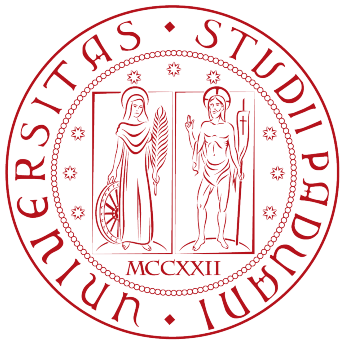

UNIVERSITÀ DEGLI STUDI DI PADOVA

Facoltà di Ingegneria

Tesi di Laurea Magistrale in

Ingegneria Informatica

Un sistema ubiquo basato su agenti per l'assistenza ai passeggeri in ambito aeroportuale

Relatore
Prof. Carlo Ferrari

Laureando
Gianluca Lorenzin

Anno Accademico 2011-2012

Ai miei genitori,
con amore e riconoscenza

Sommario

I sistemi ubiqui, detti anche sistemi pervasivi, stanno sempre più entrando a far parte delle vite quotidiane delle persone, assistendole e aiutandole a compiere operazioni, molto spesso senza che le persone stesse si rendino conto della loro presenza. Sistemi di questo genere sono destinati a diventare sempre più presenti nella società futura, permettendo di avere una maggiore qualità sui diversi servizi che saranno offerti alle persone. Questa tipologia di sistemi rappresenterà molto probabilmente il futuro dei sistemi distribuiti, permettendo l'esplorazione di nuovi scenari che fino ad oggi era solo possibile immaginare. Questo lavoro di tesi si pone come obiettivo quello di realizzare un sistema ubiquo sfruttando un'infrastruttura ad agenti, al fine di dimostrare in un caso reale l'applicabilità e i vantaggi di questo tipo di approccio. Il sistema sviluppato riguarda l'ambito del trasporto aereo e più in particolare le operazioni che le persone devono compiere all'interno di un aeroporto. Il sistema permetterà di fornire assistenza ai passeggeri, al fine di aiutarli nello svolgere tutte quelle operazioni che necessariamente bisogna compiere prima di salire sull'aereo e una volta scesi a destinazione, al fine di facilitare le procedure per i passeggeri e velocizzare le varie operazioni, così da diminuire i vari tempi di attesa. All'interno del progetto è stata anche sviluppata un'applicazione che sfrutta l'intero sistema e che rappresenta il mezzo attraverso cui l'utente finale interagisce con esso. L'applicazione è presente in due versioni, quella per computer e quella per la piattaforma Android, in modo da poter essere installata anche su una larga fetta di smartphone attualmente in commercio.

Abstract

Ubiquitous systems, also called pervasive systems, are increasingly becoming part of people's everyday lives, assisting and helping them to perform operations, often without people themselves are aware of their existence. Such systems are expected to become increasingly present in future society, thus allowing a higher quality on the various services that will be offered to persons. This type of systems most likely represent the future of distributed systems, allowing the exploration of new scenarios that until now was only possible to imagine. This thesis work aims to realize an ubiquitous system through an agents infrastructure in order to demonstrate the applicability in a real case and the advantages of this approach. The developed system concerns the context of air transport and in particular the operations that people have to make within an airport. The system will provide assistance to passengers in order to assist them in performing all those operations that need to be accomplished before boarding the plane, and once get off at destination, in order to facilitate the procedures for passengers and speed up various operations, thus decreasing the waiting times. Within the project has also developed an application that uses the entire system and that the end user will use to interact with it. The application is available in two versions, one for computer and one for the Android platform, so it can also be installed on most of the smartphones currently on the market.

Indice

Capitolo 1

Introduzione

1.1 Ambito e obiettivi della tesi

La relazione che le persone hanno con la tecnologia cambia in continuazione, ma è soprattutto negli ultimi anni che stiamo assistendo ad un cambiamento più che mai marcato. La crescente diffusione di telefoni cellulari sempre più "smart", assieme all'aumento della disponibilità di accesso alla rete Internet, hanno fatto si che la società stessa subisse profondi cambiamenti.

Questo trend ha portato alla nascita di nuovi modelli per quanto riguarda le infrastrutture a supporto delle nuove tecnologie. Tra i modelli più recenti troviamo il cosiddetto *ubiquitous computing*, il cui vero potenziale molto probabilmente non è ancora stato sfruttato al massimo. L'ubiquitous computing prevede un ambiente in cui ogni oggetto come può essere un vestito, un elettrodomestico, un'automobile, una casa e anche il corpo umano stesso possiede uno o più chip al suo interno per connettere quel particolare oggetto alla rete praticamente infinita formata da altri oggetti dotati di chip a loro volta e tutto ciò senza intervenire in maniera intrusiva nella vita delle persone, ma anzi molto spesso restando invisibili e fondendosi nell'ambiente stesso. Esiste un altro termine per definire bene la situazione appena descritta, ovvero *calm technology* che indica quella tecnologia che dà delle informazioni all'utente finale ma che non richiede l'attenzione di quest'ultimo.

I campi applicativi di questo tipo di sistemi sono molti: si immagini ad esempio di avere una rete di sensori all'interno della propria abitazione, i quali controllano la temperatura e possono decidere di alzarla nel caso in cui questa scenda sotto una certa soglia e ci siano delle persone all'interno dell'abitazione, mentre nel caso in cui non ci fosse nessuno in casa magari il riscaldamento non verrebbe attivato così da risparmiare energia; oppure si pensi al monitoraggio di vari ambienti, anche cittadini, al fine di fare rilevazioni utili per fornire servizi alle persone, le quali potrebbero accedere a questi dati attraverso i propri smartphone o computer; si pensi all'ambito

dei trasporti, dove più sensori sono presenti nei vari mezzi e sono in grado di comunicare tra loro, notificando in tempo reale un ingorgo o un incidente, al fine di far deviare i percorsi degli automobilisti in caso di bisogno oppure, nell'ambito del trasporto pubblico, si potrebbe avere una maggiore coordinazione tra autobus, metropolitane, treni e aerei al fine di fornire un servizio migliore ai viaggiatori. Questi sono solo alcuni esempi delle possibili applicazioni pratiche che si possono ottenere dall'utilizzo dei sistemi ubiqui.

Ed è proprio nell'ambito dei trasporti pubblici che si pone questo lavoro di tesi. In particolare il progetto riguarda il trasporto aereo e più in particolare le operazioni che devono compiere i passeggeri prima di imbarcarsi sull'aereo e una volta arrivati a destinazione. L'idea è quella di fornire uno strumento di supporto soprattutto per quelle persone che si trovano per la prima volta a dover prendere un aereo e quindi a dover fare i conti che le varie procedure e regole, a volte non molto chiare, che bisogna seguire all'interno di un aeroporto. Il sistema che verrà realizzato sarà un sistema ubiquo che permetterà ai passeggeri in transito negli aeroporti di ottenere vari tipi di informazioni personalizzate, cioè legate alla particolare situazione in cui si trova una certa persona, tutto questo in una maniera non intrusiva. Sarà infatti l'utente stesso a decidere quando visualizzare queste informazioni, le quali potranno essere diverse a seconda della posizione dove ci si trova. Oltre a questo, sarà possibile utilizzare dei particolari monitor posizionati i punti strategici dell'aeroporto per avere informazioni riguardanti il proprio viaggio e tutto questo senza dover premere un solo bottone, ma semplicemente avvicinandosi a questi monitor informativi.

Oltre all'infrastruttura verrà anche realizzata un'applicazione, attraverso la quale le persone potranno interagire con il sistema e viceversa; questa applicazione verrà realizzata in due versioni, una per i computer e una per la piattaforma Android, così da poter essere eseguita su un vasto numero di smartphone che utilizzano questo sistema operativo. L'applicazione permetterà tra le altre cose di ricevere notifiche in tempo reale su eventuali cambiamenti, come l'orario di partenza o lo stato del volo, sarà inoltre possibile avere suggerimenti per la particolare operazione che bisogna compiere e indicazione sul dove bisogna recarsi per portarla a termine; oltre a queste saranno presenti anche altre funzioni il cui scopo sarà comunque quello di rendere più facile lo spostarsi all'interno dell'aeroporto.

Il sistema sfrutterà il modello ad agenti: i motivi della scelta di questo particolare modello sono dovuti al fatto che il paradigma ad agenti, sfruttando l'architettura P2P, è particolarmente indicato per applicazioni che richiedono un certo livello di scalabilità e per un sistema di questo tipo, che potrà essere utilizzato da migliaia di persone contemporaneamente, l'aspetto riguardante la scalabilità non è di certo secondario; altro aspetto interessante di questo paradigma è che un agente ha la possibilità di migrare su host diversi e questa è una caratteristica che permette di ottenere funzionalità che con altri modelli architetturali probabilmente sarebbe stato difficile imple-

mentare; oltre a questi motivi c'è stata anche la volontà di voler mettere alla prova su un caso pratico un paradigma, quello ad agenti, che è in continua evoluzione e in continuo studio, per vederne le potenzialità ed eventualmente i difetti quando viene applicato a situazioni di questo tipo.

1.2 Struttura della tesi

Oltre a questo capitolo introduttivo, sono presenti altri cinque capitoli che coprono i seguenti argomenti;

- *capitolo 2*: panoramica sui sistemi ubiqui e sul paradigma ad agenti. In questo capitolo verranno dati i fondamenti teorici di quelli che sono i due argomenti principali di questa tesi, andando a spiegare abbastanza nel dettaglio cosa sono i sistemi ubiqui, o sistemi pervasivi, e il paradigma ad agenti, facendo anche un confronto tra il modello client/server e quello peer-to-peer, quest'ultimo utilizzato proprio dai sistemi ad agenti;

- *capitolo 3*: analisi iniziale e progettazione del sistema. Questo capitolo parla dell'analisi iniziale che è stata fatta per capire meglio come funziona un aeroporto, di quali sono le procedure e di che cosa gli utenti avevano bisogno; oltre a questo è presente l'analisi delle architetture del sistema (quella hardware e quella software) con la spiegazione dei vari elementi che le compongono;

- *capitolo 4*: panoramica sulle due piattaforme utilizzate. Il capitolo inizia con la descrizione della piattaforma JADE, utilizzata per sviluppare il sistema ad agenti, nella quale si parla dell'architettura di questa piattaforma, dei servizi e dei tool che mette a disposizione degli sviluppatori. Il secondo argomento di questo capitolo è la piattaforma Android della quale verrà descritta l'architettura con un'analisi anche dei vari componenti che ne fanno parte, del come è strutturata un'applicazione e alcuni cenni su come praticamente si può svilupparne una;

- *capitolo 5*: descrizione dello sviluppo dell'applicazione. In questo capitolo è presente l'analisi dei vari aspetti dell'aeroporto che sono stati simulati, degli agenti sviluppati, delle due versioni dell'applicazione e di alcune interazioni tra gli agenti;

- *capitolo 6*: conclusioni tratte da questo lavoro di tesi e possibili sviluppi futuri. Questo capitolo finale parla delle conclusioni riguardanti l'utilizzo del paradigma ad agenti, il sistema e l'applicazione sviluppati e gli strumenti utilizzati; infine vengono proposti dei possibili sviluppi che per mancanza di tempo non è stato possibile implementare, ma che aumenterebbero le funzionalità del sistema realizzato.

Capitolo 2

Sistemi ubiqui e sistemi multiagente

Questo capitolo parla degli aspetti teorici che fanno da cardine al sistema che è stato realizzato e che è oggetto di questa tesi. Vengono descritti i sistemi ubiqui (sezione 2.1), chiamati anche sistemi pervasivi e i sistemi multiagente (sezione 2.2), che rappresentano una delle possibili scelte quando si vogliono realizzare sistemi ubiqui.

2.1 Sistemi ubiqui

In questa sezione verranno discussi i sistemi ubiqui, una particolare categoria di sistemi distribuiti in cui l'instabilità è il comportamento normale e i dispositivi che li compongono sono spesso piccoli, a batteria, mobili, con una connessione esclusivamente senza fili e fanno parte dell'ambiente che ci circonda senza a volte che ce ne accorgiamo.

Come prima cosa verrà spiegato meglio che cosa sono questi sistemi e da dove nasce il termine UbiCom che li rappresenta, dovuto a Mark Weiser; poi verranno elencate e spiegate le caratteristiche chiave di questo tipo di sistemi.

2.1.1 Che cosa sono i sistemi ubiqui

Viviamo in un mondo che giorno dopo giorno si sta digitalizzando sempre di più, con moltissimi dispositivi digitali creati apposta per assistere e automatizzare le azioni delle persone, per arricchire le interazioni sociali tra di esse e quelle con la realtà vera e propria. Il mondo fisico si arricchisce ogni giorno di sensori e dispositivi di controllo che possono facilmente localizzare la nostra posizione e automaticamente eseguire delle operazioni sulla base di questa, come ad esempio accendere delle luci non appena ci avviciniamo. Ci sono poi dispositivi come i lettori di carte magnetiche che permettono

di aprire porte, anche se la carta magnetica ce l'abbiamo in tasca, oppure dispositivi che permettono il download di informazioni dalla rete Internet, come PC, tablet e smartphone.

Tutte queste tecnologie sfruttano in maniera intensiva la rete Internet e quindi c'è bisogno che l'accesso ad essa sia il più possibile disponibile; questo ha fatto si che l'utilizzo di reti wireless subisse un rapido incremento ed ha portato ad una miniaturizzazione dell'infrastruttura per supportarle. I circuiti elettronici e i dispositivi sono fatti in modo da essere più piccoli, meno costosi e che possano funzionare in maniera più sicura e con meno energia. C'è un gran numero di dispositivi mobili che possono accedere a servizi locali e remoti, come gli smartphone, che fungono anche da macchine fotografiche, videocamere, riproduttori multimediali e console per videogiochi.

Il termine "ubiquo" viene ben descritto dalla frase *"appearing or existing everywhere"*, che unito poi a "computing" forma il termine Ubiquitous Computing (UbiCom) che è usato per descrivere i sistemi ICT (Information and Communication Technology), che fanno si che le informazioni e le operazioni siano disponibili ovunque, e ne supportano l'utilizzo intuitivo da parte delle persone, rimanendo invisibili ad esse.

Un mondo in cui i computer scomparivano e restavano sullo sfondo di un ambiente che consisteva di stanze ed edifici intelligenti venne pensato anni fa e più precisamente nel 1991, da parte di Mark Weiser, che ricopriva uno dei ruoli di rilievo all'interno del Computer Science Lab della Xerox PARC, ed è proprio a lui che si deve l'invenzione del termine *Ubiquitous Computing*. L'ubiquitous computing definisce un mondo nel quale le persone sono circondate da dispositivi calcolatori e da un'infrastruttura che le supporta in tutto quello che fanno. Sempre Weiser ha detto le seguenti parole che spiegano le tre ere dell'informatica (vedi Figura 2.1), l'ultima delle quali è proprio rappresentata dall'avvento dei sistemi ubiqui:

> *"Ubiquitous computing names the third wave in computing, just now beginning. First were mainframes, each shared by lots of people. Now we are in the personal computing era, person and machine staring uneasily at each other across the desktop. Next comes ubiquitous computing, or the age of calm technology, when technology recedes into the background of our lives."*

2.1.1.1 Le proprietà chiave dei sistemi ubiqui

Le caratteristiche che distinguono i sistemi ubiqui (chiamati anche sistemi pervasivi) da quelli distribuiti sono le seguenti: come prima cosa i sistemi ubiqui sono localizzati in ambienti focalizzati sulle persone ed interagiscono con esse in maniera non intrusiva. Secondo, i sistemi ubiqui vengono utilizzati e fanno parte di ambienti fisici e sono in grado di rilevare l'ambiente che

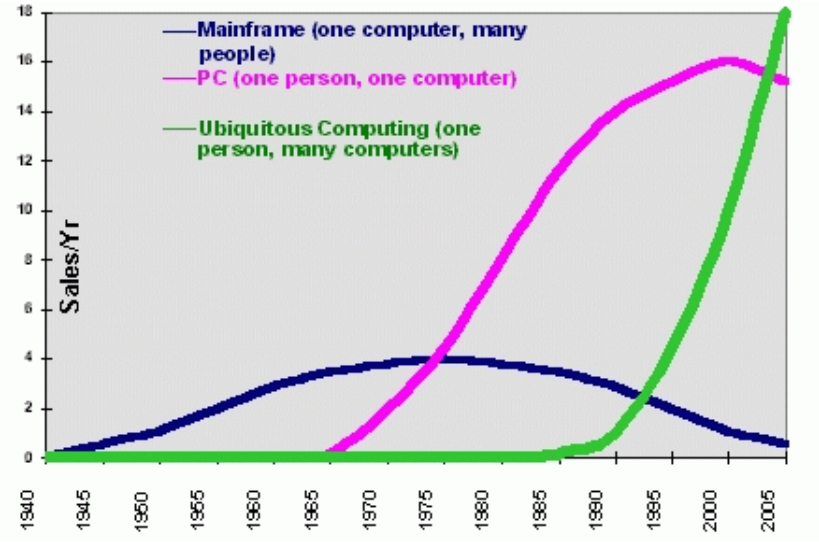

Figura 2.1: Le tre maggiori tendenze nel mondo del sistemi di calcolo

li circonda, adattandosi e interagendo con esso, oltre a controllarlo. Quindi, l'ubiquitous computing può essere riassunto in tre requisiti fondamentali:

1. i computer devono essere connessi alla rete, distribuiti e accessibili in maniera trasparente;
2. l'interazione persona-computer deve essere nascosta il più possibile;
3. i computer devono essere "sensibili al contesto" in modo da ottimizzare le loro operazioni in base all'ambiente che li circonda.

   Sono state proposte altre due caratteristiche chiave per i sistemi ubiqui:

4. i computer possono operare in maniera autonoma, senza l'intervento umano, gestendosi da soli;
5. i computer possono gestire una molteplicità di azioni dinamiche e di interazioni, controllate da una intelligenza decisionale e da un'organizzazione delle interazioni intelligente. Questo può portare a una forma di intelligenza artificiale in grado di gestire:

   - interazioni incomplete e non deterministiche;
   - cooperazione e competizione fra i membri del sistema ubiquo;
   - una maggiore interazione attraverso la condivisione del contesto, della semantica e degli obiettivi.

L'aggiunta degli ultimi due punti riguarda il lavoro dei sistemi ubiqui in due gruppi di ambienti: il primo gruppo riguarda gli ambienti centralizzati

sulle persone e che quindi vedono una grossa interazione con quest'ultime; il secondo gruppo invece riguarda gli ambienti formati da oggetti fisici inanimati. Le cinque caratteristiche chiave dei sistemi ubiqui e i tre tipi di ambienti (ICT, fisici e umani) non sono mutuamente esclusivi, anzi si sovrappongono molto spesso e si combinano tra loro.

Dalla discussione appena fatta, dovrebbe risultare chiaro che sono soprattutto due i punti fondamentali per un sistema ubiquo: l'integrazione fisica con l'ambiente e l'interoperabilità "spontanea" con gli altri componenti del sistema.

Per quanto riguarda l'integrazione fisica, un sistema ubiquo deve necessariamente fornire una qualche integrazione tra i nodi di calcolo e il mondo fisico. Ad esempio, basti pensare ad una sala riunioni "intelligente", dove vari dispositivi sono presenti nelle sedie, nei tavoli, nella lavagna ed ognuno di essi percepisce la realtà che lo circonda e sulla base di questa offre dei servizi agli utenti finali, collaborando con gli altri dispositivi presenti. I sistemi ubiqui sono inoltre localizzati in ambienti (environments) per così dire discreti, cioè delimitati in un qualche modo, come stanze, case, aeroporti e uffici; per questo motivo, i progettisti di questi sistemi devono dividere i sistemi ubiqui dentro ambienti con dei confini che ne limitano il contenuto. Un chiaro confine del sistema è spesso il mondo fisico stesso, come per gli esempi che abbiamo detto prima; una cosa importante è che il confine dell'ambiente dove è posizionato il sistema ubiquo definisce solo lo scopo di quest'ultimo ma non ne vincola l'interoperabilità con i componenti al di fuori dei suoi confini. Per capire meglio cosa si intende, si pensi al sistema ubiquo presente all'interno di una sala riunioni il cui scopo è quello di fornire servizi e supporto alle operazioni che normalmente avvengono all'interno di questa stanza, ma non per questo è limitato nelle comunicazioni ad esempio con l'ufficio del responsabile, dove è presente un altro sistema ubiquo, e con il quale può scambiarsi informazioni di vario genere.

L'altro punto fondamentale è l'interoperabilità "spontanea". In un ambiente saranno presenti vari componenti, cioè piccole unità software che forniscono un'astrazione come servizi, client, risorse o applicazioni. In un sistema ubiquo questi componenti devono interoperare spontaneamente al modificarsi dell'ambiente, cioè collaborare con componenti che possono cambiare sia la loro identità che le loro funzioni, a seconda di come evolve l'ambiente che li circonda. Un componente che interagisce in maniera spontanea può cambiare il suo interlocutore durante le normali operazioni, magari perché egli stesso si è spostato in un altro ambiente o perché un altro componente è entrato a far parte del suo e questo cambiamento lo fa senza il bisogno di nuovo software o di nuovi parametri. Un esempio per capire meglio questo concetto potrebbe essere quello di una persona che porta all'interno della sala riunioni "intelligente" il proprio computer e da questo può inviare le proprie presentazioni al proiettore; in questo caso, sia il computer che il proiettore devono collaborare tra loro, anche se è la prima volta che fan-

no parte dello stesso ambiente. Per questi motivi bisognerebbe progettare un sistema ubiquo sul presupposto che l'insieme formato dagli utenti finali, dall'hardware e dal software è altamente variabile e non predicibile.

2.2 Sistemi multiagente

Andiamo ora a parlare dei sistemi multiagente, che abbrevieremo con l'acronimo MAS, ovvero Multi-Agent Systems. Questa nuova tecnologia sta riscuotendo un certo successo negli ultimi anni perché rappresenta un nuovo paradigma per la concettualizzazione, la progettazione e l'implementazione dei sistemi software, in particolare per quelli distribuiti. Questa nuova tecnologia non sfrutta il modello client-server (C/S) che si può rivelare inadatto in certe situazioni, ma usa invece il modello peer-to-peer (P2P), attraverso il quale gli agenti possono comunicare tra loro in modo paritario. Prima quindi di parlare dei sistemi multiagente veri e propri, facciamo una breve descrizione dei due modelli architetturali appena accennati per poi farne un confronto dove verranno evidenziati i vantaggi e gli svantaggi di entrambi.

2.2.1 Il modello architetturale client-server

In questo modello, abbiamo che i processi del sistema si dividono in due categorie: server e client. Un server è un processo che mette a disposizione un certo servizio, come ad esempio un server http. Un client invece è un processo che richiede l'utilizzo del servizio al server e lo fa inviandogli delle richieste e attendendo poi delle risposte. In pratica, il client prepara un messaggio, all'interno del quale specifica il servizio richiesto ed eventualmente altre informazioni necessarie, dopodiché lo invia e attende la risposta. Il server che riceve la richiesta, la elabora e restituisce il risultato, sempre attraverso un messaggio, al client. Risulta quindi chiaro che nel modello C/S, è il client che prende l'iniziativa e che inizia la comunicazione, mentre il server è un processo totalmente reattivo, che risponde alle richieste del client; questo però rappresenta un problema, in quanto i client hanno bisogno degli indirizzi dei server per poter comunicare con essi, mentre i server devono conoscere gli indirizzi dei client per poter fornire loro le riposte. La soluzione al problema è abbastanza semplice e consiste nel mettere in ascolto il server su una porta conosciuta dai client (well-known port); in questo modo, una volta che il client si è connesso al server, può comunicare con esso, senza che il server debba essere configurato con qualche informazione sul client.

Una cosa molto importante, è che i due processi che devono comunicare fra loro, parlino lo stesso linguaggio, basta ad esempio pensare al protocollo HTTP utilizzato nella rete Internet, il quale viene utilizzato dai browser (cioè i client) che inviano le richieste delle pagine html ai server http, i quali

rispondono inviando la pagina cercata. Un aspetto fondamentale che differenzia il modello C/S da quello P2P, è che nel primo caso, le comunicazioni che avvengono sono solo tra i processi client e i processi server, ma non può capitare che due processi dello stesso tipo comunichino tra loro. In questo modello architetturale, di solito tra il server e il client è quest'ultimo ad essere più complesso, mentre i client sono di solito più semplici (anche se questo non è sempre vero).

Come è stato accennato in precedenza, questo modello architetturale è stato molto usato e continua ad esserlo, grazie alla semplicità di realizzazione e di manutenzione. Ciò nonostante, l'architettura C/S ha alcuni difetti, come il fatto di non permettere l'utilizzo della potenza di calcolo dei client, ma solo dei server; inoltre ci sono casi nei quali questa architettura non rappresenta la scelta migliore. Basti pensare a situazioni in cui è necessario che i vari processi del sistema distribuito comunichino tra loro in maniera paritaria. Per applicazioni di questo tipo, la scelta migliore è l'architettura P2P.

2.2.2 Il modello architetturale peer-to-peer

Da un punto di vista ad alto livello, i processi (peer) che costituiscono un sistema P2P sono tutti uguali, cioè i processi che formano il sistema distribuito rappresentano tutte le funzioni che devono essere portate a termine. Di conseguenza, l'interazione tra processi è quasi tutta simmetrica: ogni processo agisce come client e come server allo stesso tempo (a volte viene utilizzato il termine *servent* per indicare questo fatto).

Il modello P2P può essere visto come il mettere su uno stesso computer un processo server e uno client. Quindi ogni computer può accedere ai servizi presenti nel sistema utilizzando il processo client e allo stesso tempo fornire servizi attraverso il processo server. Questo modello implica però una cosa, cioè che ogni processo deve conoscere l'indirizzo di tutti gli altri, o almeno quello dei processi con cui deve comunicare, problema questo che non può essere risolto come era stato fatto nel caso dell'architettura C/S.

Risulta abbastanza chiaro che è presente una certa complessità a causa della scoperta, della comunicazione e della gestione del grande numero di computer e processi coinvolti nel sistema distribuito; per cercare di gestire al meglio questa complessità, tipicamente il modulo software di una rete P2P è strutturato su tre livelli (layer): il *base overlay layer*, il *middleware layer* e l'*application layer*.

Il *base overlay layer* si occupa del problema della scoperta degli altri partecipanti nel sistema P2P e della creazione del meccanismo che permette a tutti i nodi di comunicare tra loro. Questo livello garantisce che tutti i partecipanti siano a conoscenza della presenza degli altri. Le funzionalità fornite in questo livello sono quelle di base, necessarie ad un sistema P2P per funzionare.

Il *middleware layer* invece fornisce altri componenti software che possono essere utilizzati da differenti applicazioni. Il termine "middleware" viene usato per riferirsi ai componenti software che vengono principalmente invocati da altri componenti e usati come supporto all'infrastruttura usata per costruire altre applicazioni. Tra le funzioni che fanno parte di questo livello troviamo quella capace di creare un indice distribuito per il reperimento delle informazioni nel sistema, le funzioni pubblica/sottoscrivi (publish/subscribe) (queste funzioni permettono alle applicazioni di inviare messaggi a punti di contatto logici, di solito descritti per mezzo di un oggetto e allo stesso modo le applicazioni possono indicare il loro interesse per messaggi di un certo tipo, dopodiché il middleware di comunicazione si prende carico del fatto che i giusti messaggi siano inviati alle giuste applicazioni) ed infine i servizi di sicurezza. Le funzioni messe a disposizione in questo livello non sono necessarie per tutte le applicazioni, ma sono fatte per essere utilizzate da più di una sola applicazione.

L'ultimo livello è l'*application layer*, il quale mette a disposizione i pacchetti software che sono fatti per essere usati dagli utenti finali e sviluppati in modo da poter sfruttare la caratteristica di decentralizzazione dell'architettura P2P.

Un grosso vantaggio del modello P2P è dato dal fatto che la collaborazione di più processi e computer permette di ottenere una maggiore potenza di calcolo rispetto all'utilizzo di un singolo computer e questo fa si che le architetture P2P permettano una maggiore scalabilità rispetto a quelle C/S (motivo per cui è stata scelta proprio questa architettura per il sistema oggetto di questa tesi).

2.2.3 Confronto tra le architetture C/S e P2P

Per confrontare le due architetture andremo a vedere dei punti chiave che bisognerebbe considerare quando ci si trova sul punto di progettare un sistema e si deve scegliere quale delle due usare; per ognuno di questi punti si andranno a vedere i vantaggi e gli svantaggi di ognuna delle due architetture:

- *facilità di sviluppo*: per sviluppare applicazioni C/S sono disponibili da tempo un gran numero di programmi applicativi che rendono semplice questo compito. Molti componenti software come web server, web-application server e software di messaggistica sono disponibili in rete (sia open source che proprietari), fornendo un'infrastruttura pronta per essere utilizzata nel realizzare questo tipo di applicazioni. Per quando riguarda le applicazioni P2P invece è disponibile un numero più piccolo di pacchetti software ed inoltre, questi sono relativamente recenti con tutti i problemi che ne consegue, come ad esempio la presenza di bug che non sono da sottovalutare. Inoltre testare e debuggare un server centralizzato è certamente più semplice che farlo su

un software distribuito che richiede interazioni tra molte componenti. Per questi motivi, dal punto di vista della semplicità di sviluppo, è vantaggioso realizzare applicazioni C/S;

- *gestibilità*: con questo termine si intende la facilità di gestione del software finale, una volta che è stato distribuito agli utenti. Infatti bisognerà continuare a fare manutenzione, assicurandosi che l'applicazione non si blocchi (e in quel caso che poi riparta), fare copie di backup dei dati importanti generati, eseguire gli aggiornamenti, risolvere tutti i bug che vengono trovati e molte altre cose. Operazioni come il backup dei dati, gli aggiornamenti e la risoluzione dei bug sono più facili su un'applicazione centralizzata, piuttosto che su una che gira su diverse piattaforme. Nel caso C/S infatti, abbiamo che il server gira su un'unica piattaforma che viene scelta attentamente (cioè vengono scelte la caratteristiche hardware e software), mentre la scelta per il client non è così importante; quindi risulta relativamente facile gestire il server perché gira su un'unica piattaforma e per quanto riguarda il client molto spesso è necessaria sono una piccola manutenzione. Nel caso del P2P invece, l'applicazione è distribuita su piattaforme diverse e che fanno parte di diversi domini amministrativi, il che rende difficile avere un'omogeneità nella scelta delle piattaforme utilizzate; questo rende chiaramente la gestione di questo tipo di applicazioni più complessa. Linguaggi come Java, ovvero *platform-indipendent*, hanno semplificato in parte la gestione delle applicazioni P2P, assieme anche al forte uso della piattaforma Microsoft Windows che ha limitato il numero delle piattaforme utilizzate. Comunque, risulta evidente che in generale è più gestibile un applicazione C/S rispetto rispetto ad una P2P;

- *scalabilità*: la scalabilità di un'applicazione è misurata in termini del più alto numero, o dimensione, di interazioni con l'utente che l'applicazione può avere, mantenendo delle performance ragionevoli. Le applicazioni P2P utilizzano molti computer per risolvere un problema o un compito e per questo forniscono una soluzione più scalabile rispetto alle applicazioni C/S, le quali si affidano su un unico computer per portare a termine lo stesso compito. In generale, usando più computer, invece che uno solo, si tende ad aumentare la scalabilità di un'applicazione. C'è da dire però che anche una soluzione centralizzata sul server può essere implementata attraverso più computer. Basta utilizzare un dispatcher che smista le varie richieste ai server paritari (cioè ogni server mette a disposizione gli stessi servizi), suddividendo così il carico lavoro; inoltre il dispatcher tiene traccia di dove sono state inviate le richieste dei client, così da inviarle sempre allo stesso server. In generale, usando un numero uguale di computer

per risolvere un certo problema, utilizzarli come server fornisce un sistema più scalabile rispetto all'utilizzarli come sistema P2P; questo perché l'infrastruttura P2P richiede molte comunicazioni fra i vari nodi per portare a termine un'operazione e quindi un maggiore overhead se comparato con l'altra soluzione. In molti casi, una soluzione centralizzata è più efficiente di una distribuita. Ciononostante, bisogna sottolineare il fatto che per costruire una soluzione centralizzata del sistema, bisogna avere a disposizione dei computer dedicati e lo spazio fisico dove metterli, mentre nel caso P2P possono venire utilizzati computer già esistenti, magari con performance inferiori rispetto agli altri e che non vengono usati di continuo. Questo permette all'infrastruttura P2P di utilizzare molti più computer ad un costo davvero basso e quindi di avere una soluzione comunque scalabile ma ad un costo di molto inferiore rispetto a quella C/S, aspetto questo da non sottovalutare;

- *domini di amministrazione*: uno dei fattori chiave per decidere come strutturare un'applicazione è il modo in cui i computer che verranno utilizzati per essa sono amministrati. In generale, con un approccio C/S, i computer che fanno da server sono necessariamente sotto lo stesso dominio amministrativo e quindi tipicamente questo approccio non viene usato se i server devono invece far parte di diversi domini amministrativi. Un sistema P2P invece, viene spesso creato usando computer di vari domini amministrativi diversi e per questo motivo, se la necessità è proprio questa, la scelta migliore e più naturale è quella di seguire l'approccio P2P;

- *sicurezza*: la gestione della sicurezza implica diverse cose, come l'assicurarsi che al sistema accedano solo gli utenti autorizzati, che le credenziali utilizzate da questi ultimi siano autentiche e che utenti "maligni" non diffondano virus o trojan horse all'interno del sistema. In generale, la sicurezza su un sistema centralizzato è più facilmente gestibile rispetto a quella su un sistema distribuito; in quest'ultimo infatti i componenti che si occupano della sicurezza devono essere replicati in ogni host e questo aumenta il costo dell'infrastruttura che si occupa della sicurezza. Inoltre, la presenza di più host fa crescere la vulnerabilità del sistema, in quanto l'attacco può arrivare da punti diversi. Per questi motivi, la sicurezza è certamente più facile da gestire nei sistemi C/S rispetto che in quelli P2P;

- *affidabilità*: l'affidabilità di un sistema viene misurata sulla base della capacità che ha esso di continuare a svolgere i propri compiti anche quando una o più componenti si guastano e cessano di funzionare correttamente. L'approccio di solito usato per aumentare l'affidabilità di un sistema, è mettere delle componenti ridondanti ad altre, in modo

che se una si guasta è possibile attivarne un'altra che svolge gli stessi compiti. Questo risultato si può ottenere sia usando architetture C/S che P2P: nel primo caso, la soluzione adottata per aumentare la scalabilità risolve anche il problema dell'affidabilità (se un server si rompe, il dispatcher eviterà di inviare richieste a quello ma utilizzerà i restanti che svolgono comunque gli stessi compiti); nel secondo caso invece, è la natura stessa dell'infrastruttura che utilizza molti computer che fanno le stesse operazioni e quindi anche se uno di essi si rompe o cessa di funzionare per altri motivi (basta pensare alle reti P2P per il file sharing dove gli utenti possono spegnere o accendere il computer in qualsiasi momento) il sistema continua a funzionare correttamente. Come nel caso della scalabilità, la differenza tra l'affidabilità nei sistemi C/S e quella nei sistemi P2P è il costo al quale questa viene ottenuta: l'approccio P2P infatti ha un'intrinseca affidabilità che viene quindi raggiunta a basso costo, a differenza dell'approccio C/S dove la replica delle componenti può avere costi elevati.

2.2.4 Il paradigma ad agenti

Come prima cosa è importante iniziare dalla definizione di agente, anche se è bene dire che non ne esiste una versione universalmente riconosciuta:

> *"An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives."*

Da questa definizione risulta quindi che un agente è posizionato all'interno di un ambiente, ha degli obiettivi da raggiungere e per fare ciò è in grado di compiere della azioni in totale autonomia. Molti ambienti avranno una certa complessità e un agente non ne potrà avere il controllo pieno, ma solo parziale e limitato a quello che, attraverso le sua azioni, potrà influenzare. Dal punto di vista dell'agente, questo significa che l'azione che lui esegue può avere risultati ed effetti diversi da quelli che si potrebbero aspettare e che quindi in generale la possibilità di un fallimento è sempre considerata possibile. Possiamo riassumere questa cosa dicendo che gli ambienti dove vanno a lavorare gli agenti sono non deterministici.

Di solito, un agente ha un certo repertorio di azioni che può compiere a sua disposizione, le quali rappresentano le sue capacità per modificare l'ambiente che lo circonda. C'è anche da dire che non tutte le azioni disponibili possono essere eseguite in qualsiasi istante, ma c'è la necessità della presenza di pre-condizioni affinché si possa eseguire una determinata azione; ad esempio un agente che si occupa di comprare oggetti di una qualche natura da un altro agente, non può farlo se non ha disposizione un qualcosa per pagare.

Il problema chiave di un agente è quello di riuscire a capire quale delle azioni a sua disposizione usare, in modo da raggiungere nel miglior modo l'obiettivo prefissato.

Sono stati nominati più volte gli ambienti (environment) dove gli agenti agiscono; questi ambienti sono stati suddivisi in varie tipologie, sulla base delle proprietà che essi hanno:

- *accessibile vs inaccessibile*: un ambiente accessibile è un ambiente nel quale l'agente è in grado di ottenere informazioni aggiornate, complete e accurate sullo stato dell'ambiente stesso. Purtroppo, la maggior parte degli ambienti reali risultano essere inaccessibili;
- *deterministico vs non deterministico*: un ambiente deterministico è un ambiente nel quale l'azione compiuta dall'agente produce esattamente l'effetto che ci si aspettava, non c'è incertezza sullo stato in cui ci si troverà dopo che l'azione è stata compiuta;
- *statico vs dinamico*: un ambiente statico rimane invariato nel tempo, eccetto che per le azioni degli agenti. Viceversa, un ambiente dinamico cambia in continuazione indipendentemente dalle azioni degli agenti. Il mondo reale è chiaramente un ambiente altamente dinamico;
- *discreto vs continuo*: un ambiente è discreto se c'è un numero finito e conosciuto di azioni ed eventi che possono capitare all'interno di esso.

2.2.4.1 Le proprietà degli agenti

Parliamo adesso delle capacità chiave che ci si aspetta gli agenti abbiano:

- *reattività*: un agente dev'essere in grado di percepire l'ambiente che lo circonda ed essere in grado di rispondere in maniera veloce ai cambiamenti di quest'ultimo, al fine di raggiungere i propri obiettivi;
- *proattività*: un agente deve essere in grado di prendere l'iniziativa, non dovendo per forza aspettare stimoli esterni, sempre al fine di raggiungere i propri obiettivi;
- *socialità*: gli agenti devono essere capaci di di interagire con gli altri agenti presenti nel sistema e più in generale nell'ambiente (i quali potenzialmente possono essere anche persone fisiche).

Andiamo ad analizzare queste capacità partendo dalla *proattività*. Un agente è proattivo se può eseguire azioni dette *goal-directed*: queste sono azioni descritte semplicemente da un piano da seguire per ottenere il risultato voluto. Questo piano si basa su delle assunzioni (precondition) e sugli effetti che si avranno se queste assunzioni sono valide e si segue il piano (postcondition). Gli effetti del piano o della procedura sono l'obiettivo (goal), cioè

quello che si vuole ottenere. Se le assunzioni sono vere nel momento in cui viene applicata la procedura, allora ci si aspetta che questa venga eseguita correttamente e che una volta terminata, gli effetti si siano verificati e quindi che l'obiettivo sia stato raggiunto. Il modello appena descritto si basa su due ipotesi: la prima è che l'ambiente non può cambiare autonomamente, perché altrimenti può essere che le assunzioni che erano valide nel momento in cui la procedura era iniziata, non lo siano più durante la sua esecuzione e questo porta a degli effetti non predicibili, se non al blocco della procedura stessa. La seconda ipotesi è che l'obiettivo non deve cambiare finché la procedura viene eseguita, ma deve rimanere valido fino al suo termine. Queste due ipotesi non possono essere sempre considerate vere, anzi ci sono molti ambienti dove nessuna delle due è valida, e in questi casi l'eseguire alla "cieca" le procedure senza preoccuparsi che le assunzioni fatte siano e restino verificate non è certamente la cosa più corretta da fare. In questo tipo di ambienti è meglio che gli agenti abbiano dei comportamenti reattivi.

Per quanto riguarda la *reattività*, abbiamo che un agente è reattivo se risponde agli eventi che accadono nell'ambiente che lo circonda, dove questi eventi possono coinvolgere sia gli obiettivi degli altri agenti, sia le assunzioni delle procedure che l'agente deve compiere al fine di raggiungere l'obiettivo.

La vera difficoltà non è creare un agente con solo comportamenti proattivi o solo reattivi, ma crearlo bilanciando le due tipologie. Infatti non vogliamo certamente un agente che esegue una procedura alla cieca, cioè che la esegue e se questa fallisce allora la ricomincia e via così; infatti può essere che il motivo per cui la procedura non va a buon fine sia la mancanza del verificarsi delle ipotesi che essa richiede e finché queste non sono valide è inutile tentare di eseguire la procedura. Quello che vogliamo è che l'agente non appena rileva un qualche cambiamento nell'ambiente che fa si che le ipotesi siano soddisfatte, esegua la procedura. Dall'altra parte, non vogliamo nemmeno che un agente reagisca sempre agli eventi che si verificano nell'ambiente, altrimenti potrebbe essere che non si focalizza su l'obiettivo che dovrebbe invece raggiungere. Risulta quindi chiaro che bilanciare le due cose non è cosa semplice.

Infine parliamo della *socialità*, che in un certo senso è una cosa semplice da ottenere, basti pensare ai milioni di computer che ogni giorno scambiano informazioni con altri computer e con le persone; ma la capacità di scambiarsi bit di informazione non è proprio un'abilità "sociale". Pensiamo al mondo delle persone, dove ognuno ha degli obiettivi totalmente diversi da quelli di un'altra persona e quindi, per raggiungerli, c'è la necessità di negoziare e cooperare con gli altri. Potremmo aver bisogno di capire e ragionare sugli obiettivi degli altri ed eseguire azioni che altrimenti non avremmo fatto (come ad esempio pagare dei soldi) al fine di cooperare con essi e raggiungere i nostri obiettivi. Questo tipo di abilità sociale è più complessa e meno facile da capire rispetto al semplice scambiarsi bit di informazioni.

2.2.4.2 Comunicazione e coordinazione

Uno dei punti chiave dei sistemi multiagente è la comunicazione. Infatti, gli agenti hanno bisogno di comunicare con gli utenti, con le risorse di sistema e con gli altri agenti nel caso in cui ci sia bisogno di cooperare, collaborare, negoziare o altro. In particolare, gli agenti interagiscono tra loro usando uno speciale linguaggio di comunicazione, chiamato *agent comunication language*, che fornisce una separazione tra l'atto comunicativo e il contenuto vero e proprio del messaggio. Per capire meglio questa cosa, si pensi che sono stati identificati un certo numero di cosiddetti *performative verbs*, che corrispondono a diversi tipi di *speech acts* (quest'ultimi sono espressioni che implicano un'azione, la quale può cambiare il mondo che ci circonda); esempi di questi performative verb sono *request*, *inform* e *promise*. Il primo agent communication language di una certa rilevanza fu KQML (Knowledge Query and Manipulation Language), che fu sviluppato agli inizi degli anni '90 da parte dell'ARPA Knowledge Sharing Effort, del governo degli Stati Uniti. KQML è un linguaggio e un protocollo per lo scambio di informazioni e conoscenze che definisce un certo numero di performative verb e permette che il contenuto del messaggio sia rappresentato da un linguaggio chiamato KIF (Knowledge Interchange Format).

Di recente, l'agent communication language più studiato e usato è il FIPA ACL (Agent Communication Language), che possiede molti aspetti che erano presenti anche nel KQML. La caratteristica principale di FIPA ACL è la possibilità di usare diversi linguaggi per esprimere il contenuto di un messaggio e di gestire le conversazioni attraverso dei protocolli di interazioni predefiniti.

La coordinazione è un processo che gli agenti usano al fine di assicurarsi che l'intera comunità di agenti stia lavorando in maniera coerente. Ci sono varie ragioni perché degli agenti devono intraprendere la coordinazione e tra queste troviamo:

- gli obiettivi degli agenti possono causare conflitti tra le azioni degli agenti;
- gli obiettivi degli agenti possono dipendere uno dall'altro;
- gli agenti possono avere diverse capacità e conoscenze;
- gli obiettivi degli agenti possono essere raggiunti più velocemente se diversi agenti ci lavorano assieme.

La coordinazione tra gli agenti può essere gestita con diversi approcci che includono l'*organizational structuring*, il *contracting*, il *multi-agent planning* e il *negotiation*.

L'*organizational structuring* fornisce un supporto attraverso la definizione di ruoli e di relazioni tra gli agenti. Il modo più semplice per assicurarsi un

comportamento coerente e l'assenza di conflitti all'interno del sistema multiagente è quello di nominare un agente supervisore, il quale avrà un visione di insieme dell'intero sistema e che sarà a capo della gerarchia che si andrà a creare. Questa rappresenta la tecnica di coordinamento più semplice, la quale utilizza la classica architettura master/slave o C/S per la distribuzione delle operazioni da compiere e l'allocazione delle risorse agli agenti slave da parte dell'agente master. E' comunque difficile dal punto di vista pratico utilizzare questo tipo di approccio perché risulta complicato creare un un controllo centralizzato, dato che va contro la natura decentralizzata dei sistemi multiagente.

L'approccio invece che si chiama *contracting* si basa su una cosiddetta market structure decentralizzata, nella quale gli agenti possono assumere due ruoli: manager e contractor. La premessa di base di questa tecnica di coordinamento è che se un agente non è in grado di risolvere un problema che gli è stato assegnato usando le sue risorse e le sue conoscenze, allora lo decomporrà in sotto-problemi e cercherà di trovare altri agenti con le necessarie risorse e conoscenze, disposti a risolvere questi sotto-problemi. Il meccanismo di assegnazione di questi sotto-problemi è formato dai seguenti passi:

1. l'agente manager annuncia il cosiddetto *contract* agli altri agenti;
2. presentazione delle offerte da parte degli altri agenti, chiamati cotracting, in risposta all'annuncio;
3. valutazione delle offerte degli agenti contracting da parte dell'agente manager, che cercherà di assegnare un sotto-problema al o ai contractor che hanno presentato le offerte più appropriate.

Un altro approccio è quello del *multi-agent planning*. Con l'obiettivo di evitare azioni e interazioni in conflitto o inconsistenti, gli agenti possono costruire un multi-agent planning, cioè un piano in comune con gli altri agenti, dove vengono scritte tutte le future azioni e interazioni richieste per raggiungere gli obiettivi, prevedendo anche delle interruzioni nell'esecuzione di questo piano per fare delle eventuali modifiche "al volo". Il multi-agent planning può essere centralizzato e distribuito. Nel *centralized multi-agent planning*, c'è di solito un agente coordinatore al quale tutti gli altri mandano i loro piani parziali; sarà poi questo agente a fare l'analisi e ad individuare e rimuovere le inconsistenze e le interazioni in conflitto presenti nei piani, così da ottenere un unico piano generale corretto. Per quanto riguarda invece il *distribuited multi-agent planning*, l'idea è quella che ogni agente possiede un modello del piano degli altri agenti; questi, comunicheranno tra loro in modo che ciascuno di essi possa aggiornare il proprio piano e modificare il modello di quello degli altri, finché non saranno eliminati tutti i conflitti.

L'ultimo approccio è *negotiation*, ovvero la trattativa, che probabilmente è la tecnica più affidabile. In particolare, la trattativa è il processo di

comunicazione di un gruppo di agenti al fine di raggiungere un accordo su una qualche questione. La trattativa può essere competitiva o cooperativa, in base al comportamento degli agenti coinvolti. La trattativa competitiva (competitive negotiation), è usata in situazioni dove gli agenti hanno obiettivi indipendenti tra loro; in questo caso gli agenti di default non collaborano, non condividono informazioni e non sono disposti a "sacrificarsi" per il raggiungimento di un obiettivo non loro. La trattativa cooperativa (cooperative negotiation), è invece usata nelle situazioni in cui gli agenti hanno un obiettivo o un'operazione comune da raggiungere o da eseguire.

2.2.4.3 La mobilità

Se ci si basa sulle definizioni standard che sono state date in letteratura, un agente mobile ha tutte le caratteristiche di un agente normale (cioè è autonomo, reattivo, proattivo e sociale), ma in più è anche capace di muoversi, ovvero può spostarsi tra i vari host al fine di eseguire determinati compiti.

Dal punto di vista dei sistemi distribuiti, un agente mobile è un programma con una sua identià che può spostare il suo codice, i suoi dati e il suo stato tra gli host presenti nella rete. Per ottenere questo, gli agenti mobili sono in grado di sospendere la propria esecuzione in qualsiasi istante e di riprenderla non appena hanno terminato di spostarsi nella nuova posizione.

Un agente mobile è formato da tre parti: codice, stato e dati. Il codice è quella parte dell'agente che viene eseguita quando migra in una piattaforma. Lo stato sono le informazioni sull'esecuzione dell'agente, incluso il program counter e l'execution stack. I dati invece sono le variabili usate dall'agente durante la sua esecuzione.

Nei sistemi che presentano agenti mobili, possono esserci due tipi diversi di migrazione: la migrazione forte (strong migration) e la migrazione debole (weak migration).

La migrazione forte è più complessa e rappresenta il caso in cui l'esecuzione dell'agente viene congelata, poi avviene la migrazione, ed infine l'esecuzione riparte dall'istruzione successiva a quella di dove si era fermata. Questa tecnica richiede la memorizzazione e la protezione dello stato dell'agente durante il processo di migrazione. La realizzazione di questo tipo di migrazione può essere molto complesso perché richiede l'accesso a parametri interni dell'esecuzione dell'agente, che generalmente sono disponibili solo al sistema operativo, e quindi dipendenti dalla particolare architettura.

Invece, la migrazione debole, non invia anche lo stato dell'agente e quindi risulta essere più semplice, perché l'esecuzione di quest'ultimo riprende dall'inizio ogni volta. Questo tipo di migrazione richiede che l'agente sia implementato come una macchina a stati finiti nel caso in cui si voglia mantenere anche lo stato.

Nella piattaforma JADE che è stata utilizzata in questo lavoro di tesi (e che sarà presentata nella sezione 4.1), la mobilità degli agenti è controllata

attraverso il semplice metodo `doMove()` che fa parte della classe `Agent`: `void doMove(Location destination)`. Il parametro `destination` deve essere un oggetto che implementa l'interfaccia `Location` e in JADE ci sono due classi che lo fanno, ovvero `ContainerID` e `PlatformID` (per maggiori informazioni su queste due classi si veda la documentazione di JADE).

Una volta che il metodo è stato invocato, inizia il processo di spostamento dell'agente nel container di destinazione specificato. Lo stato dell'agente passa da ACTIVE a TRANSIT e in questo modo l'agente termina l'attività corrente e si sospende finché la piattaforma non lo sposta. Una volta arrivato a destinazione lo stato dell'agente ritorna ACTIVE e questo può ricominciare la sua esecuzione.

2.2.4.4 Le specifiche FIPA

La FIPA (Foundation for Intelligent Physical Agents) è nata nel 1996 come un'associazione internazionale no-profit con lo scopo di sviluppare un insieme di standard in relazione alla tecnologia degli agenti software. Durante gli anni sono state proposte diverse idee riguardo agli agenti, ma alla fine quelle ritenute di maggiore importanza sono quelle sulla comunicazione, la gestione e l'architettura degli agenti.

Per quanto riguarda la comunicazione, la FIPA ha proposto un suo modello, il FIPA-ACL (Agent Communication Language). Questo modello si basa sulla speech act theory, ovvero la teoria che dice che i messaggi rappresentano azioni o atti comunicativi (noti anche come speech act o performative). Un semplice esempio potrebbe essere il messaggio "Il mio nome è Mario", che quando usato dà al ricevente una qualche informazione. Il FIPA-ACL mette a disposizione 22 communicative act, tra cui troviamo *inform*, *request*, *agree*, *not understood* e *refuse*. Questi danno il senso della maggior parte delle forme base di comunicazione. Negli standard FIPA è scritto che un agente è completamente compatibile con le specifiche FIPA se è in grado di ricevere qualsiasi messaggio che rappresenta un communicative act nel formato FIPA-ACL e rispondere almeno con un messaggio di *not-understood* se il messaggio ricevuto non può essere elaborato.

Sulla base di questi communicative act, la FIPA ha definito anche un insieme di protocolli di interazione, ognuno dei quali consiste su una sequenza di communicative act, al fine di coordinare le azioni che richiedono l'invio di più messaggi, come quelle viste nel paragrafo 2.2.4.2.

Oltre alla comunicazione, il secondo fondamentale aspetto di un sistema ad agenti che viene affrontato dalle specifiche FIPA è la gestione degli agenti: un framework all'interno del quale gli agenti compatibili con le specifiche FIPA possono esistere, funzionare ed essere gestiti. Esso definisce il logical reference model per la creazione, registrazione, localizzazione, comunicazione, migrazione e per le operazioni degli agenti. In Figura 2.2 (PRESA DA

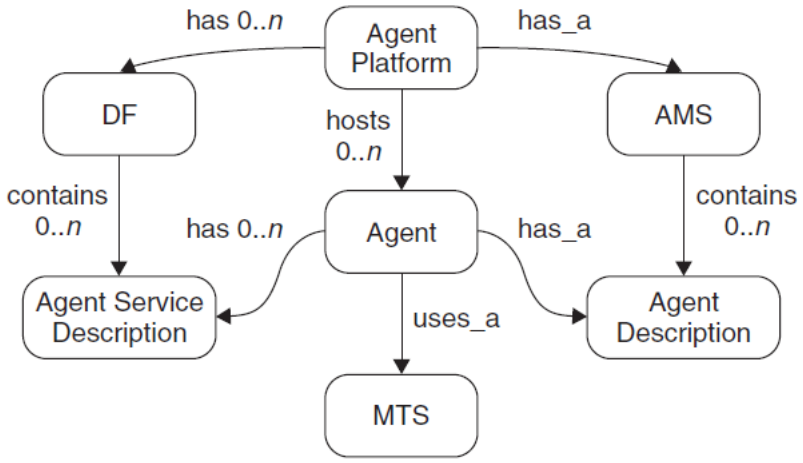

Figura 2.2: L'architettura proposta dalla FIPA

Developing Multi-Agent Systems with JADE) sono rappresentate tutte le componenti del management reference model:

- *Agent Platform (AP)*: questa è l'infrastruttura fisica nella quale gli agenti sono eseguiti, ovvero computer, sistemi operativi, componenti per la gestione degli agenti FIPA, gli agenti stessi e ogni altro tipo di supporto software. La progettazione interna dell'AP è lasciata allo sviluppatore, e non fa parte delle specifiche FIPA;

- *Agent*: questo è un processo che vive nell'AP e normalmente offre uno o più servizi che vengono resi disponibili agli altri agenti. L'implementazione degli agenti è lasciata allo sviluppatore e l'unica cosa che FIPA ha specificato è la struttura dei messaggi scambiati tra di essi. Ogni agente ha una propria nozione di identità descritta dal FIPA Agent Identifier (AID), che ne associa un'etichetta che permette di distinguere l'agente da tutti gli altri. Inoltre, una agente si può registrare con più indirizzi di trasporto ai quali può essere contattato;

- *Directory Facilitator (DF)*: il DF è un componente opzionale dell'AP, che fornisce il servizio di pagine gialle agli altri agenti. Esso mantiene una lista completa degli agenti e può fornire le informazioni più aggiornate su di essi. Ogni agente che vuole pubblicizzare i propri servizi, deve prima registrarsi con il DF fornendo una propria descrizione. In qualsiasi istante l'agente può chiedere al DF di modificare la descrizione oppure di de-registrarsi. Un agente può anche usare il DF per cercare agenti la cui descrizione soddisfa certi criteri, anche se il DF non dà la certezza che i dati forniti siano corretti;

- *Agent Management System (AMS)*: questo è un componente obbligatorio di un AP ed è responsabile della gestione delle operazioni che vi avvengono, come la creazione e l'eliminazione di agenti e il controllo della migrazione degli agenti in entrata e in uscita dall'AP. Ogni agente si deve registrare con l'AMS in modo da ottenere un AID, il quale viene mantenuto anche dall'AMS al fine di sapere quali agenti sono presenti nell'AP e il loro stato. La vita di un agente all'interno di un AP termina con la cancellazione dall'AMS, dopo la quale, l'AID diventa di nuovo disponibile per essere assegnato ad un altro agente. Inoltre, la descrizione degli agenti può essere cercata attraverso l'AMS, il quale mantiene anche la descrizione dell'AP. L'AMS può richiedere ad un agente di eseguire una specifica azione, come ad esempio terminare la sua esecuzione, ed ha l'autorità per costringerlo a farla nel caso in cui questa venga ignorata. In un AP può esistere un solo AMS, anche se l'AP è distribuito su più host;

- *Message Transport Service (MTS)*: questo è il servizio fornito dall'AP per il trasporto dei messaggi FIPA-ACL tra gli agenti che fanno parte di quel AP e anche con quelli di un altro AP.

Infine, oltre alla comunicazione e alla gestione degli agenti c'è anche l'architettura, che tra il 2000 e il 2002 è stata creata con il nome di FIPA Abstract Architecture, al fine di non far diventare un problema le continue modifiche alle specifiche fondamentali dell'implementazione della piattaforma. Questo è stato reso possibile facendo un'astrazione degli aspetti chiave dei meccanismi critici, come il trasporto dei messaggi, racchiudendoli all'interno di un'unica specifica. L'obiettivo di questo approccio è permettere la creazione di un sistema che si integra perfettamente all'interno del suo ambiente di calcolo mentre può continuare ad interoperare con il sistema ad agenti che è posizionato in un altro ambiente. Alcuni dei punti fondamentali specificati all'interno della FIPA Abstract Architecture sono:

- *agent messages*: i messaggi rappresentano la forma di comunicazione fondamentale tra gli agenti. La struttura di un messaggio è formata da un insieme di valori chiave scritti nel FIPA-ACL; il contenuto del messaggio si può esprimere attraverso un content language, come ad esempio FIPA-SL o FIPA-KIF e le espressioni e i termini usati possono essere legati a una particolare ontologia;

- *message transport service*: questo servizio serve per l'invio e la ricezione di messaggi di trasporto tra gli agenti. Questo è considerato un elemento obbligatorio di un sistema ad agenti FIPA;

- *agent directory service*: questo servizio è come un repository condiviso per le informazioni, all'interno del quale gli agenti possono pubblicare

le entry delle loro agent directory e possono cercare quelle degli altri. Anche questo componente è obbligatorio in un sistema ad agenti FIPA;

- *service directory service*: questo servizio è come un repository nel quale gli agenti e servizi possono trovare altri servizi. I servizi includono ad esempio quello per il trasporto dei messaggi, l'agent directory, il servizio di gateway e di buffering dei messaggi. Anche questo componente è obbligatorio in un sistema ad agenti FIPA.

Gli altri componenti che non sono stati nominati tra quelli precedenti, assieme agli *agent messages*, sono da ritenersi come elementi opzionali dell'architettura.

Capitolo 3

La progettazione

Questo capitolo tratta la progettazione dell'intero sistema, partendo dall'idea iniziale (sezione 3.1), passando poi per la prima analisi che è stata fatta delle procedure aeroportuali (sezione 3.2), fino ad arrivare all'architettura hardware e quella ad agenti del sistema (sezione 3.3).

3.1 L'idea iniziale

In questa sezione viene discusso l'ambito in cui si pone questo lavoro di tesi e che cosa si è voluto andare a sviluppare. In particolare vengono evidenziate le varie problematiche che devono affrontare i passeggeri in aeroporto prima di imbarcarsi su un volo o all'arrivo a destinazione; il sistema che sarà sviluppato avrà come obiettivo proprio quello di cercare di risolvere questo tipo di problemi, agevolando e velocizzando le varie procedure che i passeggeri devono affrontare. Oltre al sistema sarà anche sviluppata un'applicazione (ne verranno realizzate due versioni), che sarà utilizzata dagli utenti finali.

3.1.1 L'ambito di lavoro e che cosa si è voluto realizzare

Al giorno d'oggi, le persone che viaggiano in aereo sono sempre più numerose. Oltre ai viaggiatori abituali, che si spostano soprattutto per ragioni di lavoro da un paese all'altro, abbiamo anche una grossa fetta di persone che si sposta per motivi turistici o comunque diversi dal lavoro. Se è quindi vero che le persone abituate a viaggiare in aereo sanno bene quali sono le procedure da seguire all'interno degli aeroporti, è vero anche che ci sono persone che viaggiano meno e che quindi possono fare più fatica a capire tutte le procedure che bisogna seguire. La cosa poi non viene resa più semplice dal fatto che queste procedure non sono sempre chiare e semplici da seguire, soprattutto per una persona che entra per la prima volta in un aeroporto: chiunque infatti si ricordi la prima volta che vi è entrato, conoscerà la confusione che si prova nel cercare di capire cosa si deve fare e dove ci si deve

Figura 3.1: Persone in attesa dentro un aeroporto

recare. Oltre a questo, si aggiunge il fatto che dopo gli avvenimenti dell'11 settembre, le misure di sicurezza si sono intensificate e questo comporta un tempo maggiore per il controllo di ogni singola persona; c'è da dire che si potrebbero velocizzare i tempi se tutti i passeggeri sapessero cosa fare una volta arrivati ai controlli di sicurezza, mi riferisco ad esempio al fatto di togliere tutti gli oggetti metallici che si hanno addosso, onde evitare di dover ripassare più volte sotto il metal detector, oppure di non portare liquidi oltre la quantità consentita. Questi appena detti sono piccoli accorgimenti che se non rispettati fanno perdere del tempo non soltanto a noi stessi, ma anche a tutte quelle persone che come noi si devono imbarcare; basta pensare al numero di passeggeri che ogni giorno passano all'interno di un aeroporto per capire subito quanto sia importante il cercare di tagliare i vari tempi di attesa che inevitabilmente si creano alla varie file.

E' esattamente all'interno di questo contesto che si pone il progetto di questa tesi. L'obiettivo è quello di fornire uno strumento di supporto a tutte quelle persone che si trovano all'interno di un aeroporto, guidandole attraverso tutte quelle operazioni che necessariamente bisogna fare prima di imbarcarsi e una volta scesi a destinazione. L'applicazione che è stata sviluppata non rappresenta quindi una semplice guida all'aeroporto, ma è uno strumento che, basandosi sulla situazione specifica della persona che la utilizza (questo perché ad esempio non tutti i passeggeri devono compiere le stesse operazioni prima della partenza), la guida passo passo, fornendogli allo stesso tempo suggerimenti sulle varie operazioni che deve fare e indicazioni di dove dirigersi per compierle. In questo modo è possibile facilitare l'esperienza all'interno dell'aereoporto delle persone, sia che esse siano dei viaggiatori abituali oppure delle persone che prendono l'aereo per la prima volta.

Quello che verrà sviluppato è quindi un sistema completo, che dovrà essere presente all'interno dell'aeroporto, per il supporto ai passeggeri; questo sistema implementerà tutti i meccanismi necessari al reperimento e alla diffusione delle informazioni utili allo svolgimento delle varie operazioni. Verrà inoltre sviluppata un'applicazione che sfrutta questo sistema e che rappresenta il mezzo attraverso il quale l'utente finale si interfaccia ad esso. Vista la crescente diffusione di telefoni smartphone, si è pensato di sviluppare due versioni dell'applicazione: una versione per computer e una per un particolare tipo di telefoni di ultima generazione, ovvero per quelli che hanno un sistema operativo Android. Una volta sviluppata quindi l'applicazione per i computer, verrà fatto il porting su piattaforma Android, al fine di renderne possibile l'utilizzo su una categoria di telefoni in grande diffusione.

3.2 L'analisi delle procedure aeroportuali

La presenza delle procedure aeroportuali e la difficoltà che si può avere nel seguirle, rappresenta il motivo principale per cui si è voluto creare il sistema e l'applicazione oggetto di questo lavoro di tesi. Come è stato già detto in precedenza, una procedura è quella sequenza di operazioni che un passeggero deve compiere all'interno dell'aeroporto, prima di potersi imbarcare sull'aereo e dopo che è sceso a destinazione. Essa è formata da un certo numero di operazioni che vanno svolte in luoghi diversi all'interno dell'area aeroportuale ed entro certi tempi (altrimenti si rischia di perdere l'aereo). E' stato quindi fondamentale capire quali sono in genere le procedure utilizzate e tracciarne uno schema di massima.

3.2.1 Una prima analisi

La prima cosa fatta è stata quella di informarsi sulle procedure in vigore all'interno degli aeroporti; per procedure si intendono tutte quelle operazioni che una persona deve compiere prima di imbarcarsi a bordo di un aereo. Ci si è resi subito conto che non tutti passeggeri devono compiere le stesse procedure: infatti c'è chi magari ha prenotato il volo via Internet, si è stampato il biglietto a casa e ha solo il bagaglio a mano e quindi non serve che faccia la fila per il check-in, oppure c'è chi ha prenotato in un'agenzia di viaggi e che quindi deve recarsi nell'area tour operator dell'aeroporto per ritirare i vari documenti per il viaggio. Quindi una cosa essenziale per il tipo di applicazione che si voleva creare, era la capacità di adattarsi alle diverse esigenze che ogni persona aveva.

Per ottenere una cosa del genere, si suppone che al momento della prenotazione del viaggio in agenzia, oppure dopo l'acquisto del biglietto via internet o all'aeroporto, venga data all'utente una login e una password che permettano di accedere all'applicazione e di avere a disposizione tutte le

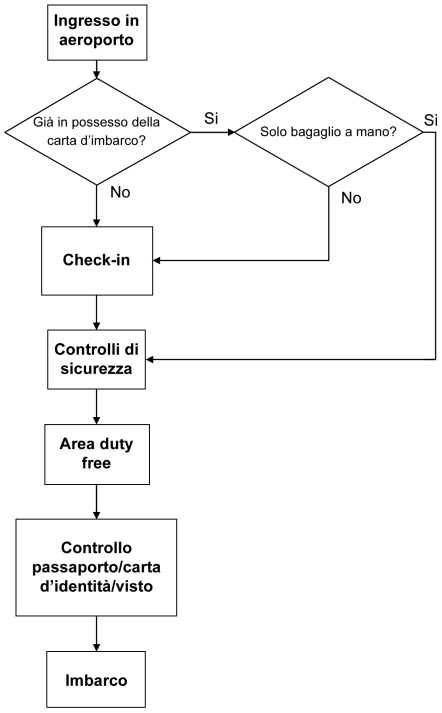

Figura 3.2: Procedura di Partenza

istruzioni personalizzate. Infatti il sistema riconoscerà la login con cui l'utente accede e da questa recupererà la procedura corretta che questo deve seguire e fornirà tutte le informazioni utili di ogni singola operazione man mano che ce ne sarà bisogno.

Ritornando all'analisi iniziale delle procedure che è stata fatta, si riporta a titolo di esempio la procedura di Partenza, illustrata nella Figura 3.2, la quale consiste nelle seguenti fasi: una volta entrati nell'aeroporto, la prima operazione da fare è il check-in, durante la quale viene pesato il bagaglio da stiva per vedere se rispetta i limiti di peso per poi essere imbarcato; oltre a questo viene data anche la carta d'imbarco, con indicati i posti che sono stati assegnati. Questa operazione può essere saltata nel caso in cui l'utente abbia fatto la prenotazione via internet, stampato il biglietto a casa e non abbia con sè il bagaglio da stiva ma solo quello a mano. Fatto ciò si passa ai controlli di sicurezza, dove si deve passare sotto un metal detector e il bagaglio a mano viene controllato con uno scanner, al fine di individuare oggetti pericolosi o che non possano essere portati a bordo dell'aereo; questa è un'operazione che chiaramente per motivi di sicurezza tutti devono fare. Dopo i controlli si arriva nell'area dell'aeroporto chiamata duty free, questo perché di solito vi sono presenti negozi che non applicano le imposte sui prodotti e quindi questi sono venduti ad un prezzo minore rispetto al solito. Oltre ai negozi sono presenti anche bar, ristornati e servizi vari, questo

perché normalmente si può passare qualche ora all'interno di quest'area, in attesa dell'apertura dei gate di imbarco. Prima dell'imbarco bisogna passare per i controlli dei documenti, dove vengono appunto controllati documenti come il passaporto, la carta d'identità o il visto a seconda della destinazione che si deve raggiungere. Fatto ciò si attende l'apertura del gate, dove viene controllato la carta d'imbarco e infine si esce dall'aeroporto o per entrare direttamente nell'aereo oppure per montare su una piccolo bus che trasporterà i passeggeri fino alla pista dove li attende l'aereo.

3.2.2 Le singole operazioni

Una volta delineate a grandi linee queste procedure si è iniziato ad andare nello specifico: quindi si è analizzata ogni singola operazione di ogni singola procedura. Per ognuna di queste operazioni si sono decise le informazioni che potevano essere utili all'utente nel momento in cui si apprestava a farle come ad esempio, quando la persona si deve recare al banco del check-in, sarebbe utile sapere il numero del banco, oppure quando entra in aeroporto sarebbero utili le informazioni sul volo come l'orario e se questo è in ritardo o meno. Oltre alle informazioni considerate importanti che vengono date a priori, sono stati pensati anche dei suggerimenti che sarà l'utente stesso a scegliere se visualizzare o meno; esempi di suggerimenti possono essere quelli che dicono cosa è vietato portare oltre i controlli di sicurezza oppure quelli che ricordano di controllare che i documenti siano in regola per il viaggio che si deve fare. In questo modo, man mano che l'utente avanza nella procedura, ha tutte le informazioni importanti sotto controllo e può scegliere se visualizzare o meno i suggerimenti, i quali forniscono comunque delle informazioni utili, soprattutto se non si è molto pratici degli aeroporti.

Tra le informazioni a disposizione all'utente si è voluto che ce ne fosse anche una che desse l'idea di quanto ci fosse da aspettare alla fila del check-in, che normalmente è quella più lunga. Inizialmente si è pensato di dare una stima temporale dell'attesa, ma ci si è resi conto che questo non era possibile per diversi motivi: come prima cosa, se si fosse voluta basare questa stima sulla posizione delle persone, ci sarebbe stato bisogno di localizzare le persone in attesa ai check-in in maniera precisa, in quanto di solito le file ai diversi banchi sono una vicina all'altra, ma una tale precisione richiede un'apposita infrastruttura che di certo non può essere data per scontata e quindi questa idea è stata abbandonata. Si è poi pensato di basarsi sul numero di persone che risultavano in attesa sulla base delle informazioni da loro fornite all'applicazione; purtroppo anche questa stima non sarebbe stata accettabile in quanto è troppo ottimistico pensare che quasi tutte le persone abbiano e utilizzino l'applicazione, inoltre anche se così fosse, non è detto che l'utilizzino in maniera corretta e quindi i dati forniti potrebbero non essere utilizzabili ai fini di una stima. Utilizzando uno dei metodi appena descritti si correva il rischio di dare una stima troppo ottimistica oppure

troppo pessimistica e a quel punto non aveva neanche più senso darla una stima, se questa poteva differire di molto dalla realtà. Per questi motivi si è pensato di fare come segue: invece di dare una stima temporale sull'attesa, viene dato il numero di passeggeri che devono ancora fare il check-in per il volo che si deve prendere, in questo modo ci si può regolare se conviene mettersi in fila oppure se c'è tempo per fare qualcos'altro; il numero delle persone che mancano viene ricavato dal database dell'aeroporto, nel quale è ragionevole pensare siano presenti tutte le informazioni riguardati le persone che devono fare il check-in per un determinato volo.

Tra le varie informazioni utili che vengono date ci sono quelle riguardanti il volo, come l'orario previsto per la partenza, la destinazione, il terminal, il gate e lo stato (cioè se ad esempio è in orario, in ritardo o cancellato). Oltre all'informazione sul numero di persone che devono ancora fare il check-in per il proprio volo, viene anche detto qual'è il numero del banco e verranno date in qualche modo le informazioni riguardanti la sua posizione. Queste ultime verranno date per tutte le singole operazioni, al fine di rendere più facile muoversi all'interno dell'aeroporto (è bene specificare però che non è stato creato un navigatore vero e proprio, ma un meccanismo che notifica quando ci si trova nelle vicinanze del punto corretto).

Per quanto riguarda invece i suggerimenti, questi non sono da ritenersi fondamentali per svolgere le varie operazioni ma permettono di accelerare i tempi, soprattutto se non si è molto pratici di aeroporti. Ad esempio, se ci si trova in fila al check-in, può venire suggerito di preparare già tutti i documenti personali e che se si è in gruppo basta che uno soltanto faccia la fila per tutti, mentre se si stanno ritirando i bagagli all'arrivo, si suggerisce di stare attenti di non scambiare i propri con quelli di qualcun altro e nel caso in cui questi siano stati smarriti, di rivolgersi all'ufficio apposito. Nella versione dell'applicazione per il computer, i suggerimenti saranno comunque sempre visibili, mentre nella versione per il telefono sarà l'utente a scegliere se visualizzarli o meno.

3.3 L'architettura

In questa sezione si parlerà di come è strutturata l'architettura hardware e software del sistema. In particolare si inizierà illustrando la suddivisione in aree dell'aeroporto, al fine di suddividere il carico di lavoro che peserà sui vari componenti, per poi arrivare a spiegare le varie parti che compongono l'architettura hardware e quella ad agenti, motivando anche la scelta di questo particolare paradigma per la realizzazione del sistema.

3.3.1 La suddivisione in aree

Visto il grande numero di persone che ogni giorno passano all'interno di un aeroporto, un fattore fondamentale è la scalabilità dell'intero sistema

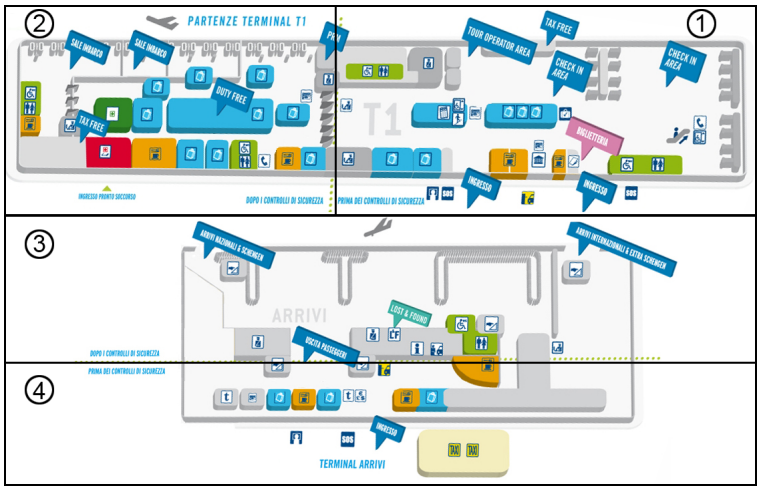

Figura 3.3: Suddivisone dell'aeroporto in aree: 1-area pubblica delle partenze, 2-area oltre i controlli di sicurezza delle partenze, 3-area prima dei controlli doganali degli arrivi, 4-area pubblica degli arrivi.

su cui si basa l'applicazione. Infatti il numero di dispositivi che si collegano/scollegano e che inviano/ricevono dati sarà molto elevato ed è necessario gestire questa grande mole di connessioni contemporanee, senza che le performance dell'intero sistema calino troppo o che peggio ancora questo vada in crash. Con l'obiettivo di evitare situazioni di questo genere è stata pensata e progettata l'intera architettura del sistema.

Come prima cosa si è deciso di dividere l'aeroporto in quattro aree distinte, vedi Figura 3.3, ed assegnare ad ognuna di esse un responsabile che ne gestirà in parte il traffico; questa soluzione risulta migliore rispetto ad una dove c'è un unico responsabile per l'intero aeroporto, il quale sarebbe certamente sovraccaricato dalle richieste varie che gli arriverebbero e potrebbe perciò avere un degrado delle performance non accettabile. Le quattro aree in cui è stato diviso l'aeroporto sono:

- *area pubblica delle partenze*, ovvero tutta quella parte di aeroporto accessibile a qualunque persona, sia che questa sia in partenza o meno; in quest'area troviamo i banchi dei tour operator, i check-in, i controlli di sicurezza, i banchi informazione e tutti quei servizi che un aeroporto può mettere a disposizione, come ristoranti, bar, toilette e molti altri;

- *area oltre i controlli di sicurezza delle partenze*, cioè la parte che si raggiunge non appena si oltrepassano i controlli di sicurezza; in quest'area troviamo la zona duty free dove è possibile fare acquisti nei vari negozi che di solito sono presenti oppure mangiare nei ristoranti e bar, sono poi presenti i controlli dei documenti, i gate d'imbarco e servizi

vari. Questa è l'area dove di solito si passa più tempo, in quanto è qui che si aspetta di imbarcarsi sul proprio volo;

- *area prima dei controlli doganali degli arrivi*, cioè quella parte che si raggiunge non appena si scende dall'aereo, fino ai controlli doganali; in quest'area sono presenti i nastri trasportatori che fanno passare i bagagli dei voli che sono appena arrivati, i banchi informazione nel caso in cui il bagaglio sia stato smarrito, i controlli dei documenti ed infine i controlli doganali dove sono presenti degli agenti che spesso fanno dei controlli a campione tra i passeggeri del volo;

- *area pubblica degli arrivi*, ovvero quella parte che va da dopo i controlli doganali fino all'uscita dell'aeroporto.

Le aree in questione sono abbastanza estese e oltre a questo c'è il problema delle varie barriere fisiche, come i muri, a far si che un singolo access point wireless (AP) non sia sufficiente a coprirle in maniera soddisfacente. Per questo motivo, ogni singola area è stata suddivisa in settori, il cui numero dipende dalla grandezza dell'area stessa, e in ogni settore è presente un AP al quale si collegheranno i vari dispositivi; grazie a questa suddivisione sarà anche possibile capire con una certa precisione in quale settore dell'aeroporto un dispositivo è presente, semplicemente andando a vedere a quale AP è collegato.

3.3.2 L'architettura hardware

Passiamo adesso ad analizzare ogni singolo componente dell'architettura hardware riportata in Figura 3.4 iniziando dagli host, ovvero i dispositivi mobili nel quale l'applicazione sarà installata e che permetterà di interagire con l'intero sistema. I dispositivi potranno essere personal computer, smartphone e tablet (su quest'ultimi l'implementazione non è stata fatta, in quanto non se ne aveva la disponibilità); tutti si collegheranno al sistema attraverso una connessione wireless.

Passando poi alle aree, si è deciso che per ognuna di esse sarà presente un server dedicato, che conterrà tutte quelle componenti di sistema che riguardano quella particolare area; in questo modo riesco a decentralizzare i dati che sono necessari in una certa area e, dopo la fase iniziale di reperimento della informazioni dall'intero sistema, le richieste verranno risolte al livello di area e non arriveranno a quelle componenti più in alto nella gerarchia che verranno così alleggerite dal carico di lavoro. All'interno del server di area verrà mantenuta una lista degli hosts presenti in quella particolare area e, cosa importante, per ognuno di questi host sarà presente una componente che manterrà le informazioni del particolare dispositivo (informazioni come ad esempio il punto della procedura a cui è arrivato e il settore dove si trova) e che sarà responsabile del mantenimento di queste informazioni nel caso in

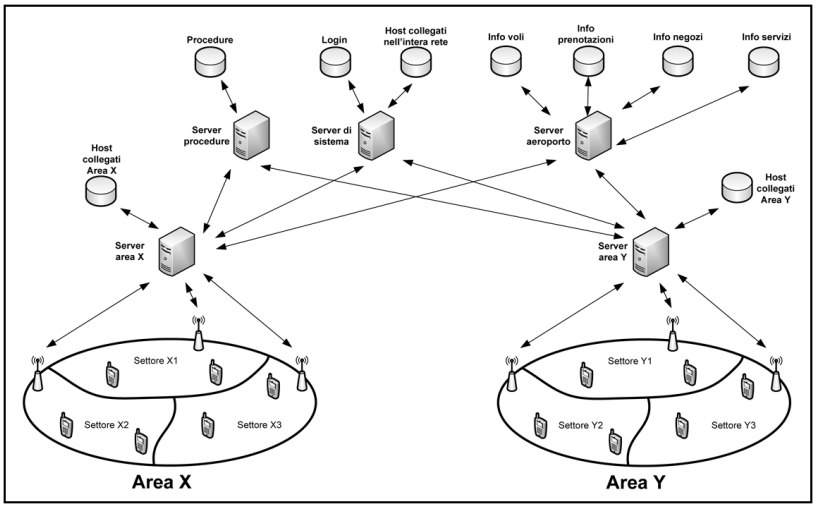

Figura 3.4: L'architettura hardware del sistema

cui l'host vada offline e poi ritorni online, in questo modo i dati non verranno persi e saranno nuovamente disponibili.

Salendo con la gerarchia troviamo il server delle procedure, quello di sistema e infine quello che farà da interfaccia per accedere ai dati veri e propri dell'aeroporto.

Il server delle procedure avrà il compito di rispondere alle richieste di download delle procedure, ovvero tutte quelle operazioni che una persona deve compiere all'interno dell'aeroporto; a questo scopo, verranno mantenute al suo interno tutte le possibili procedure che potrebbero essere richieste.

Il server di sistema invece è quello che contiene le informazioni generali riguardanti tutti gli host presenti nel sistema; quando un host si registra, verrà creata una nuova entry nell'apposita tabella con le informazioni riguardanti il nuovo dispositivo; oltre a questi dati, il server registrerà anche tutte le varie componenti che compongono l'intero sistema. Si può dire che questo server sarà l'entry point di tutte le componenti del sistema.

Infine abbiamo il server che farà da interfaccia tra il sistema e i dati dell'aeroporto, quindi per accedere ad esempio alle informazioni sulle prenotazioni e sui voli bisognerà utilizzare questo server, che con i dovuti meccanismi, andrà a recuperare le informazioni necessarie dai server dell'aeroporto. Questo è necessario perché chiaramente i dati aeroportuali sono di fondamentale importanza per il corretto funzionamento dell'aeroporto, prima che del sistema, e quindi bisogna accedervi attraverso procedure controllate ritenute sicure.

3.3.3 Il modello ad agenti

Uno dei motivi principali che ha portato alla scelta dell'utilizzo del modello ad agenti è stato sicuramente la facilità con cui si ottiene la scalabilità in questo tipo di approccio. E' infatti stato chiaro fin dalle prime fasi di progettazione, che un sistema di questo tipo deve essere in grado di scalare molto bene, in quanto il numero di dispositivi che si collegheranno ad esso è potenzialmente molto elevato. Il modello ad agenti sfrutta l'architettura P2P, di cui abbiamo già parlato nella sezione 2.2.2, la quale offre una scalabilità intrinseca al modello stesso, visto che in una tale rete tutti i componenti collaborano assieme in maniera paritaria, fornendo in questo modo una maggiore potenza di calcolo rispetto all'utilizzo di un singolo nodo di calcolo.

Oltre al motivo della scalabilità, il modello ad agenti ha come altro punto di forza la mobilità degli agenti stessi. Anche se non tutti gli agenti all'interno del sistema saranno mobili, ce ne saranno alcuni che lo sono ed è quindi necessario che questi siano in grado di migrare all'interno del sistema al fine di portare a termine determinati compiti. In questo caso il modello ad agenti rappresenta la soluzione più facile e più indicata per risolvere questo tipo di problemi.

Tra le varie caratteristiche di un agente ci sono anche la reattività (capacità di percepire l'ambiente che lo circonda e reagire di conseguenza) e la proattività (capacità di prendere l'iniziativa autonomamente): la prima si rivelerà utile in questo progetto, in quanto gli agenti che sono presenti nei dispositivi mobili dovranno essere in grado di recuperare tutte le varie informazioni presenti nell'ambiente e nel sistema per portare a termine il proprio obiettivo, che sarà quello di assistere in maniera corretta l'utente nelle varie operazioni. Oltre a questo gli agenti dovranno anche essere in grado di prendere decisioni in modo autonomo, senza basarsi su quello che fanno gli altri agenti o dal particolare ambiente dove si trovano.

3.3.4 L'architettura ad agenti

Dopo aver visto e analizzato l'architettura hardware e spiegato i motivi della scelta del paradigma ad agenti, passiamo ad analizzare l'architettura relativa agli agenti, andando a vedere quali sono i collegamenti tra gli agenti principali del sistema (vengono esclusi in particolare tutti gli agenti che svolgono una funzione di simulazione di un qualche aspetto dell'aeroporto e gli agenti che svolgono operazioni secondarie).

Cominciamo quindi andando ad analizzare gli agenti che offrono funzionalità di interesse generale per l'intero sistema (l'architettura ad agenti è riportata in Figura 3.5): abbiamo gli agenti AirportSystemManager, ProcedureManager e AirportInformationManager. L'AirportSystemManager rappresenta in un certo senso l'entry point di tutti gli altri agenti, i quali do-

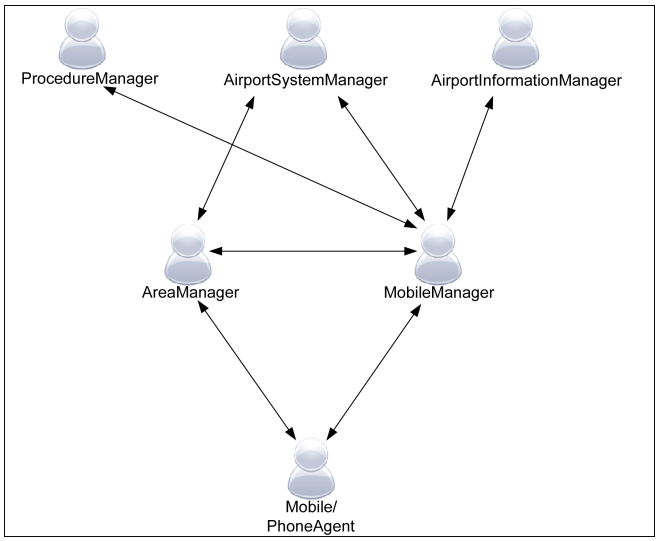

Figura 3.5: L'architettura ad agenti del sistema

vranno contattarlo non appena prendono parte al sistema; lo scopo quindi dell'AirportSystemManager è quello di mantenere delle informazioni generali riguardanti gli agenti che compongono il sistema. Un altro agente molto importante è il ProcedureManager il cui scopo è quello di mantenere le procedure che i vari passeggeri dovranno seguire e di inviare queste procedure agli agenti MobileManager che ne faranno richiesta. Infine abbiamo l'AirportInformationManager il cui ruolo è quello di interfaccia tra le richieste degli agenti MobileManager e i dati veri e propri dell'aeroporto; in particolare, dato che questi ultimi sono dati importanti per il corretto funzionamento dell'aeroporto, prima che del sistema stesso, è importante che vi si acceda in maniera corretta e con procedure controllate, al fine di non creare errori all'interno del database aeroportuale.

Parliamo ora di due agenti che hanno funzioni più specifiche, ovvero l'AreaManager e il MobileManager. A capo di ogni area sarà presente un agente AreaManager, il quale ha il compito di gestire e mantenere le informazioni riguardanti gli agenti di una determinata area. Il MobileManager è un agente che viene creato nel momento in cui un utente si collega al sistema, e viene associato all'agente che è presente sul dispositivo (Mobile se il dispositivo da cui si collega l'utente è un computer oppure PhoneAgent nel caso in cui sia un dispositivo con sistema operativo Android); il MobileManager si preoccuperà di recuperare tutte le informazioni riguardanti la procedura e di inviare le singole operazioni all'agente sul dispositivo, non appena l'utente ne farà richiesta. Il suo scopo in poche parole è quindi quello di fare da interfaccia tra l'agente presente nel dispositivo dell'utente e l'intero sistema; infatti, una volta terminata la fase iniziale di connessione al sistema da par-

te del dispositivo, durante la quale esso contatta l'AirportSystemManager e anche l'AreaManager, nelle fasi successive e per tutto il tempo in cui il dispositivo rimarrà collegato al sistema, questo comunicherà unicamente con il suo MobileManager associato. In questo modo l'agente sul dispositivo è più "leggero", in quanto molte operazioni vengono compiute dal MobileManager, e questo è un punto importante dato che i dispositivi mobili hanno le note problematiche relative alla durata della batteria, alla limitata capacità di calcolo e alla limitata memoria.

Infine abbiamo gli agenti Mobile e PhoneAgent, che sono in realtà due agenti che forniscono le stesse funzionalità, ma in modi diversi in quanto l'agente Mobile sarà presente nella versione per computer dell'applicazione, mentre il PhoneAgent in quella per sistema operativo Android. Lo scopo di entrambi questi agenti è quello di fare le richieste al sistema, in particolare al MobileManager associato, da parte dell'utente che interagirà con l'interfaccia dell'applicazione; una volta che questo agente riceve poi le risposte, farà in modo che vengano visualizzate all'utente, sempre attraverso l'applicazione.

Capitolo 4

Le piattaforme usate

Questo capitolo parla delle due piattaforme utilizzate per sviluppare il sistema. In particolare verrà descritta la piattaforma JADE (sezione 4.1), utilizzata per sviluppare il sistema multiagente e la piattaforma Android (sezione 4.2), per la quale è stata sviluppata una versione dell'applicazione che sfrutta il sistema e che verrà utilizzata dagli utenti finali.

4.1 Jade

Questa sezione parla della piattaforma utilizzata per sviluppare il sistema ad agenti oggetto di questo lavoro di tesi, ovvero della piattaforma JADE. Si daranno dei cenni storici per capire come, quando e grazie a chi è nata; verranno inoltre spiegate le varie funzionalità che JADE mette a disposizione degli sviluppatori e che ne fanno un valido strumento per sviluppare sistemi ad agenti.

4.1.1 Cenni storici

I primi sviluppi di quella che poi sarebbe diventata la piattaforma JADE, iniziarono nel 1998 da parte di Telecom Italia (all'epoca chiamata CSELT), con lo scopo di validare le prime specifiche FIPA.

Nonostante all'inizio non si fosse certi di raggiungere l'obiettivo di sviluppare una piattaforma, grazie al supporto di un parziale finanziamento da parte della Commissione Europea, si decise di andare oltre la semplice validazione delle specifiche FIPA, sviluppando quindi una vera e propria piattaforma middleware. L'obiettivo era quello di fornire dei servizi accessibili e fruibili dagli sviluppatori di applicazioni, sia che essi fossero esperti o

neofiti con poca o nessuna conoscenza delle specifiche FIPA. Quindi gli sforzi vennero fatti nella direzione di rendere semplici e usabili le API software.

JADE è diventata open source nel 2000 ed è stata distribuita da Telecom Italia sotto la licenza LGPL (Library Gnu Public Licence). Questa licenza assicura tutti i diritti di base per facilitare l'uso del software incluso nei prodotti commerciali: il diritto a fare e distribuire delle copie del software, il diritto ad avere accesso al codice sorgente e il diritto a modificare questo codice sorgente al fine di apportare dei miglioramenti. Differenza importante fra la licenza GPL (General Public License) e LGPL è che quest'ultima non impone alcuna restrizione sul software che utilizza JADE, permettendo quindi che anche software proprietario venga utilizzato assieme a quello coperto da licenza LGPL. Dall'altra parte però, la licenza LGPL richiede che ogni lavoro derivato o basato su JADE, sia reso disponibile alla comunità di sviluppatori con la stessa licenza.

In modo da facilitare il coinvolgimento delle aziende, nel maggio 2003 Telecom Italia Lab e Motorola Inc. definirono un accordo di collaborazione e formarono il JADE Governing Board, un'organizzazione no-profit di aziende impegnate a contribuire allo sviluppo e alla promozione di JADE. Questa organizzazione lasciava la libertà alle aziende di entrarne a farne parte, come anche di uscirne, a seconda delle necessità. Nel 2007 ne facevano parte Telecom Italia, Motorola, France Telecom R&D, Whitestein Technologies AG e Profactor GmbH.

All'inizio, quando JADE venne resa pubblica da parte di Telecom Italia, essa veniva usata quasi esclusivamente dalla comunità FIPA, ma al crescere delle sue funzioni ben oltre le specifiche FIPA, iniziò ad essere utilizzata anche dalla comunità degli sviluppatori. E' interessante notare che grazie a JADE c'è stata un'ampia diffusione anche delle specifiche FIPA, dando un insieme di astrazioni software e di strumenti che nascondevano le specifiche stesse; in poche parole, gli sviluppatori potevano implementare programmi in accordo con le specifiche, senza però doverle conoscere.

Una delle prime estensioni di JADE arrivò da LEAP, un progetto parzialmente finanziato dalla Commissione Europea che contribuì in maniera significativa, tra il 2000 e il 2002, al porting di JADE su ambiente Java Micro Edition e Wireless Network.

La versione di JADE utilizzata in questo lavoro di tesi è la 4.1 rilasciata il 13 luglio 2011.

4.1.2 Le funzionalità offerte

JADE è una piattaforma software che fornisce delle funzionalità di base al livello middleware, le quali sono indipendenti dalla specifica applicazione e semplificano la realizzazione di applicazioni distribuite che sfruttano gli agenti software. Un grosso vantaggio è che JADE è stata implementata attraverso un linguaggio molto conosciuto come Java, fornendo in questo

modo delle semplici API da utilizzare. Le funzionalità che JADE mette a disposizione degli sviluppatori sono:

- un sistema completamente distribuito formato da agenti, ognuno dei quali è eseguito su un thread separato, potenzialmente su una macchina remota, ed in grado di comunicare con gli altri in maniera trasparente (ad esempio la piattaforma mette a disposizione delle API che fanno diventare trasparente l'infrastruttura per le comunicazioni fra agenti);

- completa conformità alle specifiche FIPA;

- trasporto efficiente dei messaggi asincroni attraverso delle API di localizzazione trasparenti all'utente. La piattaforma seleziona il mezzo di trasporto più efficiente e, quando possibile, evita la codifica/decodifica degli oggetti Java. Quando vengono attraversati i confini della piattaforma, i messaggi vengono automaticamente trasformati dalla rappresentazione Java interna di JADE a quella compatibile con la sintassi, codifica e protocolli di trasporto FIPA;

- implementazione del servizio di pagine bianche e di pagine gialle (white page and yellow page service). Il servizio di pagine bianche permette di rintracciare gli agenti della piattaforma fornendo gli indirizzi ai quali possono essere raggiunti, mentre il servizio di pagine gialle permette di trovare gli agenti che forniscono certi servizi. I sistemi federati possono essere implementati in modo da rappresentare domini e sotto domini come un grafo di directory federate;

- una gestione semplice ed efficace del life-cycle di un agente. Quando un agente viene creato gli viene assegnato in automatico un identificativo univoco e un indirizzo di trasporto che sono usati per registrare l'agente nel servizio di pagine bianche della piattaforma. Sono inoltre rese disponibili delle API e dei tool grafici per gestire i life-cycle degli agenti sia locali che remoti;

- supporto alla mobilità degli agenti. Sia il codice dell'agente e, sotto certe restrizioni, lo stato dell'agente possono migrare tra processi e computer. La migrazione è fatta in modo trasparente, tanto che gli agenti possono continuare a comunicare fra loro anche durante il processo di migrazione;

- un meccanismo di registrazione degli agenti che intendono ricevere notifiche quando accadono degli eventi nella piattaforma; possono essere eventi riguardanti il life-cycle degli agenti, oppure eventi come lo scambio di messaggi;

- un insieme di tool grafici a supporto degli sviluppatori per poter monitorare e debuggare le applicazioni. Ad esempio le conversazioni tra agenti possono essere viste ed emulate per via grafica, è possibile andare a vedere dei parametri interni degli agenti (come ad esempio la coda dei messaggi inviati e ricevuti) oppure fare un debug step-by-step dell'esecuzione dell'agente;
- supporto per le ontologie e i linguaggi usati (ontologies and content languages). Il controllo ontologico e la codifica del contenuto dei messaggi vengono fatti in automatico dalla piattaforma, mentre lo sviluppatore può scegliere il linguaggio e l'ontologia (ovvero il vocabolario) usati;
- una libreria di protocolli di interazione che modellizza i classici schemi di comunicazione al fine di raggiungere uno o più obiettivi;
- integrazione con varie tecnologie web-based come JSP, servlets, applets e la tecnologia web service;
- supporto per la piattaforma J2ME e l'ambiente wireless. JADE è disponibile per le piattaforme J2ME-CDC e J2ME-CLDC, grazie ad un insieme di API che coprono gli ambienti J2ME e J2SE;
- un'interfaccia in-process per il lancio/controllo di una piattaforma da un'applicazione esterna;
- un kernel che si può estendere in modo che gli sviluppatori possano aggiungere funzionalità alla piattaforma.

Queste sono le caratteristiche principali che fanno di JADE uno strumento completo e relativamente facile da utilizzare per lo sviluppo di sistemi basati sull'architettura ad agenti.

4.1.3 L'architettura

Una piattaforma JADE è composta da uno o più container di agenti, che possono essere distribuiti nella rete. Gli agenti vivono all'interno di questi container, i quali non sono altro che processi Java che forniscono il runtime di JADE e tutti i servizi di cui c'è bisogno per ospitare ed eseguire gli agenti stessi. C'è un container speciale, chiamato *main container*, che rappresenta il punto di avvio dell'intera piattaforma: esso infatti è il primo container che viene avviato e tutti quelli che vengono creati dopo dovranno registrarsi con un main container.

L'identità degli agenti è rappresentata dall'oggetto Agent Identifier (AID), formato da vari campi che sono conformi alla struttura e alla semantica definite dalla FIPA. Gli elementi base e più importanti dell'AID sono il nome dell'agente e i suoi indirizzi. Il nome di un agente è un identificativo univoco a livello globale ed è costruito da JADE concatenando il nickname, definito

dall'utente, al nome della piattaforma. Gli indirizzi dell'agente invece sono indirizzi di trasporto uguali a quelli della piattaforma, dove ogni indirizzo di una piattaforma corrisponde a un Message Transport Protocol (MTP), dove i messaggi coerenti alle specifiche FIPA possono essere inviati e ricevuti.

Quando il main container viene lanciato, vengono creati ed avviati automaticamente anche due agenti speciali, i cui ruoli sono definiti dallo standard FIPA Agent Management:

1. l'Agent Management System (AMS) è l'agente che supervisiona l'intera piattaforma. E' il punto d'accesso per tutti quegli agenti che necessitano di interagire al fine di accedere ai servizi di pagine gialle e pagine bianche, come anche il gestire il loro life-cycle. E' necessario che ogni agente si registri con l'AMS al fine di ottenere un AID; questa operazione viene fatta in automatico da JADE quando l'agente viene avviato;

2. il Directory Facilitator (DF) è l'agente che implementa il servizio di pagine gialle, usato da tutti gli agenti che vogliono registrare i servizi da essi offerti oppure che vogliono cercare quelli offerti da altri. Inoltre il DF permette agli agenti di iscriversi al fine di ottenere delle notifiche quando la registrazione di un servizio o la sua modifica soddisfa certi criteri. E' possibile avviare più agenti DF in modo da distribuire il servizio di pagine gialle nei vari domini.

4.1.4 I behaviours

Un agente, come è già stato detto, deve compiere delle operazioni al fine di raggiungere un certo scopo; queste operazioni, nella terminologia di JADE, prendono il nome di *behaviour*. Un behaviour rappresenta un operazione che un agente deve portare a termine e viene implementato come un oggetto che estende la classe `jade.core.behaviours.Behaviour`. Per fare in modo che un certo behaviour venga eseguito da un agente è necessario aggiungerlo a quest'ultimo attraverso il metodo `addBehaviour()` della classe `Agent`; i behaviour possono essere aggiunti in qualsiasi istante durante la vita dell'agente.

Ogni classe che estende `Behaviour` deve implementare due metodi astratti ovvero `action()`, che ritorna l'operazione contenuta nel behaviour da eseguire, e `done()` che restituisce un valore booleano per indicare se il behaviour è stato completato e quindi può essere rimosso, oppure se non è ancora terminato. Un agente può eseguire più behaviour concorrentemente, comunque la cosa importante da ricordare è che lo scheduling di questi non è per-emptive ma cooperative, il che significa che quando un behaviour viene schedulato per l'esecuzione viene chiamato il suo metodo `action()` e rimane in esecuzione finché non è terminato, ovvero finché il metodo `done()` non ritorna il valore `true`.

JADE mette a disposizione vari tipi di behaviour per facilitare alcuni compiti agli sviluppatori; resta sottointeso che tutti i vari tipi possono essere estesi al fine di creare nuovi behaviour. I principali tipi sono:

- *One-shot*: questo tipo di behaviour è fatto per essere eseguito in un'unica fase, cioè il suo metodo `Action()` viene useguito una volta sola, dato che l'implementazione di `done()` ritorna come valore `true`;
- *Cyclic*: questi behaviour sono fatti in modo da non essere mai completati, ovvero il loro metodo `done()` restituirà sempre `false`. Questi behaviour sono molto comodi per eseguire operazioni cicliche, come ad esempio il controllo della coda dei messaggi in ingresso per vedere se ce ne sono di nuovi;
- *Generic*: questo è il tipo base di behaviour, nel quale sarà lo sviluppatore a specificare la condizione di terminazione, cioè quando il metodo `done()` restituirà il valore `true`.

Ci sono anche altri tipi di behaviour, molti dei quali sono stati utilizzati nell'implementazione del sistema che è oggetto di questa tesi, come ad esempio i `TickerBehaviour` che possono essere eseguiti ad intervalli di tempo regolari (in particolare sono stati usati per l'invio dei messaggi di keep-alive da parte degli agenti Mobile e PhoneAgent verso il MobileManager, vedi la sezione 5.2); per avere maggiori informazioni su tutti i vari tipi di behaviour che JADE mette a disposizione si rimanda al materiale presente in rete.

4.1.5 L'ontologia e il content language

Tra le varie funzioni che JADE mette a disposizione degli sviluppatori c'è quella che permette di utilizzare una particolare ontologia riguardante il contesto dell'applicazione che si sta sviluppando. L'ontologia *(ontology)* è praticamente il vocabolario dei termini che fanno parte dell'ambito a cui appartiene l'applicazione, i quali possono essere utilizzati all'interno di quest'ultima. L'utilizzo dell'ontologia viene in aiuto soprattutto per quanto riguarda la comunicazione tra gli agenti: in base alle specifiche FIPA, il contenuto dei messaggi che vengono scambiati può essere una semplice stringa oppure una sequenza di byte. In certi casi però gli agenti potrebbero aver bisogno di comunicare delle informazioni più complesse rispetto a delle semplici stringhe o dei byte. Quando c'è questa necessità, bisogna utilizzare una certa sintassi, che in base alla terminologia FIPA si chiama *content language*, in modo che il destinatario del messaggio possa capire come è strutturato, dividerlo correttamente ed estrarre ogni specifico pezzo di informazione. La FIPA non obbliga ad usare un particolare content language, ma ha definito il linguaggio SL e consiglia di utilizzarlo nelle comunicazioni con l'AMS e il DF. Prima di andare avanti facciamo un breve esempio (in riferimento

all'ambito di questo lavoro di tesi) per chiarire la questione: se avessi necessità di mandare delle informazioni riguardati un volo aereo, come partenza, destinazione, orario di partenza/arrivo, e dovessi mandare queste informazioni semplicemente attraverso una stringa, dovrei concatenare tutti i valori in un'unica stringa e inviarla al destinatario, il quale dovrebbe innanzitutto capire che cosa gli sto mandando e poi dovrebbe dividere la stringa nei punti giusti al fine di capire i valori che gli sono stati inviati, con il rischio che se un valore non si trova nel posto corretto, verrebbe scambiato per un altro (se inverto la partenza con l'arrivo il destinatario non può saperlo); la cosa diventa ancora più problematica se si pensa che magari la partenza e l'arrivo sono a loro volta dei dati strutturati, perché potrei voler specificare anche il terminal e il gate dei due aeroporti. Risulta quindi chiaro che facendo in questo modo risulta particolarmente difficile e scomodo inviare informazioni strutturate; conviene in questi casi utilizzare un content language e una certa ontologia per strutturare le informazioni (in particolare in questo lavoro di tesi, quando c'è stata la necessità di scambiare informazioni strutturate tra gli agenti, è stato utilizzato il linguaggio SL e una particolare ontologia creata ad-hoc per l'ambito di cui faceva parte il progetto, cioè l'aeroporto. Per vedere l'ontologia usata vedi la sezione 5.4.

Quando un agente riceve un messaggio, il cui contenuto è stato scritto attraverso il linguaggio SL, esso è in grado fare il parse di quel messaggio e capire il contenuto dell'informazione. Inoltre, mittente e ricevente condividono la conoscenza di particolari termini, usati per esprimere i vari concetti all'interno del messaggio; questo insieme di concetti e i "simboli" usati per esprimerli fanno parte dell'ontologia. Questa, a differenza del content language che è indipendente dal contesto dell'applicazione, è invece fortemente legata al particolare dominio. Ad esempio, i concetti di volo aereo o di banco del check-in, sono importanti nell'ambito di un'area aeroportuale nella quale l'applicazione dovrà funzionare, ma se l'ambito fosse stato ad esempio quello ospedaliero certamente questi due concetti non avrebbero avuto senso, o quantomeno non sarebbero stati utili.

Abbiamo detto in precedenza che è possibile usare l'oggetto String di Java per il contenuto del messaggio, abbiamo anche detto però che questo è molto scomodo per inviare informazioni strutturate, perché costringe il ricevente a fare il parse della stringa al fine di recuperare le varie informazioni al suo interno. Un metodo più comodo sarebbe quello di inviare gli oggetti Java che rappresentano i concetti, così da non avere nessun problema con il fatto che i dati sono strutturati; questo è proprio quello che è possibile fare sfruttando l'ontologia e i content language. Prima di poter usare questo meccanismo è però necessario definire tutte le classi degli oggetti e delle azioni che si intendono usare nei messaggi (questi oggetti dovranno estendere la classe `Concept` nel caso esprimano concetti e `AgentAction` nel caso esprimano azioni): all'interno di ogni classe ci devono essere tutte le variabili interne che servono e per ognuna di esse ci deve essere un metodo che

permette di andare a scrivere il valore e un altro che permette di andarlo a leggere (un frammento della classe Flight è riportato qui sotto a titolo di esempio); tutti questi oggetti saranno a loro volta variabili di un oggetto che estenderà la classe `Ontology` (per maggiori dettagli su come si costruisce questo oggetto si rimanda all'ampia documentazione reperibile in rete).

```
public class Flight implements Concept
{
    private String number;
    private String from;
    private String to;
    ...

    public String getNumber(){return number;}

    public void setNumber(String number){this.number = number;}

    public String getFrom(){return from;}

    public void setFrom(String from){this.from = from;}

    public String getTo(){return to;}

    public void setTo(String to){this.to = to;}
    ...
}
```

Nel momento in cui un agente invia uno degli oggetti che fanno parte dell'ontologia, deve convertire la rappresentazione interna dell'oggetto in quella corrispondente dell'ACL content expression e il ricevente dovrà fare la conversione opposta; il destinatario inoltre eseguirà una serie di controlli sulla semantica delle informazioni ricevute al fine di verificare se rispettano le regole dell'ontologia condivisa tra i due agenti (se ad esempio un campo obbligatorio dell'oggetto inviato è vuoto viene generato un errore). JADE esegue tutte queste operazioni automaticamente e quindi lo sviluppatore non si deve preoccupare del meccanismo di marshalling/unmarshalling che viene eseguito sugli oggetti.

4.1.6 I pacchetti

I sorgenti della piattaforma JADE sono organizzati in una gerarchia di pacchetti e sotto pacchetti Java, dove ogni pacchetto, in teoria, contiene l'insieme di classi e interfacce che implementano una specifica funzionalità. I pacchetti principali sono:

- `jade.core` implementa il kernel di JADE, l'ambiente di runtime distribuito che supporta l'intera piattaforma e i suoi strumenti. In questo pacchetto è contenuta la classe `jade.core.Agent` come anche tutte le classi runtime necessarie per implementare i container. Sono inoltre compresi dei sotto pacchetti, ognuno dei quali implementa uno specifico servizio a livello del kernel. Questi sono:
  - `jade.core.event` che implementa il servizio di notifica degli eventi. Questo permette a chi si è iscritto di ricevere le notifiche di sistema generate dagli eventi dai vari componenti della piattaforma;
  - `jade.core.management` che implementa il servizio di gestione del life-cycle degli agenti;
  - `jade.core.messaging` che implementa il servizio di distribuzione dei messaggi;
  - `jade.core.mobility` che implementa la mobilità degli agenti e il servizio di clonazione, incluso il trasferimento sia dello stato che del codice dell'agente;
  - `jade.core.nodeMonitoring` che permette ai container di controllarsi a vicenda al fine di scoprire i container non raggiungibili o non più esistenti;
  - `jade.core.replication` che permette la replica del main container in modo che l'intera piattaforma non vada in crash a causa di problemi nel main container;
  - `jade.core.behaviours` che contiene dei sotto pacchetti che a loro volta contengono i vari behaviour. Un behaviour rappresenta un'operazione che un agente può eseguire.
- `jade.content` e i suoi sotto pacchetti contengono tutte quelli classi che aiutano il programmatore a creare e manipolare i contenuti dei messaggi in accordo con un determinato linguaggio e ontologia;
- `jade.domain` contiene l'implementazione degli agenti AMS e DF, come specificato dagli standard FIPA, oltre alle loro estensioni specifiche di JADE;
- `jade.gui` contiene dei componenti Java generali per creare delle GUI per gli agenti;
- `jade.impt` contiene l'implementazione del JADE IMPT (Internal Message Transport Protocol);
- `jade.lang.acl` contiene le classi per il supporto al FIPA Agent Communication Language (ACL), inclusa la classe `ACLMessage`, il parser,

il codificatore e delle classi di aiuto che rappresentano i template dei messaggi ACL;

- `jade.mtp` contiene l'insieme di interfacce Java che possono essere implementate dal JADE MTP. Inoltre contiene due sotto pacchetti con l'implementazione basata sul protocollo HTTP e una basata sul protocollo IIOP;
- `jade.proto` contiene l'implementazione di protocolli generali di interazione, inclusi certi specificati da FIPA;
- `jade.tools` contiene l'implementazione di tutti gli strumenti grafici di JADE;
- `jade.wrapper` e le classi `jade.core.Profile` e `jade.core.Runtime` permettono alle applicazioni Java esterne di usare JADE come una libreria;
- `FIPA` è un pacchetto che include il modulo Interface Definition Language (IDL) specificato da FIPA per l'MTP basato su IIOP.

4.1.7 L'estensione JADE-LEAP

L'estensione LEAP della piattaforma JADE ha lo scopo di permettere l'esecuzione degli agenti su dispositivi con limitate risorse come telefoni cellulari che eseguono codice Java o dispositivi che eseguono il Microsoft .Net Framework. L'estensione LEAP è stata sviluppata principalmente all'interno del progetto LEAP IST ed è stata resa disponibile agli sviluppatori da marzo 2005. La versione utilizzata in questo progetto di tesi è la 4.1, rilasciata il 13 luglio 2011.

Con l'introduzione delle reti GPRS, UMTS e WLAN, è oggi possibile rimanere connessi praticamente in ogni momento e in ogni luogo; inoltre i dispositivi come i PDA (Personal Digital Assistant) e i telefoni cellulari (chiamati ormai smartphone) hanno visto un continuo crescere delle proprie possibilità messe a disposizione degli utenti. Entrambe queste evoluzioni hanno fatto si che le reti wireless e quelle wire, si integrassero sempre più assieme. E' in questo scenario che risulta necessario riuscire a sviluppare applicazioni che siano distribuite in parte nella rete fissa e in parte sui dispositivi mobili, come quelli citati in precedenza.

Purtroppo la piattaforma JADE così da sola non può essere eseguita anche sui dispositivi mobili con limitate risorse per i seguenti motivi:

- l'esecuzione base del runtime di JADE, richiede qualche Mbyte di memoria, che spesso è già troppo per le limitate risorse dei dispositivi mobili;

- per funzionare, JADE richiede Java 5, ma la maggior parte dei dispositivi mobili supporta solo le specifiche CDC (Connected Device Configuration), PersonalJava o molto più spesso MIDP (Mobile Information Device Profile);
- i collegamenti wireless hanno differenti caratteristiche rispetto a quelli su rete fissa come l'alta latenza, la bassa banda, la connessione che va e viene e l'indirizzo IP che può variare dinamicamente.

Per risolvere tutti questi problemi è stata creata l'estensione LEAP e permettere quindi lo sviluppo di agenti JADE anche per dispositivi mobili.

L'estensione LEAP, quando viene combinata con JADE, va a sostituire alcune parti del kernel JADE, andando così a formare il runtime environment chiamato JADE-LEAP (che significa JADE eseguito da LEAP), il quale può essere eseguito su un'ampia gamma di dispositivi mobili. Per ottenere questo, JADE-LEAP è stato diversificato in base alle due configurazioni di Java Micro Edition, ovvero:

- *pjava*: per eseguire JADE-LEAP su dispositivi mobili che supportano J2ME CDC o PersonalJava (la quale è stata dichiarata obsoleta nel 2003), come la maggior parte dei PDA odierni;
- *midp*: per eseguire JADE-LEAP su dispositivi mobili che supportano solo MIDP 1.0 (o successivi), come la maggior parte dei telefoni cellulari che supportano Java;
- *android*: per eseguire JADE-LEAP su dispositivi che supportano Android 2.1 (o successivi).

E' presente anche una versione *dotnet* per eseguire JADE-LEAP sui computer e sui server di una rete fissa, che utilizzano Microsoft .NET Framework v1.1 (o successivi). Anche se internamente le realizzazioni delle varie versioni di JADE-LEAP sono diverse, tutte mettono a disposizione dello sviluppatore lo stesso set di API, offrendo quindi una certa omogeneità a fronte di una diversità sia dei dispositivi che del tipo di reti (vedi Figura 4.1). A partire dalla versione 3.7, JADE-LEAP era una piattaforma completamente diversa da JADE e questo faceva si che non era possibile collegare un container di JADE-LEAP al main container di JADE o viceversa; a partire però dalla versione 4.0 (e quindi anche la versione utilizzata per questo lavoro di tesi, ovvero la 4.1), JADE e JADE-LEAP sono diventate una singola piattaforma, nella quale JADE-LEAP per *pjava*, *midp* e *android* sono semplicemente delle versioni modificate di JADE con lo scopo di eseguire gli agenti di JADE anche su dispositivi mobili.

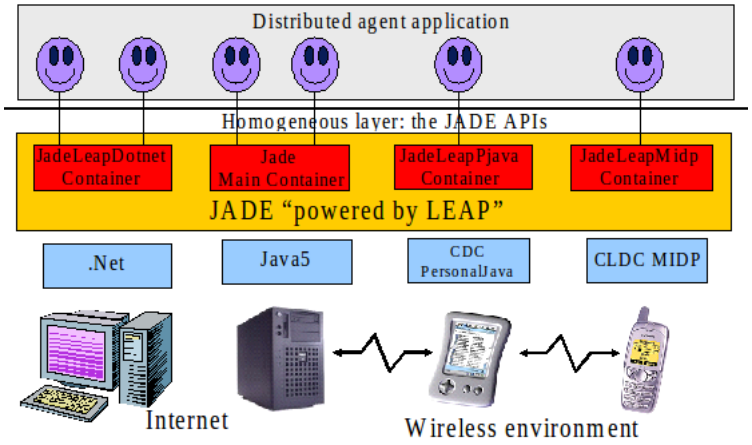

Figura 4.1: L'ambiente di runtime di JADE-LEAP

4.1.7.1 Le modalità di esecuzione di JADE-LEAP

L'ambiente JADE può essere eseguito in due modalità: *Stand-alone* e *Split*. Nella modalità di esecuzione *Stand-alone*, un container completo viene eseguito sul dispositivo mobile dove JADE è utilizzato; invece, nella modalità di esecuzione *Split*, il container è diviso (split) in un FrontEnd, il quale verrà eseguito sul dispositivo mobile e in un BackEnd, che invece verrà eseguito su un server remoto; queste due parti del container sono legate assieme da una connessione permanente (vedi Figura 4.2).

L'esecuzione *Split* è particolarmente indicata per i dispositivi con limitate risorse e capacità per diverse ragioni: innanzitutto il *FrontEnd* del container, cioè la parte che viene eseguita sul dispositivo, richiede molte meno risorse per essere eseguita rispetto al container completo; la fase di avvio è più veloce, perché tutte le comunicazioni che servono per collegarsi al main container, vengono portate a termine dal *BackEnd*, e quindi non viene utilizzata la connessione wireless del dispositivo mobile; infine, l'utilizzo del collegamento wireless viene ottimizzato. Un aspetto molto importante di tutto questo, è che lo sviluppatore non si deve preoccupare di tutti questi aspetti, infatti le API sono assolutamente le stesse, sia che il container venga eseguito in modalità *Split*, che in modalità *Stand-alone*. In questo lavoro di tesi è stata utilizzata l'esecuzione *Split* del container, per tutti i motivi appena elencati.

4.1.8 Gli strumenti di amministrazione e di debugging

Le applicazioni multi-agente sono in generale abbastanza complesse: molto spesso sono distribuite su vari host, sono formate da centinaia di processi e in tutto questo abbiamo gli agenti che possono essere creati, distrutti e migrare da un posto all'altro. E' chiaro quindi che tutti questi fattori implicano una

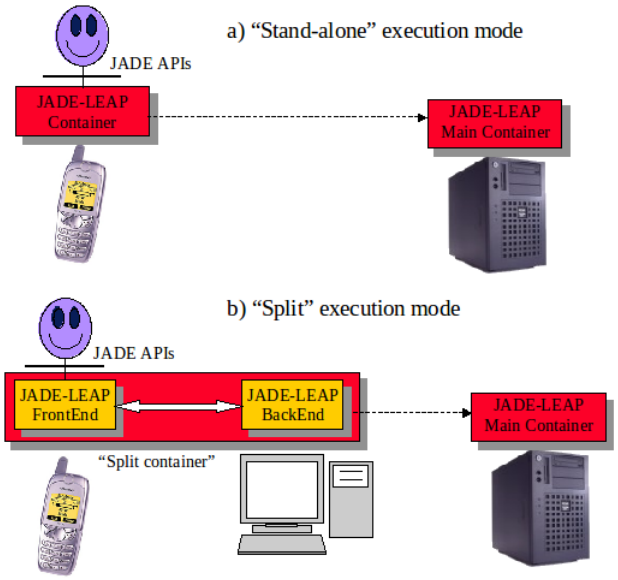

Figura 4.2: Le modalità di esecuzione di un container JADE-LEAP

difficoltà di gestione e debugging dell'intero sistema. Per cercare di risolvere questo problema, JADE ha a disposizione un servizio di notifica degli eventi, il quale è alla base della console di gestione chiamata JADE RMA (Remote Monitoring Agent) e inoltre ha altri tool grafici messi a disposizione sempre per aiutare lo sviluppatore.

Nelle sezioni successive andremo ad analizzare nello specifico tutti questi strumenti che sono di fondamentale aiuto durante le fasi di sviluppo e debugging di un'applicazione.

4.1.8.1 La console di gestione della piattaforma

Il JADE RMA (Remote Monitoring Agent), riportato in Figura 4.3, è uno strumento di sistema che implementa una console grafica per la gestione della piattaforma. Esso fornisce un'interfaccia grafica per monitorare e amministrare la piattaforma distribuita JADE, formata da numerosi host e container, sparsi nella rete; dall'RMA è possibile avviare altri tool. E' importante notare che l'RMA è un agente a tutti gli effetti, quindi al suo avvio deve anch'esso registrarsi presso l'AMS, in modo da ricevere le notifiche da parte della piattaforma.

L'RMA mette a disposizione una rappresentazione, sottoforma di directory, della topologia della piattaforma, dove i nodi sono di 3 tipi: piattaforma degli agenti, container e agente.

Selezionando il nodo di un agente è possibile eseguire diverse operazioni, come sospendere l'agente, riprendere l'esecuzione, salvarlo, fargli inviare

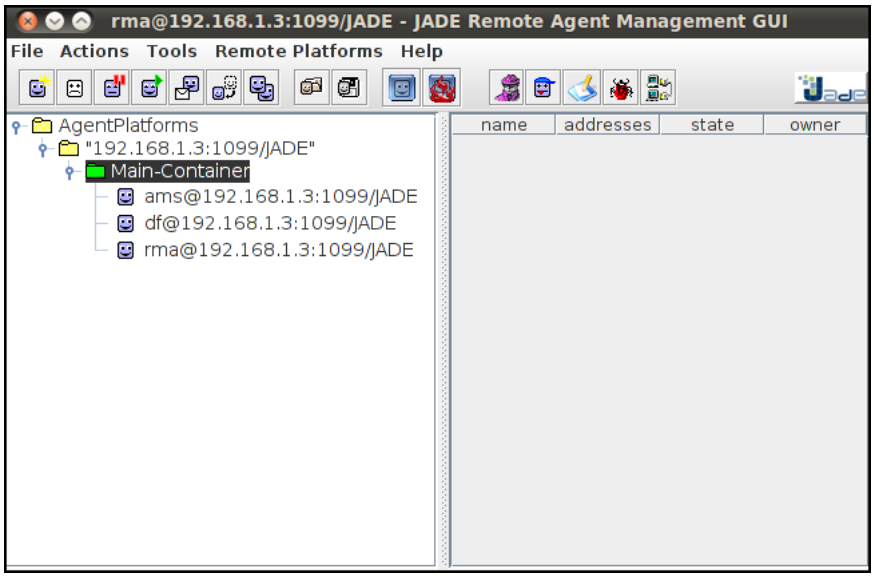

Figura 4.3: Remote Monitoring Agent (RMA)

messaggi creati ad-hoc oppure personalizzati.

Invece se si seleziona un nodo container posso creare nuovi agenti al suo interno, caricare un agente esistente, installare o rimuovere un MTP, salvare o caricare un container inclusi tutti gli agenti al suo interno e la terminazione del container.

Infine, se seleziono un nodo piattaforma è possibile vedere il suo nome e tutti i suoi servizi disponibili, oltre che alla gestione degli MTP della piattaforma. Da notare che è possibile gestire più piattaforme dallo stesso RMA, anche se il controllo sulle piattaforme remote è più limitato.

4.1.8.2 Il DummyAgent

Il DummyAgent è un'utile strumento per inviare degli "stimoli" agli agenti sottoforma di messaggi ACL, utili per testare la riposta che questi agenti danno e il funzionamento dei loro behaviour. Quello che può fare quindi il DummyAgent è inviare e ricevere messaggi, facilmente componibili grazie alla GUI messa a disposizione, oltre a poterli caricare/salvare da/su file. Il DummyAgent viene usato molto in fase di sviluppo in quanto rende possibile testare il comportamento che hanno gli agenti quando ricevono determinati messaggi.

4.1.8.3 L'agente Sniffer

Questo strumento, riportato in Figura 4.4, è utilizzato per debuggare, o semplicemente per documentare le conversazioni tra agenti. Anche lo Sniffer

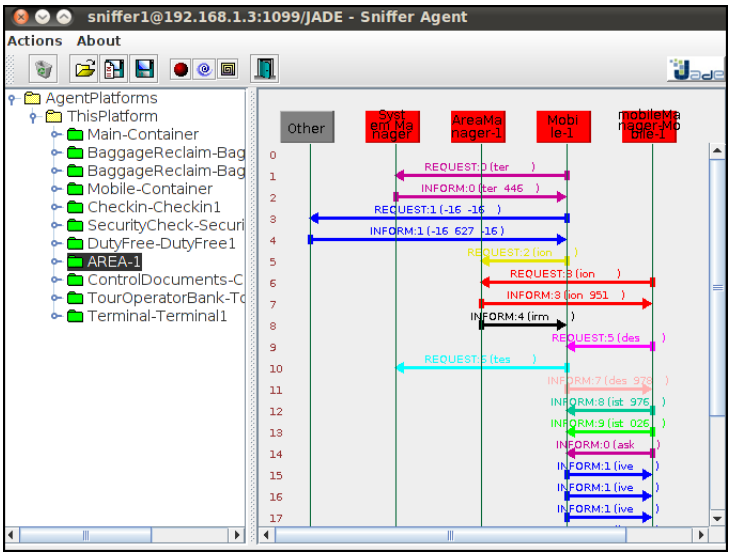

Figura 4.4: Sniffer Agent

è un'agente a tutti gli effetti e quindi si deve registrare con l'AMS della piattaforma al fine di ricevere le notifiche sugli eventi riguardanti lo scambio di messaggi fra agenti specifici, che sono selezionabili dall'utente.

Quando si decide di "ascoltare" un agente, tutti i suoi messaggi in entrata e in uscita vengono visualizzati sulla GUI dello Sniffer; è inoltre possibile andare ad analizzare ogni singolo messaggio che viene scambiato, oltre a poter salvare su un file di testo l'intera conversazione.

4.1.8.4 L'agente Introspector

Questo agente è utile per debuggare un behaviour di un singolo agente; infatti grazie all'agente Introspector è possibile seguire e controllare il life-cycle dell'agente e la coda dei messaggi in entrata/uscita. Permette inoltre di controllare la coda dei behaviour schedulati, inclusa la possibilità di vederne l'esecuzione step-by-step.

4.1.8.5 L'agente Log Manager

L'agente Log Manager è uno strumento che semplifica la gestione dinamica e distribuita del servizio di logging fornendo un'interfaccia che permette di cambiare durante il runtime i livelli di logging di ogni componente della piattaforma JADE. Questo include tutte quelle componenti che vengono eseguite su nodi remoti, inclusi i messaggi di logging specifiche dell'applicazione.

4.1.8.6 Il servizio di notifica degli eventi

Il servizio di notifica degli eventi (Event Notification Service - ENS) è un servizio che gestisce la notifica distribuita di tutti gli eventi generati da ogni nodo della piattaforma. Ogni volta che un evento è generato da un container (come ad esempio la nascita di agente o l'invio di un messaggio), questo viene intercettato dall'ENS e inoltrato a tutti gli agenti che in precedenza hanno richiesto di ricevere le notifiche di quel tipo di eventi. Se nessun agente si è iscritto, l'overhead dovuto all'esecuzione dell'ENS è comunque trascurabile.

Ci sono quattro tipi principali di eventi che possono verificarsi:

- *Platform-type*: questo tipo di eventi sono legati al verificarsi di eventi che in qualche modo coinvolgono il main container. Questi eventi sono legati ai cambiamenti nel life-cycle di un agente (ad esempio quando nasce, muore e migra) e quelli del container (ad esempio quando viene aggiunto e rimosso);

- *MTP-type*: questi eventi sono generati dalla piattaforma quando un MTP viene attivato/disattivato e quando un messaggio è inviato/ricevuto da un MTP, in poche parole quando c'è una qualche comunicazione al di fuori della piattaforma;

- *Message-passing-type*: questi sono eventi generati quando un messaggio ACL viene inviato, ricevuto, inoltrato o inserito nella coda dei messaggi di un agente. Tipicamente questi eventi vengono utilizzati dall'agente Sniffer per monitorare le comunicazioni;

- *Agent-internal-type*: questi sono eventi legati ai cambiamenti dello stato dei behaviour di un agente. L'agente Introspector li usa per monitorare il funzionamento interno degli agenti.

Questi sono dunque gli strumenti messi a disposizione dalla piattaforma JADE per aiutare gli sviluppatori durante le fasi di sviluppo e di debugging di applicazioni multi-agente che, come abbiamo detto prima, possono risultare anche molto complesse e fanno si che strumenti come questi siano di fondamentale importanza.

4.2 Android

In questa sezione si parlerà del sistema operativo Android, che rappresenta la piattaforma per cui è stata sviluppata una versione dell'applicazione, oltre a quella per computer. Verranno spiegati i vari componenti dell'architettura di questa piattaforma, si parlerà poi dei vari elementi che compongono un'applicazione Android, dando infine anche dei cenni pratici sul cosa bisogna fare per crearne una.

4.2.1 La piattaforma

Android è un sistema operativo embedded che si basa su un kernel Linux per i servizi fondamentali (core system service) come la sicurezza, la gestione della memoria, la gestione dei processi e il network stack.

Le applicazioni per Android sono scritte utilizzando il framework Java, anche se in realtà non è davvero Java; infatti, librerie standard come ad esempio la classe Swing, non sono supportate, mentre altre sono state sostituite da librerie proprie di Android, le quali sono ottimizzate per un utilizzo in un ambiente dove si hanno risorse limitate, come è appunto quello embedded. Per i motivi appena detti, Android non utilizza la Java virtual machine, ma la Dalvik virtual machine; ogni applicazione Android viene eseguita su un processo a parte assieme alla propria istanza della Dalvik VM. Questa virtual machine è stata scritta per fare in modo che un dispositivo ne possa eseguire più istanze in contemporanea in maniera efficiente. La Dalvik VM esegue file nel formato Dalvik Executable (.dex) che è ottimizzato per i dispositivi con memoria limitata; per creare questi file basta compilare prima un file .java con il compilatore Java e poi trasformare i file .class nel formato .dex (questo viene fatto grazie ad un tool messo a disposizione).

Il sistema operativo Android è open source, quindi è possibile vedere e usare il codice sorgente dell'intero sistema. Questo permette agli sviluppatori di sfruttare il sistema in ogni sua singola parte e di personalizzarlo a seconda delle necessità; bisogna anche dire però che i dispositivi Android contengono anche del software proprietario che è inaccessibile agli sviluppatori (come ad esempio quello per la navigazione tramite Global Positioning System, cioè il GPS).

Google comprò l'azienda che inizialmente sviluppò questo sistema operativo, la Anroid Inc., nel 2005. A inizio novembre del 2007, con la fondazione

della Open Handset Alliance[1], Google diede la notizia della distribuzione di Android.

4.2.2 L'architettura

Parliamo ora dell'architettura della piattaforma Android ed in particolare dei livelli chiave che compongono lo stack di essa, riportati in Figura 4.5. Ogni livello utilizza i servizi messi a disposizione dal livello sottostante.

Cominciando dal basso abbiamo il Kernel Linux, il cuore di Android. Linux permette di astrarre il livello hardware, permettendo così di utilizzare Android su una vasta gamma di piattaforme hardware con caratteristiche diverse. Internamente, Android utilizza Linux per la gestione della memoria, dei processi, della rete e di altri servizi del sistema operativo. L'utilizzo di Linux resterà comunque trasparente all'utente e in buona parte anche allo sviluppatore, se non per l'utilizzo di qualche strumento, come il comando `adb shell` che apre una shell Linux dalla quale è possibile dare comandi che verranno eseguiti sul dispositivo.

Salendo di un livello troviamo le librerie native di Android. Queste sono tutte scritte in C o C++, compilate per la particolare architettura hardware usata dal telefono e poi installate dal produttore del telefono. Alcune tra le principali librerie native includono i seguenti elementi:

- *Surface Manager*: Android utilizza un window manager che invece di disegnare direttamente sul buffer dello schermo, disegna su delle immagini a parte che vengono poi combinate con altre al fine di ottenere l'immagine finale che l'utente vede. Questo permette al sistema di creare tutti i vari effetti come le trasparenze;

- *grafica 2D e 3D*: gli elementi in due e tre dimensioni possono essere combinati un'unica interfaccia utente con Android. La libreria userà l'hardware 3D se il dispositivo ce l'ha, altrimenti utilizzerà un renderer;

- *Media codecs*: Android può riprodurre e registrare video e audio in vari formati come AAC, AVC (H.264), H.263, MP3 e MPEG-4;

- *SQL database*: Android mette a disposizione SQLite, un database leggero utilizzato fra gli altri anche da Firefox e dall'Apple iPhone;

- *Browser engine*: per visualizzare velocemente il contenuto delle pagine HTML, Android utilizza la libreria WebKit, la stessa utilizzata nel browser Google Chrome, Apple Safari, Apple iPhone e nella piattaforma Nokia S60.

[1]Open Handset Alliance è un'associazione di 84 aziende (tra cui Google, HTC, Sony, Dell, Intel, Motorola, Texas Instruments, Samsung, LG, T-Mobile e Nvidia) che lavorano nei campi dell'hardware, del software e delle telecomunicazioni e che si pone come scopo quello di sviluppare gli standard per i dispositivi mobili.

Queste librerie esistono per essere utilizzate dai programmi dei livelli superiori e, dalla versione 1.5 di Android, è possibile costruire le proprie librerie native.

Sempre posizionato al di sopra del livello del kernel, troviamo l'Android runtime, che include la Dalvik virtual machine e le Core Libraries. La Dalvik VM è l'implementazione di Google della Java VM, ottimizzata per i dispositivi mobili. Tutto il codice di un'applicazione Android è scritto in Java e viene eseguito dalla Dalvik VM, la quale differisce dalla Java VM per due importanti motivi:

- la Dalvik VM esegue file *.dex*, che sono convertiti nel momento della compilazione dal formato *.class* e *.jar*. I file *.dex* sono più compatti ed efficienti dei file *.class*, aspetto importante viste la limitata memoria e durata della batteria dei dispositivi sui quali deve girare Android;
- le Core Libreries presenti in Android sono diverse sia da quelle di Java Standard Edition (Java SE) sia da quelle di Java Mobile Edition (Java ME), anche se comunque ce ne sono molte in comune.

Sopra a questi livelli c'è l'Application Framework, il quale mette a disposizione i cosiddetti building block che verranno utilizzati per creare le applicazioni. Il framework è pre-installato su Android, ma è possibile estenderlo con i propri componenti, nel caso ce ne sia bisogno. Le parti più importanti di questo livello sono:

- *Activity Manager*: controlla il life cycle delle applicazioni e mantiene il "backstack" utile alla navigazione fra le varie activity dell'utente (in poche parole, quando l'utente passa da un'activity a un'altra, quella che non è più visualizzata viene messa dentro uno stack gestito con politica LIFO, così che quando l'utente vuole tornare all'activity precedente può farlo);
- *Content providers*: questi oggetti racchiudono al loro interno i dati che devono essere condivisi fra le varie applicazioni, come ad esempio i contatti della rubrica;
- *Resource manager*: gestisce le risorse dell'applicazione, ovvero tutto quello che sarà presente nell'applicazione ma che non è codice;
- *Location manager*: permette a un dispositivo Android di capire dove si trova geograficamente;
- *Notification manager*: eventi come l'arrivo di messaggi e appuntamenti possono essere notificati all'utente in una maniera non intrusiva.

Infine arriviamo al livello più alto, ovvero quello chiamato Applications, che conterrà appunto tutte le applicazioni che l'utente potrà usare. Ovviamente il lavoro di sviluppo è concentrato soprattutto sulla realizzazione di

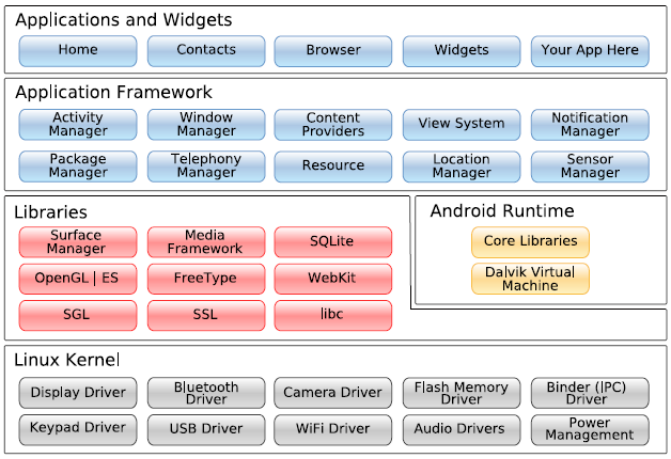

Figura 4.5: L'architettura del sistema Android

nuove applicazioni che, da quando il sistema operativo Android è stato reso disponibile, sono disponibili in un numero davvero elevato, tanto da stare per superare quelle disponibili sull'AppStore di Apple.

4.2.3 Le componenti principali di un'applicazione

Vediamo ora quali sono le principali componenti che vanno a formare un'applicazione all'interno del sistema operativo Android:

- *Activity*: è la componente dell'applicazione che permette all'utente di interagire con essa; ogni activity mette a disposizione una finestra nella quale è possibile disegnare l'interfaccia grafica (in poche parole, ogni schermata di un'applicazione è un'activity diversa). Normalmente ogni applicazione ha una *main activity* che è quella che viene visualizzata quando l'applicazione si avvia;

- *Content providers*: permettono di memorizzare e recuperare i dati nel dispositivo, rendendoli accessibili a più applicazioni; questo è l'unico modo che mette a disposizione Android per condividere dati fra applicazioni diverse, in quanto non c'è un'area dati comune. Di base, Android mette a disposizione dei content provider per i classici tipi di dati (ad esempio audio, video, immagini). Se si vuole condividere con le altre applicazioni dei propri dati, allora bisogna creare un proprio content provider oppure aggiungerli a uno già esistente;

- *Services*: a differenza delle activity e dei content provider che possono essere terminati in qualsiasi istante, i service sono fatti apposta per rimanere in esecuzione per un tempo relativamente lungo e indipendentemente dall'applicazione, inoltre vengono eseguiti in background

e sono quindi privi di interfaccia grafica. Tipiche operazioni svolte dai service sono il download di file dalla rete, l'esecuzione di file audio e di operazioni di I/O;

- *Broadcast Receivers e Intent Receivers*: rispondono alle richieste di servizio da parte di un'altra applicazione. Un Broadcast Receivers risponde ai messaggi in broadcast da parte del sistema che annunciano un certo evento. Questi annunci possono partire dallo stesso Android (ad esempio se la batteria è scarica) oppure da un programma in esecuzione sul sistema. Un'Activity o un Service mettono a disposizione le proprie funzionalità alle altre applicazioni attraverso un Intent Receivers, ovvero un piccolo pezzo di codice eseguibile che risponde alle richieste di dati o servizi che arrivano dalle altre activity. L'activity richiedente lancia un Intent, lasciando poi che sia Android ha capire quale applicazione deve riceverlo ed elaborarlo.

4.2.4 Sviluppare un'applicazione

Questo paragrafo vuole essere solo una panoramica sui passi principali che bisogna compiere per realizzare un'applicazione per la piattaforma Android; per una spiegazione più precisa si rimanda all'ampia documentazione reperibile in rete.

Per sviluppare un'applicazione per Android è possibile utilizzare qualsiasi IDE, ma Google ne consiglia uno in particolare, ovvero Eclipse. I motivi di questa scelta sono essenzialmente due: il primo riguarda la semplicità d'uso di Eclipse, che fa si che la curva di apprendimento per utilizzarlo sia davvero minima e il fatto che Eclipse sia totalmente gratuito e completo di tutte le funzionalità Java; il secondo motivo, è che l'Open Handset Alliance ha rilasciato un plugin di Android per Eclipse che permette di creare progetti specifici per Android, compilarli e utilizzare l'emulatore Android per eseguirli e testarli. E' comunque possibile utilizzare anche altri IDE, come ad esempio NetBeans, solo che in questo caso non esistono plugin e quindi è necessario configurare tutte le varie opzioni e i vari file manualmente. In questo lavoro di tesi è stato utilizzato Eclipse e l'apposito plugin per sviluppare l'applicazione Android.

Non è però sufficiente un IDE per poter sviluppare applicazioni Android, serve infatti anche l'Android SDK, il quale mette a disposizione tutte le librerie necessarie per creare delle applicazioni che possano essere eseguite su questa piattaforma. Oltre a queste librerie, l'Android SDK contiene anche file di aiuto, documentazione, un emulatore Android e molti altri strumenti per lo sviluppo e il debugging delle applicazioni.

Per sviluppare un'applicazione è prima necessario creare un nuovo progetto Android all'interno di Eclipse, nella creazione del quale verranno chieste alcune informazioni, come la versione minima delle API che verrà uti-

lizzata, il nome dell'applicazione e la *main activity*, la quale rappresenta la prima schermata che l'utente vedrà una volta avviata l'applicazione. Fatto ciò, il progetto verrà creato e al suo interno saranno già presenti alcuni file e directory: i file più importanti, generati in automatico da Android, sono certamente `AndroidManifest.xml` e `R.java`.

Nel file `AndroidManifest.xml`, che viene auto-generato da Android, bisogna dichiarare tutte le componenti dell'applicazione, dalle activity ai service che si intendono usare, inoltre questo file viene utilizzato anche per i seguenti scopi:

- identificare i permessi utente che l'applicazione richiede, come ad esempio l'accesso a Internet e il controllo dello stato del wifi del dispositivo;
- dichiarare l'API Level minimo richiesto dall'applicazione, che dipende da quali API l'applicazione utilizza;
- dichiarare le funzionalità hardware e software che l'applicazione deve utilizzare, come ad esempio il bluetooth, la telecamera o lo schermo multitouch;
- le librerie di API che devono essere collegate all'applicazione, come ad esempio la Google Maps library.

Come esempio, qui sotto viene riportato un frammento del codice che si trova all'interno di questo file:

```
<?xml version="1.0" encoding="utf-8"?>
  <LinearLayout
      xmlns:android=http://schemas.android.com/apk/res/android
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
  >
    <TextView
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="Hello World, HelloWorldText"
    />
  </LinearLayout>
  ...
```

In parole povere, il compito del file `AndroidManifest.xml` è quello di definire i contenuti e il comportamento dell'applicazione.

Il file `R.java`, anch'esso auto-generato da Android e aggiunto al progetto dal plugin di Android, contiene i puntatori alle varie risorse presenti all'interno della directory `res` (e all'interno di tutte le sue sub-directory) e verrà

utilizzato proprio per accedervi dall'applicazione. Non bisogna mai modificare questo file manualmente, in quanto viene fatto in maniera automatica ogni volta che vengono aggiunte delle risorse al progetto. Un esempio del contenuto di questo file è stato riportato qui sotto:

```
/* AUTO-GENERATED FILE. DO NOT MODIFY.
*
* This class was automatically generated by the
* aapt tool from the resource data it found. It
* should not be modified by hand.
*/
package testPackage.HelloWorldText;
public final class R {
  public static final class attr {
  }
  public static final class drawable {
    public static final int icon=0x7f020000;
  }
  public static final class layout {
    public static final int main=0x7f030000;
  }
  public static final class string {
    public static final int app_name=0x7f040000;
  }
}
```

Nella spiegazione dell'utilizzo del file `R.java`, è stata nominata la directory `res`: questa directory contiene tutte le risorse del progetto, ovvero tutte quelle componenti che sono utilizzate all'interno di esso. Quando viene creato un nuovo progetto Android, questa directory contiene tre sub-directory: `drawable`, `layout` e `values`. Nelle prime due sono contenute rispettivamente le immagini che si vogliono usare nel progetto e i vari layout, mentre nella terza sono contenute ad esempio le stringhe globali utilizzate. Si accede a tutte queste risorse attraverso il file `R.java` spiegato in precedenza.

Nella directory `src` sono invece presenti tutti i file sorgenti delle varie activity che compongono l'applicazione e di tutti gli altri file che vengono usati all'interno del progetto.

Per testare l'applicazione ci sono due possibilità: la prima consiste nell'utilizzare l'emulatore messo a disposizione dall'Android SDK, il quale permette di scegliere il dispositivo e la versione delle API presenti sulla piattaforma installata su di esso, oppure nel caso in cui si disponga di un dispositivo fisico, è possibile installare l'applicazione su di esso per vederne il comportamento. Questa seconda strada è stata quella utilizzata in questo lavoro di tesi per sviluppare e testare l'applicazione Android; in particolare si è avuto a

disposizione un telefono HTC Desire A8181, con installata la versione 2.2 della piattaforma Android (API Level 8).

Capitolo 5

Lo sviluppo del sistema

In questo capitolo si parlerà degli aspetti pratici dell'implementazione del sistema e dell'applicazione che lo sfrutta. Come prima cosa verranno illustrati gli aspetti dell'aeroporto che sono stati simulati (sezione 5.1), in quanto non era ovviamente a disposizione un'infrastruttura del genere; poi saranno spiegati nel dettaglio tutti gli agenti che compongono il sistema (sezione 5.2), specificando per ognuno il ruolo e la sua posizione all'interno dell'architettura (eccetto che per gli agenti che offrono delle funzioni di simulazione e che quindi non sono collocati da nessuna parte all'interno dell'architettura); infine verranno analizzate le interazioni principali fra gli agenti (sezione 5.3), al fine di mostrare come la comunicazione si sviluppa.

5.1 La simulazione

Iniziamo parlando in questa sezione dei vari aspetti dell'aeroporto che è stato necessario simulare al fine di ottenere un funzionamento quanto più realistico dell'intera infrastruttura. Verrà spiegato non solo cosa è stato simulato ma anche come è stato simulato, evidenziando gli aspetti che si è preferito tralasciare a vantaggio di altri ritenuti più importanti.

5.1.1 Che cosa simulare

Oltre allo sviluppo dell'applicazione, buona parte del tempo è stata spesa anche per simulare gli aspetti rilevanti ai fini del corretto funzionamento del sistema. Infatti, non potendo chiaramente avere a disposizione un aeroporto per fare dei test, si sono dovuti simulare tutti quegli aspetti che vengono coinvolti nel sistema e che sono necessari al suo funzionamento.

La prima cosa che è stata simulata è lo spostamento dei dispositivi, o meglio delle persone che hanno i dispositivi; come mappa è stata utilizzata quella dell'aeroporto Valerio Catullo di Verona. Quando l'intero sistema viene avviato, viene visualizzata una finestra con la mappa dell'intero aero-

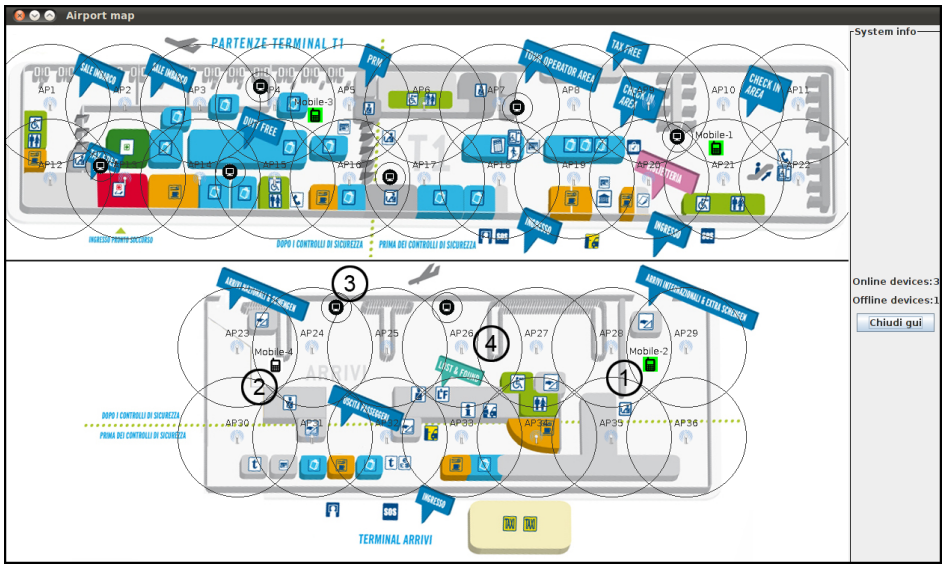

Figura 5.1: Mappa di sistema: 1- icona che indica il dispositivo online, 2- icona che indica il dispositivo offline, 3- icona che indica un monitor informativo, 4- icona che indica un access point wireless

porto assieme anche agli access point wireless, e ai loro raggi d'azione, che sono presenti al suo interno; questa è la mappa di sistema (vedi Figura 5.1), il cui unico scopo è quello di mostrare la posizione e lo stato dei dispositivi presenti. Non appena un dispositivo viene acceso e viene attivato il suo wifi (per come sono stati simulati gli access point vedere più avanti), compare sulla mappa l'icona che indica la presenza del telefono e il fatto che è stato solo rilevato, ma che questo non ha ancora fatto il login al sistema; non appena il login viene fatto, l'icona diventa verde, ad indicare appunto che il dispositivo adesso è connesso correttamente al sistema e può interagire con esso. Tuttavia, non è possibile simulare lo spostamento dei dispositivi mobili dalla mappa generale di sistema, che è stata pensata semplicemente allo scopo di visualizzare la posizione di tutti gli host all'interno dell'aeroporto. Per spostarli bisogna accedere alle mappe disponibili sui singoli dispositivi: cliccando sul bottone apposito, si aprirà una nuova finestra che visualizzerà la mappa e un quadratino nero che indica la posizione del dispositivo; per spostarlo basta cliccare in un altro punto della mappa e la sua posizione viene aggiornata, ovviamente anche nella mappa di sistema. Per quanto riguarda la versione dell'applicazione sul telefono Android, una volta che si preme il bottone della mappa, questa viene aperta non sul telefono stesso ma sul computer dove è presente il main container, e da lì è possibile simulare lo spostamento esattamente nello stesso modo spiegato prima. Il motivo per cui non è possibile cambiare la posizione direttamente dal telefono è che in realtà per ottenere questo bisognerebbe entrare nei particolari della grafi-

ca 2D di Android, in quanto dovrebbe anche essere possibile ingrandire e spostare la mappa, visto che lo schermo del telefono è troppo piccolo per visualizzare chiaramente l'intera immagine dell'aeroporto. Dato che l'oggetto di questo lavoro di tesi non era incentrato sulla grafica ma su altri aspetti e che per di più, questo meccanismo per modificare la posizione, fa parte della simulazione e quindi in un'ipotetica versione reale dell'applicazione tutto questo non sarebbe presente, è stato ritenuto più importante sviluppare in maniera migliore le parti che sono fondamentali del sistema e tralasciare un pò questi aspetti puramente di simulazione, il cui funzionamento è comunque più che soddisfacente.

Un'altra cosa molto importante che è stata simulata sono gli access point wireless (AP): a questo fine è stato creato un agente che ha lo scopo di simulare il comportamento degli AP reali. In particolare, questo agente riceverà le coordinate della posizione del dispositivo sottoforma di una coppia (x, y) e, basandosi su un file di configurazione, dove sono elencati tutti gli AP con il loro identificativo, il loro raggio di copertura e la loro posizione, risponderà indicando a quale AP si è collegati e la potenza del segnale. Per simulare il segnale degli AP non sono stati usati modelli particolari, in quanto non erano oggetto di questa tesi; quindi, per calcolare la potenza del segnale rilevata dal dispositivo si è utilizzata la formula (5.1) dove (x_0, y_0) sono le coordinate della posizione dell'AP, mentre (x, y) sono le coordinate del punto dove si trova il dispositivo:

$$0 \leq (1 - \frac{(x - x_0)^2 + (y - y_0)^2}{r^2})100 \leq 100 \tag{5.1}$$

Attraverso la formula (5.1) si calcola il valore che rappresenta la potenza rilevata dal dispositivo come un numero compreso tra 0 e 100; verrà poi scelto l'AP che è più vicino al dispositivo, cioè quello che garantisce una maggiore potenza di segnale. Ritornando al file di configurazione degli AP, questo verrà utilizzato anche dal sistema per disegnare nella mappa generale gli AP, così che possano essere visibili, anche con il loro raggio d'azione.

Durante l'implementazione del sistema è stato deciso di aggiungere un'ulteriore funzionalità, ovvero quella dei monitor informativi. Questi sono dei monitor disposti in giro per l'aeroporto e che danno informazioni sul volo che la persona deve prendere. Una volta che la persona ne autorizza l'uso attraverso le opzioni presenti nell'applicazione, ogni volta che il dispositivo verrà avvicinato a uno di questi monitor, su di esso verranno visualizzate le informazioni attinenti alle operazioni che la persona deve compiere. Non avendo a disposizione dei monitor, anche questi sono stati simulati: quindi, quando il sistema si carica va a leggere da un file di configurazione le informazioni sui monitor presenti e li disegna sulla mappa. Una volta che l'utente sposta il dispositivo vicino ad uno di essi, comparirà una finestra che visualizzerà le varie informazioni, simulando appunto il funzionamento

del monitor; passati alcuni secondi, il monitor smette di visualizzare quelle informazioni e nel caso della simulazione la finestra viene chiusa.

Altra cosa che è stata simulata è il cambiamento di alcuni dati sui voli come l'orario, lo stato, il gate e il terminal, questo perché l'applicazione permette di ricevere delle notifiche quando accadono cambiamenti di questo tipo. E' stato quindi creato un agente che, attraverso un'interfaccia grafica, permette di andare a modificare alcuni campi nella tabella dei voli e di far si che le persone che devono prendere il volo le cui informazioni sono state cambiate ricevano una notifica di avvertimento.

5.2 I componenti del sistema

In questa sezione si parla dei vari container e agenti che compongono il sistema. Verranno inizialmente presentati i container (di cui abbiamo già parlato nella sezione 4.1.3), all'interno dei quali verranno creati gli agenti; questi ultimi saranno descritti uno ad uno specificandone anche la posizione all'interno dell'architettura presentata nella sezione 3.3.

5.2.1 I container

Come è già stato spiegato nel paragrafo 4.1.3, una piattaforma JADE è formata da uno o più container; nel caso particolare di questo sistema abbiamo i seguenti container:

- *Main-Container*: è il container principale dal quale viene avviata l'intera piattaforma e nel quale sono presenti tutti i servizi di base di JADE (vedi paragrafo 4.1.3 per una spiegazione più approfondita). All'interno di esso verranno anche creati gli agenti di base per il funzionamento del sistema, vale a dire il SystemSimulationLoader, l'AirportSystemManager, l'AirportInformationManager, il ProcedureManager e l'AccessPointManager; anche il PhoneMapManager e il DbChanger vengono creati all'interno di questo container, il primo quando viene utilizzato anche il dispositivo Android e il secondo quando si vogliono fare delle modifiche al database dell'aeroporto, ad esempio al fine di simulare le variazioni dell'orario di un particolare volo;

- *Mobile-Container*: all'interno di esso vengono creati tutti gli agenti Mobile che rappresentano la versione per computer dell'applicazione;

- *AREA-i,* $1 \leq i \leq 4$: questi sono i container delle varie aree in cui è suddiviso l'aeroporto, in particolare c'è nè uno per ogni area. Nell'implementazione che è stata fatta è possibile simulare da un minimo di 1 a un massimo di 4 aree, ricordando però che simulandone meno di 4 c'è il rischio che il sistema si blocchi nel caso in cui si sposti il

dispositivo in un'area per la quale non è stato creato il container. All'interno del container AREA-i, sarà presente un agente AreaManager e poi tutti gli agenti MobileManager associati ai dispositivi presenti nell'area i-esima (questi poi migreranno fra le varie aree seguendo gli spostamenti dei dispositivi associati);

- *NomeZona-NomeMonitor*: questi container servono per il corretto funzionamento dei monitor informativi. Esempi di nomi di container di questo tipo sono *Checkin-Checkin1*, *BaggageReclaim-BaggageReclaim1* e *BaggageReclaim-BaggageReclaim2*, dove *Checkin* e *BaggageReclaim* sono i nomi delle zone dove sono presenti i monitor informativi, mentre *Checkin1*, *BaggageReclaim1* e *BaggageReclaim2* sono i nomi specifici dei monitor. All'interno di ognuno di questi container è presente un agente MonitorManager, il quale dovrà gestire l'accesso al monitor da parte degli agenti Visualizer, i quali dovranno mostrare a video delle informazioni di supporto all'utente; questi agenti migrano dai container di area, nei quali vengono creati dai MobileManager, ai container dei monitor;

- *split container*: questo è il container che viene creato quando l'agente PhoneAgent presente nel dispositivo si collega alla piattaforma. Essendo uno split container, esso è diviso in due parti: una è il BackEnd e l'altra è il FrontEnd; il BackEnd è la parte presente nel server remoto (in questo caso il computer dove c'è il main container), mentre il FrontEnd è la parte che resta sul telefono (maggiori informazioni a riguardo sono riportate nella sezione 4.1.7).

5.2.2 Gli agenti

Dopo aver descritto le componenti del sistema da un punto di vista macroscopico, passiamo ad un'analisi più precisa andando ad analizzare i vari agenti che sono distribuiti nell'intero sistema e che ne realizzano il funzionamento. Per ognuno di essi verrà spiegato lo scopo e la sua localizzazione all'interno dell'architettura del sistema spiegata nella sezione 4.1.3.

5.2.2.1 Il SystemSimulationLoader

Questo agente ha lo scopo di creare tutti gli agenti che fanno funzionare il sistema e che simulano alcuni aspetti dell'aeroporto, come è già stato spiegato nella sezione 5.1. Come argomenti, questo agente prende in ingresso il numero di aree (minimo 1 e massimo 4) e il numero di dispositivi (anche zero, in quanto è possibile crearli anche successivamente) che si vogliono simulare. Da notare che se non si simulano tutte le aree, si avrà un comportamento errato da parte del sistema non appena il dispositivo entrerà in un'area che

non è stata simulata. Il SystemSimulationLoader come prima cosa crea l'agente AirportSystemManager che rappresenta il "cervello" del sistema (vedi paragrafo 5.2.2.2), poi vengono creati il ProcedureManager (vedi paragrafo 5.2.2.9) e l'AirportInformationManager (vedi paragrafo 5.2.2.3) che sono due agenti che simulano rispettivamente gli AP e i monitor informativi (le informazioni su quest'ultimi vengono caricati da un file xml, il quale contiene per ogni monitor un identificativo, il nome della zona dove è posizionato e le coordinate (x, y) di dove si trova) ed infine gli agenti AreaManager (vedi paragrafo 5.2.2.4) ed eventualmente Mobile (vedi paragrafo 5.2.2.5) se sono stati indicati negli argomenti. Una volta che questo agente ha terminato il suo compito, cioè ha creato e fatto partire gli agenti che servivano, viene distrutto, in quanto non verrà più utilizzato.

5.2.2.2 L'AirportSystemManager

Questo agente, come già detto precedentemente, rappresenta l'entry point per tutti gli altri agenti e si trova nel server di sistema. Una volta avviato, come prima cosa controlla se il database è presente e in caso contrario lo crea, caricando i dati sulle login da un file di configurazione. Fatto ciò visualizza la mappa di sistema (un particolare della mappa è riportato in Figura 5.2) dove vengono subito disegnate nelle posizioni corrette le icone che rappresentano gli AP con il loro nome e il raggio d'azione, insieme ai monitor informativi; le informazioni che servono per disegnare queste cose vengono caricate da altri file di configurazione. Terminata questa fase iniziale, l'AirportSystemManager si mette in attesa dei messaggi da parte degli altri agenti: i messaggi possono contenere richieste da parte dei dispositivi mobili per il login al sistema, richieste da parte degli agenti AreaManager per registrarsi presso il sistema (questo serve ad indicare che sono presenti e attivi), richieste di eliminazione di un dispositivo perché ha chiuso l'applicazione o perché è offline da troppo tempo, richieste di modifica delle icone sulla mappa, richieste di modifica del database perché ad esempio un dispositivo ha cambiato area e allora bisogna aggiornare la tabella. Questo agente insomma rappresenta il punto di riferimento per quasi tutti gli agenti che popolano il sistema.

5.2.2.3 L'AirportInformationManager

Lo scopo di questo agente è quello di fornire un'interfaccia per l'accesso ai dati dell'aeroporto. Queste informazioni sono fondamentali per lo svolgimento di molte operazioni all'interno dell'aeroporto e quindi devono essere manipolate con una certa attenzione, in quanto un semplice errore potrebbe creare problemi molto più grandi, non solo per questo sistema, ma per l'intero aeroporto. L'AirportInformationManager ha quindi lo scopo di ac-

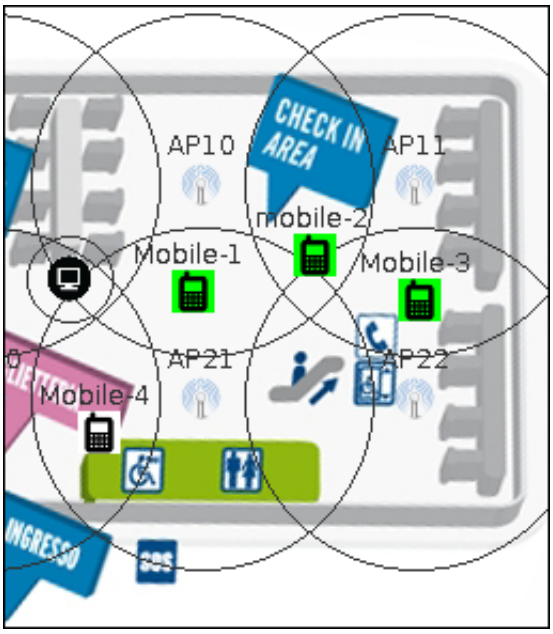

Figura 5.2: Particolare della mappa generale che viene gestita dall'AirportSystemManager

cedere a questi dati su richiesta degli altri agenti e di ritornare ad essi i valori cercati. Esso si trova nel server che gestisce l'accesso ai dati dell'aeroporto.

Una volta avviato, questo agente controlla se il database è presente e in caso contrario lo crea, inserendovi i dati riguardanti i voli, le prenotazioni, il traffico ai gate, ai terminal, ai check-in e ai nastri trasportatori per il ritiro dei bagagli; oltre a questi, carica anche le informazioni che riguardano i bar, i ristoranti e i banchi dei tour operator. Tutte queste informazioni vengono caricate da vari file di configurazione, alcuni in formato xml (quelli che richiedevano più informazioni e che quindi è stato più comodo strutturare utilizzando l'xml) e altri in semplice formato txt (in genere sono quelli che richiedevano poche informazioni). Una volta che l'agente ha caricato tutte queste informazioni, si mette in attesa di messaggi da parte degli altri agenti; tra le varie richieste che possono arrivare, la più importante è forse quella per richiedere i dati "live". Infatti, come verrà spiegato più in dettaglio nella sezione riguardante l'agente MobileManager (vedi paragrafo 5.2.2.6), nelle procedure che vengono fornite sono presenti solo i nomi delle informazioni che devono comparire all'interno di un'operazione, ma non ci sono i dati veri e propri; per cercare di spiegare meglio questa cosa, prendiamo ad esempio l'operazione di recarsi al banco del check-in. In questa operazione viene dato come informazione il numero del banco che sta imbarcando i bagagli per il proprio volo; indipendentemente dagli agenti che sono coinvolti nelle operazioni (verranno spiegati più avanti), una volta che questa operazione

viene fornita, al suo interno è presente solo il campo "check-in" ma non c'è scritto qual'è il numero del banco, perché dipende dal volo e dal giorno. Quindi, ci sarà un agente che richiederà all'AirportInformationManager le informazioni "live" riguardanti il banco check-in di un certo volo in una certa data, questo le recupererà e le invierà all'agente che le aveva richieste. Nella realtà, l'AirportInformationManager riceve una lista di oggetti (come ad esempio Flight, Checkin, Terminal, per maggiori informazioni vedi il paragrafo 5.4.2) che si vogliono riempire con i dati "live", ed è proprio quello che viene fatto; grazie anche al codice identificativo della persona che ha fatto la richiesta (che viene inviato assieme alla lista di oggetti), è possibile risalire a tutte le informazioni riguardanti il suo volo, le quali verranno messe all'interno degli oggetti appositi e inviate all'agente che ne aveva fatto richiesta, per poter così essere usate con tutte le informazioni corrette.

Un'altra richiesta importante che può arrivare all'AirportInformationManager è quella di registrazione, o di cancellazione, all'alert service: questo servizio permette di ricevere notifiche nel caso in cui ci siano dei cambiamenti sulle informazioni del volo che la persona deve prendere. Al momento gli aggiornamenti che si possono ricevere riguardano alcuni dati dell'oggetto Flight, come ad esempio il cambio d'orario oppure di stato del volo, e dell'oggetto Checkin, come ad esempio il numero del banco. Questo servizio funziona nel seguente modo: il MobileManager invia all'AirportInformationManager una lista di oggetti (ad esempio Flight), dei quali vuole ricevere aggiornamenti nel caso in cui subiscano variazioni. Non appena riceve questa lista, l'AirportInformationManager salva queste informazioni in una struttura dati. Nel momento in cui avviene una modifica, un messaggio avverte l'AirportInformationManager il quale andrà a ricercare all'interno della struttura apposita se qualche agente è interessato alle modifiche appena avvenute. In caso positivo, viene inviato un messaggio agli agenti interessati, specificando al suo interno che cosa è stato variato.

Altre richieste possono riguardare la lista delle attività commerciali (negozi, bar, ristoranti) oppure tutte le informazioni riguardanti una specifica attività, poi può essere richiesta la lista delle posizioni dei monitor informativi, oppure quella dei voli in una certa data o di uno specifico volo (queste ultime possono arrivare perché dai dispositivi è possibile vedere una tabella che elenca tutti i voli in partenza o in arrivo di una certa giornata).

5.2.2.4 L'AreaManager

A capo di ogni area c'è un'agente di questo tipo, il quale ricopre le funzioni di amministratore e di controllore dei dispositivi che si trovano all'interno di essa; un agente AreaManager è presente in ogni server di area. Anche questo agente non appena si è avviato, controlla la presenza del database e se questo non c'è lo crea; subito dopo si registra con l'AirportSystemManager, indicando in questo modo che è presente ed attivo. Fatto ciò si mette in

attesa di riceve messaggi: possono arrivare richieste di registrazione da parte dei dispositivi mobili che entrano nell'area oppure di cancellazione da parte di quelli che escono dell'area per passare in un'altra oppure che vengono eliminati perché offline da troppo tempo o più semplicemente perché hanno chiuso l'applicazione. Inoltre ci possono essere richieste per cambiare lo stato di un dispositivo, ad esempio da online a offline o viceversa; tutti questi cambiamenti vengono registrati su un database.

5.2.2.5 Il Mobile

Questo agente, che rappresenta il dispositivo mobile nella versione per i computer, è dotato di interfaccia grafica per permette l'utilizzo dell'applicazione all'utente. Attraverso l'interfaccia (vedi Figura 5.3), l'utente può abilitare e disabilitare il wifi del dispositivo (il funzionamento del wifi viene simulato) e collegarsi al sistema. Una volta che viene premuto il tasto "Connetti al sistema", comparirà un pop-up dove l'utente dovrà inserire l'username e la password che gli sono state assegnate in precedenza. E' possibile anche accedere in modalità "ospite", usando come username e password la parola "guest"; questa modalità è stata pensata per quelle persone che si trovano in aeroporto che non devono prendere un volo, ma che magari stanno aspettando l'arrivo di parenti o amici. Nella modalità "ospite" è possibile accedere solo ad alcune sezioni dell'applicazione, come ad esempio alla pagina che permette di vedere le informazioni sui voli in partenza o in arrivo in un dato giorno, oppure alla sezione riguardante i servizi che l'aeroporto mette a disposizione, come bar, ristoranti e negozi. Una volta effettuato il login, è possibile accedere a diverse sezioni: innanzitutto c'è la mappa, dalla quale è possibile simulare lo spostamento all'interno dell'aeroporto, come già spiegato nel paragrafo 5.1; poi c'è la sezione riguardante i task, ovvero dove saranno presenti tutte le indicazioni per eseguire le varie operazioni prima di imbarcarsi sull'aereo oppure all'arrivo a destinazione (questa sezione verrà spiegata nello specifico più avanti); nella sezione "Voli" si possono vedere tutte le informazioni riguardanti i voli in una certa data, con la possibilità di filtrare anche i risultati della ricerca; infine c'è la sezione riguardante le attività commerciali presenti all'interno dell'aeroporto, che permette di eseguire delle ricerche selezionando la categoria e la posizione (ad esempio prima dei controlli di sicurezza oppure dopo) dell'attività, ottenendo così informazioni specifiche sull'attività scelta, come gli orari di apertura/chiusura e una breve descrizione.

La registrazione al sistema avviene in due fasi: nella prima vengono inviate all'AirportSystemManager lo username e la password inserite dall'utente per verificare se sono corrette; in caso positivo, l'AirportSystemManager invia la conferma all'agente Mobile e nella risposta indica anche l'area in cui il dispositivo si trova (questo servirà nella fase successiva di registrazione presso l'AreaManager), i codici identificativi delle procedure di partenza e

Figura 5.3: Interfaccia della "Home" dell'agente Mobile

arrivo ed infine il codice personale (utilizzati per fare le richieste al ProceduReManager e all'AirportInformationManager, al fine di scaricare la procedura corretta e di ottenere i dati "live"). La seconda fase consiste nel registrarsi presso l'AreaManager a capo dell'area in cui ci si trova: una volta inviata la richiesta, l'AreaManager provvederà a creare l'agente MobileManager associato al dispositivo che si occuperà di svolgere molte operazioni per conto del Mobile (per maggiori spiegazioni vedere il paragrafo 5.2.2.6 riguardante l'agente MobileManager). Fatto ciò, il Mobile riceverà una conferma che tutte queste operazioni sono andate a buon fine da parte del MobileManager ad esso associato. La fase di connessione è spiegata con maggiore dettaglio nella paragrafo 5.3.1, riguardante le interazioni che avvengono nel momento in cui un agente Mobile si collega al sistema.

Come è stato detto, quest'agente ha un'interfaccia grafica: per far si che l'oggetto che rappresenta l'interfaccia grafica interagisse con l'agente, è stato sufficiente passare ad esso il riferimento dell'agente; infatti visto che i behaviour vengono messi nella coda mantenuta dall'agente ed eseguiti uno dopo l'altro e che quest'ultimo vive su un thread separato, non c'è alcun problema di sincronizzazione se si aggiungono behaviour da un oggetto esterno. Viceversa, per modificare gli elementi della GUI dal thread dell'agente in risposta agli avvenimenti di quest'ultimo è consigliabile utilizzare un oggetto di tipo `Runnable`, mettere al suo interno il codice che va a modificare la GUI e poi metterlo in coda, attraverso il metodo `invokeLater` della classe `SwingUtilities`, all'`EventDispatchThread` che lo eseguirà non appena

possibile.

Veniamo ora alla sezione principale che mette a disposizione la GUI dell'agente Mobile, ovvero la sezione "Lista task": in questa sezione è possibile avere tutte le informazioni a riguardo delle varie operazioni che vanno fatte prima di salire sull'aereo oppure una volta arrivati, come il check-in, i controlli di sicurezza o il ritiro dei bagagli. La Figura 5.4 mostra l'interfaccia che si presenta all'utente non appena entra nella sezione riguardante i task: in alto viene indicato il titolo dell'operazione e una descrizione di quello che si dovrà fare; subito sotto, vengono date delle informazioni relative alla posizione dove ci si trova, le quali segnaleranno, anche cambiando colore, quando ci si trova nelle vicinanze della destinazione, cioè del punto in cui si deve compiere la particolare operazione e, sempre cambiando colore, segnaleranno anche se non ci si trova più nelle vicinanze di essa (in questo modo si supplisce un pò al fatto che non è presente un navigatore vero e proprio, che comunque non era l'obiettivo di questo progetto, inserendo delle notifiche che daranno delle indicazioni, anche se non molto precise, sulla posizione dell'utente in modo da potergli comunque permettere di orientarsi o perlomeno di sapere se si trova nella zona corretta); poco più in basso troviamo il riquadro delle informazioni, la cui natura dipende dall'operazione che dobbiamo fare; infine troviamo i suggerimenti, che possono essere utili se ci si trova per la prima volta in un'aeroporto, oppure possono fungere da promemoria, per essere sicuri di non aver dimenticato nulla; infine troviamo i bottoni che permettono di chiudere questa sezione oppure di passare alla prossima operazione o di tornare alla precedente (è possibile scorrere tutte le operazioni in avanti e indietro indipendentemente dalla posizione in cui ci si trova).

5.2.2.6 Il MobileManager

Questo agente è il responsabile del dispositivo mobile, mantiene per esso le informazioni sulla procedura che deve seguire e ne inoltra le richieste ai vari agenti del sistema. I motivi per cui si è voluto creare questo agente sono i seguenti: l'applicazione che si è realizzata si dovrà poter installare su diversi dispositivi (in questo lavoro di tesi è stata sviluppata per computer e per telefoni con sistema operativo Android), ognuno con un suo sistema operativo e con problematiche diverse. E' inevitabile dover fare quindi il porting dell'applicazione sulle varie piattaforme, ma la cosa diventa più semplice e veloce se il numero di funzionalità implementate sull'agente che andrà nel dispositivo è limitato e riguarda quasi unicamente l'implementazione dell'interfaccia grafica nella piattaforma specifica. Questo si ottiene facendo in modo che la maggior parte delle funzionalità siano implementate sull'agente MobileManager e che quindi l'agente Mobile, il quale dovrà essere modificato a seconda della piattaforma dove verrà installato, sia poco più di un front end grafico. La seconda motivazione, è che l'applicazione dovrà girare anche

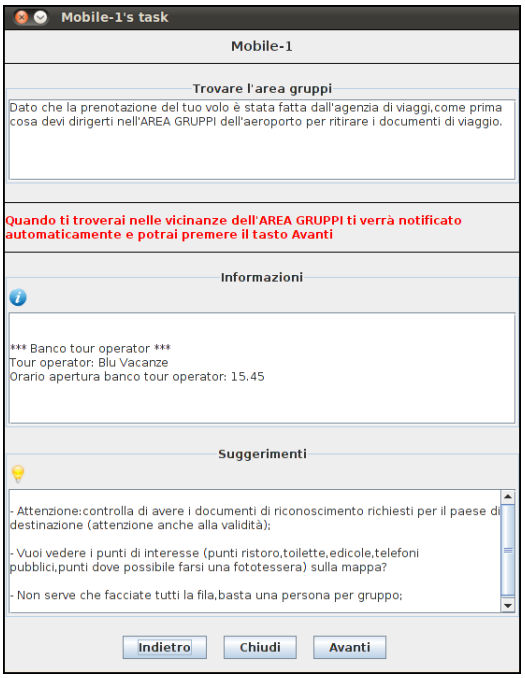

Figura 5.4: Interfaccia della sezione "Task" dell'agente Mobile

su dispositivi mobili come smartphone, i quali hanno limitate capacità di calcolo e limitata energia disponibile; è quindi importante riuscire a mantenere "leggera" l'applicazione, cercando di evitare il più possibile operazioni pesanti sotto il punto di vista del calcolo per non rallentare l'intero sistema operativo o consumare eccessivamente la batteria. Questa seconda motivazione rende ancora più importante la decisione di spostare il più possibile le operazioni "pesanti" sul MobileManager, il quale risiederà nel server dell'area dove si trova il dispositivo, e si muoverà con esso quando cambierà area (quando il dispositivo lascia un'area per entrare in un'altra, il MobileManager migra anch'esso nella nuova area, informando dell'avvenuto spostamento sia l'AreaManager dell'area dove si trovava in precedenza, sia quello dell'area dove arriva). Un altro vantaggio di questa scelta progettuale è che, quando il dispositivo diventa offline, i dati vengono comunque mantenuti dal MobileManager e resi di nuovo disponibili non appena ritorna online. Per sapere se il dispositivo è ancora attivo, questo invia dei messaggi di keep alive al suo MobileManager a intervalli temporali; se questi messaggi non vengono più visti per un certo periodo, allora il dispositivo viene considerato offline e questo viene notificato all'AreaManager e all'AirportSystemManager; infine, se il dispositivo non ritorna online entro un certo periodo, allora il MobileManager si elimina, altrimenti se il dispositivo torna ad inviare i messaggi di keep alive, il MobileManager cambia il suo stato in online e in entrambi i casi lo notifica all'AreaManager e all'AirportSystemManager. Da sottolineare

il fatto che se il dispositivo torna online prima che il suo MobileManager venga eliminato, non è necessario che l'utente rifaccia il login, in quanto le informazioni sono già tutte presenti.

Una volta avviato, quest'agente si registra con l'AreaManager dell'area dove si trova e subito dopo richiede al Mobile il codice della procedura associata e il codice personale, al fine di inviare la richiesta della procedura al ProcedureManager e la richiesta dei dati "live" all'AirportInformationManager. Una volta che ha ottenuto la procedura con i tutti i dati in tempo reale, è pronto per inviare le singole operazioni al Mobile non appena questo ne farà richiesta, cioè quando l'utente seleziona l'operazione successiva o precedente dall'interfaccia grafica (vedi paragrafo 5.2.2.5); in questo modo il Mobile dovrà tenere in memoria unicamente l'operazione che l'utente sta visualizzando e non tutta la procedura. Oltre ai keep alive, il Mobile invia al MobileManager anche l'identificativo dell'area dove si trova (il quale gli viene comunicato dall'AccessPointManager), perché se questa cambia, allora il MobileManager deve migrare a sua volta nella nuova area dove si trova il Mobile.

Altra operazione importante riguarda il recupero delle informazioni "live" necessarie nelle operazioni della procedura: per fare ciò, il MobileManager scorre tutte le operazioni contenute all'interno della procedura e da ognuna di esse recupera le informazioni (come ad esempio Flight, cioè le informazioni sul volo) che dovranno essere visualizzate e le salva su una lista; questa lista, assieme al codice personale che servirà per trovare i dati sul volo che deve prendere l'utente, verranno poi inviati all'agente AirportInformationManager, il quale andrà a leggere dal database dell'aeroporto le informazioni necessarie e le invierà come risposta al MobileManager, che a questo punto avrà tutte le informazioni pronte per essere inviate al Mobile non appena questo ne farà richiesta.

Un'altra funzione che svolge il MobileManager è quella di inoltrare le notifiche al Mobile. Queste notifiche, come già spiegato, riguardano le modifiche al database dell'aeroporto, ad esempio perché è cambiato l'orario di partenza o di arrivo di un aereo. Per ricevere queste notifiche è necessario che il MobileManager richieda di registrarsi a questo servizio all'AirportInformationManager, specificando gli oggetti (come ad esempio Flight) dei quali è interessato a ricevere notifiche in caso di variazioni. Per fare ciò, una volta che ha ha ricevuto i dati "live" li scorre in modo da vedere quali informazioni verranno usate ed in particolare se tra queste ce ne sono certe di cui si possono ricevere le notifiche in caso di aggiornamenti; se queste ci sono, allora prepara un messaggio nel quale specifica gli oggetti di cui vuole ricevere gli aggiornamenti e lo invia all'AirportInformationManager il quale si preoccuperà di inviare le notifiche nel caso avvengano delle modifiche ai dati a cui è interessato il MobileManager.

Quando il MobileManager riceve questi messaggi di notifica esegue due operazioni: la prima è aggiornare i dati "live" con le modifiche che sono

state apportate e la seconda è inoltrare il messaggio al Mobile, il quale si preoccuperà di mostrare la notifica all'utente. Questa funzione risulta particolarmente utile, in quanto informa in tempo reale delle variazioni che possono capitare e che è bene conoscere il prima possibile.

E' stato detto in precedenza che l'interfaccia del dispositivo mette a disposizione anche una sezione per vedere le varie attività commerciali presenti all'interno dell'aeroporto e un'altra per vedere una tabella con i voli di una certa data; in entrambi i casi, le informazioni sono presenti nel database dell'aeroporto e quindi sarà l'agente AirportInformationManager che dovrà recuperarle. Una volta che l'utente ha scelto cosa vuole visualizzare, la richiesta viene inviata al MobileManager il quale la inoltrerà all'AirportInformationManager; questo recupereà le informazioni richieste e le invierà come risposta al MobileManager che le inoltrerà al Mobile. Il motivo per cui c'è il MobileManager a fare da ponte a questa interazione riguarda sempre il voler semplificare il più possibile l'agente Mobile: in questo modo infatti anche se la struttura del sistema dovesse cambiare per qualche motivo in un'implementazione futura, comunque il dispositivo dovrebbe comunicare solo con il MobileManager e quindi non dovrebbe venire modificato; questo risulta estremamente comodo se si pensa che l'applicazione potrebbe essere già stata distribuita a diversi utenti e la modifica della struttura del sistema non richiede alcuna modifica dell'applicazione stessa.

Altra cosa che questo agente controlla è la vicinanza ai monitor informativi: se l'utente ha acconsentito al loro utilizzo e si trova in prossimità di uno di essi allora vengono selezionate le informazioni da visualizzare in quel monitor, a seconda della posizione dove questo si trova, queste poi vengono passate come argomento al nuovo agente Visualizer che viene creato dal MobileManager per poi essere inviato all'host che gestisce il monitor, dove sarà presente l'agente MonitorManager che si occuperà di gestire l'accesso al monitor da parte dei vari agenti Visualizer (vedi il paragrafo 5.2.2.13 per il MonitorManager e il paragrafo 5.2.2.12 per i Visualizer).

5.2.2.7 Il PhoneAgent

Questo agente è la versione per il telefono con piattaforma Android dell'agente Mobile, cioè ha ha le stesse funzioni del Mobile, però realizzate in maniera diversa (nella Figura 5.5 è possibile vedere uno screenshot della sezione relativa alle indicazioni sulle operazioni da compiere). La differenza nell'implementazione è dovuta al fatto che questo agente si trova all'interno di un'applicazione Android, un ambiente totalmente diverso da quello in cui si trova l'agente Mobile.

Una differenza importante riguarda la connessione di questo agente al main container: come è già stato spiegato nel paragrafo 4.1.7, è possibile eseguire JADE in modalità split-container oppure stand-alone, in questo caso è stata scelta la modalità split-container che è la preferita per la maggior parte

Figura 5.5: Interfaccia della sezione relativa ai task dell'agente PhoneAgent

dei dispositivi Android viste le loro limitate capacità. Una volta importata nel progetto la libreria `JadeAndroid.jar`, è stato necessario richiedere i permessi per utilizzare il servizio `jade.android.MicroRuntimeService` aggiungendo al file `AndroidManifest.xml` la seguente riga:

```
<service android:name="jade.android.MicroRuntimeService"/>
```

E' stato poi necessario fare un bind tra l'applicazione e il servizio richiesto passando alcuni parametri di configurazione; una volta collegati correttamente al servizio, si è proceduto alla creazione del container condiviso e infine alla creazione dell'agente PhoneAgent.

Un'altra cosa in cui questo agente differisce con il Mobile, è la gestione della mappa per la simulazione del movimento. Dal menu dell'applicazione è infatti possibile aprire una finestra che visualizza la mappa dell'aeroporto e dalla quale è possibile simulare lo spostamento del dispositivo all'interno dell'aeroporto; la particolarità è che questa finestra viene aperta nel computer dove è presente il main container e non sul telefono (la motivazione di questa scelta è già stata spiegata nella sezione 5.1). Per gestire questo meccanismo viene utilizzato l'agente PhoneMapManager (vedi paragrafo 5.2.2.8), il quale farà in modo di nascondere questa differenza rispetto al Mobile, permettendo una visione uniforme da parte del sistema degli agenti Mobile e PhoneAgent.

5.2.2.8 Il PhoneMapManager

Questo agente ha lo scopo di visualizzare e gestire la mappa per conto del PhoneAgent. Quando l'utente preme il tasto della mappa sul dispositivo,

viene inviato un messaggio al PhoneMapManager, il quale si trova nel computer dove è presente il main container, e mostrerà a video una finestra che visualizza la mappa dell'aeroporto e dove è indicata tramite un quadrato nero, la posizione del dispositivo. Il funzionamento della mappa è uguale a quella dell'agente Mobile e si rimanda quindi al paragrafo 5.2.2.5 per avere maggiori informazioni a riguardo; l'unica differenza è che quando viene selezionata la nuova posizione, questa dovrà essere comunicata non solo all'AccessPointManager che darà poi le informazioni riguardo all'AP al quale il dispositivo è collegato, ma anche al PhoneAgent, in modo che possa aggiornare le coordinate della posizione memorizzate nelle proprie variabili locali.

L'obiettivo del PhoneMapManager è quello di rendere trasparente all'intero sistema questo meccanismo della mappa che viene aperta sul computer invece che direttamente sul dispositivo: per fare ciò, quando vengono inviate le nuove coordinate all'AccessPointManager e anche all'AirportSystemManager (l'invio a quest'ultimo è perché esso deve aggiornare la posizione del dispositivo nella mappa di sistema), viene impostato come mittente del messaggio il PhoneAgent, anche se in realtà il messaggio viene inviato dal PhoneMapManager; in questo modo poi l'AccessPointManager risponderà direttamente al PhoneAgent, comunicandogli le informazioni riguardanti l'AP a cui è collegato.

5.2.2.9 Il ProcedureManager

Questo agente è il responsabile delle procedure aeroportuali e il suo scopo è quello di inviare tali procedure agli agenti che ne fanno richiesta; esso è collocato all'interno del server delle procedure. Al suo avvio vengono caricate tutte le procedure presenti in un file xml di configurazione e salvate all'interno di una lista; ogni procedura è identificata da un codice che ne renderà possibile il recupero nelle fasi successive. Fatto ciò, il ProcedureManager si mette in attesa di richieste da parte dei vari MobileManager per lo scaricamento di procedure: queste richieste dovranno contenere il codice identificativo della procedura (questo codice viene comunicato al MobileManager dal Mobile, il quale a sua volta l'ha ottenuto dall'AirportSystemManager al momento della registrazione) grazie al quale il ProcedureManager recupererà quella corretta e la invierà al MobileManager che ne ha fatto richiesta.

5.2.2.10 L'AccessPointManager

Questo è uno degli agenti che simulano un qualche aspetto dell'aeroporto e, in particolare, l'AccessPointManager simula il funzionamento degli access point wireless (AP). Dopo il suo avvio, l'agente si mette in attesa di messaggi da parte dei Mobile che richiedono informazioni sull'AP al quale sono collegati: nella richiesta sono presenti le coordinate (x, y) del punto di cui

si trova il dispositivo, le quali vengono utilizzate per calcolare attraverso la formula (5.1) quale sia l'AP migliore, cioè quello che dà una maggiore potenza di segnale; una volta trovato, viene comunicato al Mobile che ne aveva fatto richiesta, insieme alle informazioni riguardanti la potenza del segnale.

5.2.2.11 Il DbChanger

Questo è un altro agente di simulazione, in particolare il DbChanger simula il cambiamento delle informazioni riguardati i voli. Come capita nella realtà, dove i voli possono subire variazioni negli orari di partenza o di arrivo, oppure nel numero del gate o del terminal, anche in questa applicazione si possono simulare eventi di questo tipo. L'agente DbChanger mette a disposizione un'interfaccia grafica tramite la quale è possibile andare a cambiare alcuni campi del database dell'aeroporto; una volta che l'aggiornamento è stato effettuato, il DbChanger invia un messaggio all'AirportInformationManager per informarlo dell'avvenuta modifica (sarà poi l'AirportInformationManager ad inviare un messaggio per notificare l'aggiornamento ai vari MobileManager che sono interessati).

5.2.2.12 Il Visualizer

Lo scopo di questo agente è quello di visualizzare, tramite i monitor informativi, delle informazioni utili al passeggero che vi si trova nelle vicinanze. L'idea è di fare in modo che se anche l'utente non ha il dispositivo sottomano, ma magari lo tiene in tasca, può usufruire dell'assistenza, anche se in minima parte. Se l'utente accetta di utilizzare questa funzione, ogni volta che si avvicinerà ad uno dei monitor informativi sparsi per l'aeroporto, verranno eseguite le seguenti operazioni: l'agente MobileManager, dopo aver identificato a quale monitor si trova vicino il passeggero, crea un agente Visualizer e gli passa come argomento le informazioni che dovrà visualizzare; identificare a quale monitor si trova vicino l'utente è importante perché le informazioni visualizzate dipendono da dove si trova, perché non ha senso ad esempio mostrare il numero del banco del check-in se il monitor, e di conseguenza il passeggero che gli si trova vicino, si trovano nell'area duty free. Le informazioni che verranno visualizzate sono del tutto simili a quelle disponibili nel dispositivo e non conterranno alcun dato personale, quindi non saranno collegabili alla specifica persona. Una volta che l'agente Visualizer è stato creato, esso migra nel computer che gestisce il monitor; quando arriva a destinazione, invia un messaggio di `lock` al MonitorManager (vedi paragrafo 5.2.2.13) per richiedere l'utilizzo esclusivo del monitor. Se non c'è nessun'altro agente che lo sta usando, allora gli verrà data la conferma per l'accesso, altrimenti la richiesta verrà messa in una coda e l'agente dovrà attendere il suo turno. Una volta che ha ottenuto il permesso, il Visualizer

mostrerà a video le informazioni per alcuni secondi, dopodiché invierà un messaggio di `unlock` al MonitorManager e si eliminerà.

5.2.2.13 Il MonitorManager

Questo agente è posizionato nei computer che gestiscono i monitor informativi ed ha lo scopo di gestire l'accesso al monitor che sarà condiviso fra tutti gli agenti Visualizer (vedi paragrafo 5.2.2.12), i quali devono mostrare a video le informazioni che contengono. Non appena gli agenti Visualizer arrivano nel container presente nel computer che gestisce il monitor, mandano un messaggio al MonitorManager per richiedere l'accesso al monitor; questo, se il monitor è occupato, mette la richiesta in una coda gestita con politica FIFO, altrimenti ne concede subito l'utilizzo. Non appena l'agente Visualizer ha terminato di mostrare le informazioni, manda un messaggio al MonitorManager per dire che lascia libero il monitor, e viene quindi inviato un messaggio all'agente che aveva mandato la richiesta successiva, consentendogli di visualizzare le proprie informazioni.

5.3 Le interazioni fra gli agenti

Gli agenti, come è già stato spiegato nella sezione 4.1 riguardante la piattaforma JADE, comunicano fra loro attraverso lo scambio di messaggi asincroni. Ogni agente ha una coda FIFO dove i messaggi in arrivo vengono memorizzati, dipenderà poi dalle scelte dello sviluppatore dell'agente se questo andrà a leggerli o meno. In questa sezione si vogliono andare ad analizzare le fasi più importanti delle interazioni fra gli agenti del sistema; in particolare si andranno a commentare le interazioni che avvengono al momento della registrazione di un nuovo agente Mobile al sistema (paragrafo 5.3.1), quando l'utente chiude l'applicazione e viene eliminato l'agente Mobile (paragrafo 5.3.2) ed infine il download della procedura da parte del MobileManager (paragrafo 5.3.3).

5.3.1 La registrazione di un nuovo agente Mobile

Cominciamo dall'inizio, ovvero dallo scambio di messaggi, rappresentato nella Figura 5.6, che avviene nel momento in cui un agente Mobile si registra presso l'AirportSystemManager. Come prima cosa, dopo che l'utente ha effettuato il login, il Mobile invia all'AirportSystemManager il messaggio n.1 dove indica il suo nome identificativo (ad esempio Mobile-1), l'AP a cui è collegato, la username e la password inserite dall'utente. L'AirportSystemManager, quando riceve questo messaggio, controlla nel database se la login è corretta e in caso affermativo risponde con il messaggio n.2 confermando l'avvenuta registrazione ed indicando anche i codici identificativi delle procedure di partenza e di arrivo associate all'utente, il codice personale utile

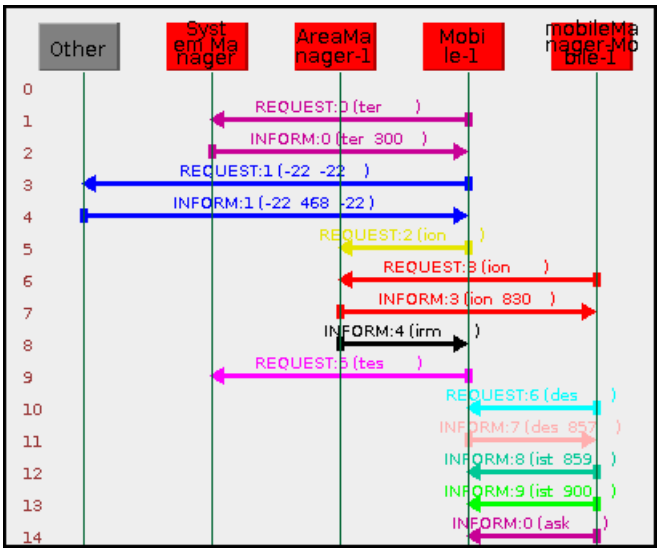

Figura 5.6: Interazioni fra gli agenti al momento della registrazione di un nuovo agente Mobile

per recuperare i dati "live" nelle fasi successive ed infine il numero di area nel quale il dispositivo si trova; quest'ultima informazione che viene data, serve al Mobile per contattare l'AreaManager corretto e registrarsi anche con esso. Infatti i messaggi n.3 e n.4 sono rispettivamente la richiesta di avere l'indirizzo dell'agente che fornisce il servizio *area service-n*, dove *n* è il numero dell'area in cui si trova il dispositivo e la risposta da parte del Directory Facilitator contenente l'indirizzo dell'agente che eroga quel servizio, ovvero l'AreaManager di quella particolare area. Una volta che il Mobile ha ottenuto questo indirizzo, gli invia il messaggio n.5 nel quale comunica la sua presenza all'interno dell'area; quando L'AreaManager riceve questo messaggio, crea un agente MobileManager e lo associa al Mobile, passandogli come argomento il nome identificativo di quest'ultimo. Non appena il MobileManager si avvia, manda il messaggio n.6 all'AreaManager per confermare l'avvenuta creazione e questo gli risponde con il messaggio n.7, dove conferma a sua volta di aver aggiornato le informazioni del database e informa anche se l'area in cui ci si trova fa parte della zona partenze o arrivi; questa informazione è fondamentale, in quanto da essa il MobileManager sa se deve chiedere la procedura per la partenza o per l'arrivo. L'AreaManager conferma anche al Mobile, con il messaggio n.8, che tutta la procedura è andata a buon fine, informandolo anche sul nome del MobileManager ad esso associato. Da questo momento in poi il Mobile parlerà solo con il MobileManager, eccetto che per i messaggi riguardanti la simulazione, come ad esempio per la comunicazione delle coordinate della posizione (messaggio n.9), i quali verranno inviati direttamente all'AirportSystemManager. Il messaggio n.9

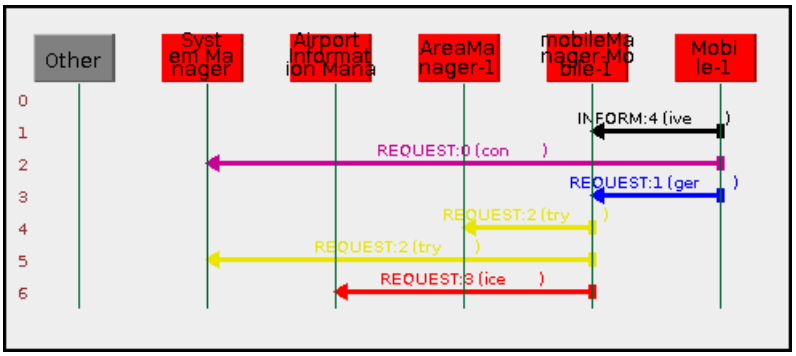

Figura 5.7: Interazioni fra gli agenti al momento dell'eliminazione di un agente Mobile

che si vede in figura riguarda un aspetto della simulazione e in particolare contiene le informazioni per disegnare correttamente l'icona del dispositivo sulla mappa di sistema; i messaggi n.10 e n.11 sono invece rispettivamente la richiesta e la risposta dei codici identificativi delle procedure di partenza e di arrivo e il codice personale per il recupero dei dati "live". I messaggi dal n.12 al n.14 sono informazioni che il MobileManager manda al Mobile perché quest'ultimo possa riempire i campi delle opzioni che l'utente può scegliere quando vuole vedere le varie attività commerciali o i voli di una certa data.

5.3.2 La chiusura dell'applicazione da parte dell'utente

Andiamo ora ad analizzare lo scambio di messaggi che avviene nel momento in cui l'utente decide di chiudere l'applicazione, riportato in Figura 5.7. Prima di iniziare, si vuole sottolineare il fatto che in questo caso si considera l'eliminazione di un agente Mobile, che è l'agente per la versione per computer dell'applicazione, ma che l'eliminazione di un agente PhoneAgent, cioè l'agente per la versione per piattaforma Android, è esattamente la stessa, cambiano solo alcuni messaggi di simulazione che però non sono rilevanti.

Il messaggio n.1 è un messaggio di keep alive inviato dal Mobile al MobileManager per indicare che è ancora attivo (questi messaggi vengono inviati ad intervalli di tempo regolari). Prima che venga inviato un nuovo messaggio di keep alive, l'utente chiude l'applicazione e questo fa si che viene inviato il messaggio n.3 dal Mobile al MobileManager per richiedere l'eliminazione dal sistema; il messaggio n.2 invece è un messaggio che serve alla simulazione, in quanto indica all'AirportSystemManager di eliminare l'icona che rappresenta il dispositivo dalla mappa di sistema. I messaggi seguenti, ovvero il n.4 e il n.5 sono le richieste fatte dal MobileManager di eliminazione delle informazioni riguardanti il Mobile rispettivamente all'AreaManager e all'AirportSystemManager; infine, il messaggio n.6 è la richiesta di rimo-

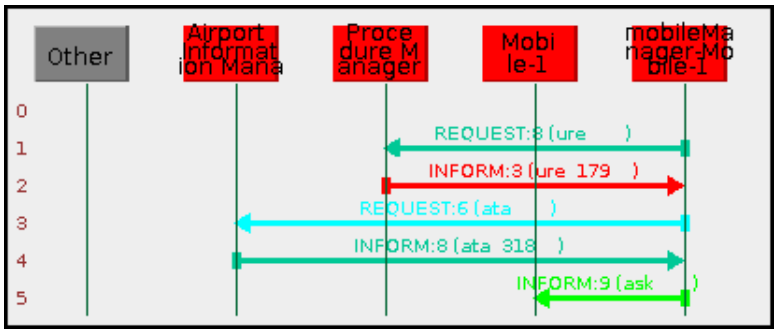

Figura 5.8: Interazioni fra gli agenti al momento del download della procedura

zione dal servizio alert service del dispositivo che è appena stato eliminato, altrimenti le notifiche in caso di cambiamenti nelle informazioni del volo, continuavano ad essere inviate ad un dispositivo che in realtà non esisteva più. Una cosa da notare è che in questa interazione non sono presenti messaggi di risposta alle varie richieste che vengono fatte: il motivo è che gli agenti che inviano queste richieste, una volta terminato l'invio, si eliminano; ad esempio, dopo il messaggio n.3, l'agente Mobile cessa di esistere, così come anche il MobileManager dopo il messaggio n.6 e quindi non è più necessario inviare dei messaggi di conferma delle avvenute cancellazioni (se per caso qualcosa andasse storto e delle informazioni rimanessero memorizzate nel database, non ci sarebbero comunque problemi per il sistema).

5.3.3 Il download della procedura

Nell'interazione presente nella Figura 5.8, è possibile vedere lo scambio di messaggi che avviene nel momento in cui il MobileManager va a recuperare la procedura per il dispositivo associato; si ricorda che è necessario fare due richieste per ottenere la procedura completa con i dati "live", in quanto con la prima richiesta si ottiene la procedura contenente però solo i nomi delle informazioni che devono essere recuperate, mentre con la seconda richiesta si ottengono i dati "live" da inserire nella procedura, che a quel punto sarà pronta per essere utilizzata dal Mobile (o dal PhoneAgent).

Il messaggio n.1, che è la richiesta di download della procedura, contiene il codice identificativo di quest'ultima, il quale servirà al ProcedureManager per recuperare la procedura corretta e inviarla indietro al MobileManager, attraverso il messaggio n.2. A questo punto il MobileManager ha ottenuto la procedura priva dei dati "live", ma con solo il nome delle informazioni che sono necessarie; a titolo di esempio si riporta la lista delle informazioni necessarie di un'operazione che fa parte di una delle procedure, così come viene ottenuta a questo punto dello scambio di messaggi:

```
<task>
    ...
    <listOfInformations>
        <information>CheckIn</information>
        <information>Flight</information>
    </listOfInformations>
    ...
</task>
```

in questo caso dovranno essere recuperate le informazioni "live" riguardati il check-in e il volo. Il messaggio n.3 richiede esattamente questi dati, inviando una lista di oggetti (come ad esempio CheckIn e Flight, i quali al loro interno hanno varie variabili locali che andranno riempite con i dati reali) all'AirportInformationManager, il quale dovrà riempirli con i dati veri e propri; nel messaggio viene anche inserito il codice identificativo dell'utente, che servirà per capire quale volo deve prendere, o ha preso, l'utente e quindi recuperare i dati corretti. Il messaggio n.4 è la risposta dell'AirportInformationManager il quale, dopo aver recuperato tutti i dati richiesti dal database dell'aeroporto e averli inseriti dentro gli oggetti corretti, li invia al MobileManager con il messaggio n.4. Infine, con il messaggio n.5 il MobileManager invia al Mobile (o al PhoneAgent), la prima operazione della procedura completa dei dati "live"; quindi se prima l'oggetto Flight aveva al suo interno un campo *"time"* che rappresenta l'orario di partenza dell'aereo, adesso quel campo non sarà più vuoto ma conterrà ad esempio *1550*, cioè l'orario vero e proprio.

5.4 L'ontologia usata

Nel paragrafo 4.1.5 si è parlato dell'ontologia e dei content language, entrambi strumenti molto utili quando si tratta di dover inviare informazioni strutturate tra gli agenti. Per questo progetto non è stato implementato un nuovo linguaggio ma è stato utilizzato quello suggerito dalle varie guide di JADE, ovvero il linguaggio SL, mentre in questa sezione si parlerà dell'ontologia che è stata creata ed utilizzata nel sistema, in quanto essa ha giocato un ruolo importante nella semplificazione delle comunicazioni.

Va comunque specificato che non in tutte le comunicazioni fra agenti si è fatto ricorso all'uso dell'ontologia, in quelle più semplici infatti, dove magari l'informazione era composta da un solo valore, si è utilizzata una semplice stringa.

5.4.1 Come è stata creata l'ontologia

Prima di parlare del contenuto dell'ontologia di questo progetto, andiamo a vedere come praticamente è stato sviluppato questa specie di dizionario

dove sono presenti tutti i concetti e tutte le azioni che sono state necessarie ai fini dell'applicazione.

Il primo passo per poter usare questo servizio è stato quello di creare la classe `AirportOntology`, all'interno della quale sono presenti gli schemi dei vari tipi di predicati (questi non sono stati utilizzati in realtà in nel progetto e verranno quindi tralasciati nel resto della trattazione), di azioni degli agenti e dei concetti. Questa classe rappresenta essenzialmente una collezione di schemi che tipicamente non cambiano durante la vita degli agenti. All'interno della classe `AirportOntology` vanno dichiarati tutti i concetti e tutte la azioni, ognuno dei quali è identificato da una stringa che ne contiene il nome. Nel metodo costruttore vanno poi creati tutti gli schemi dei concetti e della azioni, associando a ciascuno la propria classe. Ogni schema è formato da più slot, ognuno dei quali ha un nome e un tipo; uno slot può essere dichiarato `OPTIONAL`, e quindi il suo valore può essere anche `null`, altrimenti viene considerato `MANDATORY`, cioè obbligatorio e nel caso in cui non gli viene assegnato alcun valore, verrà generato un errore in fase di validazione; in oltre uno slot può avere cardinalità maggiore di 1, in quel caso è una lista di oggetti di un certo tipo. A titolo di esempio si riporta qui sotto il codice usato per definire i termini del concetto "Procedura", per specificare la classe che implementa il concetto e la dichiarazione della struttura dello schema:

```
...
// DICHIARAZIONE DEI TERMINI DEL VOCABOLARIO
public static final String PROCEDURE = "Procedure";
public static final String PROCEDURE_CODE = "code";
public static final String PROCEDURE_TASKS = "tasks";
...
/* AGGIUNTA DELLO SCHEMA DEL CONCETTO "PROCEDURE",
* CIOE' IL CONCETTO DI PROCEDURA; DA NOTARE CHE VIENE
* SPECIFICATA LA CLASSE */
add(new ConceptSchema(PROCEDURE),Procedure.class);
...
// STRUTTURA DELLO SCHEMA PER IL CONCETTO DI PROCEDURA
ConceptSchema cs = (ConceptSchema) getSchema(PROCEDURE);
cs.add(PROCEDURE_CODE,
(PrimitiveSchema)getSchema(BasicOntology.STRING),
ObjectSchema.MANDATORY);

cs.add(PROCEDURE_TASKS,(ConceptSchema)getSchema(TASK),
1,ObjectSchema.UNLIMITED);
...
```

in questo frammento di codice si vede che all'inizio vengono dichiarati i termini `Procedure`, `code` e `tasks` che serviranno per identificare i concetti

nelle fasi seguenti. I comandi successivi sono all'interno del costruttore della classe `AirportOntology` e il primo aggiunge a questa lo schema del concetto `PROCEDURE`, associandovi anche la classe che lo implementa. Infine viene data la struttura dello schema, nella quale vengono specificati i vari slot: il primo slot si chiama `PROCEDURE_CODE`, è di tipo `String` ed è obbligatorio; il secondo si chiama `PROCEDURE_TASKS`, è di tipo `Task` (il quale è stato dichiarato con gli stessi passaggi che stiamo qui presentando ma che viene tralasciato) ed è una lista perché gli ultimi due parametri dicono che ce ne possono essere più di uno (quindi è anche uno slot obbligatorio).

Quando viene aggiunto uno schema, abbiamo appena detto che bisogna anche associargli la classe che lo implementa. Chiaramente questa classe dovrà essere coerente con lo schema dichiarato e dovrà seguire varie regole, come ad esempio implementare la corretta interfaccia (`Concept` se è la classe che implementa un `ConceptSchema`, oppure `AgentAction` se implementa un `AgentActionSchema`). Sempre come esempio si riporta il codice della classe `Procedure`:

```
public class Procedure implements Concept
{
    private String code;
    private List tasks;

    public String getCode(){return code;}
    public void setCode(String code){this.code=code;}

    public List getTasks(){return tasks;}
    public void setTasks(List l){tasks = l;}
}
```

da notare subito che implementa l'interfaccia `Concept` dato che questa classe è associata ad un `ConceptSchema`; poi vengono dichiarate due variabili che hanno i nomi degli slot dichiarati in precedenza ed infine ci sono i metodi per andare a leggere e a scrivere le variabili. Per comodità, tutte le classi associate ai vari schemi vanno messe all'interno dello stesso pacchetto dove si trova la classe dell'ontologia; in questo progetto è stato creato ad esempio il pacchetto `airportOntology`, all'interno del quale sono presenti tutti i sorgenti che riguardano l'ontologia.

Quello appena presentato è il modo in cui si aggiungono tutti i concetti e le azioni degli agenti alla classe `AirportOntology`, per maggiori informazioni a riguardo si rimanda al materiale reperibile in rete.

5.4.2 Il contenuto dell'airport-ontology

Si andranno ora a descrivere i concetti e le azioni più importanti che sono state inserite all'interno dell'ontologia usata dal sistema, non si riporteranno

tutti in quanto la lista sarebbe molto lunga e alcuni dei concetti sono come blocchi elementari per costruire concetti più complessi e verranno quindi visti come parte di un altro concetto più grande:

- *Procedure*: questo concetto rappresenta chiaramente la procedura che il passeggero deve seguire all'andata o al ritorno. Al suo interno si trovano i campi *code*, che rappresenta il codice identificativo, e *tasks* che è invece una lista di oggetti di tipo `Task` i quali sono le singole operazioni che compongono la procedura;

- *Task*: questo concetto rappresenta una singola operazione che il passeggero deve fare all'interno dell'aeroporto. Tra i suoi vari campi troviamo *informations* che è una lista di oggetti `Information` i quali contengono il nome dell'oggetto dell'ontologia di cui si devono visualizzare le informazioni, poi c'è il campo *tips* che invece è la lista dei suggerimenti, ognuno rappresentato da un oggetto `Tip`. Sono presenti altri campi, come quelli per identificare la specifica operazione e la zona dove va eseguita;

- *TaskLive*: questo concetto è la versione "live" di quello precedente. Al suo interno troviamo tutte le informazioni che erano già presenti nel `Task` ma saranno inoltre presenti gli oggetti, come ad esempio `Flight` o `Checkin`, contenenti i dati veri e propri;

- *Flight*: questo concetto rappresenta il volo aereo. Al suo interno troviamo quindi campi come *number* che è il numero del volo, *from* e *to* che sono la città di partenza e di destinazione, *status* che è lo stato del volo (ad esempio in orario, in ritardo o cancellato), *airline* cioè la compagnia aerea e *terminal* che è un oggetto di tipo `Terminal` al cui interno ci sono informazioni sia sul terminal che sul gate di partenza.

- *Checkin, TourOperatorBank, BaggageReclaim, SecurityCheck, DutyFree*: questi sono solo alcuni dei concetti che rappresentano varie cose che sono presenti in aeroporto. Ognuno di essi ha informazioni diverse ma tutti hanno un campo *localization* dove si specifica il settore dove si trovano. Questa è un'informazione utile nel momento in cui bisogna dire all'utente se si trova nella zona corretta dell'aeroporto per eseguire una certa operazione;

- *CommercialActivity*: questo concetto rappresenta le varie attività commerciali che sono sparse per l'aeroporto, come bar, ristoranti e negozi. Tra i suoi campi troviamo *name* che è il nome dell'attività, *description* che contiene una breve descrizione, *location* che descrive a parole dove si trova, *openingTime* e *closingTime* che indicano l'orario di apertura e di chiusura rispettivamente;

- *Monitor*: questo concetto rappresenta i monitor informativi che sono presenti in aeroporto. Alcuni dei campi al suo interno sono *positionName* che è una stringa che indica la zona dove si trova (è necessario per capire il tipo di informazioni che deve visualizzare), *posX*, *posY* e *range* che danno le coordinate della posizione e il raggio d'azione entro il quale i dispositivi vengono rilevati;

- *FlightAlert*: questo concetto è in pratica la notifica che viene ricevuta dall'AirportInformationManager se ci sono state delle modifiche nei dati di volo (viene inviata dal DbChanger). Al suo interno ci sono due campi di tipo `String` che sono *sNumber* e *sDepartureDate* che servono per identificare il volo che ha subito variazioni e poi ci sono una serie di campi booleani come *departureTime* e *status* che se sono hanno il valore `true` significa che hanno subito variazioni e quindi l'AirportInformationManager deve andare a vedere nel database e inviare gli aggiornamenti agli agenti che ne avevano fatto richiesta;

- *RequestProcedure* e *ResponseProcedure*: queste sono due azioni che rappresentano rispettivamente la richiesta di download della procedura da parte di un agente MobileManager all'AirportInformationManager e la risposta di quest'ultimo. La richiesta contiene i campi *code* che indica il codice identificativo della procedura che si vuole ottenere e *sender* che è un oggetto di tipo `AID` ed indica appunto l'AID dell'agente che invia la richiesta; questo è necessario perché un `AgentAction` deve contenere l'azione e l'identià di chi deve compiere l'azione. Per quanto riguarda invece la risposta, al suo interno troviamo il campo *procedure* che è un oggetto `Procedure` e rappresenta appunto la procedura che è stata richiesta, ed infine il campo *receiver* che è di tipo `AID` ed indica il ricevente;

- *RequestLiveData* e *ResponseLiveData*: queste due azioni sono rispettivamente la richiesta e la risposta dei dati "live", tra l'agente MobileManager e AirportInformationManager. Nella richiesta è presente il campo *requestList* che è una lista di stringhe dove ognuna è il nome dell'oggetto di cui si vogliono ottenere le informazioni "live", poi abbiamo *personalCode* che il codice personale dell'utente che serve a recuperare i suoi dati di viaggio ed infine ancora il campo *sender* il cui significato è già stato spiegato prima. La risposta invece contiene molti campi come ad esempio *flight*, *terminal*, *checkin* e altri, ma nessuno di questi è obbligatorio; questi contengono i rispettivi oggetti, cioè il campo *flight* ad esempio è di tipo `Flight`, che a loro volta contengono i dati "live". Chiaramente saranno messi solo quei campi per cui è stata fatta richiesta; oltre a questi c'è sempre il campo *receiver*;

- *RequestFlightAlertSubscription*: questa è un'azione che serve a richiedere l'iscrizione al servizio di notifica per le modifiche che coinvolgono i dati del volo. I campi sono *queue* che in questo caso avrà il valore *Flight* e serve all'AirportInformationManager per smistare le richieste, poi c'è il campo *flightAlert* che è di tipo `FlightAlert` e al suo interno vengono specificati più nel preciso i campi di cui si vogliono ricevere gli aggiornamenti, ed infine il campo *applicant* che è l'AID del richiedente, così che quando avvengono modifiche l'AirportInformationManager sa a chi mandare le notifiche. Le richieste per ottenere il servizio di notifica per altri oggetti sono del tutto simili a questa.

Questi sono quindi solo alcuni tra i concetti e azioni presenti nell'ontologia creata ad-hoc per questo progetto. L'utilizzo di questa assieme al content language SL ha reso possibile inviare e ricevere informazioni strutturate in maniera comoda tra i vari agenti del sistema e si è quindi rivelato uno strumento di fondamentale importanza per realizzare le varie interazioni.

Capitolo 6

Conclusioni e sviluppi futuri

6.1 Conclusioni

All'inizio di questa tesi erano stati formulati diversi obiettivi da ottenere in questo progetto. Uno tra i più importanti era testare il paradigma ad agenti, applicandolo ad una problematica reale; in particolare si voleva vedere se le funzionalità offerte da un sistema ad agenti permettevano di ottenere risultati soddisfacenti ed essere in qualche modo un valore aggiunto per questo tipo di applicazioni.

Nello sviluppo di questo progetto, l'utilizzo del paradigma ad agenti per realizzare un sistema ubiquo è stato sicuramente di grande aiuto, in quanto esso si è adattato perfettamente alla necessità di modellizzare i vari componenti che andavano a formare l'architettura del sistema. In particolare, il sistema è formato da varie parti essenzialmente autonome che ben si prestano ad essere implementate attraverso gli agenti; essi infatti possono comunicare e collaborare tra loro in maniera semplice, ma allo stesso tempo eseguire in maniera autonoma tutta una serie di operazioni a discrezione dello sviluppatore. Queste funzioni possono essere aggiunte agli agenti sottoforma di behaviour, attraverso i quali è possibile andare a definire in qualsiasi dettaglio il funzionamento e il comportamento dell'agente stesso, sia per quanto riguarda le scelte autonome che dovrà compiere e sia per le interazioni che dovrà avere con gli altri agenti, al fine di portare a termini determinati compiti.

Un altro aspetto degli agenti che è stato sfruttato è la mobilità, ovvero la capacità di migrare, nel caso particolare di questo progetto, da un container all'altro. E' stato così possibile fare in modo che quegli agenti che contenevano informazioni importanti riguardanti i dispositivi mobili, potessero seguire in un certo senso lo spostamento di questi ultimi, migrando di area in area, portando con sè le informazioni che dovevano essere rese disponibili, evitando in questo modo una serie di interazioni che sarebbero state necessarie altrimenti per recuperarle. La mobilità degli agenti è stata anche

sfruttata per quanto riguarda l'invio delle informazioni da visualizzare nei monitor informativi: questo ha permesso di lasciare un agente fisso all'interno degli host che gestivano i monitor e inviare i vari agenti che avevano il compito di mostrare le informazioni i quali, una volta arrivati a destinazione, venivano gestiti dall'agente responsabile al fine di ottenere il permesso di accesso al monitor. Senza la mobilità sarebbero state necessarie numerose comunicazioni al fine di decidere quale agente aveva il permesso di utilizzare il monitor, complicando notevolmente la gestione del meccanismo.

Un altro obiettivo che si voleva ottenere era che il sistema fosse scalabile, considerato il numero di persone che mediamente transitano ogni giorno in un aeroporto, che chiaramente dipende anche dalla grandezza e dall'importanza dell'aeroporto stesso. Il paradigma ad agenti si è dimostrato ottimo anche sotto questo punto di vista: infatti, grazie al fatto che esso sfrutta il modello peer-to-peer nel quale tutti possono parlare con tutti, e grazie anche alla possibilità di decentrare le varie informazioni presenti, è stato possibile creare un sistema che almeno sulla carta dovrebbe avere un buon grado di scalabilità. Purtroppo non è stato possibile compiere dei test al fine di verificare sperimentalmente questa cosa, a causa della mancanza di un numero sufficiente di dispositivi; non è stato neanche possibile simulare un numero abbastanza grande di questi ultimi, in quanto il degrado delle prestazioni sarebbe stato dovuto non all'incapacità del sistema di rispondere ad un alto numero di richieste, ma al computer stesso che ospitava i vari agenti mobili simulati, in quanto essi richiedono comunque una certa quantità di risorse per poter essere eseguiti. Nonostante quindi l'impossibilità di verificarlo sperimentalmente, il sistema presenta tutte le caratteristiche per avere un buon grado di scalabilità, grazie soprattutto al decentramento dei dati che permette di risolvere la maggior parte delle richieste con poche interazioni; infatti, una volta terminata la fase successiva alla registrazione nel sistema di un nuovo dispositivo, nella quale vengono recuperate dalle varie componenti le informazioni necessarie, queste ultime vengono in un certo senso "portate vicino" al dispositivo che le dovrà utilizzare, in modo che le richieste successive si risolvano con poche interazioni e senza andare a interessare quegli agenti del sistema che possono rappresentare un collo di bottiglia, dato il grande numero di operazioni che devono svolgere.

Per quanto riguarda invece la piattaforma JADE, utilizzata per sviluppare il sistema ad agenti, essa si è rivelata molto buona sotto tutti i punti di vista. JADE è relativamente facile da imparare ad utilizzare, grazie alla documentazione presente in rete nel sito ufficiale del progetto e grazie alla mailinig list molto attiva, nella quale viene data risposta da parte degli esperti sui vari problemi che una persona può incontrare. L'unica vera difficoltà dal punto di vista di apprendere come funzionavano certe cose è stata dovuta al rilascio nel luglio del 2011 della nuova versione della piattaforma assieme anche all'estensione JADE-LEAP (nella quale è compresa la libreria JADE-Android che è stata utilizzata in questo progetto); in questa nuova

vesione era totalmente cambiato il modo in cui JADE-LEAP si fondeva con JADE, ma purtroppo la documentazione allegata faceva ancora riferimento alle versioni precedenti e questo ha causato non pochi problemi, risolti poi grazie all'aiuto degli sviluppatori della piattforma. A parte questo, la piattaforma JADE è un ottimo strumento per lo sviluppo di sistemi multi agente: la gestione degli agenti attraverso l'RMA (Remote Monitoring Agent) risulta facile e intuitiva, permettendo anche attraverso i vari tool messi a disposizione, di andare a risolvere i vari bug che man mano possono verificarsi. L'utilizzo dei container, che possono essere posizionati nei vari host del sistema distribuito, rende facile suddividere in aree l'intero sistema, con la possibilità inoltre di far migrare gli agenti da un container all'altro.

Un'altra piattaforma utilizzata è stata Android la quale, alla pari di JADE, ha una curva di apprendimento relativamente alta. Il fatto che il linguaggio utilizzato sia Java, semplifica notevolmente molti aspetti e permette ad uno sviluppatore di creare una semplice applicazione anche non conoscendo nei particolari la piattaforma. Oltre a questo, Android mette a disposizione un altissimo numero di API, che permettono di interagire con i vari componenti software e hardware, rendendo facili e veloci operazioni che possono essere anche complesse. La difficoltà principale è stata quella di riuscire a far comunicare l'agente di JADE all'interno dell'applicazione Android con la piattaforma JADE presente chiaramente in host diverso, questo a causa soprattutto di un'adeguata documentazione aggiornata, come è già stato detto prima.

In conclusione, il sistema ubiquo che si è ottenuto permetterà agli utenti, tramite l'utilizzo di un computer portatile o di uno smartphone, di ricevere informazioni sempre aggiornate e indicazioni varie sulle operazioni che dovranno compiere all'interno dell'aeroporto, sia quello di partenza che quello di arrivo. I dati saranno sempre disponibili nel sistema e si muoveranno assieme all'utente, lasciando che sia quest'ultimo a decidere quando farne uso; la presenza di monitor informativi permetteranno di ottenere svariate informazioni personalizzate, anche se il proprio dispositivo sarà in tasca o dentro una borsa (purchè ci si sia collegati al sistema e si sia abilitata questa funzione). Il sistema ubiquo, fondendo le informazioni date dall'utente assieme ad alcuni dati ambientali (come la posizione), produrrà delle indicazioni che verranno rese disponibili in qualsiasi momento all'utente finale, al fine di agevolare la permanenza di quest'ultimo all'interno dell'area aeroportuale.

6.2 Sviluppi futuri

Per quanto riguarda gli sviluppi futuri c'è da dire come prima cosa che le potenzialità di un sistema e di un'applicazione di questo tipo sono certamente numerose. Tra le cose più utili che si potrebbero sviluppare c'è una maggiore integrazione delle mappe con l'applicazione: in particolare potrebbe

essere molto comodo visualizzare una mappa nella quale si vedono i punti di interesse selezionabili e la propria posizione, così da rendere più facile lo spostamento all'interno dell'aeroporto, al fine di raggiungere una certa posizione. Sempre in questo ambito, sarebbe davvero ottimo essere guidati con delle indicazione stile "navigatore", alle varie posizioni dove devono essere fatte le varie operazioni, permettendo così a chiunque di sapersi orientare tranquillamente.

Oltre alle mappe, sarebbe interessante sviluppare un meccanismo che, sulla base dei tempi di attesa delle varie code, propone alternative su quello che una persona può fare, come ad esempio andare al bar o al ristorante per mangiare o passeggiare per i negozi, per poi ricevere una notifica quando i tempi di attesa sono diminuiti e quindi si consiglia di mettersi in fila. In questo modo si eviterebbero in una qualche misura le file, a volte chilometriche, che si presentano in vari punti dell'aeroporto.

Potrebbe capitare che il sistema vada in crash o che per qualche motivo non possa rispondere; in questi casi sarebbe utile un sistema di notifica delle informazioni importanti (come il cambiamento dell'orario dell'aereo) tramite SMS. Non appena il sistema non è più in grado di rispondere alle richieste degli utenti, entra in funzione un servizio che invia degli SMS agli utenti, informandoli come prima cosa dei problemi al sistema e che in caso di aggiornamenti importanti saranno inviati dei messaggi per informare i passeggeri, mettendo a disposizione di quest'ultimi anche un numero telefonico da chiamare e al quale risponderà un operatore in caso di bisogno di aiuto. Quando il sistema ricomincia a funzionare, il servizio SMS viene disabilitato e tutto ricomincia a funzionare nella consueta maniera. In questo modo, anche in caso di guasto che si può comunque verificare, i passeggeri non vengono "abbandonati", ma viene fornito loro un servizio che permette di rimanere aggiornati sugli eventuali cambiamenti.

Un'altra cosa interessante a cui si era pensato all'inizio di questo progetto ma che poi non è stata realizzata, è l'utilizzo di piccoli dispositivi da mettere all'interno dei bagagli a mano. Questi dispositivi inviano un segnale al telefono per informarlo che si trovano nelle vicinanze; nel momento in cui questo segnale non viene più rilevato, e quindi il bagaglio si trova ad una certa distanza dal proprietario, questo riceverà una notifica che lo avverte, così da assicurarsi che non si sia dimenticato il bagaglio da qualche parte.

Queste sono solo alcune delle migliorie che si potrebbero apportare al sistema sviluppato. Inoltre, al di là del particolare ambito in cui è stato utilizzato, questa tipologia di sistema è estremamente versatile e potrebbe essere utilizzata anche per fornire assistenza in altri ambienti, non necessariamente al chiuso ma anche ambienti esterni, alle persone che occasionalmente o ogni giorno, devono svolgere un qualche compito al loro interno, al fine di rendere più familiare un ambiente in parte o del tutto sconosciuto.

Elenco delle figure

Bibliografia

[1] *Aeroporto di Heathrow*. URL: http://www.heathrowairport.com/.

[2] *Aeroporto Valerio Catullo di Verona*. URL: http://www.aeroportoverona.it/.

[3] Fabio Luigi Bellifemine, Giovanni Caire e Dominic Greenwood. *Developing Multi-Agent Systems with JADE (Wiley Series in Agent Technology)*. 1ª ed. Wiley, 2007.

[4] E. Burnette. *Hello, Android: Introducing Google's Mobile*. 3ª ed. Pragmatic Bookshelf, 2010.

[5] J.F. DiMarzio. *Android: A Programmer's Guide*. 1ª ed. McGraw-Hill Osborne Media, 2008.

[6] *FIPA Communicative Act Library Specification*. URL: http://www.fipa.org/specs/fipa00037/SC00037J.html.

[7] *Introduction to JADE*. URL: http://jade.tilab.com/doc/html/intro.htm.

[8] *JADE Administration Tutorial*. URL: http://jade.tilab.com/doc/tutorials/JADEAdmin/index.html.

[9] *JADE Administrator's Guide*. URL: http://jade.tilab.com/doc/administratorsguide.pdf.

[10] *JADE ANDROID add-on guide*. URL: http://jade.tilab.com/doc/tutorials/JADE_ANDROID_Guide.pdf.

[11] *JADE API Documentation*. URL: http://jade.tilab.com/doc/api/index.html.

[12] *JADE Develop Forum*. URL: http://avalon.tilab.com/pipermail/jade-develop/.

[13] *JADE Programmer's Guide*. URL: http://jade.tilab.com/doc/programmersguide.pdf.

[14] *JADE Programming Tutorial*. URL: http://jade.tilab.com/doc/tutorials/JADEProgramming-Tutorial-for-beginners.pdf.

[15] T. Kindberg e A. Fox. "System software for ubiquitous computing". In: *Pervasive Computing, IEEE* 1.1 (2002), pp. 70–81.

[16] Mark L. Murphy. *Beginning Android 2*. 1ª ed. Apress, 2010.

[17] *Piattaforma Jade*. URL: `http://jade.tilab.com/`.

[18] Stefan Poslad. *Ubiquitous Computing: Smart Devices, Environments and Interactions*. 1ª ed. Wiley, 2009.

[19] Rick Rogers et al. *Android Application Development: Programming with the Google SDK*. 1ª ed. O'Reilly Media, 2009.

[20] D. Saha e A. Mukherjee. "Pervasive computing: a paradigm for the 21st century". In: *Computer* 36.3 (mar. 2003), pp. 25–31.

[21] *Specifiche FIPA*. URL: `http://www.fipa.org/index.html`.

[22] *Sviluppo su piattaforma Android*. URL: `http://developer.android.com/index.html`.

[23] Katia P. Sycara. "Multiagent Systems". In: *AI magazine* 19.2 (1998).

[24] Andrew S. Tanenbaum e Maarten Van Steen. *Sistemi distribuiti*. 2ª ed. Pearson Education, 2007.

[25] M. Weiser. "Hot topics-ubiquitous computing". In: *Computer* 26.10 (set. 1993), pp. 71–72.

Ringraziamenti

Come prima cosa voglio ringraziare con tutto il cuore i miei genitori per avermi permesso di raggiungere questo importante obiettivo, per avermi sostenuto e per aver sempre creduto in me durante questi anni. Un ringraziamento va anche a mio fratello Filippo, alla nonna Anna e a tutti gli altri nonni che non ci sono più, perché se ho raggiunto questo traguardo è anche grazie a loro.

Grazie ad Alessandra per essermi sempre stata vicino e per essere sempre stata presente quando ne ho avuto bisogno. Un grazie ad Andrea e Roberto, per aver condiviso assieme questi anni di studio e per l'amicizia che più volte mi hanno dimostrato.

Infine, voglio esprimere la mia più sincera gratitudine al mio relatore, il professor Carlo Ferrari, per la disponibilità e la presenza durante questo periodo di tesi.