliono nascondere all'utilizzatore molte caratteristiche proprie di un sistema distribuito, presentandogli un ambiente omogeneo. In particolare possono essere nascosti la locazione, l'eterogeneità dei dispositivi, delle risorse e dei collegamenti, nonché la condivisione concorrenziale, guasti e introduzione o rimozione di componenti.

Molto spesso i sistemi distribuiti, per presentare un ambiente omogeneo, sono organizzati come un layer software logicamente posizionato fra il livello che comprende il sistema operativo e le funzionalità di comunicazione di base e il livello delle applicazioni e degli utenti come mostrato nella seguente figura. Tali sistemi, per questo motivo, vengono definiti middleware.

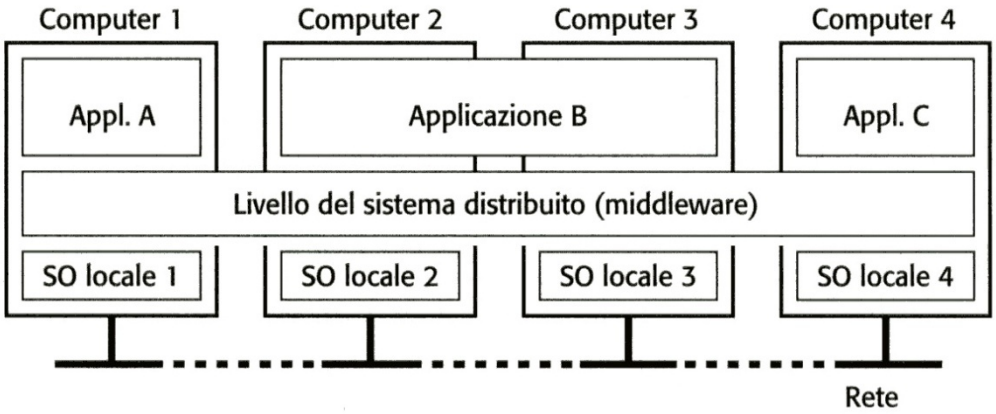


Figura 1: Architettura middleware di un sistema distribuito

Uno degli aspetti più interessanti nell'evoluzione dei calcolatori è nella proliferazione di dispositivi sempre più piccoli, dai PDA ai wearable computer (dispositivi di piccole dimensioni non intrusivi che possono essere posizionati sul corpo dell'utilizzatore come fossero parti del proprio vestiario), ai dispositivi per il sensing e il controllo remoto che, seppure di limitate capacità di memoria e potenza di calcolo, letteralmente pervadono l'ambiente, interagendo fra di essi, generando così un sistema pervasivo o ubiquo.

Il termine "Ubiqutos Computing", definito da Mark Wiser nel 1988 durante la docenza come *Chief Technologist* (Ingegnere capo) presso il Palo Alto Research Center, sta proprio ad indicare questo modello di computazione in cui l'uomo (o l'oggetto di studio) è immerso in un ambiente intelligente, dove i dispositivi interagiscono tra loro, in maniera autonoma.

La caratteristica principale di questo tipo di sistemi è la mobilità dei propri dispositivi, spesso composti da piccoli calcolatori mobili ed embedded, principalmente alimentati a batteria e con una connessione quasi esclusivamente wireless. Il comportamento normale di questi sistemi è governato dall'instabilità: dato che i dispositivi sono mobili, la topologia della rete tenderà a variare nel tempo non garantendo più un routing ottimo o addirittura rendendo impossibile l'accesso a determinati nodi o risorse; le connessioni wireless inoltre sono generalmente soggette ad interferenza e perdita di segnale a seconda dell'ambiente in cui si trovano; Inoltre l'alimentazione a batteria potrebbe richiede un utilizzo programmato dei tempi di accensione e spegnimento dei vari dispositivi. Pertanto problematiche come la gestione dell'eterogeneità dei dispositivi e dei colle-

gamenti, la tolleranza ai guasti e la scalabilità del sistema sono aspetti particolarmente importanti in questo contesto.

Un settore dove viene svolta molta ricerca sui sistemi distribuiti pervasivi è quello dell'assistenza sanitaria (healthcare). Il vantaggio nell'utilizzare questi sistemi sta nella possibilità di poter seguire un gran numero di pazienti contemporaneamente risparmiando risorse. Questi sistemi possono essere utilizzati sia in ambito ospedaliero, ad esempio per il monitoraggio dei pazienti pre o post intervento, sia in ambito domestico evitando così dei ricoveri dispendiosi per le organizzazioni sanitarie. Dal lato paziente questi vantaggi si traducono nel fatto che le persone con particolari problemi di salute, non debbano più essere confinate in ospedali o nelle loro abitazioni immobilizzati ad apparecchi di controllo fissi e spesso ingombranti, ma possano condurre una vita relativamente normale, sapendo che ogni loro problema medico verrà tempestivamente rilevato e, in caso di necessità, verrà fornito loro aiuto.

Il sistema per l'assistenza sanitaria domestica, presentato in questa tesi, rappresenta un esempio di questi sistemi. Generalmente sono composti da un numero variabile di sensori disposti sul corpo del paziente organizzati in una body area network (BAN) solitamente wireless. Lo scopo di questo tipo di sistemi è quello di monitorare le funzionalità del paziente senza impedirne i movimenti. Per la rete di sensori sono possibili due diverse configurazioni: nella prima i sensori inviano i propri dati ad una piccola unità centrale portata dal paziente che li memorizza e successivamente li deposita in un server; nella seconda la BAN è agganciata tramite un collegamento wireless a un dispositivo fisso che memorizza i dati provenienti dai sensori.

Per quanto riguarda la rilevazione delle emergenze sono possibili, in questo scenario, diverse strategie. Le più diffuse in letteratura sono principalmente di due tipi: l'analisi dei dati in place, ovvero il dispositivo che raccoglie i dati dei sensori, li analizza e decide se vi sono le condizioni per lanciare l'emergenza; oppure l'analisi dei dati dopo che questi sono stati trasmessi e memorizzati da un'unità esterna alla BAN come ad esempio un server. Entrambi i metodi offrono vantaggi e svantaggi: nel caso dell'analisi dei dati in place i vantaggi consistono nella velocità di rilevazione, mentre, dato che questa analisi viene limitata ad un ristretto numero di valori per le scarse capacità di memoria del dispositivo, potrebbero verificarsi dei falsi allarmi; nel caso dell'analisi dei dati in remoto, disponendo di più dati, si avrebbe un minor numero di falsi positivi, tuttavia i tempi di rilevazione delle emergenze si allungherebbero.

Nella tesi vengono presi in considerazione, oltre la gestione dei sensori anche gli applicativi dal lato medico nonché, per gestire le emergenze, anche quelli relativi al personale medico di supporto. Tutti elementi emersi nello studio delle caratteristiche del sistema, dopo aver analizzato altri lavori simili (capitolo 2) verranno discusse nel capitolo 3.

Infine la tesi tratta anche della possibilità di utilizzare, per questo tipo di sistema, una piattaforma ad agenti.

Con il termine agente si è soliti indicare un particolare software che è in grado di agire con un certo grado di autonomia al fine di svolgere compiti per conto dei suoi utenti. A differenza del paradigma ad oggetti, in cui la definizione di un elemento si basa su metodi e attributi, nel paradigma ad agenti la definizione è basata più sul comportamento di questi ultimi.

In letteratura esistono diverse definizioni di caratteristiche che fanno riferimento agli agenti, queste possono essere riassunte nelle seguenti proprietà:

- Autonomia: gli agenti possono decidere di svolgere compiti o attività anche complessi senza bisogno dell'intervento umano
- Abilità sociali: gli agenti sono in grado di comunicare con i loro simili per coordinarsi e per completare una certa attività
- Reattività: a seconda del contesto in cui opera, un agente è in grado di reagire in modo appropriato e coerente al contesto.

Esistono diverse piattaforme per lo sviluppo di sistemi multi agente, tra queste per la realizzazione della tesi è stato utilizzato JADE (Java Agent Development Framework), realizzato in Java che supporta lo sviluppo di applicazioni distribuite, fornendo un insieme di servizi di base, conformi allo standard FIPA[1].

I dettagli relativi alla la descrizione funzionale del sistema verranno esposti nel capitolo 4, mentre nel capitolo 5 verrà presentato JADE e il software utilizzato. Nel capitolo 6 infine verrà proposta una possibile realizzazione.

[1] FIPA (Foundation for Intelligent Physical Agents) è un'organizzazione fondata nel 1996 per lo sviluppo e la standardizzazione di sistemi ad agenti. Gli standard proposti comprendono una serie di specifiche sull'implementazione e la comunicazione in questo tipo di architetture. Per maggiori dettagli consultare [2] in bibliografia.

2. Stato dell'arte

Il settore dei sistemi distribuiti per l'healthcare è molto vario ed eterogeneo[2], molto spesso i progetti sviluppati riguardano diversi ambiti: si va ad esempio da quello strettamente domestico a quello ospedaliero fino a quello per l'assistenza agli anziani.

In alcuni lavori[3] si pone particolare attenzione ai dispositivi hardware utilizzati nel sistema, analizzando quali debbano essere le caratteristiche e le problematiche.

Come detto nel capitolo introduttivo il sistema è solitamente composto da una serie di sensori collegati ad un dispositivo centrale. Il collegamento è solitamente wireless e si utilizzano reti con bassa copertura come ad esempio Bluetooth o ZigBee.

In particolare l'utilizzo di un collegamento Bluetooth consente per alcuni lavori di utilizzare hardware già presente in commercio, come avviene in WAITER[4] che verrà affrontato nel capitolo 2.2.

I sensori che vengono generalmente più utilizzati in questi sistemi sono i seguenti:

[2] Vedere [17] in bibliografia
[3] Vedere [14] in bibliografia
[4] Vedere [12] in bibliografia

- accelerometro a doppio asse: sensore che misura le variazioni di acceleratore a cui è sottoposto il soggetto in studio. La misurazione viene effettuata rilevando l'inerzia di una massa sospesa in un ambiente elastico e sottoposta ad accelerazione. Con l'utilizzo di questo sensore é possibile stabilire l'attività del paziente, come verificare una caduta quando il valore misurato è molto elevato
- termometro: un sensore in grado di rilevare la temperatura corporea o le sue variazioni quando questo è posto a contatto con la cute del paziente.
- cardiofrequenzimetro: sensore in grado di rilevare la frequenza cardiaca in tempo reale, utile per verificare eventuali situazioni di stress cardiache.
- ECG (elettrocardiogramma): sensore che misura l'attività elettrica del cuore, di solito la misurazione avviene tramite una serie di elettrodi che vengono posti sul torace del paziente. Il segnale trasmesso genera un tipico tracciato composto dall'onda P, dal complesso QRS, dall'onda T ed eventualmente dall'onda U. Dallo studio di questo tracciato possono essere evidenziate aritmie cardiache.
- Sensore SpO_2: misura la pressione di saturazione dell'ossigeno a livello ematico tramite un sistema di rilevamento a luce rossa o infrarossa. È solitamente utilizzato per rilevare problemi cardiorespiratori.
- sfigmomanometro: è un sensore in grado di rilevare la pressione minima (diastolica) e massima (sistolica). Generalmente è composto da un manicotto collegato ad un dispositivo elettronico legato al braccio del paziente e, per funzionare, necessita del cardiofrequenzimetro, poiché la misurazione è basata sulla variazione del battito cardiaco quando il manicotto viene gonfiato e sgonfiato. La rilevazione richiede generalmente fra i 10 e i 20 secondi quindi, a differenza dei sensori precedenti, non può essere effettuata in real time.

Oltre ai sensori precedentemente descritti e disposti sul corpo dei pazienti, alcuni lavori[5] utilizzano anche sensori e dispositivi che fanno parte dell'ambiente in cui si muove il paziente (come bilance per misurare il peso corporeo) e che vengono utilizzati saltuariamente dal soggetto in studio. Le problematiche affrontate in questo contesto sono tipiche di una DTN (Delay Tollerant Network), ovvero di una rete in cui le informazioni, data la mobilità del paziente e con esso della BAN, possono essere momentaneamente non disponibili per poi diventarlo successivamente.

[5] Vedere [16] in bibliografia

La rete dei sensori può essere collegata ad un server che memorizza i dati. Questo collegamento può avvenire in diversi modi sfruttando reti diverse, ad esempio Amon[6] utilizza un collegamento cellulare via sms (si veda il Cap 2.1), altri come WAITER, un collegamento internet su rete gsm, ecc.

Gli applicativi sviluppati sono spesso molto simili: generalmente danno la possibilità di visualizzare i valori dei parametri dei sensori per quanto riguarda il paziente e offrono capacità di analisi avanzate dal punto di vista medico.

Nella successiva analisi vengono presentati i lavori che hanno maggiori caratteristiche in comune con il progetto sviluppato, sia da un punto di vista dei problemi affrontati, sia da un punto di vista delle soluzioni trovate.

2.1. Amon

Il primo sistema presentato nel dettaglio è Amon, un sistema di monitoraggio e allarme medico basato su più parametri che utilizza un dispositivo indossabile, progettato soprattutto per pazienti con gravi problemi cardiorespiratori.

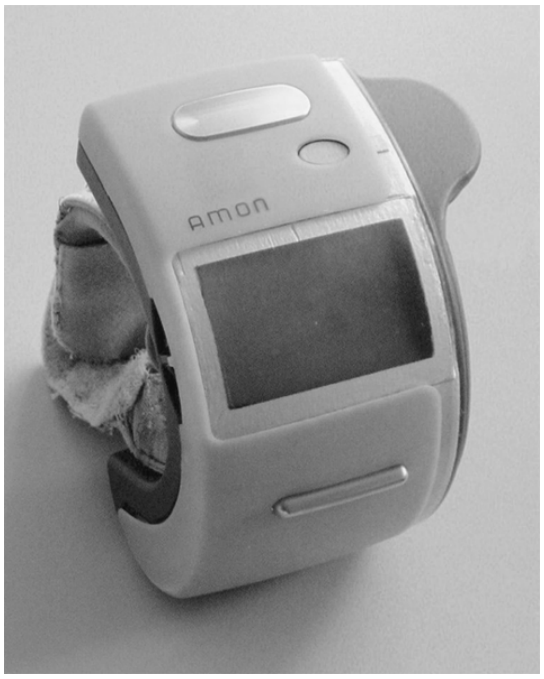


Figura 2.1: Prototipo Amon

[6] Vedere [11] in bibliografia

Dal punto di vista hardware, Amon si compone di due dispositivi: il primo a forma di bracciale che incorpora diversi sensori ed è in grado tra l'altro di misurare la pressione arteriosa, L'ECG e l'attività del paziente tramite un accelerometro a doppio asse. Inoltre fornisce un piccolo led che permette di visualizzare le varie misurazioni effettuate. Il secondo dispositivo è una unità stazionaria situata presso il centro di telemedicina (TMC). I due dispositivi sono connessi mediante un collegamento cellulare GSM/UMTS e la comunicazione avviene tramite l'invio di SMS. I dati raccolti vengono consegnati al TMC tre volte al giorno.

La rilevazione delle emergenze può essere svolta sia in place sia in modo remoto. In place vengono semplicemente stabilite delle soglie per ogni parametro preso in considerazione: quando uno di questi valori supera tale soglia scatta l'emergenza. In modo remoto invece vengono svolti calcoli specifici per ogni parametro che si vuole prendere in considerazione in modo da gestire eventuali emergenze non rilevate dal dispositivo.

L'applicativo, residente al centro di telemedicina, è stato realizzato in Java e consente la visualizzazione dei vari dati dei pazienti e offre la possibilità di analizzarli e ordinarli per gravità tramite software di terze parti. Offre inoltre la possibilità di consultare dati storici relativi al paziente (cartella clinica) e di poter segnalare eventuali emergenze.

Uno degli aspetti positivi del sistema è senz'altro l'utilizzo di un unico dispositivo che, in modo non intrusivo, è in grado di rilevare eventuali anomalie tempestivamente e in modo aggressivo. Dall'altro lato l'utilizzo della di linee di comunicazione GSM/UMTS proprietarie rende di fatto il sistema costoso, anche se, con l'impiego di questi canali, sarebbe possibile individuare un dispositivo Amon all'interno di una cella del sistema cellulare in modo da poter inviare sul posto i mezzi di soccorso senza bisogno di informare direttamente il centro medico dell'eventuale posizione del paziente.

2.2. WAITER

Il secondo sistema preso in considerazione è WAITER. Come nel caso di Amon, si tratta di un sistema per il monitoraggio di pazienti in grado di rilevare eventuali anomalie.

A differenza del sistema precedente dove si aveva un'unica entità che inviava dati ad un server centrale, WAITER, dal punto di vista hardware è composto da più di un dispositivo.

Le misurazioni dei parametri vitali del paziente avvengono mediante un apparecchio di piccole dimensioni che viene posto dietro il padiglione auricolare ed è costituito da tre sensori, il primo misura la frequenza cardiaca, il secondo la temperatura e il terzo è un accelerometro a doppio asse per misurare l'attività del paziente. Questo dispositivo è inoltre in grado di connettersi tramite bluetooth per l'invio dei dati raccolti.

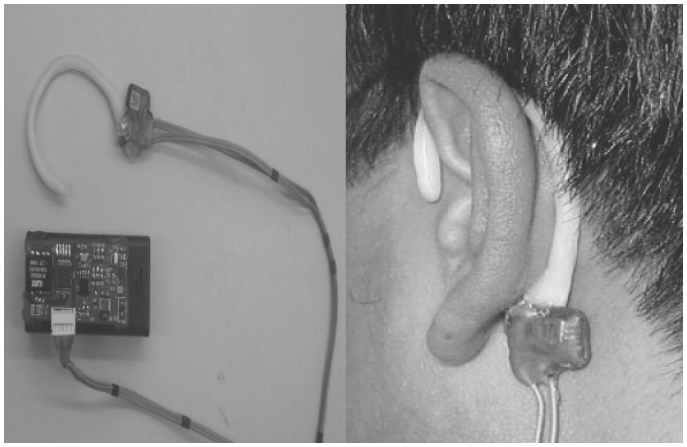


Figura 2.2: Prototipo WAITER

La novità di questo sistema è che è possibile utilizzare come dispositivo di raccolta e analisi dei dati anche un comune cellulare, dotato di ricetrasmettitore bluetooth, su cui sia installato J2ME (Java 2 Micro Edition). La trasmissione di questi dati dai sensori avviene ogni 0,05 secondi. L'applicativo sviluppato consente di visualizzare i vari dati in forma semplificata dando modo all'utente di verificare il corretto funzionamento del sistema.
I dati raccolti infine vengono spediti una volta all'ora tramite connessione GSM ad un database server presso un Healthcare Center dove il personale addetto si occuperà di visualizzare e controllare i parametri dei pazienti.

La rilevazione delle emergenze, anche in questo caso, avviene una volta che i dati sono stari ricevuti sul cellulare. Come in Amon, la rilevazione è aggressiva e avviene tramite valori di soglia. Una volta che l'emergenza è lanciata, viene contattato il server dell'Healthcare Center dove il personale addetto provvederà a validarla o meno.

L'aspetto positivo di questo sistema è senza dubbio nell'utilizzo di dispositivi già in commercio, sfruttando le capacità di questi di connettersi a più reti nello stesso momento, e nell'implementazione multipiattaforma utilizzando applicativi Java.

2.3. Sistema ad agenti di K. Miller e S. Sankaranarayanan[7]

In letteratura esistono alcuni sistemi che utilizzano il paradigma ad agenti per gestire un sistema di monitoraggio per l'assistenza sanitaria. Il più simile a quello trattato nel progetto di tesi per quanto riguarda le procedure realizzate è senz'altro quello di K. Miller e S. Sankaranarayanan, benché si occupi di un ambito leggermente diverso.

Infatti, tecnicamente, il sistema è progettato per essere utilizzato in ambito ospedaliero, anche se molte sue soluzioni potrebbero essere applicate anche in quello domestico.

Da un punto di vista hardware, come nel progetto precedente vi sono diversi dispositivi. La misurazione dei parametri vitali avviene tramite sensori (nella realizzazione svolta dagli autori si fa riferimento ad un solo sensore per la temperatura corporea, ma potrebbe essere facilmente esteso ad altri). Questi sensori inviano i loro dati ad un'unità non necessariamente propria del paziente che poi li inoltra al database dove vengono memorizzati e gestiti

A differenza delle applicazioni precedenti, dato che ci si trova in ambito ospedaliero, non è necessaria una visualizzazione dei parametri dei sensori da parte del paziente. Questi infatti possono essere depositati su un dispositivo mobile appartenente al personale infermieristico dell'ospedale per poi essere memorizzati su un database. La visualizzazione dei dati avviene quindi in maniera relativamente approfondita solo sul dispositivo mobile dell'infermiere fornendo inoltre alcune possibili interazioni con i vari sensori e in maniera approfondita sull'elaboratore del medico.

La rilevazione delle anomalie o emergenze, avviene sia in place, sia in remoto, stabilendo dei valori di soglia per ogni misurazione dei sensori.

[7] Vedere [13] in bibliografia

Per quanto riguarda l'implementazione, la novità in questo sistema è nell'utilizzo della piattaforma ad agenti Jade-Leap. Essa è leggermente diversa rispetto a quella utilizzata per lo sviluppo del sistema di questa tesi, poiché è fondamentalmente pensata per applicazioni su dispositivi mobili, quindi con ridotte capacità di memoria e di calcolo.

Gli agenti presenti nel sistema sono principalmente di quattro tipi:

- L'agente sensore: si occupa di recuperare le informazioni dai sensori e, su richiesta del nurse agent, li invia a quest'ultimo.
- Il nurse agent: si occupa di raccogliere e analizzare i dati dei sensori, e, periodicamente invia questi dati al database agent. Se viene rilevata un'emergenza viene visualizzato un messaggio di allarme.
- L'agente database: ha il compito di ricevere i dati dal nurse agent e memorizzarli nel database, dopodiché avvia un sottoagente che analizza i dati, se questo rileva un'emergenza avvia un nuovo agente addetto all'anomalia: il wardboy agent
- Il wardboy agent: è un agente in grado di migrare. Si occupa di trovare il dottore disponibile e specializzato in quel tipo di emergenza fornendogli le informazioni relative all'anomalia rilevata.

Come si vedrà in seguito, nella realizzazione del lavoro in tesi, sono stati prese come punto di partenza alcune soluzioni adottate in questo paper, cercando, dove possibile, di adattarle o migliorarle.

3. Analisi dei requisiti del sistema

Il problema preso in considerazione è quello di fornire un servizio di monitoraggio e di assistenza medica per problemi fisici non gravi in modo da poter evitare, per quanto possibile, il ricovero ospedaliero utilizzando apparecchi che non limitino la libertà di movimento della persona che li utilizza.

Nella progettazione di un sistema che coinvolga più entità in grado di collaborare per la realizzazione del servizio preso in esame, è necessario per prima cosa individuare quali siano gli utilizzatori del sistema stesso.

Data la definizione del problema, la figura principale sarà sicuramente il paziente i cui dati vitali andranno misurati. Questi andranno successivamente validati e controllati da del personale esperto quale, ad esempio, potrebbe essere un medico del centro che offre il servizio di healthcare. In caso di emergenza sarà inoltre necessario attuare eventuali procedure per intervenire, quindi utilizzeranno il sistema anche degli operatori sanitari sul territorio (d'ora in poi chiamati per brevità OST) che, ad esempio, potrebbero essere costituiti da personale addetto all'intervento sui mezzi di soccorso.

In base alle figure rilevate è possibile stabilire quali possano essere le loro aspettative sul sistema basandosi in modo particolare sui lavori presi in esame nel capitolo precedente e sui paper esaminati e presenti in bibliografia.

Nel proseguo del capitolo verranno prima analizzate le aspettative del sistema da parte delle tre figure rilevate, successivamente queste aspettative saranno tradotte in requisiti di sistema.

Richieste del paziente

Dall'analisi della letteratura a disposizione, si possono estrarre diverse caratteristiche che solitamente possono essere di interesse per un paziente sotto monitoraggio.

Generalmente i dispositivi di monitoraggio non devono intralciare le normali attività giornaliere di una persona: devono essere piccoli e semplici da utilizzare e, con pochi comandi, si vuole mostrare che i dispositivi funzionino adeguatamente visualizzando in forma semplificata i dati correnti raccolti.

La funzione principale del dispositivo deve essere quella di avvisare tempestivamente, in caso di emergenza, dando al paziente la sicurezza dell'arrivo di soccorsi in tempi rapidi.

Richieste del personale medico

Generalmente l'utilizzo del sistema da parte del medico avverrà tramite un calcolatore fisso o portatile.

La consultazione dei dati da parte del medico è decisamente più approfondita di quella del paziente, che richiederà pochi parametri da visualizzare. Infatti, di solito, le applicazioni dal lato medico consentono non solo di visualizzare i dati raccolti periodicamente sui pazienti, ma anche di analizzarli statisticamente e poter seguire la storia clinica accedendo alla cartella clinica.

Oltre a questo, un medico deve essere immediatamente informato se su un paziente viene rilevata un'emergenza. Anche nel caso del medico, infine, potrebbe esserci la necessità di segnalare eventuali allarmi dovuti a situazioni rilevate dopo la consultazione storica dei dati del paziente.

Richieste degli Operatori Sanitari Sul Territorio (OST)

Per quanto riguarda gli operatori sanitari sul territorio, la loro funzione principale è quella di intervenire in caso di emergenza. Solitamente sui mezzi di intervento è presente del personale qualificato in grado di risolvere eventuali anomalie.

Pertanto l'utilizzo del sistema che viene svolto da parte di un OST è molto simile a quello di un medico: oltre a sapere l'ubicazione del paziente sarà necessario accedere ai dati dei sensori e alla cartella clinica, in particolare ai dati relativi ad eventuali allergie o patologie sofferte dal paziente in modo da evitare complicazioni in un'eventuale somministrazione di farmaci. In genere al termine di ogni soccorso verrà compilato un apposito modulo (ad esempio il modulo 118) contenente le informazioni sull'intervento.

3.1. Requisiti del sistema

In base alla breve analisi sulle potenziali richieste del sistema è possibile stabilire quali debbano essere le caratteristiche hardware e software per svolgere una eventuale realizzazione.

La trattazione analizzerà prima i requisiti hardware e in seguito, per ciascuna delle tre figure, i requisiti software.

3.1.1. Hardware

Per il monitoraggio di un paziente saranno necessari una serie di sensori per acquisirne i vari parametri vitali. Questi sensori necessiteranno, ad esempio, di un trasmettitore bluetooth per inviare

i propri dati ad un dispositivo che li analizzi sul posto per poi spedirli ad un server dove verranno memorizzati stabilmente. Nella realizzazione, dato che erano i più diffusi in letteratura, si è pensato di utilizzare questi sensori:

- Accelerometro a doppio asse per la misurazione dell'attività del paziente, in grado di rilevare eventuali cadute dello stesso.
- Termometro per la misurazione della temperatura
- Sfigmomanometro per la misurazione della pressione
- Cardiofrequenzimetro per la misurazione delle pulsazioni cardiache.

Per la ricezione dei dati dei dispositivi sarà necessario utilizzare un dispositivo portatile: un esempio potrebbe essere un dispositivo già presente in commercio come nel caso di WAITER, quale un telefono cellulare o un PDA dotato di connessione wi-fi per il collegamento con il database, bluetooth per quanto riguarda la comunicazione con i vari sensori.

Inoltre sarà necessario disporre di uno o più server per il deposito dei dati degli utenti, nonché delle loro cartelle cliniche. Oltre a questo il server potrebbe servire anche per depositare i dati relativi ai medici e al personale OST.

Per quanto riguarda i medici, essi necessiteranno di un calcolatore portatile o fisso cui svolgere le loro azioni e che abbia la possibilità di essere connesso wired o wireless alla rete per essere collegato al database.

Infine gli OST, data la loro necessità di spostarsi, dovrebbero disporre di un dispositivo portatile quale ad esempio un notebook o anche un pda dotato di connessione wireless

3.1.2. Software

Gli applicativi a livello software saranno diversi a seconda della diversa figura individuata all'inizio di questo capitolo: paziente, medico o personale OST.

Lato paziente

L'applicativo paziente dovrà essere per prima cosa leggero, dovendo essere eseguito su un dispositivo mobile. L'interfaccia offerta dovrà essere semplice, in alcuni casi potrebbe essere anche una semplice visualizzazione dei valori dei vari sensori.

Per collegare i vari dispositivi sarà necessario instaurare una piccola rete (ad esempio bluetooth) centralizzata sul dispositivo di raccolta. Pertanto l'applicazione dovrà disporre delle procedure per connettersi su tale rete.

Oltre a questo, l'applicativo sul dispositivo dovrà essere in grado di collegarsi al database, ciò richiede anche di instaurare una connessione tcp-ip con il server, nonché disporre delle procedure di accesso in scrittura di dati in un database.

Per quanto riguarda la gestione delle emergenze sarà necessario decidere come minimizzare i falsi allarmi e come queste debbano essere lanciate.

Un aspetto che dovrà essere tenuto in considerazione inoltre, per prevenire le perdite di dati dovute ad eventuali problemi, è di predisporre opportune procedure di backup dei dati ricevuti dai sensori, facendo attenzione a non occupare uno spazio eccessivo date le limitate capacità di memoria del dispositivo.

Infine per garantire la privacy dei pazienti, affinché i loro dati sensibili non vengano letti da terze parti non appartenenti al sistema si dovranno stabilire opportuni protocolli di sicurezza come accessi con password, protocolli di criptazione delle comunicazioni fra i dispositivi, ecc.

Lato medico

L'applicativo al lato medico richiede una gestione più complessa rispetto a quella sul lato paziente. L'interfaccia offerta dovrà essere completa di tutti i dati relativi al paziente, il che comprende informazioni sulle generalità, sulla cartella clinica e sui dati storici e statistici dei sensori. Dovrà quindi fornire accesso in lettura al database per poter scaricare i dati dei sensori e le generalità dei pazienti. Sia in lettura sia in scrittura per poter scaricare e modificare la cartella clinica.

Anche sul lato medico la sicurezza è importante: per evitare che personale non autorizzato acceda a dati protetti dei pazienti sarà necessario gestire opportunamente gli accessi.

Per prevenire la perdita di dati modificati, anche l'applicativo sul lato medico dovrà disporre di opportune procedure di backup.

Infine per quanto riguarda le emergenze, l'applicativo dovrà poter ricevere, eventualmente per validare o annullare, le segnalazioni di situazioni critiche da parte del paziente e dovrà dare la possibilità al medico di avvisare di eventuali emergenze emerse dall'analisi dei dati presentati.

Lato OST

Per quanto riguarda l'applicativo utilizzato dal personale OST, esso dovrà essere molto simile a quello medico, offrendo un'interfaccia in grado di mostrare le generalità del paziente, la cartella clinica e i dati dei sensori. Inoltre dovrà essere anch'esso predisposto per accedere sia in lettura sia in scrittura al database: in lettura, come nel lato medico, per ricevere i dati dei sensori, in scrittura/lettura per la cartella clinica ed eventualmente per i moduli post intervento.

Anche in questo caso andranno predisposte sia procedure per evitare la perdita dei dati attuando politiche di backup, sia procedure di sicurezza per evitare l'accesso al sistema a personale non addetto.

Per le emergenze, dato che gli operatori sanitari sul territorio sono il personale che interviene, sarà necessaria un'opportuna procedura che stabilisca la fine dell'emergenza una volta che è stata conclusa.

4. Descrizione funzionale del sistema

Dopo aver analizzato quali debbano essere le caratteristiche hardware e software del sistema è possibile stabilire in dettaglio quali siano le funzioni e le procedure che le varie parti del sistema andranno ad assolvere.

4.1. Agenti

Dovendo realizzare un sistema pervasivo si sono volute sfruttare, nel sistema sviluppato, le potenzialità di mobilità e di autonomia offerte dal paradigma ad agenti. Pertanto per definire quali debbano essere le procedure del sistema è importante identificare per prima cosa quali agenti debbano essere presenti e successivamente quali azioni debbano compiere.

Prendendo come spunto il lavoro di K. Miller e S. Sankaranarayanan nel sistema si possono identificare due tipi di agenti: principali e secondari. I principali si occuperanno di svolgere le azioni più importanti del sistema, mentre i secondari quelle di supporto ai principali.

Gli agenti principali del sistema sono:

- Agente paziente, che svolge le operazioni di monitoraggio del paziente.
- Agente medico, che offre una serie di servizi per il medico.
- Agente OST, che riceve le segnalazioni di emergenza.
- Agente emergenza paziente, per segnalare le emergenze rilevate dall'agente paziente.
- Agente emergenza medico, per segnalare le emergenze dal lato medico.

Per quanto riguarda gli agenti di supporto, questi sono essenzialmente dei driver che consentono la comunicazione con i sensori e con il database, pertanto sono di tre tipi:

- Agente sensore: ha accesso alle procedure di comunicazione con il sensore a cui fa riferimento e invia i dati raccolti all'agente paziente.
- Agente database paziente: fornisce il solo accesso in scrittura al database e vi invia i dati. È utilizzato dal solo agente paziente.
- Agente database medico: fornisce accesso in scrittura e lettura al database. È utilizzato dall'agente medico, dall'agente emergenza paziente e dall'agente emergenza medico.

Oltre a questi possono esserci per ogni figura identificata nel capitolo precedente, degli agenti utilizzati semplicemente per loggare un utente al sistema: questi agenti prendono il nome di starter agente paziente, starter agente medico e starter agente OST.

4.2. Procedure

Agente Paziente

L'agente paziente ha il compito di ricevere i dati dei sensori, analizzarli e spedirli al database. In caso di emergenza dovrà lanciare una procedura opportuna. Come avviene in molti sistemi per l'assistenza sanitaria domestica per "emergenza rilevata" si intende un valore superiore a una soglia prefissata. Schematicamente le azioni che dovrà compiere sono le seguenti:

1. Avviare l'interfaccia paziente
2. Lanciare gli agenti sensori
3. Ricevere i dati dei sensori
4. Aggiornare i valori dell'interfaccia
5. Analizzare i dati
    - In caso di valore fuori soglia:
        a. Entrare nello stato: "emergenza lanciata" (non vengono rilevate altre emergenze mentre è in questo stato)
        b. avviare l'agente emergenza paziente fornendogli i dati dell'emergenza
        c. rimanere in attesa di eventuali messaggi di termine emergenza e, se ricevuti, terminare lo stato di allarme
6. Periodicamente lanciare l'agente database per inviare i dati dei sensori.
7. Tornare al punto 3.

Agente Medico

L'agente medico ha il compito di fornire al medico i dati da analizzare ed eventualmente segnalare le emergenze che vengono rilevate osservando i dati. Pertanto la procedura che deve eseguire è la seguente:

1. Avviare l'interfaccia del medico.
2. Ricevere input del paziente di cui si vogliono mostrare i dati
3. Avviare l'agente database per ottenere i dati richiesti
4. Rimanere in attesa di emergenze dal paziente
5. Rimanere in attesa di eventuali input del medico:
    - In caso di emergenza segnalata: avviare l'agente emergenza medico con i dati del paziente.
    - Nel caso in cui il medico aggiorni la cartella clinica: avviare l'agente database medico fornendogli i dati da inserire nel database.
    - Nel caso in cui il medico voglia cambiare paziente: tornare al punto 2.

- Nel caso in cui il medico voglia eseguire logout: terminare agente e chiudere l'interfaccia.

Agente OST

Il compito di quest'agente è sostanzialmente quello di ricevere le emergenze e segnalare che l'OST è attivo nel sistema. Le azioni che deve svolgere sono le seguenti:

1. Avviare l'interfaccia OST
2. Rimanere in attesa di eventuali emergenze
3. Rimanere in attesa di input da parte del OST
    - In caso l'OST voglia eseguire logout: terminare agente e chiudere l'interfaccia.

Agente Emergenza Paziente

L' Agente emergenza paziente è quello che sovraintende a tutta la procedura di emergenza, in pratica si occupa di avvisare il medico nel caso in cui venga rilevato un valore anomalo nei sensori. La procedura che questo agente deve svolgere è la seguente:

1. Ricevere i valori dei sensori anomali.
2. Chiedere al paziente se confermare l'emergenza (utile in caso di falso positivo)
    - Se non confermata terminare l'agente e comunicare all'agente paziente di chiudere la situazione di emergenza e terminare.
    - Se confermata:
        a. Ricercare un medico attivo
        b. Una volta trovato, migrare sul calcolatore del medico
        c. Una volta arrivato visualizzare l'interfaccia
        d. Lanciare l'agente database medico per ottenere i dati del paziente che segnala l'emergenza (cartella clinica, dati storici sensori, ecc)
        e. Ricevere i dati dal precedente agente

f. Aggiornare l'interfaccia
g. Rimanere in attesa di input:
    - In caso di emergenza non confermata: segnalare all'agente paziente di chiudere lo stato di emergenza e terminare.
    - In caso di emergenza confermata: avviare l'agente emergenza medico, inviandogli i dati del paziente e terminare.
    - In caso di aggiornamento della cartella clinica: avviare l'agente database medico fornendogli i dati da inserire nel database.

Agente Emergenza Medico

L'agente emergenza medico deve svolgere un ruolo simile a quello dell'agente precedentemente descritto, anche se le parti coinvolte sono il medico e l'OST. La procedura da svolgere sarà simile a quella dell'agente emergenza paziente:

1. Ricevere l'anomalia segnalata.
2. Chiedere al medico se confermare l'emergenza:
    - Se non confermata terminare l'agente.
    - Se confermata:
        a. Ricercare un OST attivo
        b. Una volta trovato, migrare sul calcolatore del'OST
        c. Una volta arrivato, visualizzare l'interfaccia
        d. Lanciare l'agente database medico per ottenere i dati del paziente segnalato (cartella clinica, dati storici sensori, ecc)
        e. Ricevere i dati dal precedente agente
        f. Aggiornare l'interfaccia
        g. Rimanere in attesa di input:
            - Se viene eseguita un'operazione di update dei valori della cartella clinica o altro: avviare l'agente database medico fornendogli i dati da inserire nel database.

- Se viene chiusa l'emergenza:
    - Se l'emergenza è partita dall'agente emergenza paziente: segnalare al paziente di chiudere lo stato di emergenza e terminare

Agente Sensore

È l'agente che, disponendo dei protocolli di comunicazione per il sensore a cui è associato, ha il compito di fornire i dati rilevati all'agente paziente. In generale le azioni che dovrà compiere sono le seguenti:

1. Prendere i dati del sensore.
2. Trasmettili all'agente paziente.
3. Tornare al punto 1.

Agente Database Paziente

Agente Database Paziente dispone dei protocolli di comunicazione e dei driver specifici per comunicare con database al quale invia i dati del paziente, la procedura ad esso associata è la seguente:

1. Ricevere i dati da inviare
2. Aprire una connessione con il database
3. Inviare i dati al database e, una volta inviati, chiudere la connessione
4. Terminare.

Agente Database Medico

Questo è un agente simile al precedente, ma deve essere possibile attivarlo in più modi: in lettura per leggere i dati dei pazienti per poi inviarli all'agente richiedente e in scrittura per poter memorizzare la cartella clinica. Le azioni da compiere si possono così schematizzare:

1. Ricevere tipo di attivazione
    - Se attivazione di tipo UPDATE:
        a. Ricevere i dati da aggiornare
        b. Aprire connessione con il database
        c. Inviare i dati al database
    - Se attivazione di tipo GET:
        a. Ricevere il nominativo i cui dati andranno cercati
        b. Aprire una connessione con il database
        c. Recuperare i dati e Inviare i dati recuperati al richiedente
2. Chiudere la connessione con il database e terminare

Agenti di tipo starter

Gli agenti starter si occupano di loggare l'utente a cui fanno riferimento nel sistema. Svolgeranno un accesso al database per stabilire se l'utente appartiene o meno al sistema e in seguito gli daranno l'attributo di attivo. Le attività che devono svolgere sono:

1. Avviare l'interfaccia di accesso
2. Rimanere in attesa di input (login)
3. Eseguire l'accesso al database
4. Ricercare l'utente
    - Se trovato: segnalarlo come attivo e avviare il suo agente
    - Se non trovato tornare al punto 1
5. Chiudere la connessione con il database e terminare.

5. Software Utilizzato

Nella realizzazione del progetto come detto nel capitolo di introduzione è stato utilizzato come piattaforma ad agenti JADE. Oltre a questo sono stai utilizzati altri tool e software a supporto come MySQL per la gestione del database e Java3D per la realizzazione dell'accelerometro.

5.1. JADE

JADE (Java Agent DEvelopment framework) è un ambiente di sviluppo completamente realizzato in linguaggio Java e sviluppato da Telecom Italia, disponibile open source e rilasciato sotto licenza GNU LGPL. Lo scopo principale di JADE è semplificare l'implementazione di sistemi multi agente (sistemi in cui possono risiedere molti agenti che cooperano per raggiungere uno stesso risultato) conforme allo standard FIPA.

La piattaforma ad agenti creata da JADE può essere distribuita su più macchine anche se queste non dispongono dello stesso sistema operativo. L'unico prerequisito è l'installazione di Java 1.5 o superiore.

Oltre a diversi tool che supportano sia la fase di sviluppo sia quella di debug, JADE offre un'interfaccia grafica che consente il controllo e la gestione della piattaforma ad agenti. Tramite l'interfaccia è possibile svolgere attività runtime, come: creazione, terminazione, clonazione e migrazione di agenti, nonché visualizzazione e modifica delle attività del sistema.

Container, agenti principali e AID

L'istanza principale dell'ambiente di runtime è definita con il termine "container". Nella versione standard di JADE, ogni container generalmente risiede in un host collegato alla piattaforma e ogni host può contenere uno più container. In versioni derivate da JADE, come ad esempio JADE-leap, progettato per dispositivi mobili, un container può essere distribuito su più host proprio per le minori capacità di elaborazione di queste componenti.

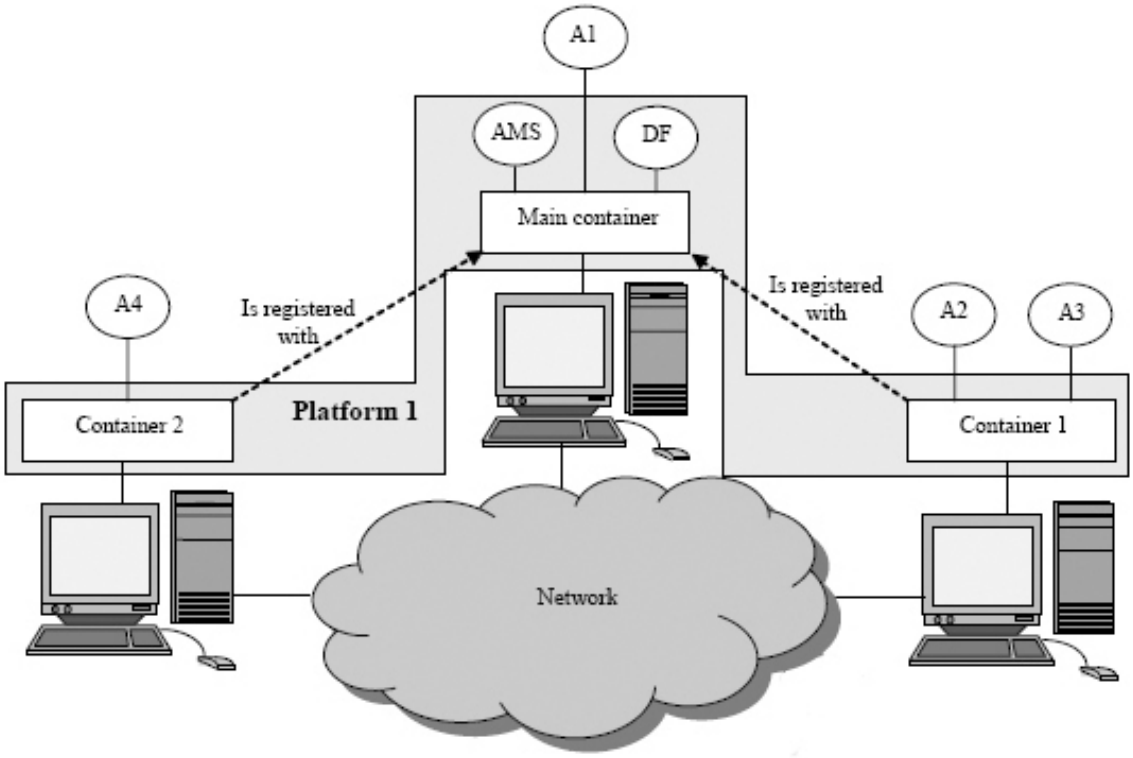


Figura 5.1.1: Piattaforma JADE

All'interno di ogni container è possibile avere uno o più agenti attivi. JADE definisce come container principale (Main container) quello che contiene due agenti molto importanti che svolgono attività di gestione e amministrazione della piattaforma: l'AMS e il DF. Se questi dovessero venir meno, verrebbe meno anche la piattaforma.

L'AMS (Agent Management System) è l'agente che viene avviato con la piattaforma. Fornisce tutta una serie di servizi di gestione e amministrazione, tra cui: si assicura che ogni agente abbia un nome univoco all'interno della piattaforma, conosce l'ubicazione di ogni agente, da la possibilità di creare e/o distruggere agenti e container anche remoti. È possibile inoltre interrogare l'AMS con protocolli di comunicazione predisposti da JADE. Un esempio di questa interazione verrà fornita quando si parlerà di migrazione.

Il DF (Directroy Facilitator) ha un compito paragonabile a quello delle pagine gialle, in altri termini gli agenti appartenenti alla piattaforma possono utilizzare questo agente per pubblicare i propri servizi offerti o, in alternativa, un agente può rivolgersi al DF per sapere quale agente offre un determinato servizio. Nel secondo caso l'agente richiedente riceverà una lista di identificativi, ciascuno dei quali è detto AID, di chi offre il servizio richiesto.

L'AID (Agent IDentifier) è l'identificativo univoco di un agente all'interno della piattaforma. L'oggetto AID in JADE si compone di un nome unico e di un indirizzo numerico. Il nome unico ha la forma di *<nickname>@<nome piattaforma>*, mentre l'indirizzo numerico è quello dell'host su cui si trova l'agente. Solitamente quest'ultimo è utilizzato solo per comunicazioni che includano più piattaforme poiché per le comunicazioni intra-platform si utilizza semplicemente il nome univoco.

Gli Agenti

L'implementazione di un agente in JADE è semplice come la realizzazione di una classe Java. inoltre tutti gli oggetti e metodi definiti in questo linguaggio possono essere utilizzati anche in JADE.

La classe che definisce l'agente deve estendere jade.core.Agent implementando il metodo setup() che serve per l'esecuzione. È possibile realizzare anche agenti che gestiscano interfacce: in questo caso, si parla di GUI agent e la classe da estendre è jade.Gui.GuiAgent. Oltre a implementare il me-

todo setup(), va implementato anche il metodo OnGuiEvent(). Tecnicamente l'interfaccia e l'agente comunicano ad eventi: l'interfaccia pubblica gli eventi sul GUI agent che si occuperà di processarli nel metodo precedentemente citato.

Come avviene per i thread Java, anche gli agenti all'interno di una piattaforma possono trovarsi in diversi stati, definiti dallo standard FIPA che rappresenta uno standard internazionale per l'interoperabilità tra agenti.

Gli stati definiti dallo standard sono i seguenti:

- INITIATED: è lo stato in cui si trova un agente appena viene creato. L'agente in questa condizione non ha ancora un nome, un indirizzo, non può comunicare e non si è registrato presso l'AMS.
- ACTIVE: è lo stato normale di esecuzione dell'agente, quando questi possiede un nome ed è registrato all'AMS, potendo usufruire dei servizi offerti dal sistema.
- SUSPENDED: l'agente in questo stato non è in grado di svolgere alcuna azione e il suo thread è sospeso.
- WAITING: l'agente è bloccato e in attesa di un determinato evento, come ad esempio per la ricezione di un messaggio.
- DELETING: l'agente viene eliminato dal sistema e non è più registrato presso l'AMS.
- TRANSIT: è lo stato in cui entra un agente che è in grado di migrare. I messaggi, inviati all'agente mentre si trova in questo stato, vengono memorizzati dal sistema per poi essere riconsegnati quando questo giunge a destinazione.

JADE da la possibilità, attraverso diversi metodi pubblici, di gestire le diverse transizioni fra gli stati. In ogni caso le transizioni seguono lo schema nella seguente figura:

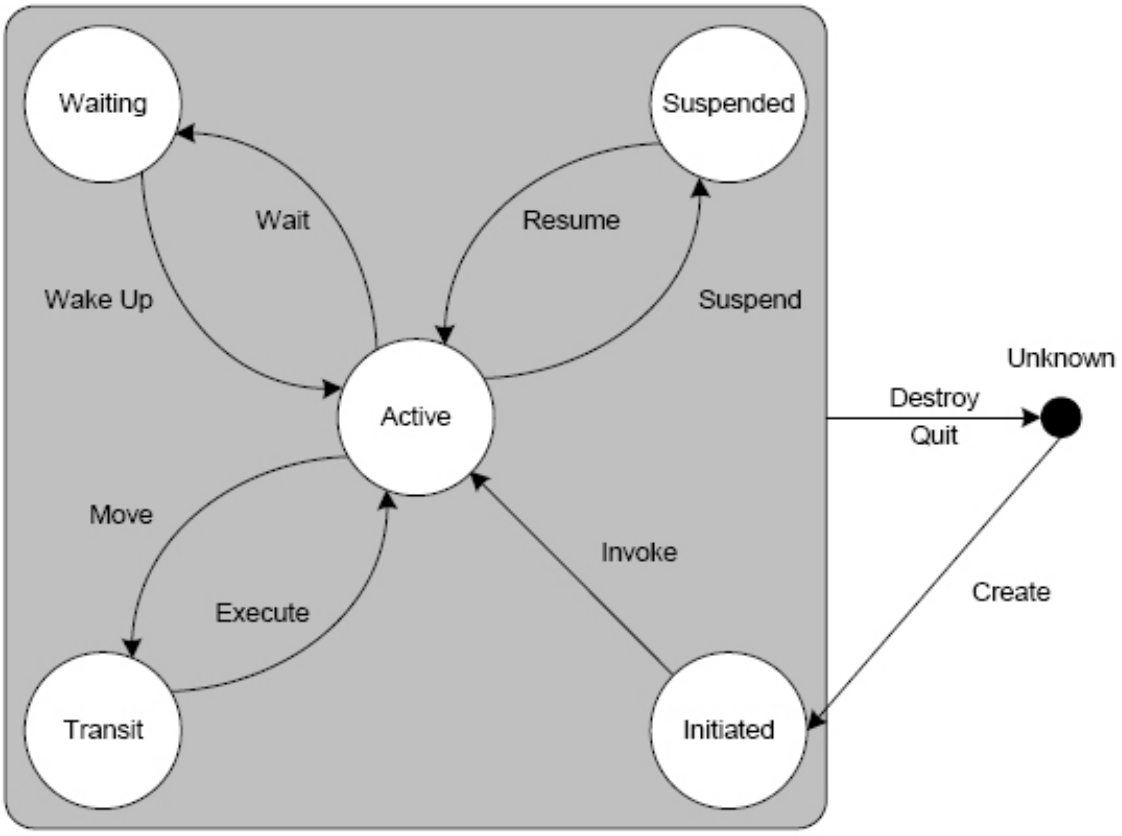


Figura 5.1.2: Ciclo di vita di un agente

I behaviours

Nella creazione di un agente è possibile stabilire diversi tipi di comportamento (behaviour). Ogni agente è in grado di gestire uno svariato numero di behaviour che possono essere eseguiti in maniera concorrenziale.

L'utilizzo di behaviours offre diversi vantaggi per un sistema ad agenti: dato che per ogni agente, corrisponde un singolo thread è possibile svolgere azioni concorrenziali anche in macchine dalle scarse capacità computazionali in quanto i behaviours, essendo più leggeri di un thread Java, aiutano nello risparmiare risorse, inoltre, appartenendo dallo stesso thread, non vi sono problemi di sincronizzazione fra behaviours concorrenti che accedono alle stesse risorse.

JADE, tramite una serie di classi astratte presenti nel package jade.core.behaviours, offre alcuni comportamenti già preimpostati in cui, di solito, vanno realizzati semplicemente il metodo che

rappresenta l'azione da compiere e quello che ne sancisce la conclusione. Fra i principali behaviour si possono ricordare:

- Il OneShotBehaviour che rappresenta un'azione che viene svolta una volta sola.
- Il CyclycBehaviour che rappresenta un comportamento ciclico.
- Il TickerBehaviour che rappresenta un'azione che viene svolta ciclicamente a intervalli di tempo costanti e decisi dal programmatore.
- Il WakerBehaviour che rappresenta un'azione in un colpo solo ritardata di una quantità temporale fissata dal programmatore.

Ovviamente sono possibili altri comportamenti non preimpostati; per ottenerli è necessario estendere la classe behaviour implementando i relativi metodi.

La comunicazione tra agenti

Una delle caratteristiche più importanti in un sistema multi agente è la possibilità che hanno questi di poter comunicare. Il paradigma adottato da JADE è quello del asynchronous message passing. Ogni agente gestisce una coda per i messaggi in ingresso dove vengono postati quelli spediti da altri agenti. Il modo di processare questi messaggi dipende dall'implementazione dell'agente.

I messaggi in JADE seguono lo standard FIPA. Questo formato comprende un gran numero di campi, fra cui:

- Il mittente del messaggio (sender).
- Una lista di destinatari (receivers).
- L'intento di comunicazione (performative) che rappresenta cosa otterrà il sender dalla comunicazione. Tra questi ci sono l'INFORM con cui il sender vuole mettere a conoscenza il/i destinatario/i di un certo fatto, la REQUEST per ottenere una risposta, QUERY_IF per conoscere se una condizione è soddisfatta, CFP (chiamata per proposta), PROPOSE, ACCEPT_PROPOSAL e REJECT_PROPOSAL per instaurare e gestire una negoziazione tra sender e receiver, ecc.

- L'informazione contenuta del messaggio (content): questo solitamente nello standar FIPA è una stringa di testo, tuttavia, JADE da la possibilità di poter inviare anche oggetti, anche se questo è sconsigliato non essendo parte dello standard.
- Il linguaggio del messaggio (language), ovvero la sintassi utilizzata nella comunicazione.
- L'ontologia del messaggio (ontology), in altri termini il vocabolario di simboli utilizzato.
- Altri campi per la gestione dei messaggi come l'utilizzo di timeout, specifiche di risposta, ecc.

Generalmente in JADE, per poter spedire un messaggio è necessario compilare solo alcuni dei campi sopra citati. Una volta composto è possibile spedire il messaggio tramite il metodo send(msg) della classe Agent. Ad esempio il codice per effettuare una comunicazione fra due agenti in cui l'argomento sia il tempo è il seguente:

```
ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
msg.addReceiver(aidDestinatario);
msg.setLanguage("Italian");
msg.setOntology("Weather-forecast-ontology");
msg.setContent("Oggi splende il sole");
send(msg);
```

dove ACLMessage.INFORM è la performative e aidDestinatario è l'identificativo univoco del receiver. Si noti infine il contenuto del messaggio che è una stringa.

Per quanto riguarda il receiver, la ricezione del messaggio avviene con il seguente frammento di codice:

```
ACLMessage msg = receive();
if (msg != null) {
// Process the message
}
```

La chiamata receive() non è bloccante, ovvero l'agente continuerà a svolgere i propri compiti anche se non ha ricevuto nulla. È possibile effettuare una ricezione bloccante utlizzando il metodo BlockingReceive().

È infine possibile stabilire quali messaggi ricevere stabilendo pattern di ricezione: l'agente processerà i soli messaggi che corrispondono a quel pattern ignorando gli altri.

Utilizzo del DF

In generale, quando viene creato, un agente non sa quali siano gli agenti presenti nel sistema. Nel caso in cui debba comunicare con un altro agente di cui non conosce l'AID, può utilizzare quel servizio di pagine gialle offerto dal DF di cui si era parlato in precedenza.

La procedura di registrazione di un servizio presso il DF avviene tramite la seguente sequenza standard di righe di codice:

```
DFAgentDescription dfd = new DFAgentDescription();
dfd.setName(getAID());
ServiceDescription servizio = new ServiceDescription();
servizio.setName("nomeservizo");
servizio.setType("tiposervizo");
dfd.addServices(servizio);
try {
  DFService.register(this, dfd);
}
catch (FIPAException fe) {fe.printStackTrace();}
```

È anche possibile cancellare la registrazione sul DF del servizio offerto. Solitamente questa operazione può essere svolta tra le istruzioni del metodo takeDown(), che viene eseguito quando l'agente viene terminato. L'utilità di tale operazione è dovuta al fatto che, in questo modo, il DF ha conoscenza dei soli servizi attivi. In ogni caso il DF periodicamente cancella le registrazioni che non sono più attive. Il frammento di codice che consente la cancellazione della registrazione sul DF del servizio offerto è il seguente:

```
try {
  DFService.deregister(this);
}
catch (FIPAException fe) {fe.printStackTrace();}
```

Nel caso in cui si ricerchi un servizio va composto un template che descriva l'oggetto della ricerca. In genere il template comprende il nome, il tipo di servizo, e altre informazioni, tuttavia non è necessario compilarlo in tutte le sue parti. Ovviamente maggiore sarà la definizione offerta, più pre-

cisa sarà la ricerca. Un esempio di righe di codice che corrispondono all'operazione di ricerca sul DF sono le seguenti:

```
DFAgentDescription template = new DFAgentDescription();
ServiceDescription sd = new ServiceDescription();
servizio.setName("nomeservizo");
servizio.setType("tiposervizo");
template.addServices(sd);
DFAgentDescription[] result = null;
{
  DFAgentDescription[] result = DFService.search(this, tem-
  plate);
}
catch (FIPAException fe) {fe.printStackTrace();}
//processa I risultati
```

In questo modo si riesce ad ottenere una lista di AID che corrispondono ad agenti che offrono il servizio identificato dal template sottoposto.

Migrazione e utilizzo dell'AMS

Un'altra caratteristica del sistema ad agenti realizzato da JADE è nel dare la capacità agli agenti di poter essere mossi all'interno di una piattaforma.

La migrazione di un agente può avvenire fra i container di una stessa piattaforma. L'agente che effettua la migrazione può essere di tipo standard (estende la classe jade.core.Agent) o di tipo GUI. Dato che i container sono distribuiti su più host, nell'effettuare la migrazione bisogna far attenzione ad eventuali risorse che potrebbero essere lasciate attive nell'host di partenza prima che l'agente migri e non essere più disponibili nell'host di arrivo. JADE per ovviare a questi problemi introduce due metodi beforeMove() e afterMove() che rispettivamente vengono eseguiti prima e dopo la migrazione: ad esempio beforeMove() può essere utilizzato per chiudere un'interfaccia e afterMove() per aprirla nel nuovo host.

Per effettuare la migrazione è necessario conoscere la posizione della destinazione. Esistono diverse possibilità su come questa ricerca può essere fatta. I principali metodi di ricerca consentono

di migrare ricercando un container date determinate caratteristiche. È, ad esempio, possibile ricercare un container dato che si conosce almeno un agente in esso contenuto. In questo caso solitamente prima si contatta il DF per avere l'AID dell'agente cercato, poi si contatta l'AMS per trovare la locazione dell'agente.

Le comunicazioni con l'AMS necessitano di un protocollo di comunicazione stabilito dallo stesso sistema: il messaggio da inviare deve contenere tutta una serie di informazioni standard. JADE offre una serie di tool che hanno proprio il compito di formattare questo messaggio secondo lo standard. In generale il codice per inviare una richiesta all'AMS per trovare un container dato l'agente è la seguente:

```
ContentManager manager = (ContentManager)getContentManager();
Codec codec = new SLCodec();
Ontology ontology = JADEManagementOntology.getInstance();
manager.registerLanguage(codec);
manager.registerOntology(ontology);
WhereIsAgentAction azione = new WhereIsAgentAction();
azione.setAgentIdentifier(agentecercato);
Action qloc = new Action(getAMS(), azione);
ACLMessage req = new ACLMessage(ACLMessage.REQUEST);
req.setSender(getAID());
req.addReceiver(getAMS());
req.setLanguage(codec.getName());
req.setOntology(ontology.getName());
manager.fillContent(req,qloc);
send(req);
```

Come si può osservare, si formula una richiesta che implica una risposta che andrà gestita dall'agente tramite una ricezione bloccante o meno.

La risposta dell'AMS, a seconda della richiesta fatta, conterrà uno o più Location (nel caso in esempio una sola poiché si chiede la locazione di un agente e gli agenti hanno un identificativo univoco). Una volta scelta La location desiderata sarà possibile effettuare una migrazione attraverso il metodo doMove(location) dell'agente.

JADE infine offre anche la possibilità di poter clonare gli agenti, in questo caso la procedura è simile per molti aspetti a quella di migrazione. Per dettagli specifici si rimanda alle guide di JADE scaricabili e consultabili presso il sito ufficiale.

Gestione dei guasti

Come si è detto all'inizio, JADE offre una piattaforma che può essere distribuita su più host. La piattaforma dipende in pratica da un solo container (il Main container). Data la natura della rete eterogenea e dato che possono verificarsi dei guasti nei dispositivi, potrebbe capitare che il Main container non sia più raggiungibile. In questo caso, con la configurazione di default, anche l'intera piattaforma verrebbe meno.

JADE prevede una serie di servizi aggiuntivi tra cui anche uno di sostituzione del Main container. Bisogna fare attenzione al fatto che, con la sostituzione, verranno ripristinati solo il Main container e degli agenti DF e AMS, per cui eventuali altri agenti presenti nel container principale andranno persi.

In pratica attivando questo servizio è possibile creare diversi Main container di cui uno è il principale, gli altri vengono considerati secondari. Logicamente questi container sono organizzati ad anello.

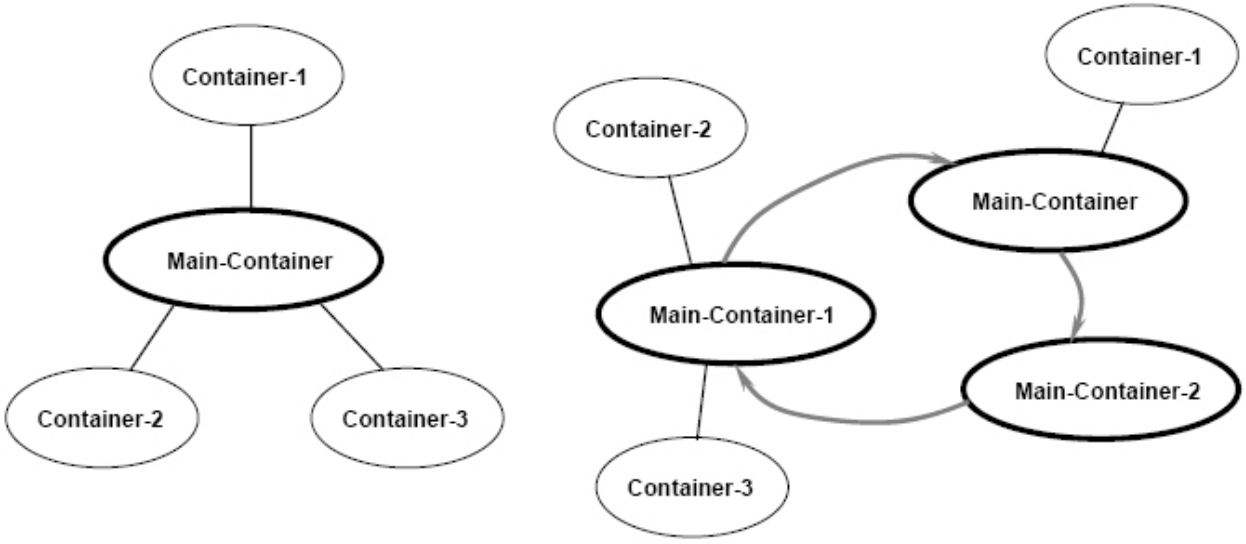


Figura 5.1.3: organizzazione dei container, a sinistra con il comportamento normale, a destra il comportamento in fault tolerance (Main-container-1 e Main-container-2 sono i Main container secondari)

A regime il compito dei secondari è quello di monitorare il container principale. In caso di crash del container principale viene eletto un sostituto che diviene il nuovo Main container, in seguito i secondari sopravvissuti e il nuovo Main container si riorganizzano ad anello.

A tutti gli effetti l'utilizzo di questo servizio rende la piattaforma con la configurazione di default e quella con i Main container secondari del tutto equivalenti dal punto di vista delle funzionalità offerte, anche se, attivando il servizio di sostituzione del container, vengono disattivate alcuni servizi, tra i quali anche quelli che consentono la migrazione (attivi di default nella precedente configurazione). Se si vogliono utilizzare ancora i servizi disattivati, questi vanno esplicitamente dichiarati durante l'inizializzazione di ogni container.

5.2. MySQL

MySQL è un sistema di gestione per database (DBMS) compatibile con il linguaggio SQL (Structure Query Language). È disponibile sia in versione libera sia commerciale: viene rilasciato con un doppio sistema di licenze, in genere GNU GPL, ma, in caso si presenti la necessità, è possibile acquistare una licenza commerciale.

È scritto in C e C++ ed è compatibile con diversi sistemi operativi tra cui Linux, Mac OS X, Solaris, SunOS, Microsoft Windows, ecc.

Comprende tutte le caratteristiche considerate più importanti per la gestione e creazione di basi di dati, tra cui transazioni, vincoli a livello di record, chiavi esterne, interrogazioni annidate e ricerca testuale, ecc.

MySQL rende disponibili una serie di tool grafici per la gestione e la manutenzione del database server. Fra questi si possono ricordare: MySQL Administrator, scritto in Java, che consente di gestire tutta una serie di funzioni come avvio e interruzione del servizio server, visite, password, backup, ecc; e MySQL Query Browser, anch'esso scritto in Java, che si utilizza per la creazione, visualizzazione, cancellazione di database. Tutte operazioni che andrebbero svolte da riga di comando facilitando quindi il lavoro agli sviluppatori.

In particolare MySQL è un sistema di gestione di database relazionali (RDBMS), in altri termini è un sistema che gestisce un'insieme di relazioni. La relazione in questo contesto è rappresentata da una tabella di dati composta da colonne (dette anche attributi) e righe (dette anche record o tuple). Gli attributi rappresentano il tipo di dato che andrà memorizzato per quella colonna, la tupla è un insieme di valori che rappresentano il dato memorizzato per un'entità della tabella. Generalmente è possibile stabilire un determinato insieme di attributi che sono unici per ogni riga: questi attributi vengono definiti chiave primaria e servono ad identificare univocamente il record all'interno della tabella.

L'accesso, la creazione e la modifica delle tabelle avviene attraverso query, ovvero interrogazioni al database in un linguaggio che rispetta lo standard SQL ANSI.

Per quanto riguarda la creazione di una tabella andranno definiti il nome (della tabella), gli attributi (e loro tipologie di dato) ed eventualmente lo strumento di memorizzazione che si vuole utilizzare. Il linguaggio di MySQL offre diverse possibilità per svolgere questa operazione, un esempio è il seguente:

```
create table nome_tabella
(
  definizione primo attributo
  definizione secondo attributo
  …

) [type = strumento di memorizzazione];
```

La modifica di una tabella creata può avvenire tramite il comando ALTER seguito dal nome della tabella e dalla modifica richiesta, mentre per la cancellazione si usa il comando DROP seguito dal nome della tabella.

Per quanto riguarda l'inserimento viene utilizzato il comando INSERT al quale deve essere seguito il nome della tabella su cui inserire i dati. Per la cancellazione invece il comando è DELETE e consente, attraverso una serie di opzioni, di poter eliminare i singoli dati. Altri comandi che possono essere utili in questo contesto sono UPDATE, per aggiornare il contenuto di un record; REPLACE, simile a INSERT e ALTER TABLE per eliminare il contenuto di un'intera tabella.

L'accesso ai dati della tabella avviene tramite una sequenza standard di istruzioni: per prima cosa si identifica l'insieme di attributi di interesse tramite la formula SELECT, in seguito viene specificato quali tabelle vengono coinvolte dall'interrogazione tramite il comando FROM e infine si possono stabilire delle condizioni sulla base delle quali viene svolta la ricerca con il comando WHERE. In sintesi:

```
SELECT elenco attributi
FROM elenco tabelle
WHERE condizioni
```

A questi comandi è possibile aggiungerne altri facoltativi per raggruppare i risultati (GROUP BY) stabilendo delle condizioni sui gruppi (HAVING), oppure per ordinarli (ORDER BY) e per limitarne il numero (LIMIT).

Per utilizzare MySQL con altri linguaggi solitamente si utilizzano dei driver. Attualmente esistono driver per Java, C, C++, Perl, Python, .Net, ecc. Per quanto riguarda l'utilizzo in Java, solitamente si utilizza un protocollo standard per comunicare con il database. In genere questo protocollo è composto da:

- Registrazione dei driver
- Apertura della connessione effettuando il login con username e password
- Apertura di uno statement
- Esecuzione della query e ricezione eventuali risultati
- Chiusura dei risultati (se presenti)
- Chiusura dello statement
- Chiusura della connessione

Il codice che svolge questa procedura è il seguente:

```
try
{
  Class.forName("com.mysql.jdbc.Driver").newInstance();
}
catch (Exception e){/*gestione dell'errore*/}
```

```
Connection conn = null;

/*  apre connessione */
try
{
  conn = DriverManager.getConnection(url, user, pwd);
  Statement stmt = null;
  ResultSet rset = null;
  /*  apre uno Statement */
  if(!conn.isClosed()) stmt = conn.createStatement();
  /*  esegue query */
  //…
  /*  chiude i risultati query */
  rset.close();
  /*  chiude Statement */
  stmt.close();
  if(!conn.isClosed()) conn.close();
}
catch (SQLException e){/*gestire errore*/}
```

Nella realizzazione del sistema è stato scelto MySQL per le sue elevate prestazioni, documentate nella sezione benchmark del sito ufficiale, che sono pari o più elevate rispetto ai concorrenti commerciali. Inoltre è il DBMS gratuito più diffuso al mondo. In Internet il suo utilizzo avviene principalmente assieme al linguaggio PHP per la realizzazione di forum, pertanto si presta ad applicazioni in cui sono richieste un gran numero di connessioni.

5.3. Java 3D

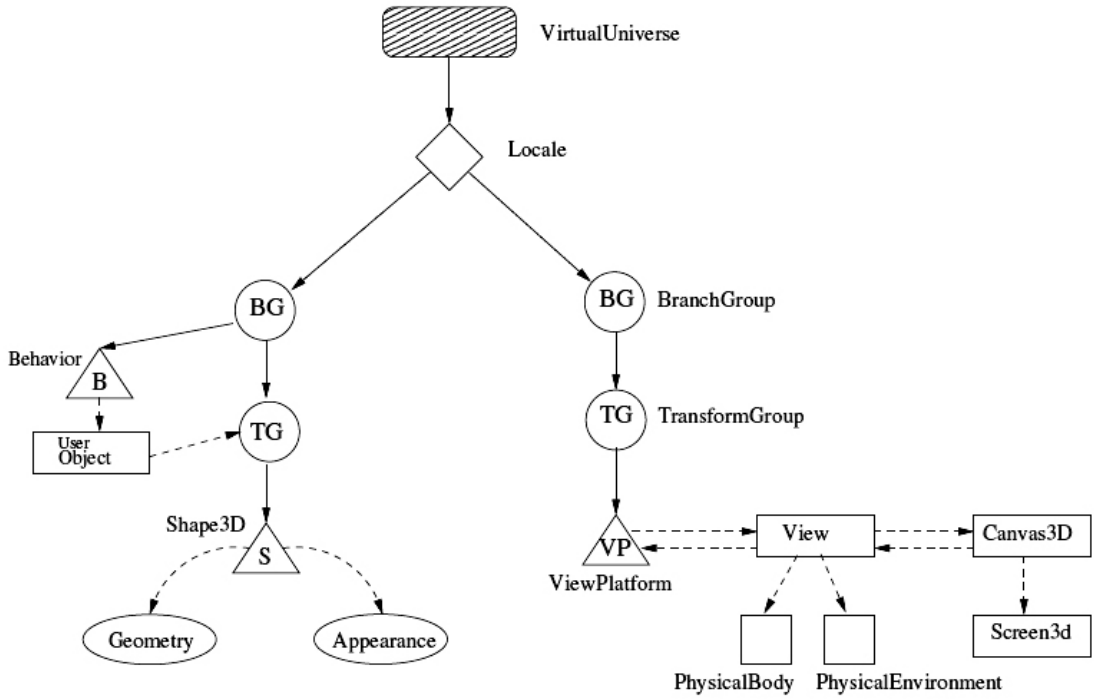
Java 3D è una Application Programming Interface (API) sviluppata dalla Sun Microsystems per la programmazione di applicazioni e applet di ambienti tridimensionali utilizzando il linguaggio Java. Lo scopo principale di questa api è quello di fornire agli sviluppatori strumenti di alto livello per la creazione e la manipolazione di geometrie tridimensionali e la costruzione di strutture per il rendering.

Dal punto di vista tecnico Java 3D si appoggia sulle librerie 3D del sistema ospite, come ad esempio DirectX o OpenGL.

Quando si utilizza questa api, vengono create istanze di oggetti Java 3D che vengono posizionati all'interno di una struttura dati detta "scene graph", che è un albero aciclico che specifica completamente il contenuto di un universo virtuale.

I nodi di questo "albero di scena" sono formati da classi di Java 3D. In particolare i nodi interni, detti "group node" applicano delle trasformazioni al sistema di coordinate dell'universo virtuale, mentre i nodi foglia (leaf node) descrivono gli oggetti visuali che popolano la scena tridimensionale. Inoltre i nodi foglia possono avere dei riferimenti ad altri oggetti (detti node component) che definiscono la geometria e l'aspetto di un oggetto visuale.

Per ogni scena che viene rappresentata si possono distinguere due sottoalberi: il view branch graph e il content branch graph. Il primo specifica i parametri di vista come la locazione e la direzione di quest'ultima; il secondo, si occupa del contenuto dell'universo virtuale, come le geometrie, l'aspetto, il posizionamento e il comportamento degli oggetti visuali, anche se questo è un suono o una luce. Insieme questi due sottografi specificano quale deve essere il lavoro del motore di rendering.



5.3: Lo scene graph di Java 3D

Il motore di rendering di Java 3D si occupa di posizionare i vari oggetti visuali all'interno dell'universo virtuale visitando l'albero dello scene graph, applicando le trasformazioni al sistema di coordinate descritte nei nodi interni e disegnando gli oggetti rappresentati dai nodi foglia.

Nell'implementazione effettuata Java 3D, per la semplicità offerta nella realizzazione di una scena grafica, è stato utilizzato per simulare un accelerometro a doppio asse.

6. Realizzazione

6.1. Simulazione di sensori

Non disponendo di reali sensori per la realizzazione del progetto in tesi, si è pensato di simularli utilizzando delle interfacce realizzate in Java che consentissero l'interazione per la modifica dei parametri presi in esame.

Come precedentemente detto nel capitolo relativo al software utilizzato, per realizzare l'accelerometro si è utilizzato Java3D.

Dato che un accelerometro misura l'accelerazione a cui è soggetto un corpo, questo è stato rappresentato da una sfera in uno spazio 3D. Nella realizzazione della sfera si è usata una primitiva di Java 3D (com.sun.j3d.utils.geometry.Sphere) al quale è stata data una colorazione e delle luci per risaltare l'effetto tridimensionale. Inoltre per rendere più chiara la posizione dell'oggetto all'interno dello spazio tridimensionale sono stati aggiunti gli assi cartesiani x e y attraverso la creazione di due linee 3D. La posizione della sfera nello spazio 3D viene catturata mediante il metodo get2dCoordinate() che restituisce un vettore delle coordinate. Spetterà poi all'agente che controlla

questo sensore ricavare l'accelerazione relativa allo spostamento effettuato dalla sfera nello spazio.

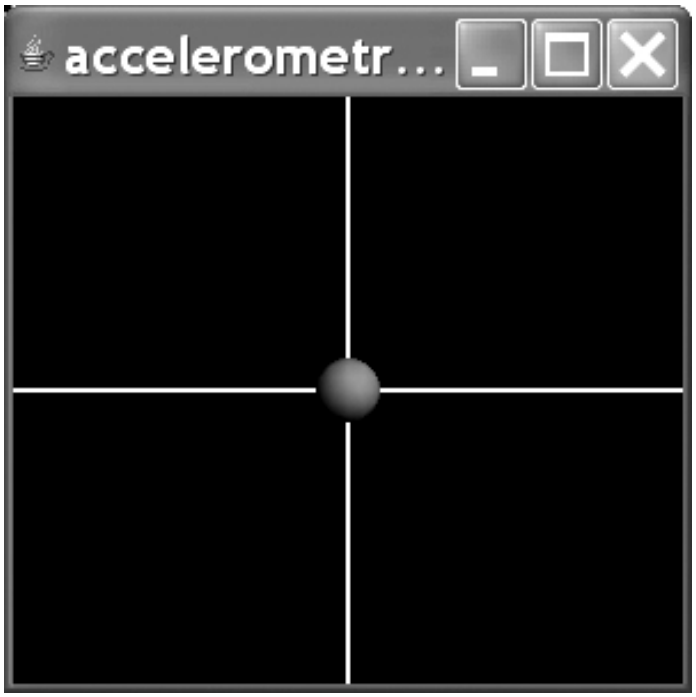


Figura 6.1.1: Interfaccia dell'accelerometro

L'interazione con la sfera, per simulare eventuali movimenti della persona sotto monitoraggio, è svolta aggiungendo al gruppo di cui fa parte l'oggetto 3D un mouse behaviour di tipo traslatorio (com.sun.j3d.utils.behaviors.mouse.MouseBehavior). In questo modo si può simulare un accelerometro a doppio asse semplicemente utilizzando il mouse e spostando la sfera. Nell'eventualità che si voglia simulare un accelerometro a triplo asse è sufficiente modificare il mouse behaviour affinché svolga anche l'azione di "zoom", ovvero lo spostamento sull'asse z.

Gli altri sensori non necessitano di misurare spostamenti di una massa in uno spazio tridimensionale, pertanto si è pensato che, nella simulazione, fosse sufficiente modificare un parametro predefinito con una semplice combinazione di tasti, quindi è stato utilizzato un sottoinsieme delle classi appartenenti al package javax.swing, in particolare javax.swing.JSpinner per la realizzazione di una casella il cui valore può essere aumentato o diminuito con la semplice pressione di un tasto.

Figura 6.1.2: Interfacce del termometro, del cardiofrequenzimetro e dello sfigmomanometro

L'interfaccia nel caso dei sensori quali termometro, sfigmomanometro, cardiofrequenzimetro è composta principalmente da un pannello diviso in due parti: nella prima vi è il nome della misura, nella seconda un valore modificabile attraverso le apposite frecce. La modifica sui valori cambia a seconda del sensore: nel caso della temperatura, il passo di cambiamento è di 0,1; nel caso della pressione è di 1, cosi come nel caso della frequenza cardiaca. Nell'interfaccia dello sfigmomanometro, infine, il pannello è dotato di due righe identiche, una per la pressione minima, l'altra per la massima. Gli agenti che "dialogano" con questo sensori si limitano semplicemente a invocare un metodo pubblico che restituisce il valore corrente sull'interfaccia.

6.2. Driver dei sensori

Gli aspetti di gestione della comunicazione con i sensori, in altri termini la funzione di driver, sono affidate agli agenti sensori. Quelli realizzati sono in tutto quattro e si occupano oltre a leggere i dati dei dispositivi a loro corrispondenti anche a inviarli all'agente paziente a cui fanno riferimento.

JADE offre la possibilità di fornire un vettore di oggetti come argomento ad un agente quando viene creato. Pertanto, per semplificare le comunicazioni in modo da non coinvolgere il servizio di pagine gialle del sistema per dover ricercare la destinazione a cui inviare i dati, si è pensato che, dato che questi agenti (come si è detto nel capitolo relativo alla descrizione funzionale del siste-

ma) vengono lanciati dall'agente paziente, abbiano come argomento l'indirizzo del loro "genitore", ovvero dell'agente paziente che deve ricevere i dati.

I quattro agenti vengono avviati in un container separato rispetto a quello che contiene l'agente paziente poiché si è pensato che, nel caso in cui fosse possibile utilizzare dei sensori con una minima capacità di calcolo, questi potessero contenere gli agenti sensori. Inoltre la separazione dei container è pensata anche per l'utilizzo di Jade-Leap che è in grado di estendere un unico container a più dispositivi qualora questi non abbiano elevate capacità di calcolo.

Tutti gli agenti di tipo sensore una volta lanciati memorizzano la destinazione dei loro messaggi leggendo l'argomento di lancio e in seguito avviano l'interfaccia (il sensore) a loro corrispondente. Inoltre tutti possiedono un loro behaviour di tipo TickerBehaviour che a tempi fissati (0,5 secondi) richiede il valore del sensore corrispondente richiamando il metodo pubblico della classe che realizza l'interfaccia. L'utilizzo di un TickerBehaviour distinto per ogni sensore consente di poter utilizzare tempi di campionamento diversi e indipendenti gli uni dagli altri: ad esempio nel caso dello sfigmomanometro 0,5 secondi non sono sufficienti nella realtà per svolgere una misurazione.

Come si è detto precedentemente gli agenti che governano il termometro, lo sfigmomanometro e il cardiofrequenzimetro, semplicemente leggono il valore che compare sull'interfaccia. Nel caso invece dell'accelerometro, il valore che viene passato al sensore è la posizione del centro della piccola sfera nello spazio tridimensionale. Per calcolare l'accelerazione dell'oggetto viene per prima cosa calcolata la velocità media fra due intervalli di campionamento secondo la seguente formula:

$$\bar{V}_{corrente} = \frac{\Delta X}{\Delta T} = \frac{X_{corrente} - X_{precedente}}{tempo\ di\ campionamento}$$

In seguito viene calcolata l'accelerazione media del corpo nell'intervallo di campionamento secondo la seguente formula:

$$\bar{a}_{corrente} = \frac{\Delta V}{\Delta T} = \frac{V_{corrente} - V_{precedente}}{tempo\ di\ campionamento}$$

Dato che il tempo di campionamento è molto piccolo (0,5 secondi), si può approssimare questo valore all'accelerazione istantanea.

6.3. Applicazione del paziente

L'implementazione dell'applicazione del lato paziente è sostanzialmente divisa in due parti: la prima è l'interfaccia che deve essere semplice e immediata da leggere, l'altra il vero cuore dell'applicazione composto dall'agente paziente che svolge tutte le attività di interesse del suo utente.

L'interfaccia è stata realizzata in modo simile a quella dei sensori, in altre parole è composta da un pannello diviso in una griglia 2x6. I dati dei sensori visualizzati sono:

- L'accelerazione lungo l'asse x
- L'accelerazione lungo l'asse y
- La temperatura
- La frequenza cardiaca
- La pressione minima
- La pressione massima

Sul lato sinistro tramite gli appositi JLabel vi sono i nomi dei valori, sul destro i dati dei sensori posti in JTextField. A differenza delle interfacce dei sensori i dati non sono modificabili. Per ogni sensore l'interfaccia possiede un metodo pubblico per l'aggiornamento dei valori. Se un sensore possiede dati multipli, come nel caso dell'accelerometro (accelerazione lungo l'asse x e quella lungo l'asse y), questi sono gestiti assieme.

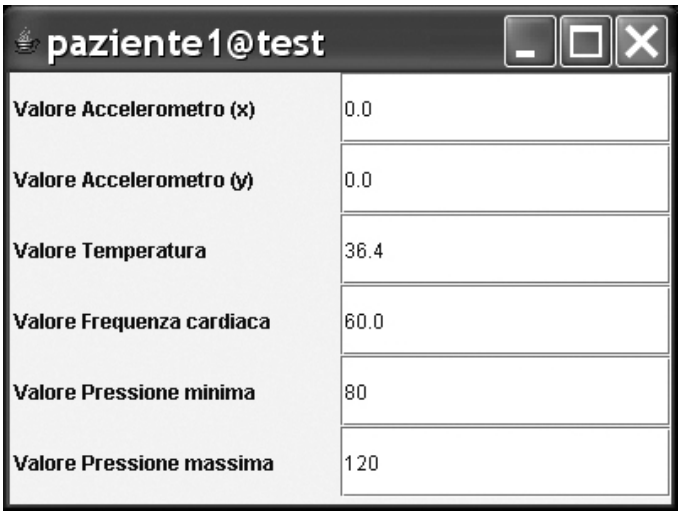


Figura 6.3: Interfaccia paziente

L’agente paziente ha il compito di aggiornare l’interfaccia con i dati dei sensori verificando che non vi siano situazioni di emergenza, inoltre deve depositare i valori raccolti sul database.

L’utente non avvia direttamente l’agente paziente, invece questa attività è svolto da un agente di lancio definito come starter agente paziente. Questo agente ha il compito di avviare l’agente paziente nel proprio container, assegnando ad entrambi un nome. In questo caso è stato scelto per entrambi il codice tessera sanitaria (si ricorda che gli agenti e i container devono avere una nominazione univoca all’interno della piattaforma e il codice della tessera sanitaria è univoco). L’agente starter gestisce una piccola interfaccia su cui va semplicemente inserito il codice della tessera sanitaria. Premendo il tasto OK viene creato il container e in esso è avviato l’agente paziente. Fatto ciò lo starter termina chiudendo la propria interfaccia.

All’avvio l’agente paziente registra se stesso presso il noto servizio di pagine gialle di JADE, in modo tale da consentire la ricerca dello stesso agli agenti di emergenza per l’invio di messaggi come verrà spiegato nel capitolo relativo al funzionamento in caso di emergenza.

In seguito l’agente avvia la propria interfaccia con i seguenti valori di default:

- Accelerometro: accelerazione x = 0, accelerazione y = 0;
- Sfigmomanometro: pressione massima = 120, pressione minima = 80;
- Cardiofrequenzimetro: frequenza cardiaca = 60;
- Termometro: temperatura = 36,4.

Dopo aver avviato l’interfaccia, l’agente paziente crea, sia per gli agenti che fungono da driver per i sensori, sia per quello che funge da driver per il database, due container nominati rispettivamente con il proprio nome (codice della tessera sanitaria) preceduto dalla dicitura “container sensori” nel caso dei container sensori e “container DB driver” nel caso di quello relativo al driver database. All’interno di quello dei driver sensori vengono subito avviati gli agenti sensori dando loro come argomento l’AID dell’agente.

La gestione dei dati dei sensori che arrivano all’agente paziente come messaggi, avviene all’interno di un comportamento ciclico (CyclicBehaviour) che funziona grosso modo come una segreteria: a seconda del tipo di messaggio arrivato, questo viene opportunamente processato.

All’arrivo di un messaggio da parte di uno dei sensori, per prima cosa, viene estratto il contenuto in modo da ottenere dei valori utilizzabili. Questi valori in seguito vengono controllati secondo del-

le soglie predeterminate. Le soglie per semplificare l'implementazione, a seconda del sensore, sono state così fissate:

Valore	***Minimo***	***Massimo***
Accelerazione (totale)	-	10
Pressione Massima	100	140
Pressione Minima	60	100
Temperatura	36	37
Frequenza cardiaca	30	150

Nel caso in cui un valore sia fuori soglia l'agente entra nello stato di emergenza avviando nel suo stesso container, l'agente che si occupa dell'emergenza dandogli tutti i dati necessari. In questo stato e fintantoché non viene ricevuto un messaggio di annullamento o un messaggio che segnali la fine dell'emergenza, l'agente non rileva altre emergenze.

L'utilizzo di un behaviour di tipo ciclico sul lato paziente che rimane sempre in attesa di nuovi messaggi e di un behaviour di tipo periodico sul lato sensore offre un importante vantaggio: consente infatti di poter gestire per ogni sensore tempi di campionamento differenti e di poterli modificare a piacimento senza dover mettere mano sul codice dell'applicazione del paziente.

Come si è detto, l'agente paziente, dato che il dispositivo su cui opera non dispone di una gran quantità di memoria, deve inviare i dati raccolti al database periodicamente per il loro deposito. Pertanto si è pensato di introdurre un ulteriore behaviour di tipo periodico (TickerBehaviour) che svolga proprio questa azione. In questo behaviour, dato che viene compiuto a intervalli di tempo che devono essere maggiori rispetto al tempo di campionamento dei dispositivi, viene lanciato periodicamente l'agente che implementa tutte le procedure necessarie per la comunicazione con il database: l'agente database paziente, al quale vengono affidati i dati da inviare.

Per limitare la quantità di dati da memorizzare sul database, si è scelto di ridurre i dati in invio, per ogni sensore, al valore medio, massimo e minimo nell'intervallo di tempo fra un collegamento alla base di dati e il successivo. Per sicurezza infine i dati inviati vengono memorizzati anche sul dispositivo del paziente fino ad un massimo di cinque, dopodiché viene sovrascritto il primo e così via.

6.4. Applicazione del medico

Come avviene nell'applicazione del paziente, anche quella del medico si può dividere in due parti: l'interfaccia e l'agente medico.

L'interfaccia dell'applicazione è stata disegnata per essere più completa rispetto a quella del paziente: presenta in alto un pannello contenente i dati anagrafici del paziente quali nome, cognome, indirizzo, data di nascita, numero di telefono e numero della tessera sanitaria. Subito sotto questo pannello è presente uno spazio in cui il medico può inserire una stringa di testo per specificare il tipo di emergenza che ha rilevato. Questa stringa di testo sarà utilizzata per lanciare l'emergenza da parte del medico.

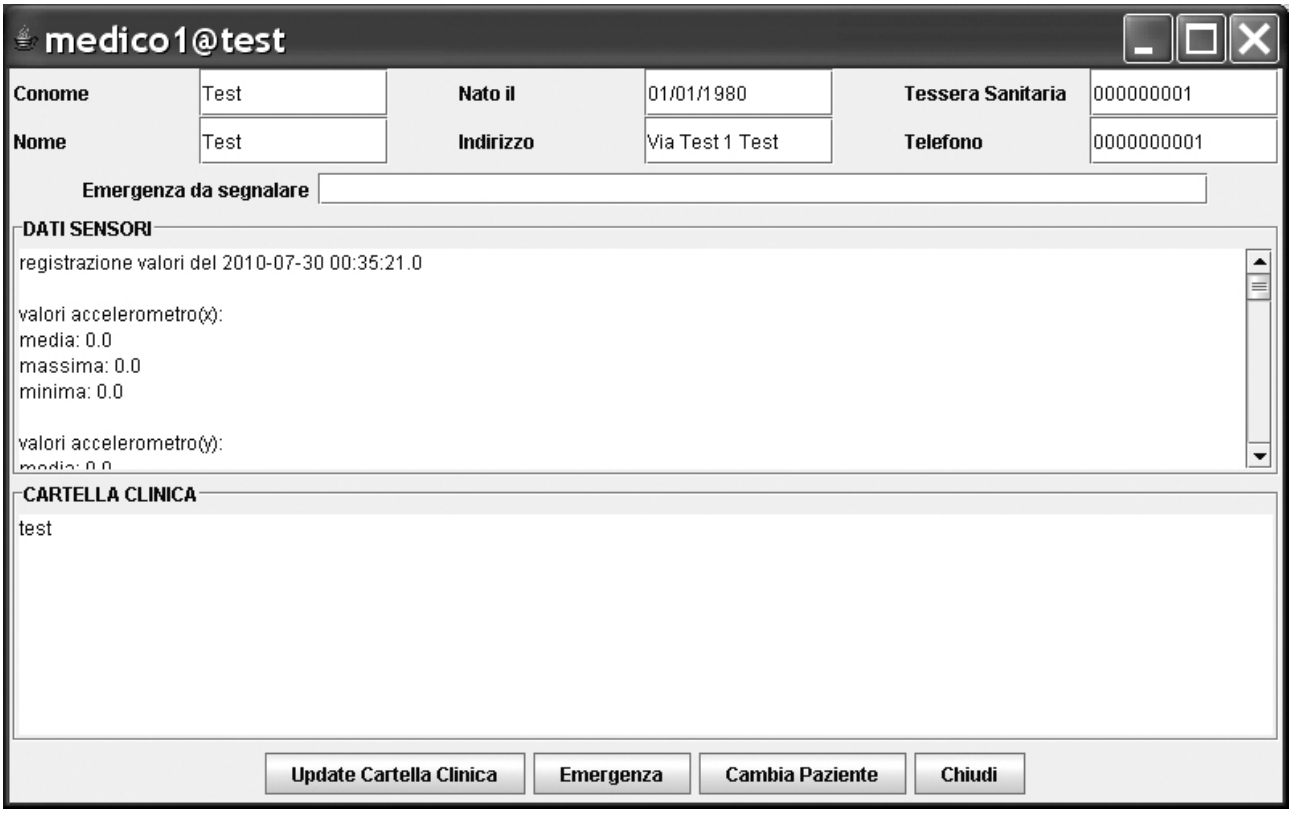


Figura 6.4: Interfaccia medico

Per quanto riguarda la cartella clinica e i dati dei sensori, dato che queste parti sono composte da innumerevoli parametri (ad esempio per la cartella clinica possono esserci anamnesi, terapie seguite, allergie, ecc), si è pensato, per semplificare la realizzazione dell'interfaccia, di organizzare il

tutto in due JTextArea (javax.swing.JTextArea) contenenti la prima i dati dei sensori in ordine cronologico, la seconda la cartella clinica gestita come un testo.

L'utente non può modificare ne i dati dei sensori ne i dati anagrafici dei pazienti, l'unica parte modificabile è l'area di testo della cartella clinica per l'eventuale aggiornamento sul server.

Infine subito sotto questo pannello è presente quello con dei pulsanti per l'interazione con il medico. Per primo vi è quello per eseguire l'aggiornamento della cartella clinica nel caso in cui vi siano state fatte delle modifiche, poi quello per lanciare un emergenza (e in questo caso va compilato l'apposito campo), infine gli ultimi due servono a cambiare paziente che si vuole consultare e a chiudere e disconnettere dal sistema il medico.

Come detto precedentemente, l'applicazione del medico è gestito dall'agente medico. Questo, come accade nel caso di quello del paziente viene lanciato da uno starter che si occupa di dare un nome univoco all'agente e al container che lo contiene (nella realizzazione si è pensato di utilizzare per entrambi il codice personale del medico). Una volta lanciato, per prima cosa, questo agente si registra presso il DF per consentire l'eventuale ricerca dell'agente medico da parte di altri agenti e in seguito lancia anch'egli un agente driver in un container separato per comunicare al database che il medico è attivo nel sistema.

A differenza dell'agente paziente che doveva semplicemente aggiornare un'interfaccia con i dati che arrivavano dai sensori, questo agente deve gestire degli input provenienti dall'interfaccia, pertanto è stato realizzato come GUI Agent implementando inoltre il metodo onGuiEvent(GuiEvent ge). In questo metodo viene gestita la parte relativa ai pulsanti dell'interfaccia: quando l'utente preme uno dei pulsanti genera un evento che viene pubblicato sul GUI Agent che risponde eseguendo l'azione stabilita. Nel caso di questo agente di tratta di: aggiornare la cartella clinica lanciando il driver database, lanciare un'emergenza con l'avvio dell'apposito agente, cambiare il paziente esaminato oppure chiudere l'interfaccia scollegando il medico dal sistema (anche in questi ultimi due casi viene lanciato l'agente addetto alle comunicazioni con il database).

Il protocollo di comunicazione con il driver database prevede che quest'ultimo venga avviato con determinate informazioni a seconda del suo utilizzo. Quando viene avviato per svolgere una lettura sul database questo semplicemente invia i dati raccolti all'agente medico. Pertanto, similmente a quanto accade con la gestione dei messaggi nell'agente paziente, si è deciso di dotare anche l'agente paziente di un behaviour di tipo ciclico che svolga il compito di gestione dei messaggi in

ingresso. Le operazioni in lettura sul database vengono svolte per i dati anagrafici dei pazienti, per la cartella clinica e per i dati dei sensori. Si è deciso di utilizzare per questi, tre messaggi separati in modo da poter gestire con più semplicità eventuali aggiornamenti della cartella clinica o dei dati dei sensori.

6.5. Applicazione dell'Operatore Sanitario sul Territorio

L'applicazione realizzata per gli OST è molto semplice: dato che la loro attività è quella di entrare in azione solo nel caso in cui si verifichi un'emergenza, si è pensato di trasferire la parte strettamente relativa alla conoscenza e alla gestione dei dati dei pazienti nell'applicazione che migra sull'host dell'OST nel caso venga verificato un evento critico.

La funzione principale dell'applicazione dell'OST è quindi solo quella di mantenere attivo l'operatore nel sistema per intervenire in caso di emergenza. L'interfaccia sviluppata è composta da un unico pannello contenete un pulsante che consente la disconnessione dal sistema.

L'agente che si occupa di svolgere le funzioni dell'OST, come avviene negli altri casi è avviato da un agente starter che si occupa di collocarlo all'interno di un container. Sia al container, sia all'agente viene dato un nome unico e in questa realizzazione esso è il codice personale dell'OST.

L'azione dell'agente è sequenziale, non vi è gestione di messaggi, non utilizza altri agenti come driver e rimane attivo fintantoché l'OST è attivo nel sistema. Dato che è un agente leggero si è pensato di dotarlo dei protocolli di comunicazione con il database, pertanto all'avvio esegue una connessione con la base di dati aggiornando lo status dell'OST. Questa procedura viene eseguita anche quando l'OST si disconnette dal sistema premendo l'apposito pulsante. Per comunicare con l'interfaccia e gestire il pulsante, l'agente è stato realizzato come un GUI Agent.

6.6. Database

Il database è una parte fondamentale del sistema: innanzitutto funge da deposito permanente dei dati dei sensori, detiene tutte le informazioni relative ai pazienti (storia clinica, dati anagrafici, ecc) e quelle relative ai medici e agli OST. Infine svolgerà un ruolo importante anche nell'identificazione di determinati agenti nel sistema

Nell'implementazione effettuata, ogni paziente è identificato univocamente all'interno del sistema dal codice della propria tessera sanitaria. In modo analogo anche medici e OST, avranno un identificativo univoco basato su un loro codice personale (che potrebbe essere assegnato loro una tantum al momento dell'iscrizione al sistema).

Nel database, le tabelle che riguardano il paziente sono tre: quella relativa ai dati anagrafici del paziente che andrà compilata quando il paziente verrà iscritto al servizio di assistenza; quella contenente le cartelle cliniche dei pazienti, generalmente gestita dai medici; e quella che servirà a contenere i dati dei sensori.

Nella tabella contenente i dati anagrafici del paziente, i cui attributi sono ad esempio nome, cognome, indirizzo, numero di telefono, ecc; si è scelto di utilizzare come chiave primaria il codice della tessera sanitaria. Questo codice è anche chiave primaria della tabella relativa alle cartelle cliniche dei pazienti, che in questa realizzazione è semplicemente un documento di testo.

Per quanto riguarda la parte relativa ai dati dei sensori, si è scelto che venga predisposta una tabella per ogni paziente contenente tutti i dati raccolti. Per evitare che i dati siano troppo numerosi si è pensato che, per ogni valore di un sensore, il database salvasse solo il valore medio, il massimo e il minimo nell'intervallo di tempo fra un invio di dati e il successivo, in altri termini se, ad esempio, il paziente invia i propri dati ogni 5 minuti, verranno memorizzati i valori medi, massimi e minimi registrati in quei 5 minuti. In tutto quindi i valori salvati sono 18 (tre per valore di cui accelerazione e pressione sono doppi) più uno come chiave primaria rappresentante un time stamp del momento in cui vengono inseriti i valori nella base di dati.

Per quanto riguarda il personale medico e gli OST, si è scelto, per semplificare, che il database venga utilizzato per tenere traccia dell'accesso al sistema da parte di una di queste due figure. Pertanto, in questo caso, le tabelle sono solo due ed entrambe conterranno come chiave primaria il

codice del medico o dell'OST e uno status di ONLINE o OFFLINE nel caso in cui la persona a cui fa riferimento sia rispettivamente presente nel sistema o meno. Questo status verrà utilizzato in seguito per ricercare il medico attivo o l'OST attivo nel caso di emergenza.

Infine, dato che gli utenti svolgono visite diverse sul database, ad esempio i pazienti solo in scrittura, i medici in lettura e scrittura, si è scelto di sfruttare la possibilità offerta da MySQL di stabilire degli accessi personalizzati, stabilendo, a seconda della figura coinvolta, quali aree possano vistare e quali no e che azione possano compiere.

È certamente possibile organizzare il database in modo più elaborato, ad esempio aggiungendo dei vincoli di chiave, memorizzando dati anagrafici dei medici, dell'OST, ecc. In questo caso, tuttavia, si è voluto focalizzare l'attenzione a livello di sistema in quanto il database è utilizzato solo a supporto all'applicazione sviluppata.

6.7. Driver database

Come avviene per i sensori, anche per la comunicazione con il database, sia per le applicazioni del medico sia per quelle del paziente (in emergenza e in funzionamento normale) sono stati realizzati due agenti.

Il primo è quello che si occupa della comunicazione fra il dispositivo di memorizzazione del paziente con il database.

Come nel caso degli agenti sensori, anch'esso è lanciato dall'agente paziente, ma a differenza dei precedenti il suo comportamento non è ciclico, ma sequenziale. L'azione compiuta infatti è semplicemente quella di aprire una connessione con il database, formulare una query con i dati da inviare e chiudere la connessione. Il tutto è realizzato seguendo il codice descritto nel capitolo relativo a MySQL.

Dato che non è necessario depositare i dati con una frequenza paragonabile a quella della lettura dei sensori, si è pensato che al termine dell'azione, l'agente semplicemente venga terminato liberando in questo modo risorse sul dispositivo del paziente su cui viene lanciato.

Anche in questo caso, come avveniva negli agenti sensori, per diminuire le comunicazioni fra gli agenti e per non coinvolgere in questo processo agenti con compiti centralizzati come il DF, l'agente database paziente viene avviato con un argument contenente i valori da trasmettere e i dati per effettuare l'accesso alla base di dati.

Il secondo agente di comunicazione con il database è quello utilizzato sia dall'agente medico sia dagli agenti di emergenza. A differenza del precedente offre non solo servizio di scrittura nel database, che viene svolto per la scrittura della cartella clinica, ma anche di lettura dello stesso.

Per molti aspetti l'implementazione di questo agente è molto simile al precedente: entrambi vengono avviati con particolari argomenti, utilizzano lo schema di comunicazione con il database citato nel capitolo di MySQL, hanno un comportamento di tipo sequenziale e, una volta finito il loro compito, terminano per liberare risorse. La particolarità di questo agente sta nel fatto che può essere avviato con 3 diversi modi:

1. Modalità GET: l'agente esegue una connessione in lettura al database per ricavare i dati anagrafici del paziente, i dati dei sensori e la cartella clinica e li trasmette all'agente che lo ha attivato.
2. Modalità UPDATE: l'agente esegue una connessione in scrittura al database per aggiornare la cartella clinica del paziente.
3. Modalità STATUS: l'agente esegue una connessione in scrittura al database per aggiornare lo status del medico dal cui agente è stato lanciato.

Infine a differenza dell'agente database paziente che era utilizzato dal solo agente paziente, quest'ultimo agente è stato pensato per essere utilizzato da più agenti, fra cui: l'agente medico (modalità GET, UPDATE e STATUS), l'agente emergenza paziente (GET per i dati del paziente e UPDATE la cartella clinica), l'agente emergenza medica (UPDATE e GET per il precedente motivo).

L'utilizzo di un agente per l'invio dei dati al database, o per la lettura di questi, consente da un lato di scaricare l'agente che richiede tale funzionalità rendendolo più leggero (si ricorda che, ad esempio, il dispositivo del paziente è pensato per essere mobile e con ridotte capacità di calcolo), dall'altro potrebbe aumentare la sicurezza del sistema nel caso in cui, anziché essere creato sul dispositivo su cui si trova l'applicazione, venga creato o fatto migrare direttamente sul server della base di dati: in questo modo i messaggi verrebbero scambiati esclusivamente fra agenti (si noti

che, in questo caso, la comunicazione fra agenti avviene attraverso stringhe che potrebbero essere criptate con un protocollo a chiave pubblica o privata) e le comunicazioni con il database avverrebbero direttamente in locale. Lo svantaggio con questo approccio è che riduce la scalabilità del sistema nel caso in cui il database sia centralizzato in quanto tutti gli utenti del sistema creerebbero, o farebbero migrare un loro agente, sulla macchina che contiene la base di dati aumentando il numero di task contemporanei in esecuzione lasciando invariate il numero di connessioni poiché avverrebbero comunque, ma in locale. Pertanto, utilizzando un database server realizzato con MySQL, si è scelto di mantenere nell'implementazione eseguita l'approccio con i driver sul dispositivo mantenendo comunque separati i container dell'agente paziente da quello del database.

6.8. Emergenze

La gestione delle emergenze nel sistema sviluppato è affidata a due agenti: L'agente emergenza paziente e l'agente emergenza medico. Entrambi possiedono caratteristiche e interfacce simili.

Questi due agenti sono diversi da quelli precedentemente presentati: devono essere in grado di interagire con il medico o con il personale OST e devono essere anche in grado di spostarsi sui loro dispositivi. Pertanto non sono solo GUI Agent, ma anche Mobile Agent, in altri termini devono implementare sia il metodo onGuiEvent(GuiEvent ge) sia i metodi doMove(Location loc) per far migrare l'agente su una certa posizione e beforeMove(), che serve a svolgere determinate azioni prima di muoversi.

Le interfacce gestite da entrambi gli agenti sono due: la prima è una sorta di finestra di dialogo per chiedere una conferma sull'emergenza, la seconda è un'interfaccia simile a quella del medico sia per l'agente emergenza paziente sia per quello medico. Questo perché dal lato medico l'emergenza ricevuta va segnalata con dati simili a quelli con cui lavora il medico (dati anagrafici del paziente, dati dei sensori e cartella clinica), dal dato OST l'utilizzo dell'applicazione è simile a quella che farebbe il medico, quindi per semplificare l'implementazione si è scelto di utilizzare costrutti già utilizzati.

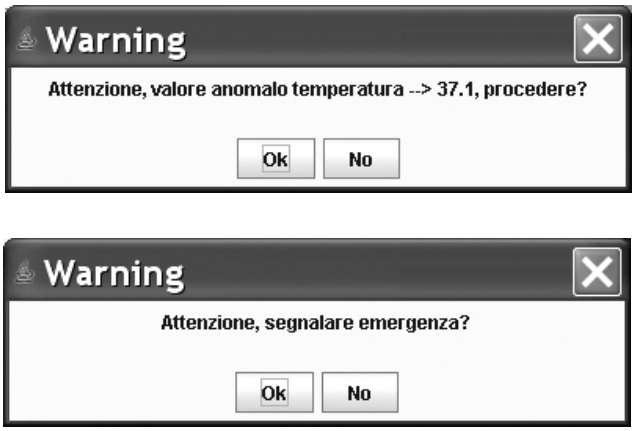


Figura 6.8.1: Interfacce di lancio, in alto quella dell'agente emergenza paziente, in basso quella dell'agente emergenza medica

Le differenze con l'interfaccia medica stanno in un numero inferiore di pulsanti (nel caso dell'emergenza paziente manca quello che permette il cambio del paziente, mente nel caso dell'emergenza paziente mancano il precedente pulsante e ovviamente quello per lanciare l'emergenza dato che l'OST ha il compito di risolverla) e nel fatto che all'avvio sull'host del medico o dell'OST queste hanno già il campo relativo all'emergenza rilevata compilato.

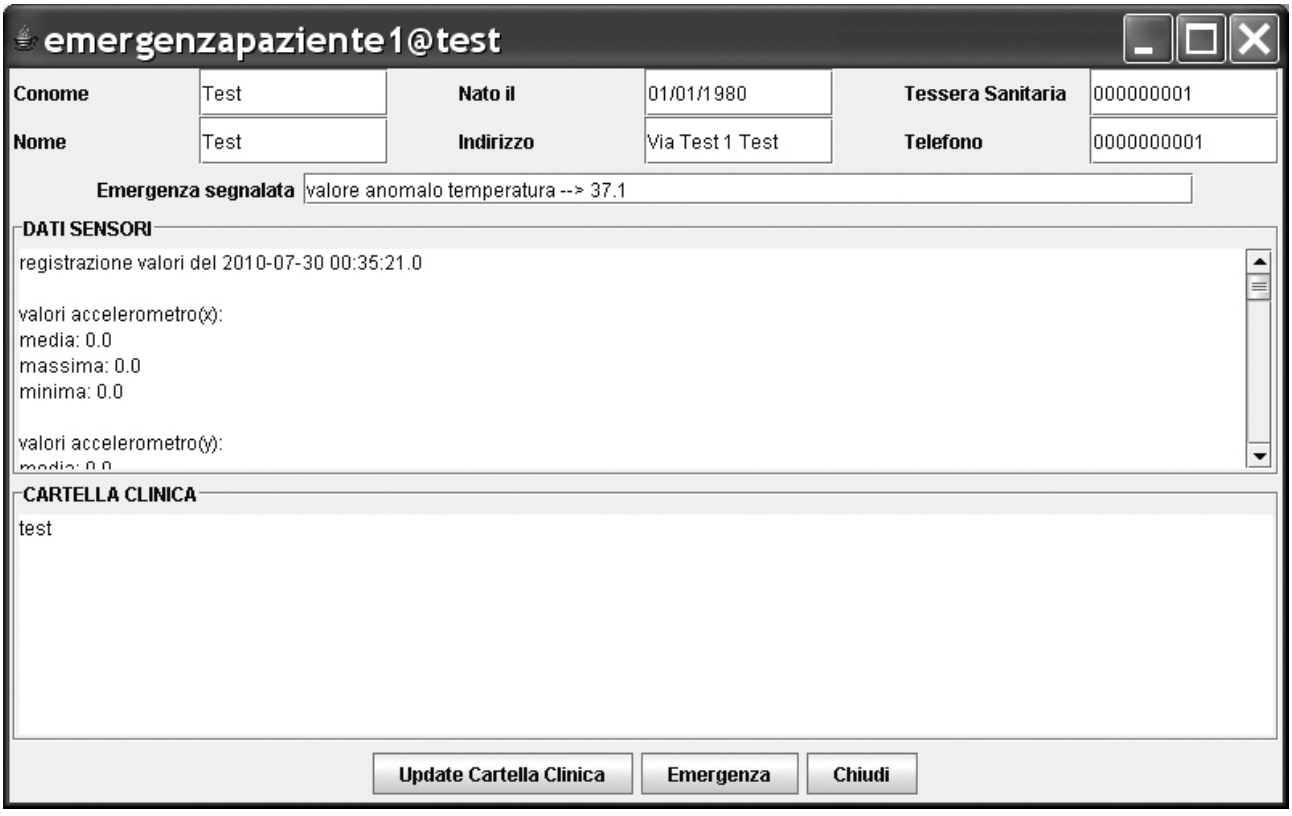


Figura 6.8.2: Interfaccia emergenza paziente visualizzata sull'host del medico

Al lato paziente, come già precedentemente detto, l'emergenza scatta non appena il valore rilevato è superiore o inferiore a una soglia fissata avviando l'agente emergenza paziente. Nel caso del medico invece l'emergenza viene attivata premendo l'apposito pulsante avviando anche in questo caso l'agente addetto a segnalare la situazione critica.

Entrambi gli agenti vengono avviati nello stesso container dell'agente che li ha creati. All'avvio inoltre ricevono in argomento una serie di dati fra cui l'emergenza rilevata e l'AID dell'agente che li ha creati. In seguito entrambi lanciano la finestra di dialogo per confermare o meno l'emergenza, se questa non viene confermata semplicemente muoiono e, nel caso dell'agente emergenza paziente, questo comunica al paziente di tornare in uno stato di funzionamento normale. Nel caso in cui l'emergenza venga confermata, per prima cosa entrambi cercano sul database la loro figura professionale di destinazione (il medico per l'agente emergenza paziente, l'OST per l'emergenza medica), una volta trovata, utilizzando il codice di ricerca su DF e AMS della location presentato nel capitolo relativo a JADE, migrano sul container dell'agente trovato. Una volta arrivati, visualizzano l'interfaccia più complessa, avviando il driver database medico per ottenere i dati da visualizzare.

Inoltre entrambi gli agenti hanno la possibilità di effettuare delle modifiche alla cartella clinica. Sia per l'emergenza paziente, sia per l'emergenza medica, queste dovrebbero rappresentare la compilazione di moduli ed eventuali prescrizioni di medicinali o terapie al paziente per risolvere la situazione che ha provocato l'emergenza.

Nell'implementazione si è scelto di non far eseguire più di una migrazione ad un agente: in questo caso se la situazione critica evidenziata dall'agente di emergenza paziente viene confermata dal medico premendo l'apposito tasto di emergenza, questo non migra sulla posizione dell'OST, ma si limita a lanciare l'agente emergenza medico.

6.9. Scalabilità

L'applicazione sviluppata, come si è detto nell'introduzione è pensata per essere un sistema distribuito e come tale deve affrontare tutte le criticità di questo tipo di sistemi, tra cui la scalabilità.

Con il termine scalabilità si intende quella proprietà che ha un determinato sistema di poter aumentare o diminuire nelle proprie dimensioni continuando ad operare a livelli accettabili. Per fare ciò è importante individuare quelli che vengono definiti "colli di bottiglia" ovvero risorse fortemente connesse per le quali possono dipendere rallentamenti nelle prestazioni nel caso in cui molti utenti le richiedano nello stesso istante (ad esempio in una rete di computer i colli di bottiglia potrebbero essere rappresentati da un server centralizzato e da un gateway).

Nel caso di questo sistema sono stati individuati due colli di bottiglia:

1. Il database: come si è detto è unico, non distribuito e realizzato in MySQL. Da questa risorsa dipendono l'invio di dei dati dei sensori di tutti i pazienti, tutte le attività dei medici e degli OST, nonché la ricerca del personale operativo in caso di emergenza.
2. Il Main container e gli agenti AMS e DF di JADE: nel capitolo relativo alla realizzazione di JADE si è parlato dell'importante funzione svolta da queste figure: in pratica gestiscono l'intera piattaforma consentendo la ricerca di altri agenti.

Ogni server MySQL può essere impostato per gestire un numero variabile di connessioni contemporanee. È anche vero che nel sistema realizzato gli agenti che si occupano di comunicare con il database, una volta eseguita la query, chiudono la connessione e questa generalmente non dura che pochi millisecondi. Pertanto è ipotizzabile, considerando che MySQL viene soprattutto utilizzato nel web per la gestione di forum basati sul linguaggio PHP, che il database così realizzato sia in grado di gestire un gran numero di utenti.

Strategie più complesse potrebbero prevedere l'utilizzazione di un eventuale database distribuito che consentirebbe una riduzione del carico sul singolo database server.

Per quanto riguarda gli agenti con funzioni centralizzate come AMS e DF, si è visto che in letteratura (ad esempio con il sistema WAITER presentato nel capitolo 2.2), il protocollo di comunicazione, nel caso in cui si dovesse spedire un messaggio, solitamente prevedeva i seguenti punti:

1. Contattare il DF per ricercare l'agente a cui inviare il messaggio
2. Ricevere la risposta del DF
3. Spedire il messaggio a destinazione

Questo tipo di protocollo non risulta molto scalabile nel caso in cui avvengano molte comunicazioni fra i dispositivi, anche se il DF gestisce una coda di messaggi in ingresso: si pensi ad esempio alla

quantità di messaggi scambiati ogni 0,5 secondi per la comunicazione fra gli agenti sensori e l'agente paziente. All'aumentare degli utenti nel sistema, il tempo di arrivo dei messaggi aumenterebbe proporzionalmente diminuendo di fatto le prestazioni.

Si è quindi pensato di escludere nel comportamento a regime (non in caso di emergenza), per quanto possibile, gli agenti quali AMS e DF: dato che alcuni agenti devono lanciarne altri con cui devono comunicare, questi vengono avviati conoscendo già l'indirizzo a cui spedire i propri messaggi escludendo di fatto dalla comunicazione il DF.

Il comportamento di ricerca presso il DF e l'AMS viene mantenuto solo nel caso in cui si verifichi un'emergenza, dato che queste teoricamente non hanno una frequenza elevata, non dovrebbero pregiudicare il funzionamento del sistema.

Nella simulazione eseguita presso il Laboratorio di Sistemi Distribuiti del Dipartimento di ingegneria dell'informazione dell'Università è stato utilizzato un personal computer con processore Intel Core i7 a 2,8GHz, 8GB di memoria RAM e 300GB di disco fisso su cui era installato un sistema operativo Linux (Ubuntu 10.04). La prova consisteva nell'avvio simultaneo di una ventina di agenti pazienti, dieci agenti medici e dieci agenti OST per testare il sistema. Dalla simulazione non sono state evidenziate perdite di prestazioni nonostante vi fossero nel sistema mediamente più di un centinaio di agenti (si ricorda che ogni agente è gestito da un thread Java).

6.10. Guasti e ridondanze

In un sistema comprendente dispositivi eterogenei come quello realizzato, sono possibili dei guasti o malfunzionamenti che potrebbero pregiudicare il funzionamento del sistema stesso.

In fase di progettazione si era pensato che, per ogni dispositivo come quelli dei pazienti, quelli dei medici, il database e i sensori, potessero avere due tipi di malfunzionamento: o smettere di funzionare o funzionare in modo anomalo. Nel primo caso la rilevazione del malfunzionamento sarebbe stata abbastanza semplice implementando un meccanismo di feedback nella comunicazione in modo da accorgersi in tempi brevi se il dispositivo non fosse più funzionante. Nel secondo caso la rilevazione della anomalia sarebbe stata più complicata dato che avrebbe richiesto un'analisi dei

dati più approfondita per verificare eventuali valori fuori dal funzionamento ordinario: ad esempio se un sensore continua a trasmettere dati molto variabili fra un campionamento e un altro in un lungo periodo di monitoraggio, questo potrebbe essere indice di un qualche malfunzionamento.

Dato che non si è potuto disporre di dispositivi veri e propri, non si sono implementate tutte le funzioni pensate, bensì solo in minima parte.

Quando un dispositivo viene meno per guasto è importante avere un meccanismo di ripristino del sistema in modo da poter riportare il tutto ad una situazione ottimale.

Dal punto di vista del paziente si è implementata una procedura di backup che prevede la memorizzazione degli ultimi cinque messaggi spediti al database, questo perché sul dispositivo del paziente ci sono delle restrizioni di spazio dovute alla carenza di memoria poiché si ha a che fare con un apparecchio mobile. Sul lato medico e nelle emergenze, invece vengono prevalentemente usate macchine con memorie decisamente più capienti, quindi si è pensato di eseguire un backup di tutti i dati inviati al database riguardanti la cartella clinica, per il cui aggiornamento viene svolta una visita in scrittura.

Per quanto riguarda il database server, MySQL dispone di un sistema di archiviazione e ripristino già implementato al suo interno: è sufficiente solo impostare lo scheduling con cui deve essere eseguita la procedura e la locazione del backup. Il server eseguirà la procedura senza interferire con le normali attività del sistema.

Un punto critico del sistema è inoltre la piattaforma ad agenti che poggia su un unico container (il Main contaner) dove risiedono gli agenti che amministrano l'intero sistema multiagente. Nel caso in cui la macchina su cui si trova il Main container dovesse guastarsi, la stessa piattaforma verrebbe meno. Per ovviare a questo problema è stata introdotta la procedura descritta alla fine del capitolo relativo a JADE, ovvero all'avvio della piattaforma sono stati predisposti dei container secondari attivando gli appositi servizi, che vengono utilizzati anche nella creazione di ciascun container presente nel sistema.

Nelle simulazioni effettuate si è provato a distruggere il Main container per verificare se il sistema si comportasse in modo equivalente all'utilizzo di un solo Main container e si è potuto vedere che effettivamente la procedura di backup genera un sistema con le stesse prestazioni a regime.

6.11. Sicurezza

Gli aspetti riguardanti la sicurezza in questo sistema sono principalmente di due tipi: quelli riguardanti le comunicazioni e quelli riguardanti gli accessi.

Per quanto riguarda le comunicazioni è stato spiegato nei capitoli precedenti che lo scambio di messaggi avviene principalmente tra agenti utilizzando delle stringhe di testo secondo lo standard FIPA, quindi sarebbe relativamente semplice realizzare un sistema di cifratura dei messaggi a chiave pubblica o privata in modo da impedire a terze parti, che intercettino i messaggi, di leggere informazioni potenzialmente sensibili.

Nel sistema inoltre, ogni figura (paziente, medico e OST) all'ingresso, esegue un login nel che prevede l'inserimento delle proprie informazioni personali. Nell'implementazione per semplificare non si utilizzano password, che comunque possono essere agilmente aggiunte modificando gli agenti starter.

MySQL inoltre offre la possibilità di poter dare per ogni utente che accede alla base di dati particolari permessi: questi possono includere diversi aspetti fra cui l'impostazione di visite ristrette solo ad alcune tabelle o dare solo i permessi in scrittura e o in lettura e così via.

Infine, per limitare le comunicazioni al solo scambio di messaggi fra agenti si era pensato di far migrare o direttamente creare gli agenti sul database server, aspetto che offre vantaggi dal punto di vista della sicurezza e svantaggi sulle prestazioni come spiegato precedentemente nel capitolo 6.7.

7. Conclusioni

L'obbiettivo della tesi è stato quello di realizzare, mediante un paradigma ad agenti, un sistema pervasivo per l'assistenza sanitaria domestica. Per fare ciò è stata utilizzata la piattaforma multiagente di JADE e un sistema di gestione di base di dati relazionali come quello di MySQL.

Si è visto che un'architettura distribuita, in questo tipo di scenario, risulta molto utile per gestire questi sistemi. Oltretutto questo paradigma risulta molto attuale e in continua evoluzione oggigiorno fornendo molti spunti di sviluppo, poiché consente da un lato di sfruttare le diverse caratteristiche dei dispositivi che compongono il sistema, dall'altro, con l'introduzione di architetture middleware, consente di affrontare problematiche come eterogeneità (dei dispositivi e dei programmi), guasti, perdita di informazioni, scalabilità e sicurezza in maniera agile ed efficiente.

Dall'analisi dello stato dell'arte è stato possibile raccogliere importanti informazioni e suggerimenti per lo sviluppo del sistema che poi sono stati elaborati o adattati a seconda del contesto affrontato.

Lo sviluppo di un sistema che dovesse includere più dispositivi con diverse caratteristiche ha richiesto l'utilizzo di una tecnologia che fa parte del mondo Java, un ambiente che notoriamente fa della portabilità, dell'integrazione con terze parti, e della openess i suoi punti di forza.

JADE, realizzato in Java, in particolare, per la natura indipendente e autonoma degli agenti si è rivelato particolarmente adatto allo sviluppo di un sistema pervasivo, anche grazie al fatto che offre molte libertà al programmatore nella gestione dei comportamenti degli agenti, fornendo un linguaggio semplice e di immediata comprensione, ma non per questo povero di funzionalità.

Del sistema realizzato, inoltre, si sono sfruttati i servizi offerti dai vari software utilizzati, evidenziando i punti di forza e cercando, nelle debolezze rilevate, dove possibile, dei chiarimenti o delle soluzioni.

Nella realizzazione si è cercato di sviluppare tutte le caratteristiche rilevate in fase di progettazione facendo attenzione a rispettare le esigenze dei possibili utilizzatori del sistema.

Il sistema sviluppato è chiaramente solo un prototipo di quale può essere un'applicazione per l'assistenza sanitaria domestica. Questo lavoro aveva lo scopo di mostrare come si potrebbe realizzare questo tipo di sistemi in questo contesto. In futuro, con la possibilità di utilizzare dei veri dispositivi e senza simularli come è avvenuto, sarebbe possibile realizzare un'applicazione più completa, elaborando in maniera più approfondita alcuni aspetti o adattando l'applicazione anche a contesti più ampi che coinvolgano più categorie di utenti come quelle rilevate.

8. Ringraziamenti

Innanzi tutto vorrei porgere un sincero ringraziamento al mio relatore, Prof. Carlo Ferrari, per essere sempre stato disponibile, paziente e per i consigli dati durante la realizzazione del sistema in tesi. Molto probabilmente senza il suo aiuto non avrei ottenuto dei risultati che all'inizio del lavoro vedevo difficilmente raggiungibili.

Ringrazio anche i miei genitori per avermi supportato e sopportato in questo mio cammino universitario e per essere sempre stati presenti sia nelle situazioni felici sia soprattutto nei momenti difficili: sono e saranno sempre un punto di riferimento.

Un ringraziamento va anche a mia sorella, non solo per l'aiuto offerto in questa tesi per termini e procedure mediche (studia Medicina Veterinaria), ma anche per avermi sempre ascoltato, offrendomi, ove possibile, i suoi consigli e suggerimenti.

Infine ringrazio tutti i colleghi ed ex colleghi, amici che mi hanno accompagnato in questa avventura universitaria. Visto che anche lui mi ha citato nella sua tesi, ricordo in particolare Andrea Martini, per il supporto, i suggerimenti, le innumerevoli chat e per avermi fatto fare una cultura su telefonia cellulare, Batman e DC Comics.

Grazie a tutti!

9. Bibliografia

[1] Sito web ufficiale della piattaforma JADE
http://jade.tilab.com/

[2] Sito web ufficiale dello standard FIPA
http://www.fipa.org/

[3] Sito web ufficiale MySQL
http://www.mysql.com/

[4] Sito web ufficiale Java 3D
https://java3d.dev.java.net/

[5] Cay S. Horstmann, Gary Cornell
Java 2 Core, volume 1: Fondamenti
McGraw-Hill

[6] Cay S. Horstmann, Gary Cornell
Java 2 Core, volume 2: Tecniche avanzate
McGraw-Hill

[7] Andrew S. Tanenbaum, Maarten Van Steen
Sistemi Distribuiti – Seconda edeizione
Pearson – Prentice Hall

[8] Giovanni Caire
JADE tutorial – JADE programming for beginners
Manuale, versione 3.7, 2009

[9] Fabio Bellifemine, Giovanni Caire, Tiziana Trucco, Giovanni Rimassa
JADE programmer's guide
Manuale, versione 4.0, 2010

[10] Fabio Bellifemine, Giovanni Caire, Tiziana Trucco, Giovanni Rimassa, Roland Mungenast
JADE administrator's guide
Manuale, versione 4.0, 2010

[11] Luke Welling, Laura Thomson
MySQL Tutorial
Pearson

[12] Urs Anliker, Jamie A. Ward, Paul Lukowicz, Gerhard Tröster, François Dolveck, Michel Baer, Fatou Keita, Eran B. Schenker, Fabrizio Catarsi, Luca Coluccini, Andrea Belardinelli, Dror Shklarski, Menachem Alon, Etienne Hirt, Rolf Schmid, Milica Vuskovic
AMON: A Wearable Multiparameter Medical Monitoring and Alert System
IEEE Transactions On Information Technology In Biomedicine 8, 2004

[13] Wanhong Wu, Jiannong Cao, Yuan Zheng, Yong-Ping Zheng
WAITER: A Wearable Personal Healthcare and Emergency Aid System
Sixth Annual International Conference on Pervasive Computing and Communications, 2008

[14] Kevin Miller & Dr. Suresh Sankaranarayanan
Monitoring Patient Health using Policy based Agents in Wireless Body Sensor Mesh Networks
World Congress on Nature & Biologically Inspired Computing, 2009

[15] Zhenmin Zhu, Xiaoli Su, Faqun Jiang, Jintao Li, Jian Ye
A User-Centric Pervasive Computing Services Model for Medical and Health-care
Sixth International Conference on Grid and Cooperative Computing, 2007

[16] George Roussos
The Anatomy of Pervasive Self Care Services
Proceedings. International Conference on Pervasive Services 2005

[17] Yuki Tamura, Masaaki Hashida, Shuji Sannomiya, Makoto Iwata
Smart Pervasive Healthcare-Assistance in Daily Life
Pervasive Computing Technologies for Healthcare (PervasiveHealth), 209 3th International Conference

[18] Joanna Alicja Muras, Vinny Cahill, Emma Katherine Stokes
A Taxonomy of Pervasive Healthcare Systems
Pervasive Health Conference and Workshops, 2006

[19] Giacomo Cabri, Francesco De Mola, Letizia Leonardi
Agent-based Plug-and-Play Integration of Role-Enabled Medical Devices
Joint Workshop on High Confidence Medical Devices, Software, and Systems and Medical Device Plug-and-Play Interoperability, 2007

[20] Antonio Moreno, Aïda Valls, David Isern, and David Sánchez, University Rovira i Virgili
Applying Agent Technology to Healthcare: The GruSMA Experience
Intelligent Systems, IEEE, 2006

[21] Ryan W. Gardner, Sujata Garera, Matthew W. Pagano
Securing Medical Records on Smart Phones
Conference on Computer and Communications Security, 2009

[22] Andrew D. Jurik and Alfred C. Weaver
Remote Medical Monitoring
IEEE Computer Society Press, 2008

[23] Davy Preuveneers, Yolande Berbers
Mobile phones assisting with health self-care: a diabetes case study
International Conference on Human-Computer Interaction with Mobile Devices and Services, 2008

[24] Chakib Tadj, Manolo Dulva Hinal ,Amar Ramdane-Cheri, Ghislain Ngantchaha
The LATIS Pervasive Patient Subsystem: Towards a Pervasive Healthcare System
International Symposium on Communications and Information Technologies, 2006. ISCIT '06.

[25] V. Chan, P. Ray and N. Parameswaran
Mobile e-Health monitoring: an agent-based approach
IET Communications, 2009

[26] Victor Chan, Pradeep Ray, Nandan Parameswaran
A Multi-Agent Collaborative Framework for Mobile E-Health
40th Annual Hawaii International Conference on System Sciences, 2007. HICSS 2007.

[27] Byung-Mo Han, Seung-Jae Song, Kyu Min Lee, Kyung-Soo Jang, Dong-Ryeol Shin
Multi Agent System based Efficient Healthcare Service
The 8th International Conference on Advanced Communication Technology, 2006. ICACT 2006.

[28] A. Martini
Le piattaforme Jadex e Jason per lo sviluppo di sistemi multiagente, analisi del funzionamento e studio comparativo delle prestazioni
Tesi di laurea dell'Università di Padova, A.A. 2009-2010

[29] A. Caccin
Sistema ubiquo applicato all'assistenza a persone con problemi di memoria
Tesi di laurea dell'Università di Padova, A.A. 2009-2010